Virtual Machine Image Management for Elastic Resource Usage in Grid Computing

Dissertation
zur Erlangung des Doktorgrades der Naturwissenschaften
(Dr. rer. nat.)

dem Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg
vorgelegt von

Niels Fallenbeck
geboren in Gießen

Marburg im Juni 2011

Vom Fachbereich Mathematik und Informatik der
Philipps-Universität Marburg als Dissertation am
9. Juni 2011
angenommen.

Erstgutachter: Prof. Dr. Bernd Freisleben, Philipps-Universität Marburg
Zweitgutachter: Prof. Dr. Matthew Smith, Leibniz Universität Hannover

Tag der mündlichen Prüfung am 26. September 2011.

Zusammenfassung

Grid Computing hat sich von einem im wissenschaftlichen Umfeld initiierten Konzept zu einem mächtigen Paradigma im Bereich des Höchstleistungsrechnens entwickelt. Während der letzten Jahre wurden Grid-Systeme entwickelt, welche die automatische Verteilung und Ausführung von Berechnungen auf (geografisch) verteilten Ressourcen ermöglichen. Diese Grid-Systeme haben in den letzten Jahren die Aufmerksamkeit immer mehr kommerzieller Anwender erregt. Um diesen Anwendern die Verarbeitung vertraulicher Daten in Grid-Systemen zu ermöglichen, müssen starke Sicherheitsmechanismen entwickelt werden, um diese Daten vor dem Zugriff unberechtigter Dritter zu schützen.

Im Gegensatz zu Grid Computing wurde das Konzept des Cloud Computing, welches im Jahr 2006 öffentlich vorgestellt wurde, von industriellen Anbietern entwickelt: Datensicherheit war von Anfang an ein wichtiger Aspekt. Um dies zu gewährleisten, wird Virtualisierungstechnologie eingesetzt, die eine sichere Trennung der Anwender ermöglicht, indem jeder Anwender eine eigene virtuelle Maschine zugeteilt bekommt, die verhindert, dass der Benutzer nur auf die in seiner virtuellen Maschine vorhandenen Daten zugreifen kann.

Um die Sicherheit in Grid-Systemen zu erhöhen, wurde daher früh die Verwendung von Virtualisierungstechnologie auch in diesem Umfeld untersucht. Dabei stellte sich heraus, dass dieser Ansatz vielversprechend ist, um die Sicherheitsanforderungen kommerzieller Anwender in Grid-Systemen zu erfüllen.

Ein Hauptteil der in dieser Dissertation vorgestellten Arbeit stellt die Image Creation Station (ICS) dar, eine Komponente, die es Anwendern erlaubt, ihre virtuellen Maschinen selbst zu erstellen und zu administrieren, und die für die automatische Verteilung dieser virtuellen Maschinen auf alle Ressourcen eines verteilten Systems verantwortlich ist.

Im Gegensatz zu Cloud Computing, das auch unerfahrenen Benutzern das Ausführen ihrer Programme in der Cloud einfach ermöglicht, sind Grid-Systeme auf Grund ihrer Komplexität deutlich schwieriger zu verwenden. Die ICS vereinfacht die Benutzung eines Grid-Systems, indem sie traditionelle Einschränkungen beseitigt. Dazu zählt vor allem die Installation eigener Software auf den Rechenknoten, die für die Ausführung der Berechnungen der Anwender verwendet werden. Dies war früher nur durch den Administrator der physikalischen Ressourcen möglich und musste für jede im verteilten System vorhandene Ressource wiederholt werden. Durch die ICS ist es Anwendern möglich, sogar kommerzielle Software in Grid-Systemen zu verwenden, ohne dass andere Anwender des Systems Zugriff auf die Software haben. Weiterhin findet eine Verschiebung der Administrationstätigkeit vom Administrator der physikalischen Ressourcen hin zu den Anwendern oder erfahrenen Kollegen statt und ermöglicht die Bereitstellung individuell angepasster virtueller Maschinen für jeden Anwendungszweck. Die ICS ist nicht nur für die Bereitstellung der Administrationswerkzeuge verantwortlich, sondern sorgt auch dafür, dass die verschiedenen virtuellen Maschinen auf allen dem verteilten System angeschlossenen physikalischen Ressourcen verwendet werden können.

Ein zweiter Aspekt der vorgestellten Lösung zielt auf die Elastizität des Systems, indem abhängig von der aktuellen Auslastung des Systems automatisch Ressourcen angefordert werden. Im Gegensatz zu bestehenden Systemen erlaubt die vorgestellte Lösung das Hinzufügen und Entfernen von Ressourcen, ohne die Ausführung von Berechnungen zu unterbrechen. Weiterhin können mit den präsentierten Komponenten nicht nur Grid-Ressourcen zur Durchführung von Berechnungen genutzt werden, sondern auch Cloud-Ressourcen dynamisch angefordert und verwendet werden, falls die vorhandenen Ressourcen nicht ausreichen, den Bedarf an Rechenkapazität zu decken. Durch das sofortige Herunterfahren nicht mehr benötigter Cloud-Ressourcen werden die Kosten für die Durchführung einer einzelnen Berechnung minimiert. Zusätzlich hat ein Anwender des präsentierten Systems die Möglichkeit, die Entscheidung zu beeinflussen, auf welchen Ressourcen seine Berechnungen durchgeführt werden. Dies ist entscheidend, wenn Daten verarbeitet werden müssen, die auf Grund gesetzlicher Bestimmungen oder sicherheitsrelevanter Bedenken die Firma oder das Land nicht verlassen dürfen. Um in heutigen Systemen ein vergleichbares Verhalten zu erreichen, ist der Anwender gezwungen, seine Berechnungen direkt an die entsprechenden Ressourcen zu senden und damit die Möglichkeit der automatischen Verteilung und optimalen Ressourcennutzung aufzugeben.

Weiterhin findet eine automatische Priorisierung der einzelnen Ressourcen statt, indem verschiedenene Metriken berücksichtigt werden, zum Beispiel das Maß an Vertrauen, das ein Anwender gegenüber einer bestimmten Ressource besitzt oder die Kosten für die Nutzung der Ressource. Die vorgestellte Lösung führt Berechnungen immer auf Ressourcen mit der höchstmöglichen Priorität aus. Oft spiegelt sich die Priorität einer Ressource in der physikalischen Distanz zwischen Anwender und Ressource wider, da der Anwender einem Cluster-System in der eigenen Firma üblicherweise mehr traut als Cloud Ressourcen und lokale Ressourcen überdies kostengünstiger zu nutzen sind. Daher minimiert die angewendete Verteilungs-Strategie die Kosten einer einzelnen Berechnung und die Gefahr, dass unberechtigte Dritte auf Daten zugreifen können, weil diese nicht notwendigerweise auf entfernte Ressourcen übertragen werden müssen.

Die in dieser Dissertation vorgestellten Komponenten ermöglichen den Aufbau eines System, das sich automatisch an die aktuelle Auslastung anpasst, indem es Grid und Cloud-Ressourcen zusammen mit lokal vorhandenen Ressourcen für die Durchführung von Berechnungen nutzen kann und jedem Anwender individuell angepasste virtuelle Ausführungsumgebungen zur Verfügung stellt.

Abstract

Grid Computing has evolved from an academic concept to a powerful paradigm in the area of high performance computing (HPC). Over the last few years, powerful Grid computing solutions were developed that allow the execution of computational tasks on distributed computing resources. Grid computing has recently attracted many commercial customers. To enable commercial customers to be able to execute sensitive data in the Grid, strong security mechanisms must be put in place to secure the customers' data.

In contrast, the development of Cloud Computing, which entered the scene in 2006, was driven by industry: it was designed with respect to security from the beginning. Virtualization technology is used to separate the users e.g., by putting the different users of a system inside a virtual machine, which prevents them from accessing other users' data.

The use of virtualization in the context of Grid computing has been examined early and was found to be a promising approach to counter the security threats that have appeared with commercial customers.

One main part of the work presented in this thesis is the Image Creation Station (ICS), a component which allows users to administer their virtual execution environments (virtual machines) themselves and which is responsible for managing and distributing the virtual machines in the entire system.

In contrast to Cloud computing, which was designed to allow even inexperienced users to execute their computational tasks in the Cloud easily, Grid computing is much more complex to use. The ICS makes it easier to use the Grid by overcoming traditional limitations like installing needed software on the compute nodes that users use to execute the computational tasks. This allows users to bring commercial software to the Grid for the first time, without the need for local administrators to install the software to computing nodes that are accessible by all users. Moreover, the administrative burden is shifted

from the local Grid site's administrator to the users or experienced software providers that allow the provision of individually tailored virtual machines to each user. But the ICS is not only responsible for enabling users to manage their virtual machines themselves, it also ensures that the virtual machines are available on every site that is part of the distributed Grid system.

A second aspect of the presented solution focuses on the elasticity of the system by automatically acquiring free external resources depending on the system's current workload. In contrast to existing systems, the presented approach allows the system's administrator to add or remove resource sets during runtime without needing to restart the entire system. Moreover, the presented solution allows users to not only use existing Grid resources but allows them to scale out to Cloud resources and use these resources on-demand. By ensuring that unused resources are shut down as soon as possible, the computational costs of a given task are minimized. In addition, the presented solution allows each user to specify which resources can be used to execute a particular job. This is useful when a job processes sensitive data e.g., that is not allowed to leave the company. To obtain a comparable function in today's systems, a user must submit her computational task to a particular resource set, losing the ability to automatically schedule if more than one set of resources can be used.

In addition, the proposed solution prioritizes each set of resources by taking different metrics into account (e.g. the level of trust or computational costs) and tries to schedule the job to resources with the highest priority first. It is notable that the priority often mimics the physical distance from the resources to the user: a locally available Cluster usually has a higher priority due to the high level of trust and the computational costs, that are usually lower than the costs of using Cloud resources. Therefore, this scheduling strategy minimizes the costs of job execution by improving security at the same time since data is not necessarily transferred to remote resources and the probability of attacks by malicious external users is minimized.

Bringing both components together results in a system that adapts automatically to the current workload by using external (e.g., Cloud) resources together with existing locally available resources or Grid sites and provides individually tailored virtual execution environments to the system's users.

Declaration

Ich versichere, dass ich meine Dissertation

> Virtual Machine Image Management for Elastic Resource Usage in Grid Computing

selbständig, ohne unerlaubte Hilfe angefertigt und mich dabei keiner anderen als der von mir ausdrücklich bezeichneten Quellen und Hilfen bedient habe. Die Dissertation wurde in der jetzigen oder einer ähnlichen Form noch bei keiner anderen Hochschule eingereicht und hat noch keinen sonstigen Prüfungszwecken gedient.

Niels Fallenbeck
Marburg im Juni 2011

Acknowledgements

First of all I want to thank Prof. Dr. Bernd Freisleben who has been my supervisor during my thesis. In many discussions he showed me new opportunities and ways to think. He always had an open ear whenever assistance was needed and influenced the way of thinking in work as well as in private. I am grateful that he gave me the opportunity to work in an international project framework and to get in contact with interesting people and exciting topics.

I also have to thank Prof. Dr. Matthew Smith for many interesting discussions and promising ideas not only during the course of this work. He paved the way for many ideas presented in this thesis.

Moreover, I want to thank all of my former and present colleagues from the Distributed Systems Group in Marburg. In alphabetical order I want to thank Lars Baumgärtner, Kay Dörnemann, Tim Dörnemann, Dr. Ralph Ewerth, Pablo Graubner, Christina Heitzer, Ernst Juhnke, Heiko Krause, Matthias Leinweber, Dorian Minarolli, Markus Mühling, Hans-Joachim Picht, Matthias Schmidt, Dr. Christian Schridde, Roland Schwarzkopf, Dominik Seiler and Dr. Thilo Stadelmann. Thanks to you all for numerous hours of interesting talks, lots of fun and friendship which cannot be taken for granted. Thanks for helping when time was running out sometimes, it was a great pleasure to work in projects together with you. Additionally, I have to thank Mechthild Kessler for great administrative support during my time at the university.

Finally, I want to thank my family, my parents, their significant others and my grandparents for great support during the last couple of years.

Last but certainly not least I wish to thank Steffi for backing me during the work on this thesis.

Thank you for outstanding years. Without you this work could not have been done.

Contents

1 Introduction

"When a distinguished but elderly scientist states that something is possible, he is almost certainly right. When he states that something is impossible, he is very probably wrong."

Clarke's First Law [36]

There have been dramatic changes in the computing landscape in recent years. Grid Computing has evolved from a theoretical construct at the end of the last century [75] to a serious computing paradigm today. The road of Grid Computing has led from the first theoretical papers over the use and development of Grid technology in academic environments to the first commercial adoptions of the Grid. In this context, the need for security has evolved, since user separation was not needed in scientific computing environments. High energy physicists used the batch resource managers side by side with chemists and mathematicians whose data was public and did not need to be secured in a special manner. But when computational resources are shared by commercial competitors, separation of the execution environments becomes crucial for the acceptance of the computing system.

This is one of the main reasons for the rapid success of Cloud computing whose most prominent representative, Amazon's Elastic Compute Cloud (EC2), was first announced as a limited beta in 2006 [114]. Amazon uses virtualization technology to provide a Virtual Machine (VM) exclusive to each of its customers. These VMs are hosted at Amazon's data centers, can be modified to each customer's needs and are used exclusively by the customer. The underlying Xen hypervisor [174] ensures a reliable and secure separation of

each user's processes and can provide resources on-demand. The customer pays only for the VMs that were active; thus, Cloud computing ideally fits into environments where the need for computational power is changing over time e.g., when peak loads arise.

Grid computing as well as Cloud computing has the potential to make high performance computational resources available for persons and companies who cannot usually afford hardware that offers comparable computing power. Hence, Grid and Cloud computing reduce the Total Cost of Ownership (TCO), not only by providing more affordable access to computing resources instead of forcing users to purchase hardware, but also by saving Information Technology (IT) management costs.

However, especially the use of Grid computing outside the open academic environment is hindered by two main factors: First, accessing the Grid and using Grid technology is cumbersome and does not yet fulfill the requirements of commercial users. Second, the Grid landscape lacks strong security mechanisms that commercial users need to secure sensitive information processing in the Grid. While network connections and data transfer between the different Grid sites are encrypted, user separation on the computational resources of the particular Grid sites is usually based on standard Operating System (OS) techniques, which do not, for example, prevent users from tracking other users' actions. The second main reason for the success of Amazon's EC2 is based on the fact that everyone can easily access Amazon's resources and start their first Cloud resources within minutes.

1.1 Motivation

Over the course of this research, many requirements were stated by industrial project partners that differ fundamentally from those of academic partners. Grid computing, developed and evolved in academic environments, has attracted the attention of commercial users in recent years. Cloud computing also came into play, developed in commercial environments and pushed forward by industrial companies. Both technologies, or paradigms, offer new possibilities for data storage and processing. At the writing of this thesis, Grid and Cloud computing have been established, but they are far from mature computing technologies.

Gartner published a collection of so-called "hype cycles" in 2010 to graphically represent the maturity (on the horizontal axis) and the adoption of technologies or applications (on the vertical axis). According to Gartner, the evolution of a certain technology is divided into five different phases. In the first phase, called "Technology Trigger," things get started due to a breakthrough in technology. After that, during the "Peak of Inflated Expectations," people or companies are very enthusiastic about this technology and take notice mainly

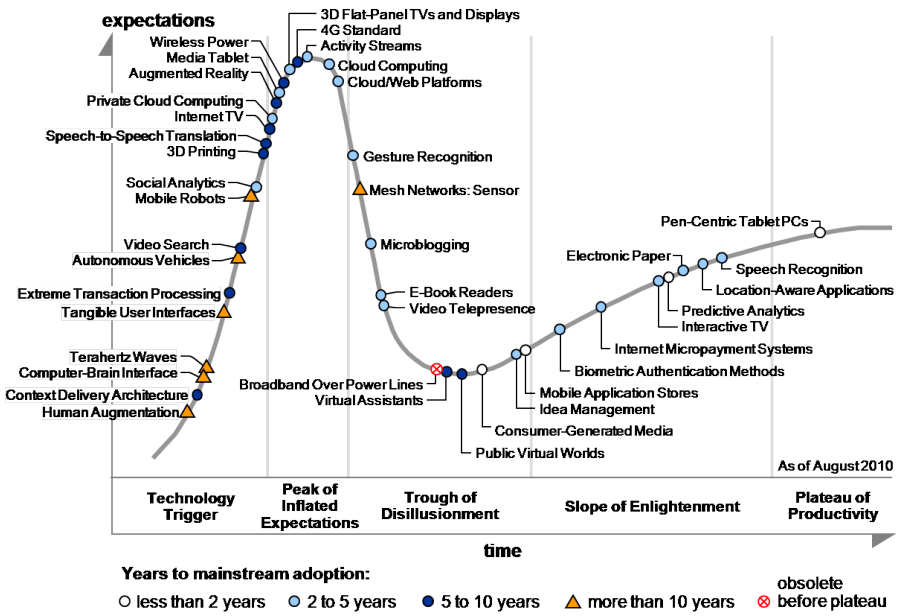


Figure 1.1: The Gartner Hype Cycle 2010 for emerging technologies sees Cloud computing "just topping the peak, and private cloud computing is still rising." (Source: `gartner.com`)

of advantages and opportunities surrounding that technology and the success stories related to it. After a while, people become more realistic about this technology and become more aware of its problems as well as the failures that others encounter while trying to use it. This phase of losing public perception is called the "Trough of Disillusionment." Afterwards, improvements are made to the technology and more use cases for its implementation become visible. This "Slope of Enlightenment" results in another raise in public interest. In this phase, technologies become more mature and are examined and used by an increasing number of people. A sign for reaching the "Plateau of Productivity" is the rise of mainstream offers of that particular technology. Figure 1.1[1] shows the hype cycle for emerging technologies. Analysts of Gartner place Cloud computing at the top of the second phase of this evolution. The first success yields critical questions about security and reliability. Some companies have started probing this new technology and found problems hindering the utilization of Cloud computing at this time.

Several projects within the research field of Grid and Cloud computing contributed to this thesis. Most of these projects focused on analyzing the requirements needed to successfully utilize these technologies in commercial and industrial environments. One topic of this work was to develop concepts and designs for new security and management mechanisms to lower the hurdle for potential users of Grid and Cloud technology. New administration concepts are necessary to counteract traditional concepts that are not suitable for the on-demand feature found especially in the Grid computing concept. For example, Grid users cannot usually install software on their own on the nodes used for

[1] Source of the Figure is a blog post published on 7th September 2010 by Gartner: `http://blogs.gartner.com/hypecyclebook/2010/09/07/2010-emerging-technologies-hype-cycle-is-here/`

computation. As in Cloud computing, tools are needed to enable users to do so and fit into the existing Grid infrastructure. By shifting the administration task from the resource provider to the customer, which is a key concept of Cloud computing, strong security concepts are necessary to eliminate the possibility of misusing either the customer's data or the provider's resources. However, Cloud computing is not suitable for every task, especially in High Performance Computing (HPC) environments. Because resources can be spread geographically, network performance may suffer and does not provide a bandwidth comparable to the network bandwidth available between Grid compute nodes. Furthermore, Cloud computing suffers from the so-called lock-in effect which prevents the easy exchange of both virtual machines and data between different Cloud providers or the Cloud customer's infrastructure. In contrast to Grid computing, no well-defined interfaces exist for inter-site data transfer.

The second topic within this thesis is the provisioning of components that can be used to link different resource sets together. While Cloud computing promises the availability of virtually unlimited resources on demand, users must pay to use these resources, usually in a pay-as-you-go manner. Most companies own resources that can be used to execute computational jobs. Traditionally, resources were planned on a large-scale to deal with peaks in demand. As a result, the resources were not used to capacity most of the time and money was wasted e.g., for maintenance and power. In addition, the initial investment was higher than needed on the average. This situation can be overcome using Cloud resources which can be acquired only as needed without requiring an initial investment. Cloud users only pay for the resources used and can deactivate unused instances to save money. Although Cloud resources can be created and accessed easily, tools should be developed to link the customers own infrastructure together with Cloud resources provided by an external resource provider.

1.2 Project Framework

The software and concepts presented in this work were developed in the framework of several projects as part of the German National Grid Initiative (D-Grid) [110] and the HPC initiative funded by the German Ministry of Education and Research (BMBF). These projects are in detail:

1.2.1 InGrid

InGrid[2] was a German D-Grid project started in 2005. The name is derived from the two major goals of the project: first, to bring **in**novative technology to the Grid and second, to make the Grid available to engineers (in German: **In**genieur). InGrid is based on a prototype application in the fields of coupled

[2] `http://www.ingrid-info.de`

multi-scale problems, coupled multidisciplinary simulations and distributed simulation-based optimization. Adaptive and scalable process models and runtime environments have been developed using Grid technologies to combine the competences in modeling, simulation and optimization for the common, efficient use of distributed resources. The project ended in 2008.

1.2.2 FinGrid

Increasing competition in the German banking sector has lead to increased pressure for restructuring and further automation in IT-related business processes in banks and financial service providers. In addition, new legal regulations such as Basel II/Basel III and changing customer needs in the direction of highly customized on-demand financial products enhance this pressure. To face these challenges, the Financial Business Grid (FinGrid)[3] project strives to identify suitable services and processes in the financial service sector and to develop Grid-based systems that enable financial service providers to reorganize their processes efficiently and to realize applications that have been impossible so far in terms of computational requirements. To guarantee relevance for the target industry, the projected research will be performed jointly with leading financial industry research partners. Three prototypes have been developed: Prototype I is a pricing and billing component for Grid-based services in cooperation with Deutsche Bank and IBM; Prototype II is a Grid-based customer portfolio performance measurement and management tool in cooperation with DataSynapse and Dresdner Bank/Commerzbank; and Prototype III is an asset-backed security factory based on Grid architecture in cooperation with FinanzIT and PA Consulting.

1.2.3 Biz2Grid

The main objective of Biz2Grid[4] is to provide foundations for an effective application of Grid technologies in enterprises. In order to achieve this goal, business and economically driven questions have to be answered and technical challenges have to be solved. Business models have to be developed that are adequate for an application in the Grid. In addition, Biz2Grid addresses technical research questions. Currently, the seamless adaption of applications to Grid technologies is hardly realizable. As a result, the project aims (i) to distribute and parallelize real world applications and (ii) to conceptualize and implement economic business models at the same time. The application case studies used in Biz2Grid research are supplied by BMW and iSILOG GmbH. One problem focuses on using the partner's in-house workstations effectively and flexibly. Since simulation jobs can be very time consuming, the industrial project partners are interested in elaborating idle workstations during the night, which promises to speed up job execution. In contrast to high perfor-

[3] http://www.fingrid.de
[4] http://www.biz2grid.de

mance compute nodes which are available all the time and are used only for job execution only, a workstation has a different usage profile: If it is used during the day by an employee, it may not be available for job execution. In addition, if a workstation is switched off, it cannot be used for job execution during the night, too. These characteristics and the fact that the simulation software used in this project needs the Windows OS prevent the use of the traditional batch schedulers that are used in HPC environments. Schedulers are needed that can deal with a dynamic computing landscape and are able to schedule computational jobs to Windows-driven compute nodes.

1.2.4 Plasma-Technology-Grid

The Plasma-Technology-Grid (PT-Grid)[5] project was started in May 2009 and aims to provide a flexible high performance computing landscape in the area of plasma physics. Components developed over the course of this thesis are used in the project to provide secure execution environments for distributed computational jobs. The project's goals are to provide a common infrastructure for the heterogeneous plasma community and to develop Grid-based tools for commercial project partners. Furthermore, a system should be developed for pricing and billing the developed Grid services within this project.

1.2.5 TIMaCS

The aim of this project is to provide "Tools for Intelligent System Management of Very Large Computing Systems" (TIMaCS)[6]. This project aims to provide an automated management and decision framework for large computing environments. Core components of the framework are monitoring components to observe all systems that are part of the framework and components that react to issues that arise within in the computing system, called events. The system that handles events automatically reacts to problems and delegates them to a system administrator if the system detects a problem that cannot be solved automatically. Since the system should be able to learn new strategies for solving the problems within a specific computing system, the administrator can propagate the solution to the system so that it will attempt to use the same (successful) solution if the problem arises again in the future. Furthermore, the framework should provide consolidation capabilities for e.g., automated migration of running virtual machines from overloaded physical machines to idle ones.

1.2.6 Further Cooperations

Close cooperation with the D-Grid Integration Project (DGI) and the community project MediGrid has been established to ensure that security requirements in the security working group of the D-Grid are exchanged. In the MediaGrid

[5] `http://www.pt-grid.de`
[6] `http://www.timacs.de`

project, tools are also used that were developed by the Distributed Systems Group in Marburg.

1.3 Contributions of this Thesis

The research contributions of this thesis are:

- The utilization of virtualization technology in traditional shared user environments, such as computing clusters in universities and research laboratories, leads to an increase in security by providing environments that can be used exclusively by a single user. This thesis provides the Image Creation Station (ICS), a component to manage Virtual Machines (VMs) in clusters as well as in distributed computing landscapes like a Grid or a Cloud. In addition to the security aspect, the utilization of virtualization technology enables users to install their own software in VMs that are used for job execution. Up until now, users usually could not influence the software available on clusters and Grid systems. Moreover, it was impossible to use commercial software in most cases. Providing self-administered VMs, users can decide on their own which software should be available on the compute nodes. The ICS facilitates software vendors that provide preconfigured VM images, which can easily be used by even inexperienced users.

  Using the ICS as a VM management component in Cloud environments, limitations like the lock-in effect, which prevents the use of virtual environments provided by one Cloud provider on resources belonging to a different provider, can be overcome.

- For efficient utilization of a distributed system, schedulers must be put in place to facilitate an optimized utilization of the available resources. While existing schedulers can usually only schedule jobs to a fixed set of resources, they cannot react to changes during runtime. In most cases, they must be restarted to adapt to a new situation, aborting running and pending jobs. Moreover, users have no influence on scheduling decisions with respect to their personal preferences e.g., if sensitive data is needed for job execution, users cannot specify the resources permitted for job execution.

  The Dispatch Daemon (dispatchd) has been developed to enable traditional systems to deal with resources that change dynamically over time. It extends the functionality of the widely used GridWay meta scheduler to enable the addition and removal of resources during runtime. The dispatchd it one of the main building blocks of a system called the “Elastic Onion”. With its user-centric design, a user represents the core of the Elastic Onion. The core is surrounded by different layers representing

different kinds of resources. While the lower layers represent resources available e.g., within the user's company, resources on outer layers are provided by Grid and Cloud providers. Users are not only able to deal with changing resources, they can specify on a job-by-job basis which layer can be used for job execution. The architecture of the Elastic Onion allows users to influence whether or not data can leave particular locations, such as the company's own server.

- Up until now, Cloud computing resources have not been easy to use in a distributed computing architecture, such as the Grid, for two reasons. First, most of the schedulers available cannot deal with a changing number of computing resources of the particular resource site (comparable to the lack of meta schedulers able to add new resource sites during runtime as described above). Moreover, these schedulers are not able to request new Cloud resources on demand and shut down these resources when they are no longer needed. GridWay, a widely used Grid scheduler, has been extended to deal with this new computing paradigm's peculiarities. While the Dispatch Daemon (dispatchd) can schedule to Cloud resources, an additional component, the Request Daemon (reqd), has been developed to automatically acquire new resources when needed. To minimize computational costs and energy consumption, reqd is also able to determine if particular resources are idle and decide to shut down these resources.

1.4 Published Papers

Several research papers have been published during the course of this research:

1. Niels Fallenbeck, Hans-Joachim Picht, Matthew Smith, and Bernd Freisleben. Xen and the Art of Cluster Scheduling. *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing in conjuncture with the ACM/IEEE Conference on Supercomputing*, pages 4–11, 2006

2. Matthew Smith, Matthias Schmidt, Niels Fallenbeck, Christian Schridde, and Bernd Freisleben. Optimising Security Configurations with Service Level Agreements. *Proceedings of the 7th International Conference on Optimization: Techniques and Applications (ICOTA7)*, ICOTA:367–368, 2007

3. Matthias Schmidt, Matthew Smith, Niels Fallenbeck, Hans-Joachim Picht, and Bernd Freisleben. Building a Demilitarized Zone with Data Encryption for Grid Environments. *Proceedings of First International Conference on Networks for Grid Applications*, pages 8–16, 2007

4. Matthew Smith, Matthias Schmidt, Niels Fallenbeck, Tim Dörnemann, Christian Schridde, and Bernd Freisleben. Secure On-demand Grid Computing. *Journal of Future Generation Computer Systems, Elsevier*, 25(3):315–325, 2009

5. Matthias Schmidt, Niels Fallenbeck, Kay Dörnemann, Roland Schwarzkopf, Tobias Pontz, Manfred Grauer, and Bernd Freisleben. *Aufbau einer virtualisierten Cluster-Umgebung*, pages 119–131. 2009

6. Roland Schwarzkopf, Matthias Schmidt, Niels Fallenbeck, and Bernd Freisleben. Multi-Layered Virtual Machines for Security Updates in Grid Environments. *Proceedings of 35th Euromicro Conference on Internet Technologies, Quality of Service and Applications (ITQSA)*, pages 563–570, 2009

7. Matthias Schmidt, Niels Fallenbeck, Matthew Smith, and Bernd Freisleben. Secure Service-Oriented Grid Computing with Public Virtual Worker Nodes. *Proceedings of 35th Euromicro Conference on Internet Technologies, Quality of Service and Applications (ITQSA)*, pages 555–562, 2009

8. Matthias Schmidt, Niels Fallenbeck, Matthew Smith, and Bernd Freisleben. Efficient Distribution of Virtual Machines for Cloud Computing. *Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2010), IEEE Press*, pages 567–574, 2010

9. Niels Fallenbeck, Matthias Schmidt, Roland Schwarzkopf, and Bernd Freisleben. Inter-Site Virtual Machine Image Transfer in Grids and Clouds. *Proceedings of the 2nd International ICST Conference on Cloud Computing (CloudComp 2010), Springer LNICST*, 2010

10. Eugen Volk, Jochen Buchholz, Stefan Wesner, Daniela Koudela, Matthias Schmidt, Niels Fallenbeck, Roland Schwarzkopf, Bernd Freisleben, Götz Isenmann, Jürgen Schwitalla, Marc Lohrer, Erich Focht, and Andreas Jeutter. Towards Intelligent Management of Very Large Computing Systems. *Proceedings of Competence in High Performance Computing (CiHPC)*, 2010

11. Matthias Schmidt, Niels Fallenbeck, Roland Schwarzkopf, and Bernd Freisleben. Virtualized Cluster Computing. In *Research Report High-Performance Computing in Hessen*, pages 147–148, 2010

12. Katharina Haselhorst, Matthias Schmidt, Roland Schwarzkopf, Niels Fallenbeck, and Bernd Freisleben. Efficient Storage Synchronization for Live Migration in Cloud Infrastructures. *Proceedings of the 19th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP), IEEE Press*, pages 511–518, 2011

1.5 Organization of this Thesis

The remainder of this thesis is organized as follows:

Chapter 2 introduces technologies which build the foundation of the work done in this thesis. Thereafter, three industrial use cases are presented which were used to carve out a list of requirements that need to be fulfilled by a distributed system embracing cluster and Cloud resources in commercial environments. The chapter closes with an overview of the work that already has been done in this area of research. In Chapter 3, an architecture is presented which fulfills the stated requirements. Some of the newly developed components are based on matured and widely-accepted software components used in distributed environments today, while others were newly developed with respect to the commercial requirements. Implementation details of the components are given in Chapter 4, before an evaluation of the proposed solution is presented in Chapter 5. Finally, Chapter 6 provides a conclusion of the work as well as an outlook for work that still needs to be done and research questions that must be examined in the future.

2 Background

"If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility... The computer utility could become the basis of a new and important industry."

John McCarthy, speaking at the MIT Centennial in 1961 [82]

2.1 Introduction

This chapter introduces the background of the work done during the course of this thesis. It presents the environment in which the work is takes place, carves out some of the problems from which the existing environment suffers, and sketches a possible solution to overcoming the presented weaknesses. It is subdivided into three main parts.

In the first part, fundamental technologies as well as definitions are introduced which build the foundation of the work presented in this thesis. A brief historical overview is given that presents the evolution of computer science from the early beginnings to the systems available today. Basic computing paradigms are presented as well as the underlying technologies that are vital for Grid and Cloud computing as known today.

The second part presents three cases for industrial use that are directly derived from industrial projects. Since these projects were located in different industrial sectors, each case puts different demands on the architecture of the

computer system to be developed. A catalog containing requirements derived from the cases will conclude this part of the chapter.

The last part will focus on existing work related to the research area within this thesis. Existing research papers will be presented in an overview and examined with respect to the requirements carved out in the previous section.

2.2 Fundamentals and Definitions

This section introduces fundamental terms and technologies that build the foundation for the work done in this thesis. First, Subsection 2.2.1 will give a brief overview of the history of Grid and Cloud computing and briefly introduce basic ideas and related approaches. Thereafter, definitions for both of the terms will be provided and the differences between these two will be presented.

In the second part, Subsection 2.2.2 will provide an overview of virtualization technology as a key enabler for Cloud computing nowadays. Its history will be sketched from the early years to the numerous virtualization concepts that can be found today. Some of these concepts will be presented briefly and fit into the existing landscape of Cloud computing. Moreover, the advantages and challenges associated with virtualization technology are presented as well.

Afterwards, Subsection 2.2.3 will present the Xen Grid Engine (XGE), a tool which is used to manage VMs on a physical compute cluster. The XGE can be used either in batch scheduling mode with an already installed Cluster Resource Manager (CRM) or as a stand-alone solution for resource provisioning.

2.2.1 History of Grid and Cloud Computing

The quotation at the beginning of this chapter was made by John McCarthy, a pioneer in Artificial Intelligence (AI) and inventor of the Lisp programming language, when he presented the idea of Utility Computing in 1961. Over the next two decades, big companies like banks rented computing power and storage from large data centers owned by IBM and other providers. Even companies like Western Union discussed providing services to its customers in 1965 [229]. This first phase of Utility Computing ended in the early 1980s when smaller companies were able to afford their own computer systems and no longer relied on the large mainframe computing systems in the data centers.

2.2.1.1 Metacomputing

In 1992, the term Metacomputing was coined in a paper by Smarr and Catlett [203]. The term refers to the concept of executing computational tasks on a virtual supercomputer that connects different resources. These resources embrace not only computational resources but also storage and special devices,

e.g. for visualization, and they can be geographically distributed and are interconnected with high-bandwidth network links. Using computer systems in such a way enables new classes of applications [16, 237] that were previously impossible. To use Metacomputing efficiently, many new challenges must be resolved. Resources must be efficiently selected and the system must be adjustable according to scale. Moreover, the system has to deal with a heterogeneous set of resources and their unpredictable structure, which usually belong to multiple administrative domains. Compared to today's challenges, all of these are still relevant for large systems that can be found today in the area of Grid and Cloud computing that promise the possibility of geographically distributed computation. Grid and Cloud systems are bigger than existing systems and may include several special purpose hardware which extend the system's heterogeneity.

2.2.1.2 Distributed Computing

The idea behind the concept of the distributed computing paradigm is to connect geographically distributed (computing) resources to a single (virtual) computer that provides more computing power than the particular resources that are part of it.

One of the main challenges a distributed computing system must deal with is a global storage system that provides access to data from everywhere in the system. In addition, the distributed computing software must provide concurrent execution of single computational tasks on the resources of the distributed system that is communicating over a network. As a prerequisite of high-performance execution on a distributed system, a program has to be split into fine-grained tasks that are loosely coupled and can be computed independently.

In contrast to parallel computing, the execution takes part on different systems that are interconnected by a high-bandwidth network, whereas the threads in a parallel computing environment are executed on different processors of a single computer system.

2.2.1.3 Service-oriented Computing

The paradigm of service orientation in IT is the encapsulation of functionality by defined services that can be (re)used by other software and services. Since functionality can be accessed by well-defined interfaces, a programmer can rearrange particular services to compose a new, more complex service.

The paradigm of reusability can also be found in the model of object-oriented computing, which is a basic design pattern in many modern programming languages.

2.2.1.4 Grid Computing

The Legion middleware [97] was presented in 1997. It envisions a "World-wide Virtual Computer" by following the object-oriented model. Everything in Legion was seen as an object with well-defined methods used to access storage or computing resources. The single virtual computer should be accessible without any knowledge of the underlying resources or the complexity of the system. However, Legion's Grid object model was not widely accepted and another middleware came to be the de-facto standard in Grid computing nowadays.

Ian Foster and Carl Kesselman presented the Globus Toolkit [74, 72], a middleware that offers not only computation management but also storage management, file staging abilities, monitoring and the implementation of security mechanisms for authentication, authorization and delegation. Usually developed as a Metacomputing middleware, it has become the de-facto standard middleware for Grid computing and also provides a toolkit for developing own Grid services. Globus had already proven its usability in the SF-Express project [24], which was the largest computer simulation of a military battle to time simulating 100,000 entities. Furthermore, the European DataGrid project[7] uses many of the Globus components and developed their own Globus-based middleware named gLite [130, 132]. This middleware is used to analyze experiments of the Large Hadron Collider (LHC) at the European Organization for Nuclear Research (CERN). The gLite middleware has also been evaluated in the EGEE [81] project, which provided the world's biggest Grid testbed to date. Besides Globus and gLite, a third middleware, UNICORE [62], has been developed that mainly targets HPC resources.

The term "Grid computing" itself appeared the first time in the late 1990's. It depicts the idea of making computing power and storage as easily available as electricity is provided by the power grid. It was coined by Carl Kesselman and Ian Foster in their work "The Grid: Blueprint for a new computing infrastructure" [123]. Grid computing combines the ideas of object-oriented computing, cluster computing, distributed computing and service-oriented approaches to provide uniform access to the underlying heterogeneous computing components which are offered by resource providers as computing elements of the Grid [76]. Foster presented a three-point checklist [71] of the main attributes established in a Grid system. In his point of view a Grid system ...

1. ... coordinates resources that are not subject to centralized control
2. ... using standard, open, general-purpose protocols and interfaces
3. ... to deliver nontrivial qualities of service.

[7] `http://eu-datagrid.web.cern.ch/`

In his paper, Foster not only names these three attributes but also provides an explanation of these points. First, a Grid system coordinates "resources and users that live within different control domains – for example, the user's desktop vs. central computing; different administrative units of the same company; or different companies; and addresses the issues of security, policy, payment, membership, and so forth that arise in these settings."[8] Second, a "Grid is built from multi-purpose protocols and interfaces that address such fundamental issues as authentication, authorization, resource discovery, and resource access."[9] It is crucial to build upon standard and open protocols otherwise "we are dealing with an application-specific system"[10]. Finally, it is worth noting that a Grid does not deliver trivial qualities of service but "allows its constituent resources to be used in a coordinated fashion to deliver various qualities of service, relating for example to response time, throughput, availability, and security, and/or co-allocation of multiple resource types to meet complex user demands, so that the utility of the combined system is significantly greater than that of the sum of its parts."[11]

These attributes were summed up by Kesselman and Foster [123] leading to the following definition for Grid Computing:

> A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.

Since Grid computing relies heavily on legacy schedulers that distribute computational tasks to the resources provided by the different sites within the Grid system, the wide acceptance of Grid computing has led to increased pressure to respect a Grid's specific needs. Schedulers like Condor [78, 224], Nimrod [26, 3], Sun Grid Engine [85, 86] (which has been renamed Oracle Grid Engine[12] after the acquisition of Sun by Oracle), and Platform Computing LSF [246] have been implemented for Grid computing. To allow the scheduling of computational jobs across multiple Grid sites, meta schedulers like GridWay have been developed [107]. The scheduler queries the current state of different Grid sites on a regular basis and schedules jobs to the Grid site which fulfills the needs of each job.

Although it is originated in the academic world, the Grid was also recognized in the commercial world. Many projects have been started bringing commercial project partners together with partners from universities and publicly-funded research centers to examine the requirements needed for a successful commercial adoption of the Grid. In the European Union (EU), the BEin-

[8] Ian Foster. What is the Grid? - A Three Point Checklist. *GRIDtoday*, 1(6), 2002, p. 2
[9] Ian Foster. What is the Grid? - A Three Point Checklist. *GRIDtoday*, 1(6), 2002, p. 2
[10] Ian Foster. What is the Grid? - A Three Point Checklist. *GRIDtoday*, 1(6), 2002, p. 2
[11] Ian Foster. What is the Grid? - A Three Point Checklist. *GRIDtoday*, 1(6), 2002, p. 3
[12] `http://gridengine.sunsource.net/`

GRID project[13] [49] was launched with no less than 25 business experiments that were carried out in industrial key sectors. In the D-Grid, the possibility of using Grid technology in different sectors of the industry has also been examined: The InGrid project[14], started in 2005 and completed in 2008, focused on bringing engineering technology to the Grid and providing Small and Medium Enterprises (SME) from the engineering sector ways to use Grid technology. The FinGrid project[15] [104] has proven the general feasibility of commercial partners within the finance sector utilizing Grid technology by developing three software prototypes. The automotive sector, as one key sector of German industry, evaluated the Grid in the Biz2Grid project[16]. IBM Research & Development Böblingen provides another example of how the co-operation of a publicly-funded Grid infrastructure and commercial middleware has been successfully demonstrated.

Additionally, the term Virtual Organization (VO) was used in the Grid computing community to handle a potentially high number of users belonging to particular organizations using the Grid. Resource providers are able to assign permissions to access their resources based on the VO membership of a particular user.

2.2.1.5 Cloud Computing

The term "Cloud computing" entered the scene in 2007, when several notable companies like Amazon, International Business Machines Corporation (IBM), Google, Apple and Microsoft, inspired by the Grid paradigm, started offering computing and storage resources to customers on-demand [241, 114]. The term was derived from the common depiction of transparent networks of resources as a Cloud. It is controversial who coined the term Cloud computing. It was borrowed from the telecommunication industry, which began offering Virtual Private Network (VPN) services with service comparable to the former point-to-point connections but at a much lower cost. A cloud symbol was used to differentiate between the administrative domain of the customer's infrastructure and that of the telecommunication company. It is undisputed, however, that Eric Schmidt, CEO of Google, described Google's approach of Software as a Service (SaaS) during the Search Engine Strategies Conference on August 9th, 2009 as Cloud computing [139].

Aaron Weiss examines the term Cloud computing and its different occurrences in the field [241]. It is no surprise that Weiss found that Cloud computing encompasses many different scenarios ranging from Web-based applications over a Grid that charges rates for processing time to distributed and parallel computing environments. Furthermore, Google's approach of using

[13] `http://www.beingrid.eu/`
[14] `http://www.ingrid-info.de`
[15] `http://www.fingrid.de`
[16] `http://www.biz2grid.de`

a large numbers of commodity hardware instead of a small number of high-performance and costly servers constitutes the idea of Cloud computing. In contrast to a bunch of machines, Weiss states that the Cloud is a "more cohesive entity."

There are many Cloud definitions around used in daily discussions that differ depending on how views Cloud computing itself [231, 193]. A technical report from the University in California in Berkeley [9] provides a detailed view of Cloud computing. The main characteristic of Cloud computing is its focus on virtualization, which it uses to hide the underlying physical resources from the user. Clouds provide scalability, which enables one to dynamically adapt to workloads that differ over time. Access to resources and management functionality is provided to the customers through a Service Oriented Architecture (SOA).

Cloud resources, such as clusters and storage servers, are typically operated by third-party providers and are rented on-demand. Consumers should perceive resources as a service and should not have to be concerned with the underlying technology used to meet computational and storage demands. A Cloud computing provider can offer resources from different layers to its customers.

Application Layer. Delivering Software over the Internet is called Software as a Service (SaaS). It allows customers to use the provided software through a standard web browser without having to install additional software locally. Examples are Google Docs, SalesForce.com customer relationship management software, and SAP Business by Design.

Platform Layer. Services on this layer are usually offered as Platform as a Service (PaaS) and provide access to an application domain-specific platform, which can be used to develop and host the user's own applications. During the development of an application customers are usually bound to some restrictions, e.g. the use of a particular programming language. On the other hand, these restrictions facilitate an autonomous adaption of the current workload and enable automatic up- and downscaling of resources. Well-known examples of platform layer Cloud services are Google AppEngine, Force.com and Microsoft Azure.

Infrastructure Layer. Providing Cloud services on the infrastructural layer is often referred to as Infrastructure as a Service (IaaS). Infrastructure service providers offer access to resources that look like physical hardware. Amazon Elastic Compute Cloud (EC2) – probably the best-known example for Compute Cloud service – and Zimory provide VMs to its customers that can be used like normal physical machines. Moreover, on this layer, Data and Storage Clouds offering reliable access to data

of potentially dynamic size. Examples are Amazon Simple Storage Service (S3) and SQL Azure.

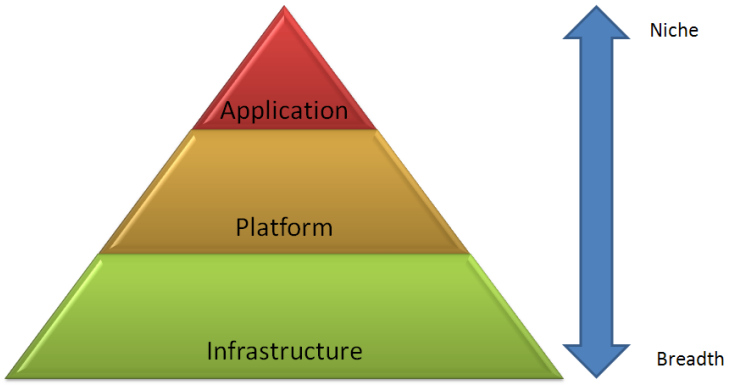


Figure 2.1: The Cloud computing pyramid [202] shows the different layers of Cloud computing and their interrelationship.

The Cloud computing pyramid shown in Figure 2.1 was proposed by Michael Sheehan in 2008 [202]. It depicts the interrelationship of the different Cloud layers introduced above. The infrastructure layer is the foundation of the Cloud and allows a generic operational capability by providing plain (computational) resources a customer can use to satisfy her needs. The platform layer is placed upon the infrastructure layer. Instead of providing a general service, it is more specialized in terms of usage. As stated above, Cloud providers may offer a platform that customers can use to develop their own applications, but it may not be used to host special software utilities. The application layer is placed on top of the pyramid, above the platform layer. In this layer, specific applications that are tailored to a specific kind of service are provided to the Cloud customers, e.g. customer relationship management software or email services.

In 2010, the European Commission Expert Group Report [99] examined the recent developments in the area of Cloud computing and analyzed the Cloud computing paradigm with respect to non-functional, economical and technological aspects:

> The concept of Cloud computing is linked intimately with those of IaaS (Infrastructure as a Service), PaaS (Platform as a Service), SaaS (Software as a Service) and collectively *aaS (Everything as a Service) all of which imply a service-oriented architecture.

The authors of that report state that several particular characteristics distinguish Cloud computing from classical resource and service provisioning environments that were described in previous sections of this chapter:

1. It is (more or less) infinitely scalable.
2. It provides either an infrastructure for platforms, a platform for applications, or applications themselves.

3. It can be used for every purpose ranging from disaster recovery and business continuity to a fully outsourced Information and Communications Technology (ICT) service.

4. It shifts the costs from Capital Expenditure (CAPEX) to Operational Expenditure (OPEX), which allows finer control of expenditure and makes acquisition of costly assets and maintenance unnecessary.

5. It exploits an already existing large-scale infrastructure available from today's Cloud resource providers.

6. Cloud offerings vary greatly and do not provide standardized interfaces.

7. Providers essentially provide data centers for outsourcing.

8. There are security concerns regarding a company's (sensitive) data stored in the Cloud.

9. There are concerns about the availability and reliability of Cloud services.

10. There are concerns regarding data being transported over anticipated broadband speeds.

Moreover, it is important to distinguish between public Clouds such as SalesForce.com or Amazon EC2 and private Clouds, which are not accessible to the public but to a limited group of customers. One scenario in which private Clouds are often used are big companies in which the company's IT department runs a private Cloud that is only accessible to other departments within the company. The mixed utilization of a public and a private Cloud is called hybrid Cloud. In a hybrid Cloud, public Cloud resources can be used to deal with peak loads that cannot be handled by the available private Cloud resources.

The National Institute of Standards and Technology (NIST) has tried to respect the different characteristics and provides a definition for the term Cloud Computing[17] in a frequently updated document [141]. A short version of this definition is as follows:

> Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

Cloud computing has gained lot of attention in recent months [116, 243]. In the first chapter, the Gartner hype cycle was presented in Figure 1.1 on page 17. Gartner analysts place Cloud computing beyond the "Peak of Inflated

[17] `http://csrc.nist.gov/groups/SNS/cloud-computing/`

Expectations" in transition to the third phase of the five-phase evolution of a technology from its early beginning to the mainstream adoption. Gartner believes that Cloud computing will reach the last phase sometime in the next two to five years. Buyya et al. [28] state that "there is an increasingly perceived vision that computing will one day be the 5th utility (after water, electricity, gas, and telephony)." With its latest paradigm – Cloud computing – a basic level of computing service is provided which is needed to fulfill the general community's necessity.

Nevertheless, with the development of Cloud computing and its wide acceptance, new challenges have appeared. Several publications focus on open problems and suggest future research topics. Giordanelli and Mastroianni [88] state that Cloud services must be offered in a standardized manner to the customers and sufficient security mechanisms must be provided to protect the customers' data in the Cloud. The Cloud Security Alliance also focuses on security aspects [1, 2]. In addition to the afore-mentioned topics, they see additional threats like the abuse of Cloud resources, an insecure programming Application Programming Interface (API) provided to Cloud customers, malicious insiders and vulnerabilities in shared technologies which give unauthorized users access to other users' data. Zhang et al. [253] also point to power management technologies as an open research issue that must be solved to let the vision of the Green Cloud come true.

2.2.1.5.1 Differences to Grid Computing Grid computing is often named in the same context as Cloud computing. At first glance, one might get the impression that researcher worked in the area of Grid computing just changed the label to Cloud computing after the term was coined by some of the industry's global players[18] [139]. Depending on the point of view, Cloud computing can be compared with Grid computing. When focusing on parallel execution of numerous computational tasks, Grid computing can be used as well as Cloud computing, but the latter uses another approach to reach this goal. While Grid computing provides a standardized interface for job submission, Cloud computing providers offer bare VMs with a minimal Linux OS installation. Users can log into the rented virtual machines and install the desired software. In contrast to Grid computing, the Cloud resource providers do not provide a scheduler to distribute the work to different virtual machines. This offers much more flexibility to the user for installing software on external resources with the possibility of powering up more resources whenever more computational power is needed. On the other hand, users are responsible for achieving ideal utilization of the rented resources on their own.

An approach to differentiating between the terms Grid computing and Cloud computing can be found in the article by Jha, Merzky and Fox [115]. They

[18] People who stated this often based it on the assumption that public funding now moves to the next big thing: the Cloud.

come to the conclusion that "Clouds can be viewed as a logical and next higher-level abstraction from Grids." They argue that Grid has missed the goals it stated in the early days [123] for two main reasons. First, the interoperability across different sites is difficult to achieve due to significant variance of the run-time and programming environments on the one hand and the difficult management of application level control among different Virtual Organizations (VOs) on the other hand. The second reason is that the Grid is difficult to use for both users and application developers. The authors see the utilization of an additional abstraction layer (virtualization) as a necessity for a (Cloud) system's acceptance and utilization.

Another approach by Weinhardt et al. [240] carves out the differences between Grid and Cloud computing by focusing on a number of essential criteria that can be used to distinguish the two paradigms. Discussing the criteria of virtualization, that is mature and essential for Cloud computing, its utilization in Grid environments is in the beginnings. While Cloud computing is easy to use and focuses on interactive applications accessible by standard web protocols, Grid computing is more complicated to use and aims to execute batch jobs using a Grid middleware. In contrast to Grids, which are not centrally controlled and rely on shared resources between its participants, Clouds are centrally controlled by a company which sells its service to customers. Based on open protocols, switching costs in Grids are low due to standardization while switching a Cloud provider is usually more complicated due to the fact, that no standardized interfaces exist and incompatibilities between the providers are common.

Vaquero et al. [231] compared the characteristics of the Cloud to the characteristics of the Grid. In fact, there are some similarities between both concepts, but also some essential differences:

Resource Sharing. While Grid Computing is about resource sharing between different sites or providers and the collaboration between them, a Cloud is typically under centralized management.[19] In fact, there is no contemporary Cloud computing provider that supports collaboration with Cloud resources hosted by another provider. This phenomenon is called Data Lock-in. Grids, as developed in academia, were designed to connect resources physically distributed among several administrative domains, while Clouds are managed by a single company. Thus, their resources are usually located in one huge data center or a few different data centers belonging to the same company.

Virtualization and Security. In Grid Computing, data and computing resources are virtualized by a middleware layer while the Cloud adds infrastructure virtualization to this list to not only raise security by shielding users'

[19] That is an opposition to the Grid definition given by foster in [71].

from one another but also to enable the management functionality offered by virtualization technology (e.g., scalability). Within the last few years, much work has been done to bring infrastructure virtualization to the Grid as well [118, 120].

High Level Services. High level services are available in the Grid because they allow users to exchange data and services with other sites. The Grid infrastructure is based on open standards published by the Open Grid Forum (OGF) or the Organization for the Advancement of Structured Information Standards (OASIS), which have become an important characteristic for Grid computing systems [22]. Cloud infrastructures do not provide standardized interfaces but usually rely on proprietary protocols hindering the exchange of data between Cloud providers and resource providers in general.

Architecture. While Grid computing offers a service-oriented architecture to its users, Cloud computing's architecture can be usually selected by the user, especially when using IaaS Cloud models. Every user can install the resources acquired from the Cloud provider by herself. If a Grid middleware is installed, Cloud resources can also be used in a Grid infrastructure.

Usability. One of the main reasons that Clouds have received widespread attention is because Clouds are easy to use. Often, a user simply needs to register at the Cloud provider's website and shortly thereafter, she can use the Cloud. In contrast, Grid computing requires its users to put more effort into learning how to make the their software work in the Grid. For example, a user does not need to register at a single site, rather she must request a certificate at a Certificate Authority (CA), import it to the software she wishes to use in the Grid, and learn how to submit computational jobs to Grid sites and transfer data into the Grid. The reasons for the differences in usability are manifold. One central point is that the industry developed Cloud computing to attract commercial users to use the Cloud for computational services; thus, it must be user-friendly.

There are even more differences between Cloud and Grid computing, but there are also many similarities. Cloud computing is a relatively new concept and will be transformed by some means or other over time. Much of the effort invested in problem solving in Grid computing may also solve the same (or similar) problems arising in Cloud computing in the future. For now, Cloud computing providers run their resources in a huge data center individually. By using proprietary protocols between the desired resources and by not providing interfaces that implement open standards as the Grid community does, Cloud providers bind their users to their own resources. Interoperability among other Cloud providers' resources is not (yet) supported. While this is understand-

able from an economic point of view, it contradicts the paradigm of the user's unawareness of where a job is executed. If Clouds are able to interact with other Clouds, any changes made in the Grid environment may be adapted to ensure a reliable interconnection between the resource sites. Nevertheless, it is important to hide the complexity found in the Grid from the customers.

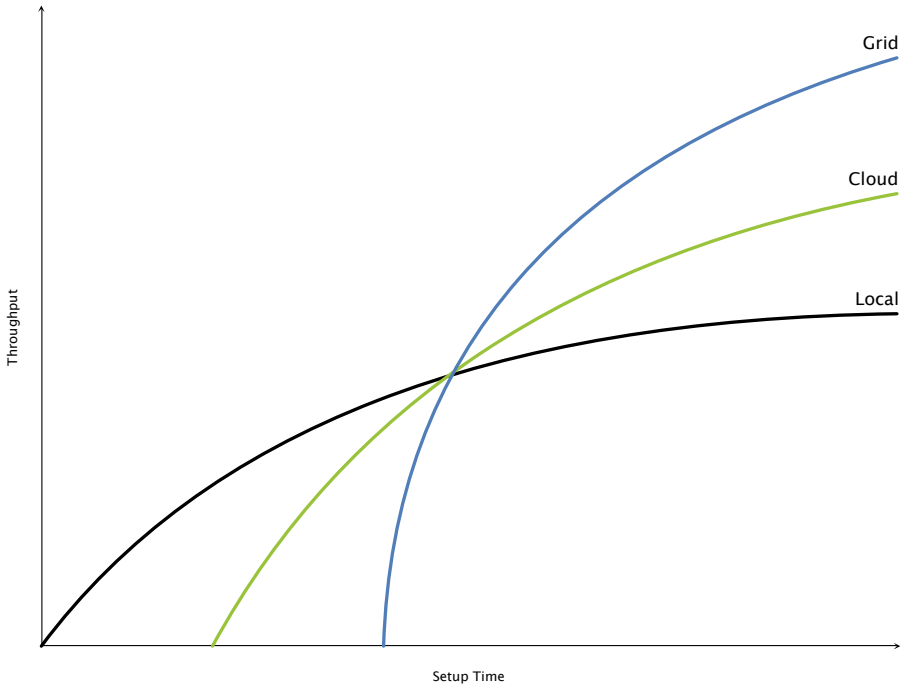


Figure 2.2: Job execution on a local machine compared to job execution in the Cloud and in the Grid.

In March 2010, Uwe Schwiegelshohn, Chairman of the D-Grid, the National Grid Initiative (NGI) of Germany, held the keynote speech at the D-Grid All-Hands-Meeting (AHM) in Dresden and presented the picture shown in Figure 2.2. In this figure, executing computational jobs on the local host is compared with executing jobs in the Cloud and in the Grid. While local job execution does not require any *Setup Time*,[20] the *Throughput* is limited by the local computational resources. When tied to the local machine, a user is not able to obtain additional CPU cores and is limited to the memory available on the local system. In contrast, a Cloud user can create numerous additional computing instances on demand to start another bunch of computational jobs at the same time. Some time is needed to initially create a computing instance and set it up to the user's needs. Once this is done, it can be duplicated to make new computing resources instantly available. Nevertheless, parallel jobs which make extensive use of communication between the processes[21] will suffer heavy performance loss when executed in the Cloud [238, 153]. The Grid offers an environment which provides the highest performance for communication by providing access to traditional high performance computing resources

[20] Setup Time denotes the time needed before a job can be executed by the user, e.g. the time needed to install software needed or to set up a particular environment.
[21] The Message Passing Interface (MPI) protocol has been developed for inter-process communication among several physical compute nodes connected via a network.

such as clusters.[22] To access these resources, a user has to use a complex Grid infrastructure provided by Grid middlewares[23] which is secured by a certificate infrastructure. Before a user can submit any computing job to a Grid site, the user must obtain a Grid certificate from a CA. Usually, the user must appear at the CA in person and show her identity card in order to acquire the needed certificate. This process may be time consuming and the job submission is also more complex than logging into a Cloud resource and starting a shell script. This effort can be neglected if massive parallel jobs need to be run which need fast network connections between the computing resources that are executing the particular jobs.

2.2.2 Virtualization

As mentioned above, virtualization is one of the main building blocks of Cloud computing systems, which, in the meantime, is also used in Grid computing.

In an information technology context, virtualization refers to the methodology of abstracting applications from the physical resources of a computer system by either partitioning a physical entity into multiple virtual entities or combining multiple physical entities into a single virtual entity. The essay released in 1959 called "Time Sharing in Large Fast Computers" [219], which was published by Christopher Strachey, can be seen as the foundation of virtualization from today's view. Strachey propagates the idea that a Central Processing Unit (CPU) could be utilized more effectively if it was shared between running programs. Until then, it was common for every program to run in a strictly sequential manner, even if the CPU had to wait for I/O. For that reason, the state of a CPU must be saved and restored when switching from one program to another. This context switch as well as the partitioning of the main memory [11] made it possible to run programs in a pseudo parallel fashion.

In the early 1960s, the two computer scientists Robert Goldberg and Gerald Popek investigated the aims of virtualization. Goldberg stated in his PhD thesis "Architectural Principles for Virtual Computer Systems" in February 1973 [91] that host computer systems must fulfill some basic requirements to be able to execute VMs:

1. The methods of executing non-privileged instructions in both supervisor and problem state must be roughly equivalent for a large subset of the instruction repertoire.

2. A method for protecting the supervisor (and any other virtual machine if there are multiple programs running) from the active virtual machine

[22] Typically the compute nodes on a Grid site are interconnected with Gigabit Ethernet or InfiniBand (IB) devices.
[23] The most common used middlewares in the D-Grid are Globus Toolkit 4 (GT4), gLite and UNICORE.

must be available. This may be accomplished, for example, through a protection system or an address translation system.

3. A method of automatically signaling the supervisor when the virtual machine attempts to execute a sensitive instruction must be available. The trap must not cause unrecoverable errors. It must then be possible for the supervisor to simulate the effect of the instruction. Sensitive instructions include:

   (a) Those instructions which alter or query the state of the machine, e.g. "Is it in supervisor state or problem state?" or "Is it in relocate mode or not?"

   (b) Those instructions which alter or query the state of the machine's reserved registers and core locations.

   (c) Those instructions which reference the storage protection mechanism, the memory system, or anything else that is specifically used by the Virtual Machine Monitor (VMM) for building and managing the virtual machine.

   (d) Any I/O instruction.

However, Goldberg also notes, that "In some case, a machine which violates hardware virtualization rules may still be able to support an HVM"[24] [91, p. 54]. Goldberg defines the two types of virtualization shown in Figure 2.3.

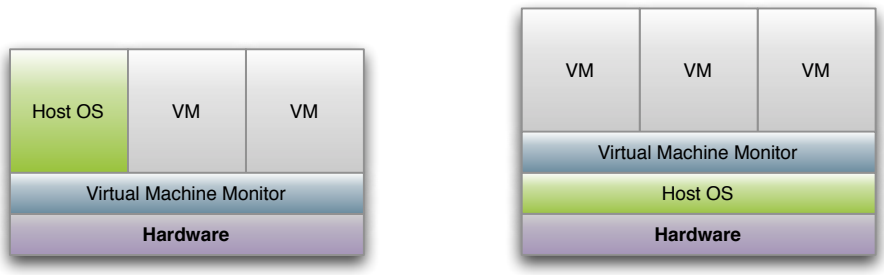


Figure 2.3: Two different types of virtualization exist. In Type I virtualization, the VMM runs directly on top of the hardware and the host OS runs on top of the hypervisor, while in Type II virtualization, the VMM runs on top of the host OS.

In Type I virtualization, the hypervisor (VMM) runs directly on top of the hardware, called *bare-metal hypervisor*. In Type II virtualization, the VMM runs on top of a normal host OS. One difference between the two approaches is the performance of the VMs. Since the VMM has direct access to the hardware when using Type I virtualization, the performance of the VMs is usually better than VMs' performance on Type II hypervisors.

In their fundamental paper "Formal Requirements for Virtualizable Third Generation Architectures" [172], Goldberg and Popek carved out several requirements that must be met in order to successfully realize virtualization. A

[24] In this quote, the abbreviation HVM means "Hybrid Virtual Machine".

piece of software, the VMM – also called hypervisor – must provide the abstraction of a VM. This hypervisor can either be a Type I or Type II hypervisor. In addition, a VMM must met three key properties:

> A virtual machine is taken to be an *efficient, isolated duplicate* of the real machine. We explain these notions through the ideal of a *virtual machine monitor* (VMM). [...] As a piece of software a VMM has three essential characteristics. First, the VMM provides an environment for programs which is essentially identical with the original machine; second, programs run in this environment show at worst only minor decreases in speed; and last, the VMM is in complete control of system resources.[25]

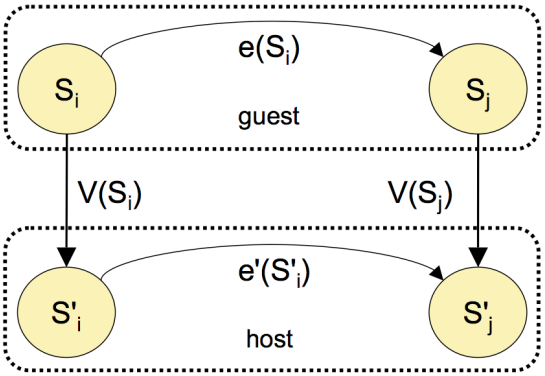


Figure 2.4: The isomorphism of virtualization (by Goldberg and Popek [172]): For each state S of a virtual machine exists an isomorphous mapping $V(S)$ to a corresponding state S' of the native machine. Within both of the machines an equivalent function $e(S)$ resp. $e'(S')$ exists that transfers the machines from one state into another.

The first of these key properties, referred to as fidelity, requires that a VM behaves exactly like a physical machine when software is executed in it. This requirement is expressed in the isomorphism of virtualization depicted in Figure 2.4. The term *isomorphism* demands that for every state S_i of a VM a mapping $V(S_i)$ exists that maps S_i to the state S'_i existing in the physical machine. In addition, for the transitional function e which transitions a VM from a given state S_i to a new state S_j an equivalent function e' exists which transitions the physical system from a given state S'_i to S'_j.

When this paper was published in 1974, IBM had already announced that it provided virtualization technology in the S/370 architecture.[26] Virtualization on mainframe computers was intended to maximize the utilization of the high-capacity and costly hardware. The basic idea was to allocate the physical resources to several concurrently running VMs and to run software programs in these VMs instead of running it natively on the hardware [152].

Today, virtualization technology is used for several reasons. Virtualization ideally reduces the complexity of a computer system (at least for its users), de-

[25] Gerald Popek and Robert Goldberg. Formal Requirements for Virtualizable Third Generation Architectures. *Proceedings of the Fourth ACM Symposium on Operating System Principles (SOSP '73)*, 17(7):412–421, 1974, p. 413

[26] IBM announced hardware assisted virtualization technology in the S/370 mainframe architecture in 1972:
`http://www.ibm.com/systems/my/z/about/timeline/1970/`

creases costs by flexibly (re)organizing the mapping between virtual resources and physical resources, provides isolated execution environments for running untrusted applications without endangering the overall reliability of the system, supports fault tolerance by checkpointing and restoring the state of a virtual resource, and allows users to execute their favorite (legacy) software in their favorite operating system environment (even if the VM's instruction set differs from the instruction set of the physical hardware).

These developments are founded on the massive increase in computational power of modern CPUs and the fact that main memory modules have grown and become less expensive over the course of the past decade. In recent years, it has become possible to use virtualization technology on commodity hardware. Although it fulfills the requirements stated by Goldberg, the widely available x86 architecture's instruction set aggravates effective virtualization [180]:

> The Intel architecture uses interrupts and traps to redirect program execution and allow interrupt and exception handlers to execute when a privileged instruction is executed by an unprivileged task. However, the Pentium instruction set contains **sensitive, unprivileged instructions**. The processor will execute unprivileged, sensitive instructions without generating an interrupt or exception. Thus, a VMM will never have the opportunity to simulate the effect of the instruction.

This is circumvented by using either a special kind of virtualization method called para-virtualization or by using hardware components that support virtualization technology to execute a VM on top of a VMM. Based on the requirements stated by Goldberg and Popek, Smith and Nair defined a VM in their 2005 book "Virtual Machines: Versatile Platforms for Systems and Processes" [205] as follows:

> In practical terms, a virtual machine executes software (either an individual process or a full system, depending on the type of machine) in the same manner as the machine for which the software was developed. The virtual machine is implemented as a combination of a real machine and virtualization software. The virtual machine may have resources different from the real machine, either in quantity or type. For example, a virtual machine may have more or fewer processors than the real machine, and the processors may execute a different instruction set than does the real machine [...].

Today's commodity hardware often has built-in virtualization support that has lead to a broad variety of desktop virtualization software such as Virtual-

Box or VMware Workstation/Fusion, allowing users to easily create and use a VM. Many companies use virtualization technology to consolidate their under-utilized server hardware by running several virtual servers on a single physical machine to cut down operational costs. Live migration enables load balancing by moving currently running VMs from a heavily loaded physical host to a physical host with less of load.

2.2.2.1 Types of Hardware Virtualization

Several types of virtualization exist for executing VMs on physical hardware, each with different advantages and disadvantages.

Full Virtualization. In full virtualization, the VM emulates enough hardware (virtual devices) that can be used by the guest OS running inside the VM without modifications. Full virtualization software often emulates few widely used devices that can be used by drivers of the guest OS. For example, qemu [15] – a well-known environment for emulating a processor as well as various peripherals – supports only one specific network interface card and one specific ISA and PCI system.[27]

Since the devices are completely emulated in the software, full virtualization has a significant overhead over physical systems; however, it is sometimes the only suitable technology such as when software should be run on physical systems with a different architecture. The aforementioned software qemu can emulate several hardware platforms, including x86, amd64, ARM, Alpha, MIPS, PowerPC and SPARC.

Further examples for full virtualization products besides qemu are VMware Workstation, VMware GSX Server and VirtualBox.

Hardware-assisted Virtualization. When using hardware-assisted virtualization, the hardware provides architectural support to ensure a reliable separation of the concurrently running VMs and support the functionality used to build VMMs [228]. Since 2008, both Intel and AMD have provided virtualization support in their modern CPUs called Intel VT-x (Vanderpool) [155] and AMD-V (Pacifica) [7], respectively. Since IBM started such developments on mainframes in the 1970s, hardware-assisted virtualization is also supported by most of the latest Intel and Sun/Oracle CPUs targeting this market.

Examples for products supporting hardware-assisted virtualization are Linux Kernel-based Virtual Machine (KVM), Xen and Parallels Desktop for Mac. Products like VMware Workstation (and VMware Fusion) or VirtualBox make also use of hardware assistance if available on the particular hardware.

[27] `http://wiki.qemu.org/download/qemu-doc.html#QEMU-PC-System-emulator`

Para-virtualization. In Para-virtualization technology, the VMM does not necessarily emulate any hardware but rather provides a software interface to communicate with the physical hardware. To use this API, the guest OS must be modified. Such hypervisor calls are named hypercalls in Xen.

Using para-virtualization usually results in higher performance since the device does not have to be emulated but is accessed via hypercalls. Xen is a well-known example that combines para-virtualization technology and hardware-assisted virtualization. If the latter is available, Xen is also able to run unmodified guest operating systems like Windows.

2.2.2.2 Categories of Virtualization

Because Section 2.2.2.1 focused on hardware virtualization, only a limited area of virtualization is covered. In computer science, three broad categories of virtualization can be distinguished:

Application Virtualization. Application virtualization means that applications are not installed in a local (operating) system, but are run as virtualized applications in an execution environment that is provided by a remote server. Users do not need not to install additional local software to use these applications. The application providers ensure that the applications are usable during peak demands by automatically assigning more resources to satisfy an application's needs. Well-known examples of such applications are Salesforce.com, Google Apps and Microsoft Office Live Workspace.

Platform Virtualization. The most prominent example of platform virtualization is the Java programming language. Bytecode produced by the Java compiler runs on the Java virtual machine. It does not matter if the source and target platform differ i.e., if the bytecode was compiled on a 64-bit machine while it is executed on a 32-bit machine. Microsoft's .NET platform is comparable to the Java approach. Platform virtualization approaches such as Microsoft Azure, Force.com and Google AppEngine are slightly different. These act as platforms that can be used by developers to host applications. Applications must be developed for each particular platform because a particular platform usually applies restrictions to its developers to ensure that it can be responsive to particular events. For example, if a peak demand arises due to a positive article in the press, additional resources are assigned to deal with the additional requests.

Infrastructure Virtualization. In general, infrastructure virtualization denotes the virtualization of physical resources. *Network virtualization* combines the underlying hardware and software network resources into a

single administrative entity. Adding encryption to the network protects the transferred data from malicious users, while an encrypted connection hides an intermediate router since it is presented as a one-to-one connection to the user. Virtual Private Networks (VPNs) allow companies to logically connect physically separated sites. In a similar manner, *storage virtualization* is a logical abstraction of physical storage. Prominent examples are Logical Volume Managers (LVMs) as used in Linux or the BSD operating systems, a Redundant Array of Inexpensive Disks (RAID), Network-Attached Storage (NAS), and the Amazon Simple Storage Service (S3). Providing only a desktop (instead of a complete virtual machine) that can be accessed by software or special hardware is called *desktop virtualization*. Prominent examples are Virtual Network Computing (VNC), thin clients such as Microsoft's Remote Desktop and associated Terminal Server products. Finally, *operating system virtualization* denotes the ability to run entire operating systems inside a virtual machine.

2.2.2.3 Virtualization in Grids and Clouds

The three broad categories of virtualization are used to varying degrees in Grids and Clouds.

On the *application virtualization* level, Salesforce.com is one of the best known examples in a Cloud context. The company offers a Customer Relationship Management (CRM) software without requiring a local software installation on the user's computer or a centralized server within the user's company. The CRM application is accessible via a standard web browser. Salesforce.com not only offers its CRM software, but also development tools that support programmers and customers in developing their own applications based on the company's platform Force.com. These applications can be provided at the AppExchange online marketplace and can be used by other customers on demand. Users pay for using the software.

Force.com uses *platform virtualization* techniques to provide users an environment for developing and running their software. Salesforce.com promises real time scalability of applications executed on the Force.com platform. For this purpose, the platform monitors the resources used by an application and provides new resources the application can use, if necessary [189]. To enable automatic scaling of applications, the platform places several restrictions on developers to avoid software designs that would not scale even if the platform assigns additional resources. Furthermore, applications are multi-tenant i.e., an application is not started for every user, rather different users share the same application. The users' activity and data will be isolated from other users by platform mechanisms, implying that every user is individually working on an application [188].

Amazon's Elastic Compute Cloud (EC2) is the best known example of using *infrastructure virtualization* in a Cloud context. Amazon EC2 users can rent VMs and get root access to these machines. VMs will be executed on resources located in data centers run by Amazon. A user can start as many machines as needed without any initial investment and pays only for the resources used. For performance reasons, Amazon provides a Xen-based infrastructure on which users can create and execute VMs. The basic component of Xen is a hypervisor that is responsible for managing the access that VMs have to the physical resources. The hypervisor runs directly on top of the hardware and not only provides drivers to access the physical hardware, but also implements the VM scheduler and the Memory Management Unit (MMU). The VMs will be executed on top of the hypervisor. In Xen, one VM has privileged access to the hypervisor and is used to control all other VMs on the same host. The hypervisor is responsible for managing access to the real resources and isolating running VMs from one another. It allows fine-grained resource allocation to different VMs in terms of CPU time or the size of main memory assigned to the VMs.

Typically, Grid computing environments do not operate with virtualized resources (mainly for performance reasons), but Grid developers understood quite early that infrastructure virtualization can be used to overcome some limitations of Grid computing. When tasks are executed in the users' own VMs, it is not possible to spy on other users' processes or jobs in their VMs. VMs can also be used to provide personalized computing environments when they are administered by the users themselves. This enables them to install the software they need without requiring root access to the physical installation and even to bring commercial software into the Grid. Motivated by these advantages, several current open-source projects are aimed at introducing infrastructure virtualization technology to Grid computing, which effectively enables Grids to offer Cloud functionality, as outlined below.

Nimbus [157] is a set of open-source tools that provides an infrastructure virtualization solution for Grids. Nimbus allows a client to lease remote resources by deploying VMs on those resources and configuring them to represent an environment desired by the user. As with Amazons EC2, most VMs exist as ready-to-go appliances. Once the VM image has been started, the user can perform modifications. Nimbus provides Virtual Clusters (VCs). A VC can consist of several VMs taking different roles e.g., Nimbus can be used to set up a VC containing a compute node, a storage node and n worker nodes on the fly, allowing on-demand creation of VCs on a site.

OpenNebula [163] is a toolkit that provides basic methods for creating and sharing custom VM images. Dynamic allocation of VMs on a pool of resources is supported, whereas the resource pool can contain physical machines at different physical locations. A VM is defined by a template covering details such as the used operating system kernel, hard disks and network settings

used. With the *onevm* command line tool, this template can be used to generate a new VM image.

Eucalyptus [162] is an open-source virtualized Grid and Cloud computing system. The Eucalyptus framework offers the same interface for external tools as Amazon's EC2 does. This enables users not only to manage the commercial Amazon Cloud but also to set up a private Cloud with Eucalyptus and manage resources with the same tools available for the commercial environment.

2.2.2.4 Challenges of Virtualization

Existing Grid and Cloud computing solutions offer a simple way to set up and manage solutions for a single site, whether it is public or private. Since the launch of Amazon EC2, Rackspace and FlexiScale, virtualization technology has matured. Nevertheless, there are several open research issues that have to be taken into account [83] to provide a high-performance and secure Grid and Cloud computing infrastructure to users:

Security. Allowing users to install their own software not only provides advantages, but also raises several security concerns. Since VMs can be administered by possibly inexperienced or even malicious users (instead of experienced system administrators), VMs cannot be seen as trusted participants running on local resources. A provider's infrastructure and the users' VMs must be secured against attacks from other VMs. In the event of severe security vulnerabilities, it must be ensured that patches are applied to all VMs, even those that are inactive at the moment. If an attack takes place, mechanisms should be available to remove malware that might have been installed. The problem is aggravated by the fact that users are able to install custom software or even operating systems that are no longer supported and will not get patched if a security flaw arises. Moreover, Trusted Computing techniques (based on Trusted Platform Module technology) must be adapted to work with VMs.

Performance. Additional abstraction layers often accompany an overhead that decreases the performance of the system. Compared to native task execution, the performance penalties of using virtualization must be minimized.

Management. Since the Grid links several high performance computing sites together, the users VMs must be available for job execution on every Grid site. To ensure that users can submit jobs even if a particular Grid site is not accessible for some reason, mechanisms must be put into place to make the users' VMs available at every site and enable users to manage their VMs where ever they are. A reliable and efficient transfer mechanism must be developed to achieve fast VM distribution without

congesting the network. This also affects Cloud providers sharing VMs among different locations.

Licensing. Commercial software that can be used in the Grid often provides a license management mechanism that is not compatible with virtualization technology. Authentication to a software may rely on hardware dongles that cannot be used from within virtual machines. License servers often base authentication on MAC or IP addresses that are static in an environment with physical hardware, but can be assigned to several VMs of different users at one time in a virtualized environment.

2.2.3 Xen Grid Engine

The XGE dates back to 2006 when a solution to interrupt long-running serial computing jobs in favor of a large number of parallel jobs was developed [66]. The basic idea was to use virtualization technology to provide execution environments for serial and parallel job execution.

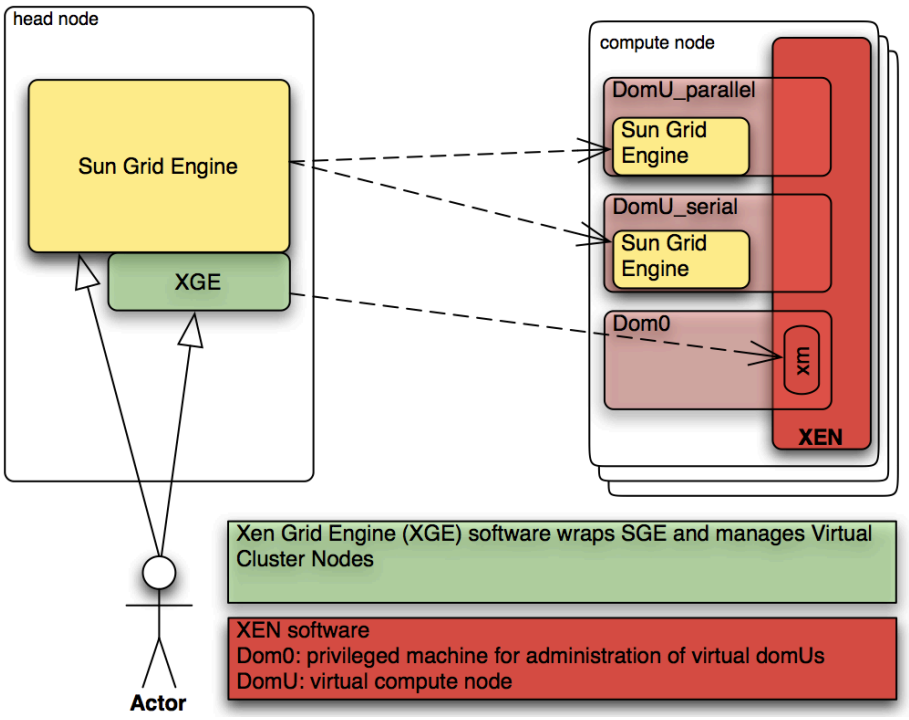


Figure 2.5: The first implementation of the XGE aimed at the ability to operate each physical compute node on a cluster in two different modes, either to execute serial or parallel jobs.

Figure 2.5 shows the architecture of the XGE. It was designed to work with the Xen virtualization environment available on the compute nodes shown on the right part of the figure. The XGE was independent of the CRM of the cluster, its functionality has been proven by using the Sun Grid Engine (SGE).[28] Only the cluster administrator had access to the XGE which was connected to the VMs used for administration tasks on the compute nodes. Following the nomenclature of Xen, the "privileged" VMs were called Domain0s (dom0s). Moreover, each compute node had two "unprivileged" VMs available, the so

[28] It is not hard to see that XGE borrows its name from the SGE.

called DomainUs (domUs): A domU_serial for executing serial jobs and a domU_parallel for executing parallel jobs. Only one of the two unprivileged VMs had been active at a time, every physical node could operate in either serial or parallel mode. Depending on this decision, either the VM hosting the environment for executing serial jobs was started or the VM hosting the parallel environment. The XGE partitioned a physical cluster into two Virtual Clusters (VCs), while the number of nodes in each VC could be adjusted dynamically.

In addition, the XGE was able to suspend running VMs executing serial jobs, start the VM for parallel job execution, and switch back to the serial VM later on. Using the suspend functionality of Xen, the cluster could interrupt long running serial jobs in favor of short running jobs that need a large number of compute nodes and to circumvent the very long waiting time for a scheduling decision. It has been shown that this strategy works well for serial jobs, but parallel jobs could not be suspended and reactivated reliably due to the network timeouts that occurred if one or more parallel VMs were suspended and awakened again later.

Based on this first version of the XGE, a new version has been developed which is still under development to this day.

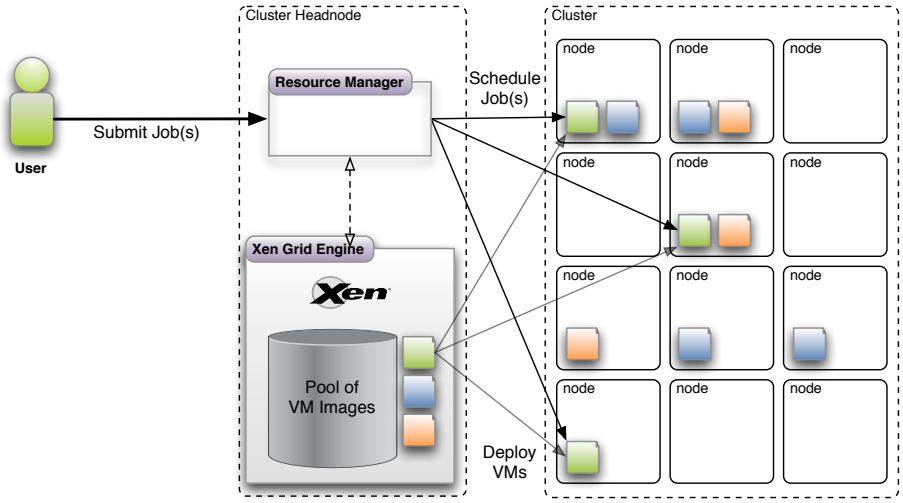


Figure 2.6: The current version of the XGE is connected to the local CRM which makes scheduling decisions. Depending on these decisions, the XGE is responsible for deploying VMs to the affected compute nodes and starting them. After job execution, these VMs are shut down.

The current version of the XGE interacts with the local CRM. Figure 2.6 shows its architecture. The local CRM, such as the SGE, makes scheduling decisions based on the current state of the cluster system. Once it decides where to schedule a pending job, the XGE deploy the appropriate VMs to the affected compute nodes. By respecting the scheduled job's owner, the XGE can provide personalized VMs that have software needed to perform the particular execution. Once a job execution has finished, the XGE shuts down the user's VMs on the nodes which then can be used to execute other jobs. This mode of operation targets environments using batch scheduling of computational jobs. Using it in this manner, the XGE is a major building block of a virtualized

cluster system that provides personalized virtual execution environments to its users.

In fact, although the XGE does not rely on a CRM, it can also be used without a scheduler. Users can request the provisioning of their VMs to execute their jobs manually. This mode of operation enables a resource provider to support the provision of service-oriented environments in which the need of a particular resource can not be estimated in advance.

The XGE has been designed to provide the best possible performance to its users. Therefore, each virtual CPU assigned to a VM is bound to a physical CPU core. Then best performance can be achieved by circumventing the concurrent utilization of a particular CPU core by multiple VMs [87]. Any desired partitioning of the physical machine can be achieved, ranging from one VM per physical core to one VM using all physical cores [242].

This thesis does not provide further information regarding the design and implementation of the different versions of the XGE. Details about the implementation of the first version of the XGE have been published in a research paper [66] and two diploma theses [65, 171]. Since the latter version is still under development, details of the implementation can be found in research papers [208, 199, 196].

In the following, the term XGE denotes this latter version of the XGE.

2.3 Requirements Analysis

The work presented in this dissertation stems form different projects funded by the BMBF (D-Grid and HPC initiative), especially the projects funded during the second call of the D-Grid initiative aimed at bringing Grid technology to the industry. The projects' main targets were to identify and solve problems associated with using the Grid in specific industrial sectors.

2.3.1 Industrial Use

Regarding computational environments, the requirements of commercial users differ greatly from those of academic users. While academic users – especially computer specialists – love to delve into a problem and often provide a solution that is suitable for experienced users, it may not be an acceptable solution for commercial users who would like to use a system in a simple manner. Commercial users' requirements focus on simplicity, usability and lowering a system's administrative burden (at least for the users).

2.3.1.1 Separating Users Securely in Shared Environments

There is another important difference in commercial users' needs compared to academic users' needs: security. In fact, security is one of the most common

concerns of both commercial software providers or commercial users when talking about Grid and Cloud environments.

Originally developed in academic environments, the first generation of Grid users was keen on developing software on their own or using open source software in an environment used concurrently by others. Furthermore, academic data is often accessible to everyone interested; therefore, data and software operating on this data does not usually need not to be secured against malicious users. Moreover, resources linked together especially by the Grid are often located at universities or public research centers and originally used by a closed community of researchers. For example, academic users interested in data created by the LHC at CERN can easily get access to the data they need. The data is distributed to several sites located around the globe to handle the enormous amount of data generated by LHC using Grid technology [130]. Access is granted based on affiliation to a research center and not open to (untrusted) third party users. It is important to note, that trust is usually the main security principle in academic Grid environments. In environments like these, it is unlikely to be attacked by a malicious user.

By enabling commercial users to use (academic) Grid resources – the main goal of the D-Grid projects funded by the BMBF – a strong separation of users and their data must be ensured. If there were any possibility that competitors can spy on other users' data or processes executed by other users on the system, Grid and Cloud technology would not be used by commercial users. Strong user separation is crucial in systems aimed at the commercial market.

Increasing competition in the German banking sector is leading to high pressure for restructuring and further automation in IT-related business processes in banks and financial services providers. In addition, new legal regulations such as Basel III and changing customer needs that are moving in the direction of highly customized on demand financial products enhance this pressure. To face these challenges, the Financial Business Grid (FinGrid) project[29] [126] strives to identify suitable services and processes in the financial services sector and to develop Grid-based systems that enable financial service providers to reorganize their processes efficiently and to realize applications that have been impossible so far in terms of computational requirements. To guarantee relevance for the target industry, the projected research will be performed jointly with leading financial industry research partners. Targeting an industrial sector that has tremendous requirements regarding data security, one major goal was to develop a secure infrastructure that matches the needs of the customers.

Using the concept of virtualization technology, users can be separated securely within one system, making it impossible for one user to access another user's data or view the processes running within another VM. The VM looks and behaves like a traditional machine that is used exclusively by the user her-

[29] `http://www.fingrid.de`

self. It is not possible to exit the VM to access the underlying physical machine or other VMs running concurrently on this machine. In particular, an user inside her VM cannot spy on data stored and processes running in other VMs, which is referred to as metadata security.

In the FinGrid project, three software prototypes had been developed that were able to run on resources provided by the D-Grid environment. Therefore, the software architecture providing secure execution environments must not be tailored to a particular piece of software; rather, it must provide a generic approach independent of the software installed in a VM.

2.3.1.2 Tailored Execution Environments for Grid and Cloud Users

The idea of using virtualization technology not only to separate users from one another but also to provide them individually tailored execution environments for their computational tasks has been developed for several years now [69, 207, 245]. In 2004, Keahey et al. [118] presented the idea of using virtual environments specifically in Grid computing, an approach that has matured to the Nimbus project [157] available today. Based on this idea, in the Plasma Technology Grid (PT-Grid) project, VMs are offered by a software provider that makes high energy physics simulation software available readily installed in VMs. Additionally, up until now, the company has only been able to offer (very limited) in-house computing resources to its customers and is looking for a dynamically scalable solution to provide as much computing power to its customers as needed.

PT-Grid[30] started in May 2009 and aims to provide a flexible high performance computing landscape in the area of high energy physics. The project's goals are to provide a common infrastructure for the heterogeneous plasma-community and to develop Grid-based tools for commercial project partners. Furthermore, a system should be developed for the pricing and billing of the developed Grid services within this project.

Commonly used within this industrial sector, ANSYS CFX "has been applied to solve wide-ranging fluid flow problems for over 20 years."[31] One normal simulation pass on a multi-core AMD Opteron system with 3 GHz takes about 1 hour and 2 GB main memory to execute the state of a simple geometry, it but would take up to several weeks and 40 GB main memory for more complex cases. With the resources available at SME, it is hardly possible to simulate time-dependent processes or automatically optimize geometries and simulation parameters. First of all, SME are not able to purchase the needed hardware resources, moreover they usually are not able to utilize these resources to their full capacity throughout the year due to the cyclic development process with changing simulation needs.

[30] http://www.pt-grid.de
[31] http://www.ansys.com/products/fluid-dynamics/cfx/

CFX Software Berlin GmbH, a software and service provider and participant in the PT-Grid project decided to examine the use of nationwide Grid resources available for its customers' simulation tasks. Considering most important aspects for commercial users of usability, reliability and security, readily installed VMs should be provided by CFX Berlin itself to save the customers the complex and time consuming software installation process in the VMs. On the other hand, customers should be able to modify the VMs to their needs, e.g. by installing additional software on their own. To minimize the overhead resulting from the use of virtualization technology, slim and efficient operating systems should be used as well as a high performance virtualization solution.

2.3.1.3 Microsoft Windows in the Grid

Focusing on Linux-based personalized computing environments up until this point, commercial software often relies on the Microsoft Windows operating system. Virtualization technology can also solve the problem of providing different operating systems dependent of the users' needs.

When trying to attract commercial users for the Grid, one of the questions that is often asked regards the support of Windows binaries in the Grid. While Cloud providers like Amazon are capable of offering the Windows Operating system to their customers, Grid providers usually are not. For example, parts of the widely adopted Grid middleware GT4 can be run on a Windows computer, but the underlying computing resources used to execute the Grid users' jobs must be operated with Linux. Nevertheless, BMW AG, a participant in the Biz2Grid project, decided to examine the use of Grid nodes for execution of Windows-based computational tasks.

The main objective of Biz2Grid[32] is to provide foundations for an effective application of Grid technologies in enterprises from the automotive sectors. In order to achieve this goal, business and economically driven questions have to be answered and technical challenges have to be solved. Business-models have to be developed that are adequate for an application in the Grid. In addition, Biz2Grid addresses technical research questions. Currently, the seamless adaption of applications to Grid technologies is hardly realizable. As a consequence, it is a challenge to (i) distribute and parallelize real world applications and (ii) conceptualize and implement economic business models at the same time.

During the project, one challenge was the flexible and effective use of in-house workstations of the project partners. Since simulation jobs can be very time consuming, the industrial project partners were interested in elaborate idle workstations during the night which promised to speed up job execution times. In contrast to high performance compute nodes which are available all the time and are used only for job execution, a workstation has a different

[32] `http://www.biz2grid.de`

usage profile. First, the workstation is used during the day by an employee and may possibly not be available for job execution. Second, a workstation may possibly be switched off and cannot be used for job execution during the night. These characteristics and the fact that the simulation software is based on the Windows OS prevent the use of traditional batch schedulers that are used in the high performance computing environment. Schedulers are needed that can deal with a dynamically changing computing landscape and are able to schedule and execute compute jobs to Windows-driven compute nodes.

Siemens Tecnomatix is the vendor of an event-based simulation software called Plant Simulation[33]. This software tool is used throughout the whole automobile product development process for a wide variety of purposes, like intra-plant and inter-plant logistics, value flow analyses, production process simulations and improvements (like bottleneck identification, throughput/inventory/cycle time analyses). Figure 2.7 shows the Graphical User Interface (GUI) of Plant Simulation. The tool runs on the Microsoft Windows platform and is a standard solution for many companies like the BMW AG for logistics simulations. Simulation experts use it to graphically create object-oriented hierarchical simulation models that can later on be run and analyzed; the simulation results e.g., throughput or resource utilization data, can be accessed using built-in diagrams and other visualization tools. It also includes an experiment manager module in which series of experiments can be defined and executed automatically.

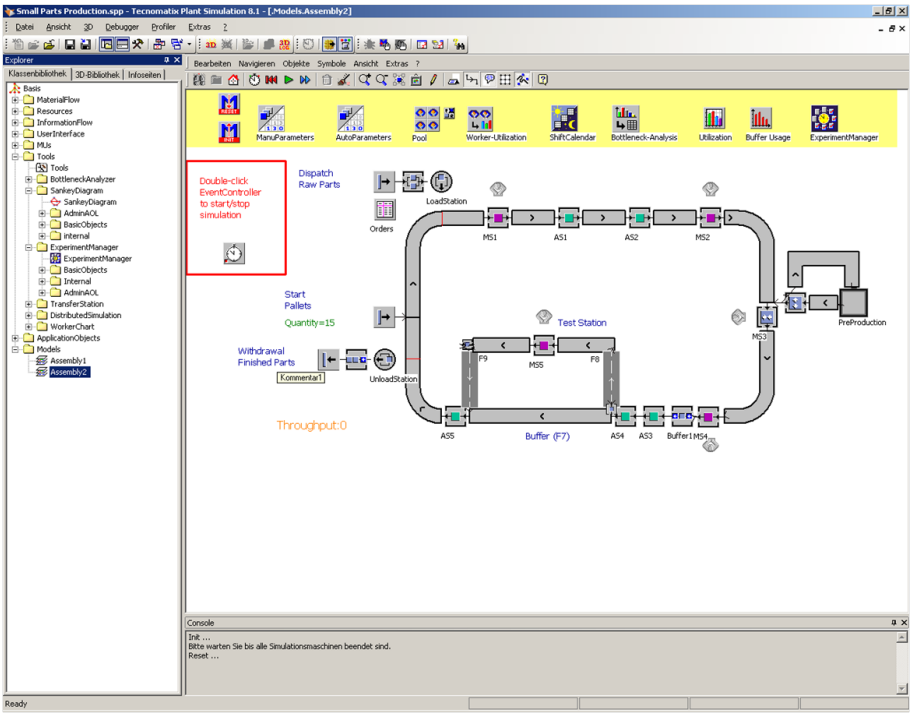


Figure 2.7: Screenshot of the Plant Simulation GUI with a simple production line.

A simulation study (also called simulation project) contains a number of experiments, each of which consists of several simulation runs leading to one

[33] `http://www.emplant.de`

observation. The runs executed within the context of the experiments are independent from one another (neither data nor control dependencies exist); consequently, those runs can be perfectly scheduled in parallel.

A simulation study requires a simulation model and a number of different input parameter types. Some typical examples of those parameters are machine performance data (throughput), machine availability data, production programs, station times and failure statistics among others. Typical results are total system throughputs and work-in-progress, locations of bottlenecks and buffer sizings. Using this information, a material flow planner can eliminate redundant buffers, optimize inventory and reduce cycle times. The simulation project's results support the production process design decisions taken in the product development process.

Some of the tasks during simulation projects are especially computation-intensive:

Statistical Validation. A stochastic simulation model will be fed with identical input parameter values but with differing seed parameters for the random number generators. This approach yields higher-quality simulation results with higher statistical support. Currently, a low number of simulation runs are executed for statistical validation due to computational and time constraints.

Input Parameter Variation. The simulation model will be fed with differing input parameter values; each configuration of input parameters constitutes a simulation experiment. The computing time increases rapidly with the number of input parameters and the number of tested values for each parameter (combinatorial explosion). Like in the case of statistical validation, currently the number of simulation experiments executed is severely limited by computational constraints and result data management issues, so that only a small portion of the parameter space is usually explored.

Each single simulation run is computationally intensive (it requires up to several hours per run) and independent of other simulation runs which makes them amenable to their execution in a Grid computing environment. Because of the fact that some simulation runs are long running and that a characteristic development of particular simulation parameters point to existing problems in the simulation model, the simulation specialist must be able to access the produced data and intermediate results at any point in time.

2.3.2 Requirements Catalog

Throughout the work in several projects, a lot of effort was invested in carving out both functional and non-functional requirements for the system to be developed. During the conceptional work performed in the projects mentioned

above, questionaries were handed out to the industrial project partners and the transcripts from the interviews were then aggregated according to John W. Creswell [40] and Kathleen M. Eisenhardt [58]. But not only questionaries were used to spot the industrial partners' requirements and guide the beginning of the software development, as the project partners were also involved in the entire software development process following the principles of iterative software development [131]. The steady contact between developers and business partners made it possible to react to changed business assumptions or changes in the business use cases and ensured the successful completion of the projects. Moreover, since not every requirement was known in advance, software could be adapted to the new requirements easily.

The following 20 requirements have been compiled based on knowledge gained from the prototypical implementations in several commercial environments as well as discussions between experts, commercial users and project partners. The requirements cover the complete lifecycle of both jobs and virtual execution environments (such as VMs) ranging from creating and managing VMs in a distributed environment to job execution within these VMs, including the security and scalability of the entire system.

Users and image providers must be able to create new VM images in an easy and comfortable manner. (R1)

Users must be able to derive new VM images from already existing ones provided either by other users or by image providers. (R2)

Users must be able to import VM images from their local virtualization software and to export VM images from the Grid or Cloud to their local virtualization environment. (R3)

These requirements target the process of creating VM images. It is crucial that the system provides an interface that even inexperienced users can use intuitively and easily. For inexperienced users, techniques must be provided to support users during the image creation process e.g., by experienced colleagues or software producers that provide readily configured VM images and act as image providers. Moreover, the system should be able to import and export existing VM images in order to circumvent the data lock-in effect and allow the exchange of existing images between different sites.

Image creation functionality must be provided by a service interface to enable the integration of the image creation component in a service-oriented environment allowing the component to be used automatically from within other services. (R4)

Software tools must be provided that support the installation of applications and other software into the VM images. (R5)

Creating VM images automatically and installing software packages via the local package manager of the Linux OS within the VM allows for the completely autonomic set up and configuration of VM images e.g., by using Business Process Execution Language (BPEL) workflows. This is useful if a VM were to be used for computation that will never be used again.

VMs must be available throughout the environment on every resource site connected to the system and not only on a single site. (R6)

Besides dedicated resources, desktop computing resources (laptops, PCs) must also be supported and used by the system to execute computational tasks. (R7)

Data necessary to complete a computational task must be automatically transferred to the compute nodes before the job is started. Result data must be transferred back to the user automatically. (R8)

These requirements focus on the challenges of VM image and data distribution as well as resource utilization. In a computing environment that embraces several independent resources located on different geographically distributed sites, efficient transfer mechanisms are needed to ensure the utilization of a existing VM image independent from the location where a computational task is executed. Mechanisms must be put into place to ensure interoperability of different resources such as high-performance clusters or a network of partially unused desktop computers or laptops e.g., in a company's office.

Firewall configuration assistants must be developed which allow the VMs to form collaborative cross-organizational environments. (R9)

Data must be stored encrypted and only be available in decrypted form during the computational process. (R10)

After simplicity, security is the second crucial aspect a system must provide to its users. It is often stated that simplicity and security contradict each other. Virtualization has been proven to provide additional security compared to physical systems and can be used to separate users from each other. It is a challenge to provide automatically reconfiguring components which adapt to the system's current state with respect to virtual environments belonging to particular users. Components providing additional security mechanisms ensuring the consistency of a user's data must be developed and made available to the customers of the proposed system.

Execution of computational jobs must be independent of users currently logged into the system, allowing jobs to be executed automatically without the need for user interaction.	(R11)
Users must be able to get information about a running jobs' progress independent of the location on which it is executed.	(R12)
The system must be able to transfer and provide access to checkpointing files (e.g., intermediate results) during the computation.	(R13)
Like the management tool for virtual machines, a tool must be provided allowing the scheduling system to be reseted and restarted easily.	(R14)
Users must be able to abort and remove previously submitted computational jobs.	(R15)
A tool must be available to the administrator, that provides information about connected computing nodes and their state.	(R16)
Users must have access to their data during the entire execution process.	(R17)

These requirements deal with job execution within a system. In addition, the system must be able to execute jobs on behalf of a user who has not to be logged into the system. While the system is responsible for managing job execution and data transfer on its own, each user needs to be able to get information about the state of her current jobs or have access to a running job's data. Moreover, a user must be able to abort a running job and to remove an already scheduled job from the system.

Based on the requirements for using multiple resource pools, additional requirements appeared that must be met. A system supporting multiple resource pools should be able to scale up and down automatically and provide access to (external) resources when needed. Since the usage of external resources,

such as a Cloud, is usually associated with increased costs compared to locally available resources, effort should be made to minimize the use of external resources without affecting the normal operation.

As new resources are needed, the computational demand should be met by the most affordable resources available matching the demand's requirements. (R18)

Requirement R18 expresses the effort to minimize operational costs which means to acquire the most affordable resources available.

To deal with peak-loads the system must be able to scale up automatically. This guarantees rapid satisfaction of the computational demand. It is fundamental for service providers that guarantee a particular reaction time to provide compute resources to their customers via a Service Level Agreement (SLA). If a provider's local resources are working to full capacity, it may be necessary to switch to external resources in order to fulfill existing SLAs and avoid paying a contractual penalty. It is important to stop using rented resources as soon as possible to cut down on the operational expenses. These needs are expressed in requirement R19.

New resources should be acquired instantly and set up autonomously and completely transparent to the system's users. Utilization of external resources should be minimized, unused external resources should be freed and deallocated as soon as possible. (R19)

By providing the ability of scale up and down and making it completely transparent to the user, she must have the ability to influence which resource not to use. If a system is used for computational tasks within a company and provides utilization of external Cloud resources when needed, a user within that company must keep control of sensitive computational data to ensure that it is not scheduled to resources outside the company. This is characterized by requirement R20.

A user working with a system that schedules tasks among different sets of (computational) resources must be able to decide which resources are allowed to be used for the execution of each task. (R20)

Several systems exist that may fulfill some of the requirements above. Nev-

ertheless, up until today no system is available that eliminates the problem of (data) lock-in and scheduling among heterogeneous systems.

2.4 Related Work

When the work of this thesis was started in 2007, Cloud computing was unknown to the public. The first projects already showed that virtualization technology could be successfully used in Grid computing environments e.g., to achieve strong user separation. Much work has been done by other research groups that affects the work within this thesis. This chapter will present an overview of related work and will discuss the approaches presented.

The use of virtualization technology has a long tradition in computer science. In June 1970, IBM announced its creation of System/370[34] and which it made available to its customers in 1972. The main software component of this system is the Control Program called VM-CP. It runs on the physical hardware and provides full virtualization of the physical machine [39]. It orchestrates the different VMs and provides VM management capabilities, device and storage management as well as memory management [100]. The concepts introduced in System/370 provide the basis for the successional operating systems such as System z, the standard OS running on IBM's mainframes nowadays. The reason for providing numerous VMs on a single piece of (supercomputer) hardware was published by Donovan and Madnick in IBM's Systems Journal in 1975 [53]:

> Security is an important factor if the programs of independent and possibly error-prone or malicious users are to coexist on the same computer system. In this paper, we show that a hierarchically structured operating system, such as produced by a virtual machine system, that combines a virtual machine monitor with several independent operating systems (VMM/OS), provides substantially better software security than a conventional two-level multiprogramming operating system approach. This added protection is derived from redundant security using independent mechanisms that are inherent in the design of most VMM/OS systems. Such a system can be obtained by exploiting existing software resources.

The increase of system security is one of the main motivations for using virtualization technology in computer science and is also the foundation of the work presented in this thesis.

[34] `http://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PR370.html`

Table 2.1: Business requirements to the technical system developed within several projects. The requirements have been carved out from interviews with experts and users as well as project partners' feedback during the software development process.

Nr.	Requirement
R1	Users and image providers must be able to create new VM images in an easy and comfortable manner.
R2	Users must be able to derive new VM images from already existing ones provided either by other users or by image providers.
R3	Users must be able to import VM images from their local virtualization software and to export VM images from the Grid or Cloud to their local virtualization environment.
R4	Image creation functionality must be provided by a service interface to enable the integration of the image creation component in a service-oriented environment allowing the component to be used automatically from within other services.
R5	Software tools must be provided that support the installation of applications and other software into the VM images.
R6	VMs must be available throughout the environment on every resource site connected to the system and not only on a single site.
R7	Besides dedicated resources, desktop computing resources (laptops, PCs) must also be supported and used by the system to execute computational tasks.
R8	Data necessary to complete a computational task must be automatically transferred to the compute nodes before the job is started. Result data must be transferred back to the user automatically.
R9	Firewall configuration assistants must be developed which allow the VMs to form collaborative cross-organizational environments.
R10	Data must be stored encrypted and only be available in decrypted form during the computational process.
R11	Execution of computational jobs must be independent of users currently logged into the system, allowing jobs to be executed automatically without the need for user interaction.
R12	Users must be able to get information about a running jobs' progress independent of the location on which it is executed.
R13	The system must be able to transfer and provide access to checkpointing files (e.g., intermediate results) during the computation.
R14	Like the management tool for virtual machines, a tool must be provided allowing the scheduling system to be reseted and restarted easily.
R15	Users must be able to abort and remove previously submitted compute jobs.
R16	A tool must be available to the administrator that provides information about connected compute nodes and their state.
R17	Users must have access to their data during the entire execution process.
R18	As new resources are needed, the computational demand should be met by the most affordable resources available matching the demand's requirements.
R19	New resources should be acquired instantly and set up autonomously and completely transparent to the system's users. Utilization of external resources should be minimized, unused external resources should be freed and deallocated as soon as possible.
R20	A user working with a system that schedules tasks among different sets of (computational) resources must be able to decide which resources are allowed to be used for the execution of each task.

At the same time, in 1973, Popek and Goldberg published formal requirements for efficient virtualization of computer systems [172]. One savory detail is that Intel's x86 architecture does not fulfill the requirements formulated in the paper and therefore could not be efficiently virtualized. In fact, there are several problems in virtualizing the x86 architecture caused by the fact that an x86 architecture CPU does not trap all instructions needed to be virtualized [180]. Much effort has been made by software and hardware developers to develop systems capable of efficiently running virtualization technology on the x86 architecture [155, 7].

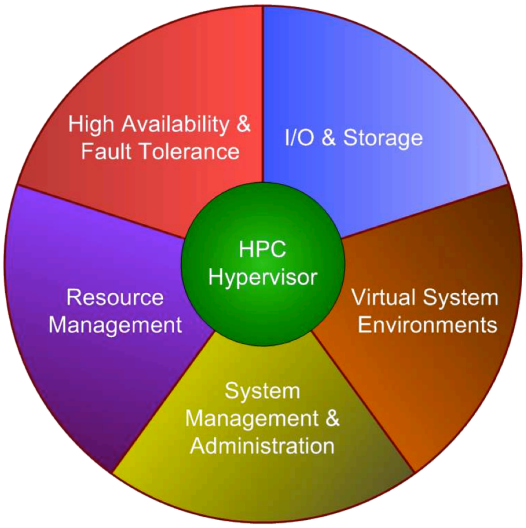


Figure 2.8: The system-level virtualization components [230].

Vallée et al. propose "five areas of enhancement" surrounding a High Performance Computing Virtual Machine Monitor as shown in Figure 2.8. To minimize interferences of the host OS with running VMs, the host OS' footprint must be minimized and processes not necessary for VM management must be stopped. Additionally, the VM's OS should only provide the necessary software components to minimize overhead during each VM's execution. Furthermore, tools for system management and administration must be provided which can deal with a potentially large number of VMs. Resource management must be optimized by efficient deployment of VMs on the compute nodes. Ideally, high availability and fault tolerance mechanisms are provided such as checkpointing, suspend/resume and live migration. Finally, the I/O performance must be improved, which is crucial for both network performance, e.g. when jobs are executed in parallel using MPI, and disks.

This section will follow the structure presented in Figure 2.8. First, work targeting to provide a HPC VMM will be presented. Then, the focus will be shifted to virtual environments designed to execute customers' computational jobs. After that, work focusing on the management of such a system will be presented. Thereafter, work will be discussed that focuses on managing the underlying resources i.e., regarding the scheduling of the virtual environments to the existing nodes. Then, techniques dealing with High-Availability (HA)

and fault tolerance will be presented which can be used in environments using virtualization technology. Being bottlenecks of current virtualization products, performance of I/O and storage in virtualized environments will be analyzed and optimization techniques will be presented.

In addition to the "areas of enhancement" proposed by Vallée et al., the work in three further areas will be discussed: Security technologies using virtualization will be addressed before economical aspects are presented, which lead over to the latest movement in the area of HPC and cluster computing called Green computing.

2.4.1 High Performance Computing Hypervisor

Currently, Xen [14, 174] is the most popular open source system in the area of virtualization. One of the main reasons is that virtual Xen machines (called domUs) gain nearly the performance of native compute nodes [34, 14] due to the usage of para-virtualization [125, 249]. In contrast to full virtualization as used by several VMware products [222], para-virtualization offers a software interface to the virtual machine that is similar but not identical to the underlying physical hardware. Without hardware support of the CPUs [155, 7], an OS must be modified to run on top of the Xen VMM. By circumventing this disadvantage with the use of new processor generations with built-in virtualization support, the advantage of providing access to the hardware with an interface instead of using a virtualization layer to emulate devices still exists. The approach of para-virtualization is usually faster than using a full-virtualization approach. For that reason, Xen is often used in environments which must provide high-performance virtual machines e.g., it can be found in several projects trying to establish virtualization technology in the area of HPC.

Kernel-based Virtual Machine (KVM) [124] debuted in 2007 and is an open source virtualization solution that relies on hardware virtualization support. It uses the existing Linux kernel infrastructure to provide an integrated hypervisor approach.[35] KVM has been integrated into the Linux kernel and is therefore available on many modern computer systems.

A comparison between Xen and KVM was made by Deshane et al. [48]. It proves that Xen reaches nearly the performance of a native Linux system when performing CPU-intensive tasks. KVM has a little more overhead than Xen but is still good. When taking disk I/O into account, e.g., when compiling a Linux kernel,[36] Xen's performance is cut to half compared to a native Linux system. Again, KVM has slightly more degradation but provides better I/O performance than Xen. They also tested the isolation of VMs on a single host and showed that both Xen and KVM have some problems with network I/O when VMs are under heavy load. Finally, they showed that Xen outper-

[35] In contrast to Xen which uses a stand-alone VMM approach.
[36] Compilation of a Linux kernel is a wide-spread and often used "benchmark".

forms KVM in terms of scalability e.g., when several VMs are executed on a single host. Since the thesis focused on the area of HPC computing, performance was one of the main reasons to use the Xen VMM for a prototypical implementation.

Yu et al. [251] propose the design of a feather-weight VM to optimize performance of a user's tasks running inside a virtual environment. Such a VM runs on top of a Windows kernel which is shared among all running VMs. Access to physical resources by a VM is intercepted. Each VM needs only minimal resources because most of the resources are shared with the base host environment. During runtime, VMs operate on the Copy-on-Write (COW) layer to ensure strong state and data encapsulation. Because each VM has a very small individual footprint, VMs can be started and shut down rapidly. Virtualization approaches are used on different levels (e.g. on process, file system and kernel levels) to ensure a strong separation between two concurrently running VMs. Nevertheless, since this approach relies on Windows, it cannot be used in standard Grid environments which operate Linux-based compute nodes for job execution. Moreover, this approach tailors VMs tightly to the underlying system/the running kernel and does not support personalized job execution environments for its users.

In spite of the advantages of using virtualization technology in HPC environments new challenges have arisen. The challenges of high-performance I/O as well as that of utilization of multicore technology in today's HPC systems have been examined by Gavrilovska et al. [96]. To gain high performance of multicore systems, they propose modifying the hypervisor's architecture from a monolithic design to a *sidecore* design. The basic idea is to use dedicated CPU cores to handle sensitive calls from VMs to circumvent time-consuming CPU mode switches from handling these calls, e.g. if a VM tries to allocate new memory pages. To increase I/O performance of virtualized systems, Gavrilovska et al. propose self-virtualizing devices in contrast to the approach of a driver-domain as realized in Xen. When using a driver domain, this single domain is responsible for multiplexing one physical device among all running virtual machines, while self-virtualized devices would handle (de-)multiplexing directly on the device. A prototypical implementation showed that both approaches are promising and can obtain an increase in performance.

2.4.2 Virtual System Environments

The term *virtual appliance* was introduced by Sapuntzakis and Lam [192, 190] to describe the disjunction between using and managing a piece of hardware or software:

> A virtual appliance is the state of a real appliance (the contents of the appliance's disks) as well as a description of the hardware (e.g. two Ethernet adapters, 256mb RAM, two hard disks, etc.).

An *appliance* is described as a piece of hardware assembled for a special task. The appliance consists of software designed to fulfill the need within its special environment and a hardware setup which is the same (or at least very similar) for all appliances. The difference between a normal personal computer and an appliance is that the user is not responsible for updating the software stack of her appliance, the appliance provider is. Since software providers avoid updating their programs on a normal user's computer because of unknown interferences with other software (which may result in an unusable computer system), the appliance vendor knows the hardware available as well as all running software on the appliance, therefore she can test and evaluate the quality of an update that she provides to her customers.

A *software appliance* is a bundle of a streamlined custom operating system components and the application. Since the software is deployed as an appliance, the software vendor has to provide it in a special appliance format. This makes the use of open source software more complicated in Grid environments because a user has to build her own appliances. In contrast providing complete virtual machines as execution environments gives the users more versatile configuration options and SSH access to their virtual machines. Since rBuilder can not be used via a Grid Service interface, the service can not be used for unattended installation of virtual computing environments and is therefore not applicable in environments like a Cloud or a Grid.

A *virtual appliance* only differs in the fact that the provided software does not run on real hardware but inside a virtual machine with defined characteristics (e.g. one CPU, one hard disk, two network interface cards, 256 MB main memory).

A solution that provides online creation of virtual appliances is presented by rPath, Inc., a company that provides tools for managing virtualized environments in enterprise environments. The rBuilder tool [183] provides a web interface for VM creation. The core of rBuilder is Conary [244], a software package management system developed by rPath and released under the Common Public License. Using the web interface, users can create new VMs, prepare software to be added to an existing VM and manage existing VMs e.g., apply updates to the installed software. It supports different virtualization technologies like VMware ESX, Xen, Parallels and Microsoft Hyper-V. Moreover, it supports different Cloud environments like Amazon's EC2. Unfortunately, rPath does not release rBuilder as open source but customers need to purchase an enterprise version of the product to handle more than 20 concurrently running appliances. Moreover, rPath does not support batch scheduling environments and can not be linked to an existing cluster CRM.

Novell presented SuSE Studio [159] in mid 2008. SuSE studio is a web interface which allows a registered user to create his own customized Linux operating system within a virtual machine (either Xen or VMware). Thus

SuSE studio and the ICS web interface provide nearly the same functionality. While SuSE targets single servers and end users, the ICS focuses on providing VMs for Cloud and Grid computing environments meaning that all ICS images are ready to use in such an environment (including distributed user management etc.). Furthermore SuSE studio only provides a web interface so neither service-oriented interaction with Business Process Execution Language (BPEL) workflows nor single Grid service calls are possible.

Red Hat's solution, the Virtual Machine Manager [109] is also a tool for creating VMs without a hassle. Red Hat provides four software tools for VM management and creation. The *Virt Install* tool is a command line tool providing an easy way to install new VMs with a Red Hat Linux OS, while the *Virt Clone* tool allows users to clone formerly created VMs. Another tool, named *Virt Image*, can create new VMs from prepared templates. Finally, the *Virtual Machine Viewer* provides a GUI for VM management. It is a Python program providing a GTK-VNC based GUI. It interacts with the libvirt library [133] to gather information about running VMs and can connect to the VMs via VNC using SSL/TLS encryption and x.509 certificate authentication. Red Hat's tools allow users to create VMs for a great variety of architecture such as Intel (both 32bit and 64bit), MIPS, Sparc and PowerPC and others. Unfortunately, only Red Hat-based Linux distributions (like Red Hat or Fedora) can be installed to the guests, and the tool set lacks the ability to import or export VMs. Since the Red Hat tools target the end-user desktop environment, multi-site management of VM images is not supported.

2.4.3 System Management and Administration

Foster et al. [73] have identified the need to integrate the advantages of virtual machines in the cluster and Grid area. It is argued that virtual machines offer the ability to instantiate an independently configured guest environment for different users on a dynamic basis. In addition to providing convenience to users, this arrangement can also increase resource utilization since more flexible, but strongly enforceable sharing mechanisms can be put in place. The authors also identify that the ability to serialize and migrate the state of a virtual machine opens new opportunities for better load balancing and improved reliability that were not possible in traditional Grid environments [73]. Finally, the level of security is increased by the use of virtual machines as computing environments [170, 175, 185].

The Globus project Virtual Workspaces [118, 255, 120, 121] and its successor Nimbus [122, 119] is a set of open source tools that provides an IaaS Cloud computing solution. It allows the dynamic creation of virtual machines in which Grid jobs are placed and executed. This provides the seclusion of Grid jobs from other jobs and running software, making it impossible for an unauthorized user to track operating system activities during job execution. These projects are using the Xen virtualization technology for the dynamic creation

and deployment of virtual machines. Both projects offer an on-demand setup mechanism to provide virtual clusters but are not usable in traditional cluster scenarios, in which a batch scheduler decides where computational jobs will be executed. In addition to on-demand provisioning of resources, the XGE provides a transparent mechanism to enable the use of virtualization technology by local batch schedulers such as the SGE and Torque. Furthermore, Virtual Workspaces/Nimbus do not offer a Grid service interface that provides software management to the virtual machines via the service interface. Moreover, Nimbus makes use of the concept of virtual appliances [77, 213], which is a promising concept when using VMs among different clusters or Cloud sites.

Automatic setup and administration of a virtual cluster is also described in [151]. The authors present a system based on NPACI Rocks [168] that can set up a virtual cluster for every user on demand. The system is based on the VMware server software to shield the environments of different customers. The use of full-virtualization causes a drawback in performance compared to the use of para-virtualization. Time slots on the physical hardware are allocated by advanced reservation and thus disable local scheduling strategies from achieving an optimal utilization of the underlying hardware. Furthermore, this concept of virtual clusters has disadvantages in the field of Grid computing because existing meta schedulers like GridWay are unable to work with a dynamically changing number of Grid sites.

Another approach to automatically set up virtual clusters was presented by Nishimura et al. [158]. The basic idea is to speed up the provisioning process by caching common software installations within a virtual disk cache. A user sends a request to the installation server containing hardware (CPU type and speed, amount of main memory, disk size and the number of nodes) and software specifications (OS type, user-level software packages). The install host contacts the resource monitor and requests to create VMs on the particular physical hosts selected to execute the request. The user's software requests will be installed into these VMs. The proposed solution relies on the OS' package management system and a cluster software installer. Individual software installation is not provided by the OS-specific package management system, preventing the installation of commercial or custom software products. Moreover, users cannot configure the VMs to their needs.

An approach following the terminology of virtual appliances uses VMs to provide personalized execution environments and was developed by Bradshaw et el. [23]. The virtual appliances are realized as Xen VMs. The system embraces a virtual appliance's entire lifecycle containing the image generation process, image management and image deployment. The proposed image generation process does not allow users to modify their images themselves but relies on the existence of image providers. These image providers are responsible for creating and managing virtual appliances for a set of users e.g., a VO or a group of users that need the same application. Once created by the im-

age provider, the appliance is handed over to a deployment service e.g., the Nimbus workspace service developed by the Globus team. This service is responsible for deployment decisions that match the resource provider's policies. Starting the virtual appliance on the computational resources is called *Contextualization*, a term also introduced by the Globus team [122]. It denotes to the assignment of a particular configuration such as the main memory's size and the number of virtual CPUs to the appliance. Nevertheless, the appliance's lifecycle management is restricted to a single site and does not support the interconnection and sharing of appliances among different resource sites.

Bjerke et al. [21] introduce tools for three aspects of VM image creation and management. First, they provide a Python library and standalone application to create new images. Several Linux distributions are supported and can be installed into the new VM images and set up by using their particular package management system. Second, an application was developed to provide a GUI to generate and manage images. This GUI is realized as a website which uses the library mentioned above. Finally, they propose VM image management by using a content-based transfer technique to minimize data transfer. Content-based transfer uses a bit by bit comparison of files and transfers only blocks that have been changed to the destination. Although it speeds up the transfer process, this approaches lacks the ability to deal with images that do not contain only software provided by the package management system. The advantages of content-based transfer rely on the fact that (at least a part of) an older version of the VM image to be transferred already resides on the destination host. Moreover, it cannot be used with multicast technology, which is also a promising approach for VM image deployment in a cluster network.

Engelmann et al. examined the use of virtualization technology to achieve simplified management and utilization of large-scale computing systems [61]. The proposed approach employs a system-level VMM to execute the virtual machines. Additionally, mechanisms are put in place "in order to provide a powerful abstraction for portability, isolation, and customization of the entire software suite of a HPC system." Unfortunately, no implementation was given addressing the identified challenges, e.g. the use of the local CRM for scheduling decisions.

The libvirt API project [133] tries to provide a unified interface for VM management, such as starting or shutting down VMs. Moreover, it provides the information regarding running VMs via the API. It supports a large number of virtualization products such as Xen, Qemu, KVM, OpenVZ, VMware products and more. It is supported by Red Hat as an "Emerging Technology Project" and promises to be a good foundation for VM management systems. Due to the wide difference in virtualization products, libvirt does not support the creation of new VMs.

Könning et al. [127] focus on the management process of adapting an ex-

isting environment from an old HPC system to a new one without the need to recompile self-written software and adapt it to the new hardware environment. This approach should also be suitable for providing a homogeneous software environment among different HPC centers. Environment configurations needed by the software are stored in an eXtensible Markup Language (XML) formatted file containing necessary information about shell environment variables and the file system structure. As virtualization solution chroot was chosen since it has nearly no overhead. Nevertheless, it has been shown that chroot is not suitable in Grid and Cloud environments in which users must not see other users' actions [206]. In addition, using chroot instead of VMs would result in significantly limiting the tasks that users can perform in their job execution environments.

An approach for deploying software to a set of nodes without using virtualization technology is presented by Sabharwal and Guijarro [187]. They introduce Avalanche, a tool that deals with the entire lifecycle of software deployments like installation, configuration, updating and deinstallation. Since it does not use virtualization technology, its software requirements are very simple: all that it needs is a remote login facility like Secure Shell (SSH). Avalanche uses a centrally installed server component and clients running on the maintained nodes. The server provides a software repository and can induce the clients to perform particular operations needed for software management. It also provides scheduler support to install software on the least used nodes. Although this system is helpful for maintaining software installation on large physical cluster systems, it is not useful in virtualized environments, in which an already existing configuration must be deployed to a set of physical nodes or a set of VMs must be created and configured on demand. Unfortunately, Sabharwal and Guijarro does not provide measurements for software installations to compare the performance of Avalanche with existing VM deployment mechanisms. Moreover, users cannot install software on their own and modify the job execution environment to their needs.

Cloud resource providers such as Amazon [6] provide tools to set up and manage VMs in their environment but do not focus on sharing these VMs with other sites outside their administrative domain. Cloud computing follows a single provider paradigm, and Cloud providers are not yet interested in providing their users with a computing platform that spans over several different commercial Cloud providers.

2.4.4 Resource Management

Freeman and Keahey present an approach for distributing and using VMs in a Grid and Cloud environment [255, 121, 120, 77]. The term "Virtual Workspace" was coined by the Globus Team in 2005 [136] and describes a personalized execution environment for (Grid) jobs. Usually, a virtual workspace is realized using a VM. The Globus team decided to use Xen para-virtualization

technology due to its small overhead compared to a full-virtualization solution. The virtual workspaces service can be operated in two different modes. First, it can be used as a Cloud middleware providing the needed amount of particular VMs to the customer upon demand. Comparable with the Amazon EC2 interface, resources are provided on demand for the time needed. The second mode focuses on batch job scheduling as common in Grid computing. Using the existing CRM, a *pilot* job is scheduled to the nodes that should execute the VMs for the Grid jobs. This pilot job is handled like a normal job by the CRM blocking the particular slot on the execution host for the estimated runtime. Thereafter, the virtual workspaces service can start user-specific VMs on the nodes executing the pilot. This allows virtualization technology to be used in the Grid without modifying the local CRM. On the other hand, using an unmodified CRM has a major drawback: Using Nimbus, users must explicitly acquire virtualization environments in order to prevent execution of their computational jobs on native hardware.

Another approach, called OpenNebula, was developed to enable efficient scheduling in virtualized environments [215, 216, 163]. Therefore, the OpenNebula framework uses leasing, and not jobs, as the fundamental resource provisioning abstraction. OpenNebula is an open-source toolkit used to easily build any type of Cloud: private, public and hybrid. OpenNebula has been designed to be integrated with any networking and storage solution to fit into existing data centers. In traditional batch execution environments, jobs are tied together with the provisioning of the resources to be used by this job and its execution. Customers are forced to use the interface provided by the CRM in order to use the resources, which is suitable in batch scheduling environments but can hardly be used in a service-oriented architecture without a middleware providing a service interface. The proposed solution for that problem is to provide raw resources to customers, who are then responsible for accessing and scheduling jobs to the provided resources.

An approach using OpenNebula combined with a traditional CRM, the SGE, has been presented by Moreno-Vozmediano et al. [146, 147, 144]. This approach aims at bringing different Clouds together for batch job scheduling. However, it has several drawbacks. The presented solution does not provide a dynamically changing set of computing resources. The CRM must be reconfigured after Cloud resources have been booted in order to schedule jobs to the new nodes i.e., because of the fact that the IP addresses of the new Cloud nodes are not known in advance. Furthermore, a direct network connection must exist between every worker node and the CRM's headnode in order to schedule jobs to the compute nodes. This may be true for publicly reachable Cloud resources, but in typical Grid computing environments, compute nodes reside in a dedicated network and are usually unable to reach an address outside of this network.

A system designed for utilization in batch scheduling environments has been

developed by Grit et al. [98] and is called Job-Aware Workspace Service (JAWS). JAWS introduces a new software layer below the middleware and the OS called Shirako [111]. JAWS is responsible for allocating physical resources to execute Xen VMs that are used for batch job scheduling. The scheduling decisions are made by Shirako's resource broker which supports sophisticated scheduling algorithms, including Earliest Deadline First, Proportional Share, and Backfill. The jobs are then executed by Plush [5], a tool designed to manage applications running over large-scale distributed systems. Nevertheless, JAWS is not compatible with existing Grid middlewares that provide a service interface for job submission by external users. Moreover, since JAWS relies on Shirako, existing cluster infrastructures could not be used without substantial initial effort. Since CRM systems like the SGE are common in the HPC environments, an experienced cluster administrator will usually not change the CRM component.

Lagar-Cavilla et al. presented a system called SnowFlock [129]. In a Cloud environment, SnowFlock enables rapid instantiation of a VM. To achieve this target, SnowFlock implements a technique called VM fork. Comparable to the Unix `fork` command for processes, the Xen VMM has been modified and userland software has been developed. The fork process is divided into four parts. First, the parent VM is suspended for a short period of time while a VM descriptor is produced, which contains the VM's metadata and guest kernel memory management data. This descriptor file is distributed to other physical hosts to spawn new VMs. Second, needed data from main memory is transferred lazily and on-demand from the parent VM to the already running child VM. Third, the amount of data to be transferred is minimized using the avoidance heuristics strategy. Finally, the VM's state is transferred simultaneously and efficiently to the other children using multicast technology. While allowing the rapid cloning of an existing VM, SnowFlock relies on some modifications of both the Xen VMM and the VMs' kernel. This system may be used in a closed virtual cluster environment in which VMs are generated in a controlled manner. When using VMs that do not contain the necessary modifications, this system cannot be implemented.

Rodríguez et al. have developed an approach for virtualized resource management in HPC nodes [182]. This approach aims at improving resource utilization through efficient partitioning and fine-grain management through multiple dimension slotting. The approach counters the waste of resources caused by an imbalanced workload: While some physical nodes host all VMs in a system, other nodes are idle. A prototype has been implemented using Xen as virtualization technology. A management component runs in the privileged dom0 on the physical nodes and can start VMs. Moreover, single physical nodes can be grouped into subsets. These subsets are seen by the local CRM as resources available for job execution. The distribution of jobs within a subset is maintained by the proposed management component. During runtime,

the management component can adjust the hardware assignment to the particular VMs executing the jobs. Focusing on providing a Cloud infrastructure, it is not a suitable solution for HPC batch job scheduling environments which make use of CRMs to obtain optimal job scheduling. Moreover, the proposed system does not support VM image exchange across multiple sites which would require users to manually distribute images on an as-needed basis.

The question of how to achieve Quality of Service (QoS) goals in an environment with dynamically changing resource needs is addressed by Padala et al. [166]. In traditional data centers, resources are assigned to each hosted application in a static manner. Because demand varies over time, resources are over-provisioned [95]. Many hosts are underutilized while other hosts are heavily loaded and there is no way to dynamically readjust resource allocation. To counter this problem, a system is proposed that implements dynamic resource allocation using Xen virtualization technology. In addition, the system takes QoS goals into account when reallocating resources. Moreover, different applications can be prioritized in order to guarantee basic functionality when all resources are under heavy use. It is a promising approach for guaranteeing optimal resource utilization in a Cloud computing environment. In batch-scheduling environments, this technology is not useful since resources are managed by a CRM and all physical resources are allocated to a job to ensure highest performance.

The Haiza Lease Manager [217] is a scheduling component which can be closely coupled to the OpenNebula framework. If customers want to access VMs, they request the desired amount of resources from Haiza. The scheduler keeps track of resource usage and provides virtual resources as soon as possible. Furthermore, Haiza supports advanced reservations, which can cause some trouble when using OpenNebula in a hybrid setup containing both virtualized and non-virtualized resources. In a traditional cluster systems, it must be ensured, that the desired number of needed resources are freed at the moment of the advanced reservation. Using virtualization technology, the needed number of VMs can be suspended in favor of providing the needed number of resources to meet the advanced reservation's needs. This approach is suitable when managing resources in a Cloud computing environment, but it does not provide any scheduling implementation for batch job execution at the moment.

Eucalyptus [160, 161, 162] has been developed as an open-source Cloud infrastructure framework that provides IaaS to its users. It is designed to be portable, modular and simple to use on infrastructure "commonly found within academic settings." An interface is provided similar to the EC2 interface that allows the use of the same software tools for the Amazon Cloud and the Eucalyptus Cloud, making it suitable for creating a private Cloud e.g., within a company, which can be easily extended by a public Cloud if more resources are needed. Eucalyptus has three major components: the Cloud controller, which provides the frontend services and the Walrus storage system, the clus-

ter controller, which provides support for the virtual network overlay, and the node controller, which manages the particular VMs on the physical nodes. Moreover, the Walrus storage system provides a simple storage service comparable to Amazon's S3 service. The node controller is designed to work with Xen as virtualization solution. Hence, no hardware virtualization support is required to install Eucalyptus when providing only Linux VMs. Nevertheless, Eucalyptus also supports KVM and support of VMware is planned. Eucalyptus is a promising approach for building Clouds that provide the same interfaces as Amazon, but like a pure IaaS system, Eucalyptus does not contain any scheduling mechanisms, meaning users are responsible for scheduling their jobs among the different VMs. Although tools for Amazon EC2 and S3 can be used with Eucalyptus, it does not support image exchange between the open source and the commercial Cloud framework.

VirtuaLinux [4] has been developed to ease cluster installation and management tasks and to provide users with a cluster system that can be adopted to their needs. Moreover, it promises to remove the single point of failure, the cluster headnode by offering services repeatedly on the cluster's nodes. VirtuaLinux is a very small Linux distribution that offers the ability to create, boot and manage Xen VMs that can be booted as needed. Based on this, a cluster can provide Unix- and Windows-based resources at the same time. Moreover, the VirtuaLinux storage system supports diskless compute nodes which often can be found in the blade systems of large data centers. The VMs' images can reside on NAS system that is available cluster-wide. The cluster administrator decides which VMs are booted on which physical nodes. Therefore, VirtuaLinux does not support user-specific VM deployment on the compute nodes depending on the jobs being executed within the particular VMs. The use of NAS systems to distribute VMs quickly and efficiently is also not suitable to manage VMs between different, physically separated clusters.

If local resources are insufficient for meeting the demand for computing resources, Cloud computing resources can be used to provide additional resources to the customers e.g., to fulfill SLAs. Vázquez et al. [232] propose a system that monitors existing (Grid) resources in order to obtain new (Cloud) resources when load exceeds a given threshold. One problem is that Grid interfaces do not necessarily match existing Cloud interfaces. Thus, a solution must be provided ensuring interoperability. One basic building block of the proposed solution is the GridWay meta scheduler used to distribute jobs among different resource sites i.e., the local Grid system and remote Clouds. A major drawback of this solution is that GridWay does not support the dynamical addition of new resources but needs knowledge of existing resource sets at the beginning of its execution, which requires GridWay be restarted every time new resources are added to the system. Moreover, the proposed solution relies on Nimbus as Cloud backend, so virtual resources must be requested explicitly before a job is executed within the VMs.

The GridWay meta scheduler [107] has been developed to achieve scheduling capabilities among a given set of Grid sites. Instead of submitting Grid jobs directly at a Grid site's headnode, users can submit their jobs at a GridWay server. GridWay itself knows the state of every connected Grid site, such as its current workload or the number of free nodes. Based on this information, computational jobs are scheduled to the Grid site best suitable to the particular Grid computational job's needs. Gridway has limited information available to make scheduling decisions and is only able to schedule to already known resources, though adding resources during runtime is not supported. Both topics are addressed by the Business Experiments in Grid (BEinGRID)[37] project funded by the EU. In this project, the eIMRT sub-project offers services for radiotherapists to speed-up and optimize radiotherapy treatments. BEinEIMRT [25] is the application of eIMRT to a Business Environment. It implements a SLA component achieving automatic negotiation with external providers and dynamically addition of external resources during runtime allowing the administrator to add and remove resources without needing to restart GridWay itself. Unfortunately, adding new resources relies on SLA negotiation, which makes it very complicated and it aggravates the management of the software. Furthermore, the proposed component was not made public by the authors. Thus, this solution could not be evaluated during the course of this thesis and implementation details remain unknown.

A solution to extend local computing resources with IaaS Cloud resources has been developed by Marshall et al. [140]. Based on the Nimbus toolkit, a resource manager has been built that can acquire additional resources from Cloud providers if the local resources become insufficient. Therefore, an additional component has been developed that enables users to monitor existing resources and add or remove external (Cloud) resources. The decision of requesting additional nodes from an external resource pool is made by policies that take the length of the local CRM's job queue into account to obtain a complete picture of the state of the local resources. Three different built-in policies are used to decide when and how many external resources should be acquired. The VM image must be prepared at the resource provider's site in advance to be used later on. This solution allows the system to scale vertically to an external provider's resources if workload increases. Scheduling decisions are made by Torque, an open-source CRM. Cloud resources are registered at the Torque headnode once they appear, so the CRM's headnode must be accessible to the Cloud resources (which is not usually the case in Grid environments). At the moment, only one external provider is supported and it is not possible to prevent specific jobs from being scheduled to the external site.

Extending existing Grid systems using Cloud resources is also discussed by Ostermann et al. [165]. They focus on extending a Grid system assembled for executing workflows which are scheduled to the resources by the

[37] `http://www.beingrid.eu/`

ASKALON development and computing environment [64, 63]. To enable the system to start and shut down Cloud resources, ASKALON's resource management component has been replaced. In contrast to the solution above, different Cloud computing providers are supported. Resources are acquired per job, which allows every user's credentials to be used to acquire new resources. This is a promising idea since it allows users to specify the amount of money they would like to spend on external job execution. On the other hand, the proposed system relies on a modification of the ASKALON middleware, which is not suitable for batch job scheduling.

Another approach focused on extending local resources with Cloud resources for workflow processing is presented by Dörnemann et al. [57, 56]. They use ActiveBPEL as the workflow modeling language. The presented solution can be used to extend an existing Grid system with resources from the Cloud. On the Grid site, the Globus toolkit is used, which not only supports the execution of workflows by ActiveBPEL [60], but also enables batch job execution via the WS-GRAM interface. In order to support Cloud resources, the ActiveBPEL engine has been extended with components for requesting Cloud resources and analyzing current system load as well as a load balancer. In addition, the presented solution only supports the execution of BPEL workflows. Although the same Grid resources can be used, Cloud resources acquired by the presented software are only being used for workflow execution. Nevertheless, it has been shown that Cloud resources can be employed to provide fault tolerance and load balancing.

A comparison between scaling strategies (scale up vs. scale out) is given by Michael et al. [142]. Hosting applications on large servers with a large number of resources is referred to as *scale up*. In contrast, *scale out* means that applications are installed on multiple small servers e.g., clusters. With Cloud computing, scale out systems have become popular, being the only way to provide the needed computational power. The authors showed that instead of a real scale up approach on a modern scale-up machine powered by IBM POWER5 CPUs, a mixed approach is more useful. Instead of starting just one instance of an application using all CPUs on the system, starting several instances with few CPUs each results in a better overall performance. Moreover, the tested scale-out system, an IBM BladeCenter, which is as expensive as the Sun system, performs nearly four times better in terms of queries per second using a search workload of Nutch/Lucene [29, 94]. A disadvantage of the latter system is the complexity of performing management tasks caused by the utilization of multiple OS instances in contrast to a single OS used on the scale-up system.

Although Grid computing is a good example that demonstrates how to achieve interoperability between different resource sites by providing a middleware that makes Grid job execution available in a unified manner, interoperability can nevertheless be further optimized. Boardman et al. [22] see the services

provided by Grid computing as "smallest common denominator on which the community can agree." They found that particular user groups developed their own middlewares reflecting their needs. Moreover, these middlewares usually do not provide interoperability among each other. The authors state that many of the existing standards developed by the OGF, OASIS, World Wide Web Consortium (W3C), and the Distributed Management Task Force (DMTF) overlap extensively and contain inconsistencies. To achieve an interoperability using the existing standards is the main target of the Grid Interoperability Now (GIN) initiative. "Interoperability Islands" should provide basic services such as Job execution or VO management among others. The proposed solution by the GIN initiative targets only the Grid community and is not suitable for Cloud computing environments since it does not eliminate the data lock-in effect that can be found in Cloud computing nowadays.

The RESERVOIR system aims to provide an interoperable Cloud infrastructure [181]. Having successful web projects like MySpace or YouTube in mind, which could gather 20 million users within two years or less, existing Cloud providers would encounter scalability problems if they hosted a comparable project. The proposed model enables multiple independent Cloud providers to cooperate seamlessly. From a business perspective, business service management and SLA management must be supported throughout the new paradigm called a *federated Cloud*. At the moment, the proposed solution only targets service applications that can be customized easily via a web interface. However, this solution does not take the management of personalized VMs between several independent sites into account. Therefore, it cannot be used in (batch) job execution environments e.g., to perform simulation tasks.

2.4.5 High Availability and Fault Tolerance

It is difficult to migrate processes from one computer to another during execution, and it usually leads to further difficulties within the system [143]. Process migration was a hot research topic in the 1980's [173, 225, 117, 13] but has seen very little use. When using VM technology, not only the particular processes are encapsulated in a closed environment but also the OS, the installed applications and all data inside the VM. Since a VM needs a VMM to be executed that provides a well-defined interface for the existing hardware and intercepts all access of the guest OS inside the VM, it facilitates new opportunities to counter the problems associated with the live migration of running processes to achieve high availability or load balancing.

Work has been done in the area of live migrating running VMs to avoid long downtimes for services hosted within these VMs. Live migration is essential in both service-oriented environments and environments that target batch execution of computational jobs to enable the resource provider to maintain physical resources without impacting the provided services. Moreover, live migration of VMs can be used to achieve load balancing if workloads within the partic-

ular VMs change over time. Finally, it can be used to switch off unnecessary physical nodes to cut down operational costs.

Clark et al. [35] implemented a "high-performance migration support for Xen" targeting on minimizing both the downtime of a VM and the total migration time. The basic idea is to use a pre-copy approach, transferring all memory pages from the source to the destination host before the VM itself is transferred to the destination host. Execution of the VM is not interrupted when memory pages are pre-copied; only during the final phase, the VM is paused on the source host, network traffic is redirected and the CPU's state is transferred to the destination. Thereafter, the VM is resumed. Memory pages that have been altered during the pre-copy operation are then pulled from the source host once the VM's execution has continued on the destination host. The presented approach lacks the ability to transfer disk images from one host to another; it is assumed that disk images reside on a network share and all hosts have access to this resource. Decentralized storage of disk images to prevent the centralized storage from bottlenecking is not supported by the proposed solution.

With the availability of VMs and the ability to create VMs easily, transfer mechanisms become crucial since many VMs are used for daily work and can reduce the need or entirely replace employee laptops that must be available within the company and for home usage. Problems that arise in this scenario must be discussed because high-bandwidth network connections between the company's servers and the employees homes are not common [191]. A machine's state is referred to as a capsule which contains the OS itself, the installed applications, all data, and any running processes if the machine is active. This capsule can be serialized and transferred from one node (e.g., the employee's workstation) to another (e.g., the home computer). Some optimization techniques are used to speed up the transfer while users can continue working with the capsule even if not all data has been transferred yet. To minimize the amount of main memory to be transferred, a ballooning mechanism is used which causes the capsule's OS to swap all data which is not absolutely necessary to disk. COW technology is used to minimize the time needed to transfer disk images between physical hosts. Furthermore, demand paging fetches only the portion of the capsule disk requested by the user's tasks. Finally, a hash mechanism has been implemented to make use of similarities between related capsules, which speed up the migration of a capsule when transferred over slow networks. The presented approach guarantees a minimum downtime of the VM during the migration process. Although they can transfer VMs through low bandwidth networks, they focus on migrating running VMs, which is not useful in the Grid scenario. In a Grid, VMs are always shut down when they are deployed into a cluster network by the ICS and will be copied to the target compute nodes before they start executing a computational job. By using multi-layered file systems, only the users' data layer has to be transferred to remote sites after they have modified it. Furthermore,

Grid sites are always interconnected by a high bandwidth network, so this approach may not help to save a significant amount of time when transferring images while even though it requires much effort to implement this technique to the management component.

The approach presented by Bhatia and Vetter in 2008 [18] also uses Xen virtualization technology to manage a cluster of virtual machines. Each physical node is observed by a running daemon. This daemon is queried on a regular basis by a central program which provides information about all connected nodes to the administrator. Moreover, depending on the system's state, the administrator can perform three types of administrative actions: dynamic load balancing, preemptive node maintenance and live migration of VMs. The first two actions rely on live migration. Since the presented system seems to be a reasonable solution for virtual clusters, batch job submission with personalized working environments is not supported by this system.

2.4.6 Input/Output and Storage

Edward Walker examined the performance of Amazon's Cloud computing service EC2 in view of high performance scientific applications [238]. Therefore, Walker conducted some macro and micro benchmarks from the NAS Parallel Benchmarks (NPB) suite [12, 47] and compared the performance of EC2 with cluster system nodes with similar CPUs and main memory. However, benchmarks show a performance degradation of 7% – 21% when running on a single host i.e., a VM containing 8 CPU cores. Moreover, when using MPI communication between the nodes, results are even clearer: Benchmarks show a performance degradation of 40% – 1000% when using 4 EC2 instances with 8 CPU cores each connected with Amazon's network compared to 4 physical cluster nodes with 8 CPU cores each connected with IB. Further investigation of the MPI performance shows "that message-passing latencies and bandwidth are an order of magnitude inferior between EC2 compute nodes compared to between compute nodes on the NCSA cluster." In his 2008 publication, Walker concluded that Amazon's EC2 has not yet matured for the use of HPC computations.

Another examination of whether Cloud computing can be used in the area of HPC has been done by Napper and Bientinesi [153]. They focus on high-performance numerical applications using the High Performance Linpack (HPL) benchmark suite [51, 52, 254] because the artificially generated workload is similar to a workload generated by scientific applications and the performance of the benchmark scales linearly with the size of the resource set. The results are similar to the results presented by Walker. When using numerical applications, an EC2 node can compete with a physical cluster node. This changes dramatically when MPI communication comes into play. In contrast to traditional HPC cluster systems, "the GFLOP/sec obtained per dollar spent

decrease exponentially when increasing computing cores" resulting in the fact that the cost for solving a linear system increases exponentially with its size.

Focusing on Cloud resources as computational resources up until here, Palankar et al. [167] examined the use of Amazon S3 in the area of Grid computing. They examined data durability, availability and transfer performance (single-threaded and concurrent file access) and conclude that transfer time depends on the location of the nodes accessing the data. Unfortunately, data accesses from Germany and France were stuck to around 300 KB/s while accesses from Stony Brook, NY, USA obtained a bandwidth of around 2 MB/s (for large files). The examination also showed that the transfer rate relies on the size of the files to transfer: the bigger the file, the better the performance. Besides that, the authors state that S3 can be used if high data availability and durability must be ensured, but due to the performance differences, caching mechanisms should be deployed on the sites accessing the data stored on S3. Furthermore, S3 does not provide sufficient security mechanisms e.g., fine-grained access control lists are not supported.

To thwart the virtualization overhead during I/O operations, Yu and Vetter examined whether high performance parallel I/O with negligible overhead can be achieved in a Xen-virtualized HPC environment [250]. They used the IOR benchmark [200] and a NAS working with Parallel Virtual File System (PVFS) [31] to carve out the configuration providing the best parallel I/O performance with Xen. Instead of using Xen's standard TCP configuration, Yu and Vetter applied optimizations like checksum offloading, TCP segmentation as well as scatter/gather support. Besides standard network interconnects, they also tested InfiniBand (IB) communication links and show that IB [201] can provide nearly the same performance in virtualized and native environments when using VMM-bypass IB support [134]. This work shows that virtualization technology can be used even in HPC environments. Nonetheless, the authors conclude that more work has to be done to optimize performance of VMs.

The same topic is examined by Raj and Schwan [177]. They focus on a self-virtualized network interface card, targeting the high performance domain. The network card itself provides virtual interfaces and manages the utilization by guests (VMs). Therefore, it provides better performance than network interfaces multiplexed by a driver-domain (e.g., the Xen dom0). The authors show that significant performance gain can be reached when using hardware-supported virtualization instead of current standard virtualized device implementations on hypervisor-based platforms.

The ViNe project [226, 227] proposes a virtual networking approach for Grids. It implements an overlay network which can easily be administrated and provides logical grouping of resources in a private network. By targeting the HPC environment (Grids), it must provide high network performance. ViNe can also be reconfigured dynamically and provides information about ex-

isting hosts in the network. ViNe has been designed to overcome limitations of VPN [89], SSH tunneling or Generic Routing Encapsulation (GRE) [68] such as no support for hosts using the same (private) IP address, high administration overhead and lack of flexibility in configuration regarding independent virtual networks. If hosts residing on different resource sites cannot reach each other, e.g., because they are in a private subnet and are not accessible to the public , a dedicated ViNe host that can be reached by all participating hosts serve as a queue server to traverse the firewalls.

Soror et al. [214] have examined the possibility of automatically configuring VMs, specifically in cases that provide database services. Their approach to optimizing the performance of VMs is a *Virtualization Design Advisor*. This advisor monitors the VMs' specific workload and adjusts resource assignment to each VM over time to gain optimal performance. In experiments, an improvement between 2% and 20% could be shown. Optimization of hardware allocation to VMs depending on the kind of applications running within them is only suitable if workloads of a specific kind are performed. Within a Cloud or Grid environment that enables users to execute whatever software they please, users will likely have little to no experience with automatic optimization of performance using the proposed advisor.

2.4.7 Security

The basic idea of VM-FIT [179] is that services are hosted in a virtualized environment which is observed by a secure hypervisor. If an attack occurs, the virtual environment can be restored from a clean image and continue operation. Moreover, VM-FIT also supports tolerance to Byzantine faults by sending incoming requests not only to one VM but to many, called a replica group. Obviously, this only works for simple services. In a HPC environment that aims to minimize job runtimes and to maximize throughput, jobs cannot be executed multiple times since this would have negative impact on either resources available to other jobs or the execution time of a particular job.

This approach has been extended by Smith et al. [210]. The public Grid headnode is usually the only host accessible from the internet, so attackers will most likely attack this headnode. For that reason, the headnode is often located in a Demilitarized Zone (DMZ). Attacking this host may have impact on a Grid site's functionality or an attacker may have access to sensitive data. The basic idea to circumvent this threat is to put all headnode services into a VM that is regularly replaced by a new VM containing the same software stack e.g., by using the same read-only VM image file. This principle is dubbed server rotation. Since Grid services are stateful in contrast to Web services, a dedicated database host must exist which is used to store the Grid connections' state and make them available to the new headnode VM. If a server rotation is planned, but active connections exist, the rotation will be postponed until these

connections are closed or a timeout occurs. This approach is suitable to secure a publicly accessible node providing Grid services to prevent attacks.

Ensuring a high level of security in computing systems is the target of the Trusted Computing technology. Trusted Computing (TC) relies on a single chip on the mainboard which needs to be trusted, the Trusted Platform Module. Based on this module, a Basic Input/Output System (BIOS) extension called Core Root of Trust for Measurement is the first stage of a secure boot process. Based on this component, each following stage of the boot process is measured by a hash function which stores the result in the chip. If this result differs from the reference result which has been stored earlier and represents a trusted system state, the boot process aborts, preventing the use of the untrusted system. The state of a particular system can even be controlled from a remote server which can prevent the use of untrusted software, which is called remote attestation.

The Trusted Computing Base of a computing system includes all components that must be trusted in order to trust another component relying on these components. Hohmut et al. "refer to the set of components on which a subsystem S depends as the *TCB as S*" [105].

The system presented by Stumpf et al. focuses on countering insider attacks by using virtualization technology and Trusted Computing Platform (TCP) components (like a Trusted Platform Module (TPM) chip) [220, 221]. In their paper, a software architecture is presented that allows documents to be edited in a secure manner. The architecture consists of four layers. On the lowest layer, the TPM chip ensures that the hardware as well as the booted software are unmodified and in a trusted state. The next layer consists of the VMM and the management VM. On the third layer, two VMs can be found: one to run normal applications like a web browser and a trusted VM that makes use of the vTPM chip, which is bound to the hardware TPM chip of the physical machine. The vTPM chip ensures that the trusted VM has not been modified. This VM provides the document editor, which is checked for manipulation every time the trusted VM boots. The last layer consists the document editor itself, which is installed on the user's machine. To maximize security, it has been written in Java to minimize the risk of buffer overflows. The document editor is the only component that can be used to access and edit the documents. Although the proposed system can provide a very secure document editing system, it cannot be adopted easily to other use cases. Moreover, TPM chips are not as widely available on HPC hardware as on consumer hardware.

Murray et al. present a system based on disaggregation to shrink the Trusted Computing Base (TCB) of a VM in a Xen system [149]. Usually, the TCB of a Xen VM contains the Xen VMM, the dom0 as privileged management VM, its kernel, the code responsible for creating new VMs, and the Python interpreter, as the latter component of Xen is written in Python. A raise in security can be

achieved by minimizing the TCB. Moreover, transferring the code necessary for creating new VMs in a trusted VM outside the management VM would remove the requirement that the privileged domain, which is usually used by the system's administrator, must be trusted. This is achieved by moving critical operations for VM creation into a dedicated trusted VM, the so-called domain builder. In contrast to dom0, the system's administrator does not have access to this VM. The new TCB is much smaller because it contains only the Xen VMM and the domain builder VM. Nevertheless, the authors do not provide any measurements that show that this approach can currently used. Again, TPMs are not very widespread in the HPC environment which render the use of the presented approach impossible.

Focusing on the HPC environment, Löhr et al. propose the use of TC technology in Grid environments [135] "to enforce *multilateral security* i.e., the security objectives of all parties involved." To achieve this goal, a Grid job submission protocol has been developed that is based on offline attestation. Only if a (previously selected) Grid provider's state is trusted, the Grid job will be submitted to the provided resources. Moreover, the proposed protocol also supports delegation i.e., enabling the Grid provider to use other Grid provider's resources if the user also trusts them. Up until today, this conceptual work has not been implemented in an existing middleware used in the Grid today. Moreover, the protocol is not compatible with existing Grid resources.

Wei et al. [178, 239] present a system that ensures the integrity of VM images and a way to share images safely e.g., between image providers and users. The presented system, called Mirage, provides an access control framework to prevent unauthorized access to VM images. Moreover, image filters can be applied to VM images to remove unwanted information e.g., from each image before transferring it to other users. Moreover, a provenance tracking mechanism tracks the derivation history of each image and a periodic check of already created images ensures the elimination of security flaws discovered after a particular image has been created. These mechanisms are very useful in Cloud and even in Grid environments, especially the filter approach is useful when dealing with VM images uploaded to the resource site by their users. Nevertheless, this approach does not support a distributed environment among several sites. Every site must contain their own repository in order to have all images available.

In large scale environments, several firewalls are usually used for network separation. To allow customers access to a given resource inside the protected networks, firewalls must be reconfigured to allow the customers to log in to a particular resource from an outside network. Especially in Grids and Clouds, users must have the ability to log on to their resources if they are active. Grid sites often prohibit users from connecting directly to the computing resources but provide a single login e.g., to the publicly accessible headnode. Nevertheless, when talking about Cloud computing, this is not a viable solution. Zhang

et al. try to solve the problem of managing and reconfiguring multiple firewalls without conflicts [252]. Thus, a policy language has been developed which can detect and resolve conflicts in firewall rules specified by the administrator. Thereafter, the rules are translated from the policy language to a particular firewall language, such as `iptables`, and can be applied to the firewall hosts. Even if this policy language has been proven to work correctly during the experiments conducted by the authors, it is more likely that endangered resources, such as publicly accessible VMs, are placed in a DMZ or a specially secured network area. The need to reconfigure several firewalls every time a VM starts (and stops) is highly unlikely. Useful in particular computing environments, the overhead of using an additional language to manage the firewalls is not suitable in Grid and Cloud computing environments.

In service-oriented environments, it is common for one machine to host several services. Since these services can be used concurrently by different users, a separation between the particular execution tasks must be provided. Java Web services are secured by the sandboxing mechanisms built in the Java VM i.e., by using the Java security manager. This approach is not suitable for Web services that rely on legacy applications like external C programs. These programs are usually executed with the same user ID on the host machine. Friese et al. [79] present a solution for separating these processes using a fine-grained confinement of native application components. In the case of Java Native Interface (JNI) process execution, the proposed solution decouples the Web service from the legacy software component using a transparent proxy. This proxy starts an independent Java Virtual Machine (JVM) process for each Web service call, ensuring that the different JNI executions are isolated. Based on the transparent JNI proxy, process separation can be achieved through more sophisticated solutions like BSD jails or Systrace. Nevertheless, this approach is limited to service-oriented environments and does not cover process separation in multi-user batch job execution environments.

An approach for observing public nodes is to use a Streaming Intrusion Detection System (SIDS) [211]. In addition to a locally installed Host Intrusion Detection System (HIDS) or Network Intrusion Detection System (NIDS), a SIDS can observe several hosts at a time by analyzing the data streams using the PIPES framework [128]. By sharing the knowledge between all observed hosts, a SIDS can detect distributed attacks against multiple sites which would not be detected by a local Intrusion Detection System (IDS) e.g., a port scan which distributes the scan processes among different sites.[38]

A completely different utilization of virtualization technology that focuses on security has been mentioned by Myers and Youndt [150]. They present the construction of a rootkit based on virtualization technology for AMD CPUs. Rootkits are used by attackers to hide their activity (like processes and files)

[38] Distributed port scans can be successfully used in environments which provide several similar hosts e.g., Grid headnodes.

and present a false view of the system to administrators and other users. While traditional rootkits rely on changes of userland programs, virtualization-based rootkits like Bluepill [186] put an existing (physical) system into a VM. From within this VM, the administrator is not able to track malicious operations performed outside this VM. The paper presents the steps needed to develop and install a hardware-assisted VM (HVM) rootkit. Moreover, a brief overview is given of counter-measures. Although this is a very interesting way of using virtualization technology, it is not useful in a Grid or Cloud environment. Moreover, this paper shows that virtualized environments increase security because the HVM rootkit cannot be installed from within a VM.

2.4.8 Economics

Although Cloud computing has been around for 5 years now, contemporary Cloud providers do not yet offer support for complex SLAs. Buyya et al. [27] analyzed the QoS and SLA mechanisms that must be supported by the provider to attract commercial customers to use Cloud resources in their business processes. First, Cloud providers must provide support for service management based on user-specific profiles and requested requirements. Second, risk management must be put in place enabling users to identify, assess and manage the risk of Cloud resource utilization. Third, market-based resource allocation strategies must be developed in respect to the customers' needs and the risk analysis. In addition, resource management must be able to adapt to changing requirements e.g., by providing particular services upon request and automatic reconfiguration of the existing resources. To achieve these goals, the management component must be able to manage different VMs in a reliable and effective manner. Moreover, the authors confirm many of the requirements which have been carved out in the industrial projects during the course of this thesis (see requirements catalog in Table 2.1 on page 60).

When using Cloud resources which are billed according to the time and storage used as well as the amount of transferred data, it is crucial to make a good estimation for resource utilization to minimize costs without affecting the overall performance of the hosted application. Deelman et al. [44] analyze the cost of the Montage Web service[39] hosted on Amazon resources for both storage and computation. One peculiarity is that data transfer is not billed within the Amazon Cloud, only data uploaded to and downloaded by the user accrues a cost. In addition to the application, a workflow management system, Pegasus [43, 45], has been put in place to orchestrate data transfers to the Cloud. They found that the cost of the computation relies heavily on the number of CPUs used to perform the computation while storage and transfer costs are negligible for their particular workload, although the storage cost decreases the more CPUs are used due to the fact that computation is completed in less time and storage is used for a shorter amount of time.

[39] `http://montage.ipac.caltech.edu/`

To depict the interrelationship between storage costs, communication costs and computational costs, the Communication to Computation Ratio (CCR) is proposed, which is defined as follows:

$$CCR = \frac{\frac{\sum_{f \in F} s(f)}{B}}{\sum_{v \in V} r(v)} \quad (1)$$

Let $F = \{f_1, f_2, \ldots, f_k\}$ be the set of files that have to be transferred to and from the resources (which are billed) and $s(f_i)$ the function that returns file f_i's size in bytes. Let B be the bandwidth of the network connection in bytes per seconds. Moreover, let $V = \{v_1, v_2, \ldots, v_n\}$ be the number of tasks that should be executed and $r(v_j)$ the function that returns the runtime of task v_j in seconds. Following this definition, their application has a CCR between 0.053 and 0.045 showing that computational costs have significantly more impact on the overall costs compared to the costs for storage and communication.

Chaisiri et al. [33] developed an algorithm for optimal VM placement across multiple Cloud resource providers. The basic idea of their approach is to use both reservation and on-demand payment plans like the ones offered by Amazon EC2 or GoGrid. By deciding to use a specified amount of resources in the future, costs are usually lower than using these resources on-demand. Nevertheless, deciding how many resources are needed at a time in the future relies on a good prediction. If the reserved resources are insufficient, one can fulfill unmet needs by acquiring additional on-demand resources. This problem is referred to as underprovisioning. On the other hand, overprovisioning refers to the problem that a customer reserved more resources that she actually needed. These resources will stay idle while generating costs. However, this model takes CPU hours, network bandwidth, transferred data and storage costs into account. The user cannot influence the decision where her jobs will be executed e.g., by excluding particular sites from scheduling. Moreover, the system does not take locally available resources into account, which could be less expensive than resources acquired from a Cloud resource provider.

Bossche et al. present the Hybrid Cloud Construction and Management system (HICCAM) [46] to optimize the costs of scheduling serial and trivially parallel[40] computational jobs in a hybrid environment containing both local and Cloud resources. The authors assume that a deadline on which a particular job must be completed is known to the scheduler. Determining estimations of task runtimes is a complex problem which is not addressed in this thesis. However, much work has been done trying to achieve a good runtime estimation in distributed systems [112, 212, 113, 233]. Scheduling decisions in the HICCAM system not only take CPU hours and the size of the main memory into account but also in- and outgoing network traffic as well as storage costs.

[40] A compute job is trivially parallel if it can be split into several independent tasks which can be computed in parallel.

Nevertheless, finding a solution that optimizes costs can be time-consuming depending on the number of pending jobs e.g., the authors report solve times of more than 27 hours. The scheduling decision only takes the price of external resources into account but cannot be influenced by the owner of the job or the administrator.

2.4.9 Green Computing

Over the past few months, Green computing has gained some attention. Green computing is about the utilization of Cloud computing technologies in order to migrate services provided by dedicated servers to VMs running on fewer physical machines in order to shut down old or energy-consuming hardware or cooling systems. Moreover, sophisticated power-saving functions provided by modern hardware components are used to decrease the power consumption of every physical system. Besides being environmentally friendly, Green computing also reduces costs for the resources' owners. Berl et al. [17] present an overview of research fields in the area of Green computing ranging from energy-efficient hardware to energy-conscious schedulers.

Since Green computing is not the focus of this thesis, only one paper should be mentioned regarding this topic. Nathuji and Schwan [154] provide a set of infrastructure management components that can be used to put through power budgeting policies. Power budgeting is the allocation of available electrical power to a system among the components that need to be active. The basic idea is to manage the power within a VM in order from a VM-centric point of view in order to decrease the entire system's use of energy. Their prototypical implementation in the Xen hypervisor improves the degradation of utilization in a data center by 43% compared to environments which do not use virtualization technology if energy budgets are enforced e.g., by shutting down machines if only a limited amount of energy is allowed to consume. This is achieved by dynamically adjusting resource assignments to VMs in respect to power budgets. The presented solution can easily be used by a resource provider providing Cloud resources based on Xen virtualization technology. Since the components presented in this thesis work with any version of the Xen hypervisor, the architecture's operation is not affected. Moreover, this solution is a good companion technology to the proposed solution in this thesis.

2.5 Summary

The first part of this Chapter has kept track of the developments in computer science from the early beginnings in the 1960s to the modern computer systems. Basic ideas and concepts have been introduced that build the foundation of modern approaches of computer science. Moreover, terms related to the work in this thesis have been defined and explained. Nonetheless, while

presenting the advantages of using virtualization technology, some new threats arose when dealing with VMs. These aspects have to be addressed when using virtualization technology in large computer systems.

Thus, three industrial use cases have been presented. It has been shown that four main aspects are crucial for commercial users:

1. A strong user separation must be provided to hide the users' data and (computational) processes on the given resources. Data is often the most valuable asset a company possesses. Strong security mechanisms must be put in place to ensure that data cannot be accessed by unauthorized third parties.

2. Users must be able to manage the VMs used as execution environments by themselves. Tools must be made available to provide VM management functionality in an easy and comfortable manner. In addition, experienced users or companies must be able to create and maintain VMs that can be used by other, less experienced users.

3. Since most commercial software is based on the Windows OS, the system must be able to execute Windows tasks i.e., by providing Windows-based computational resources and schedulers that are able to deal with Windows computational jobs.

4. Multiple different heterogeneous resource sets must be supported, allowing a system to use spare resources available within a company and obtain external Cloud resources to deal with peak demands. Utilization of these resources must be completely transparent to a user but must respect her decision whether or not data may leave a particular resource set during the computational process.

While providing the required functionality to users, resource site administrators must be able to easily handle a potentially large computing system. The demand reaches from automatic VM image distribution to patch management of a large number of partially inactive VMs.

Based on the three use cases presented and interviews with experts within the companies participating in the different research projects, a catalog of technical requirements has been derived. Meeting the stated requirements is important for attracting users as well as successful commercial adoption of publicly accessible Grid and Cloud computing systems.

In the last part of the chapter, it has been shown that a good deal of work has been done focusing on various aspects within Grid and Cloud computing. Several of the presented papers help to push technology forward in order to reach commercial acceptance of Grid and Cloud systems, but none of the presented approaches fulfills all of the requirements for industrial partners that are

carved out in the second part of this chapter. Therefore, a new system needs to be developed that takes all requirements into account and allows commercial users to use a distributed high performance computing framework to fulfill their needs.

3 Elastic Onion – A Novel Approach for Elastic Computing

"Today's so-called cloud isn't really a cloud at all. It's a bunch of corporate dirigibles painted to look like clouds. You can tell they're fake because they all have logos on them. Real clouds don't have logos."

Peter Lucas, Joseph Ballay, Ralph Lombreglia [137]

3.1 Introduction

As mentioned in the chapter before, a good deal of work has been done to solve existing problems within Cloud and Grid computing and provide a powerful infrastructure that is easy to use. But there is no single solution that addresses all of the customers' requirements stated in the requirements catalog in Section 2.3.2. Unfortunately, it is difficult to integrate existing solutions into a system to fulfill a customer's needs. Therefore, a new system has to be designed that takes all of the requirements into account. Moreover, the system should rely on well-established and matured technology and must be flexible enough to fit into existing infrastructures that are available to customers today e.g., it must be compatible to existing Grid middlewares or commercial virtualization solutions.

In addition to lacking a component that fulfills all of their customer's needs, the Cloud systems available today, such as Amazon EC2, suffer from data lock-in: If customers decide to use a particular system, it will be complicated to transfer both data and the created environments to a new resource provider later on if the user for some reason decides to change the provider. Moreover, due to these incompatibilities, it is not possible to scale automatically among different resource providers depending on the actual workload.

To overcome these limitations, a system needs to not only care about the data exchanged between the resources available in the system, but it also needs to ensure automatic scheduling and scaling among all available resources – across administrative borders – respecting the user's decision regarding which resources can be used for particular tasks.

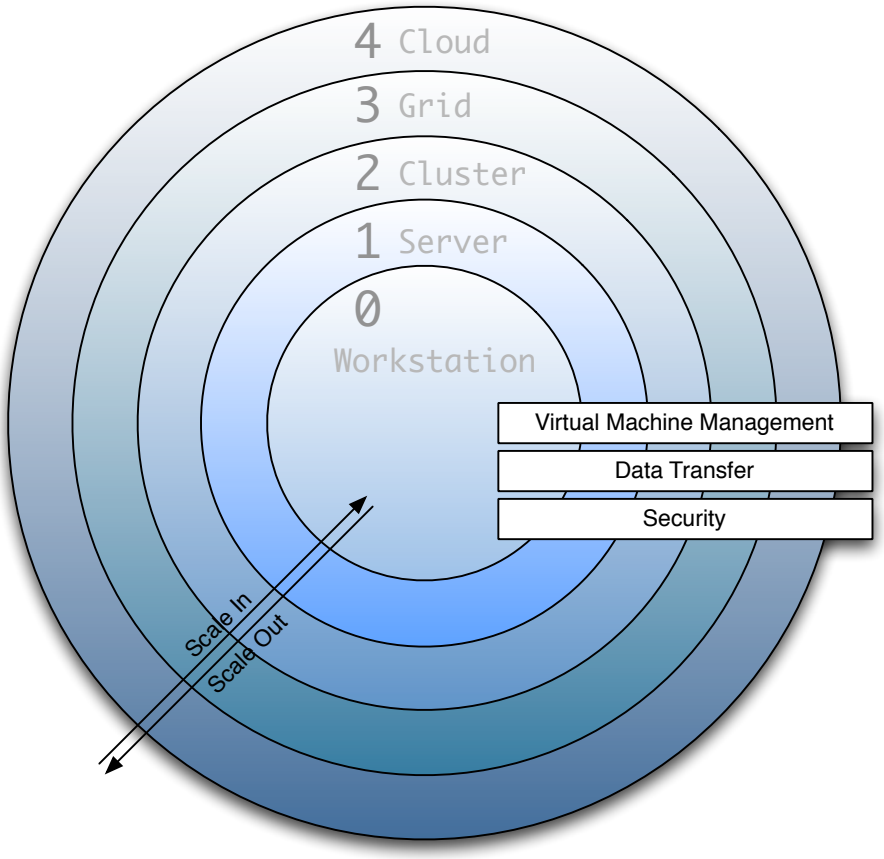


Figure 3.1: A system that acquires external resources, e.g., to deal with peak usage, and deallocates the acquired remote resources in a cost effective manner after the demand for resources has fallen off. The system's architecture can be envisioned as an onion: Resources are located on different layers around the user.

Figure 3.1 shows the different layers of such a system. Beginning at the core (Layer 0), the workstation of the user is the only resource available for computational tasks. When the number of computational tasks increases, the user can decide to use external resources (Layer 1). This can be a workgroup server or an idle computer in the office. If the load continues to increase, resources on Layer 2 e.g., a locally available cluster system could be acquired. In a company, this may be the first step into another administrative domain e.g., if the cluster system is operated by a different department. Nevertheless, clusters are often operated by the company itself. Acquisition of resources of the next

layer (Layer 3) may be hampered by the fact that handing over computational tasks to the Grid implies that computational jobs will be executed on resources under control of a third party and will leave the company's sphere. Though Grid resources are usually only available in a limited number, it is possible to acquire resources from a Cloud provider (Layer 4) if even more resources are needed. Resources might be organized into resource sets. Every resource set represents a group which consists of several single resources e.g., a Grid site or a computing cluster.

Like an onion, the system consists of different layers grouped around its core. From the inside to the outside, layers become bigger in respect to the number of resources that can be acquired from resource providers within a particular layer. The architecture has been dubbed *Elastic Onion* to illustrate the user's ability to remove and add resources of different layers with respect to the number of needed resources. Adding layers would result in an enlargement of the whole system, implying an increase in the system's performance, while removing layers would result in a smaller system close to the system's core i.e. the user's workstation.[41]

The transition from one layer to another one is fluid. Though the different layers are attributed to particular computing paradigms, there are some kinds of resources which cannot be clearly classified into a particular layer. *Desktop Grids* may be located between Layer 2 or 3 depending on which characteristics are considered. Focusing on the administrative domain, a Desktop Grid can be assigned to Layer 2 because resources on this layer are usually under the control of the company itself. Focusing on the heterogeneity of the computing resources, a desktop Grid would fit into Layer 3 because it often employs heterogeneous computer systems while a cluster system usually embraces a set of homogeneous computational resources. Considering Layer 3 and above, one must distinguish between in-house and external systems. Publicly accessible Clouds are called *Public Clouds*. Cloud systems often exist within a company and are only made accessible to the company's employees. So-called in-house Clouds or *Private Clouds* can be usually found in commercial environments. This also applies for Grid systems within one company, which are called in-house or *Private Grids*. A desktop Grid typically is an in-house Grid system but may be interconnected with external Grid resources using a Grid middleware. Connecting Private Clouds and Public Clouds results in a *Hybrid Cloud*.

The different layers of the Elastic Onion represent not only the different characteristics of the resources but also respect the (geographical) distance to the user beginning with the user's workstation itself on Layer 0 and ending in the Cloud on Layer 4. Moreover, the costs of using resources on a higher layer are usually higher than utilization costs of resources on a lower layer. On the other hand, the higher the layer, the more resources can usually be acquired on

[41] In fact, the Elastic Onion's user may also cry during removal of the onion's layers due to the additional time needed to fulfill her computational tasks. ;-)

this layer. While on Layer 0 only one physical workstation exists with a fixed number of CPU cores and main memory, virtually unlimited resources can be acquired from Cloud providers around the globe.

Figure 3.2: The two dimensions of scale. Scale up means to add components like e.g., new CPUs, memory or hard disks to an existing resource while scale out refers to adding new resources to adding system.

The ability to gain higher performance by adding more resources is called scalability [103]. As shown in Figure 3.2, scalability is divided into two broad categories: *vertical scaling* (scale up) and *horizontal scaling* (scale out) [142]. Vertical scaling or scaling up means that resources are added to an existing resource set in a system. Referring to the architecture described above, additional resources are added without changing the layer e.g., by adding additional processors, memory or storage to a particular physical computer in the system. In contrast, horizontal scaling or scaling out involves adding new resource sets to an existing system e.g., by adding new resources to an existing Grid or Cloud system or new compute nodes to a computing cluster, or by gaining access to additional resource sites like a new Grid site or Cloud resources. Therefore, horizontal scaling can impact the layer on which a system is operating.

Scaling vertically is usually much easier to manage. Since no new resource sets (such as a new Cloud or a Grid site) are added to the existing system, rather additional resources are allocated to the existing resource sets, the installed OS or CRM is responsible for efficiently utilizing the resources. Unfortunately, it is much more difficult to automatically add new CPUs or memory modules to existing computer systems (if that is even possible) than to automatically integrate spare computers to an existing set of resources. Scaling horizontally also involves increased scheduling complexity since new resource sets must be integrated into the existing system [156]. It also impacts the optimization of scheduling decisions on the different layers. On Layer 2, CRMs are put in place to manage resource usage of the clusters. CRMs manage a fixed set of resources[42] and can optimize the scheduling decisions because of a static resource pool and predicted job runtimes, which can be used to make optimization decisions. On Layer 3, meta schedulers are put into place that schedule computational jobs to different resource sites responsible for executing the jobs on their own. Optimizing the use of resources is much more difficult on this level since there are no runtime predictions that can be taken into account because jobs scheduled by the meta scheduler compete with locally submitted jobs.

[42] When plugging new resources to an existing cluster system, the CRM usually must be reconfigured to be able to deal with the new resources.

Horizontal scaling implies the absence of a static resource pool which results in aggravation or even makes it impossible to optimize scheduling. Since new resources can be acquired whenever needed,[43] sophisticated strategies like backfilling cannot be used in this scenario. On the other hand, the system may decide on its own when to acquire new resources from a higher layer. Since this usually results in an increase in operational costs, multiple parameters must be taken into account to decide whether or not new resources should be acquired. These parameters may include the usage fee of the particular external resources, a contractional penalty if a resource provider could not fulfill a SLA, costs of transferring data to the external resources, or legal restrictions.

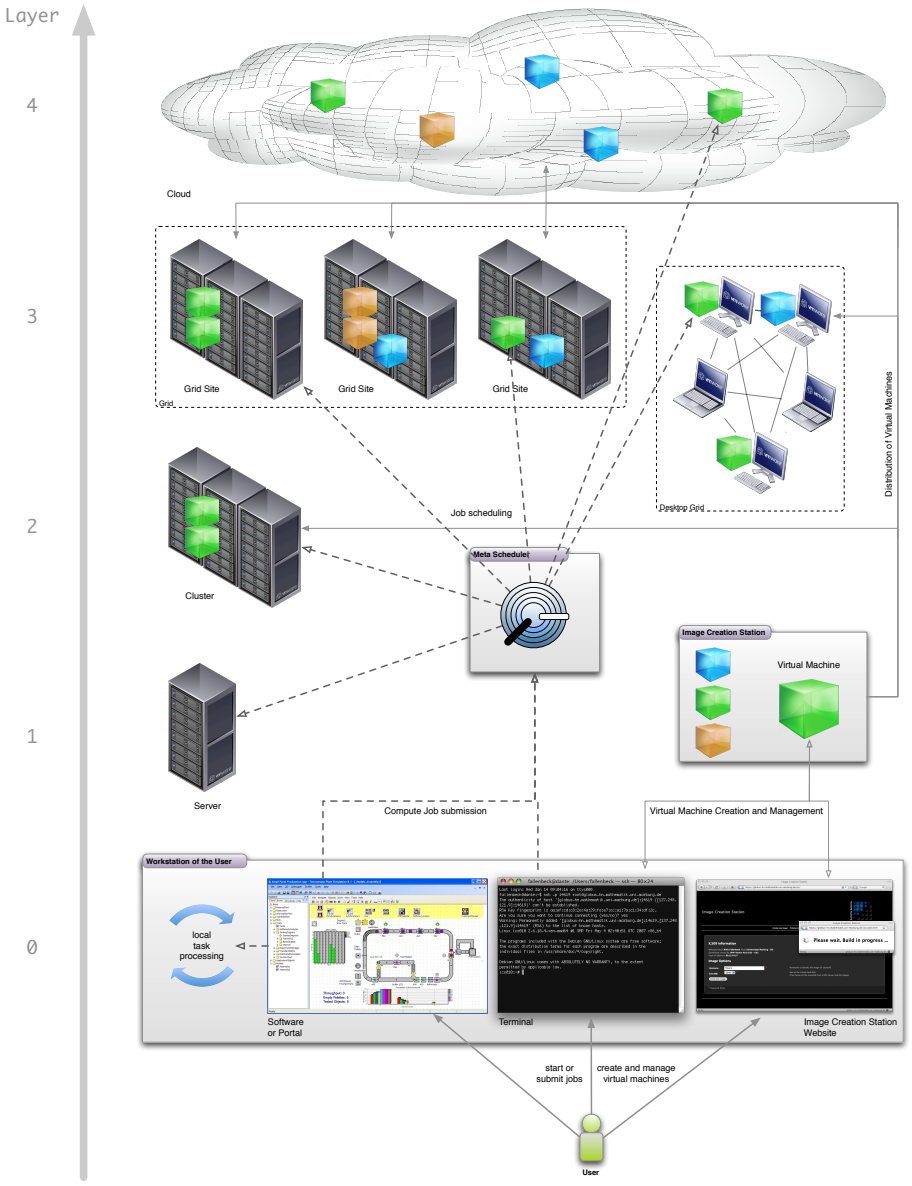


Figure 3.3: Beyond the different layers, many computing systems need to interact. From top to bottom, resource sets are shown which are interconnected by components facilitating automatic horizontal scaling. Virtualization technology is used to provide personalized environments and to shield users from other users concurrently using the same physical resources.

Figure 3.3 presents the possible interactions of different resource sets in an elastic system. The user mainly interacts with her local system. She may create personalized virtual execution environments – referred to as VMs – for job

[43] The Cloud promises to obtain a nearly infinite number of resources within minutes.

execution on remote systems. When using software that supports remote execution of tasks, the user must decide whether or not a task should be executed locally or handed over to a scheduler among other schedulers, a meta scheduler. The meta scheduler itself has knowledge of the systems available on the different layers and decides where to schedule the particular task. Depending on that decision, automatic distribution of the appropriate VM starts as well as the transfer of the data needed to execute the particular computational job.

The meta scheduler should try to schedule every computational job as close to the user as possible (on the lowest possible layer) because, in most cases, using the nearest resources is less expensive. Furthermore, the meta scheduler cannot only exceed the different layers but can also acquire new resources when needed. This is suitable in Cloud environments, which allow users to request resources on-demand. It is crucial that unused external resources are deallocated as soon as possible to reduce the operational cost (see requirement R19).

Besides deciding which cost-intensive external resources to acquire and allocate, many additional requirements must be met. Several mechanisms must be put in place to guarantee the essential usability of the system with respect to security questions, data transfer mechanisms and VM management. These techniques affect every layer because they are needed to provide a functional frame set for automatic up- and downscaling of the system.

VM Management. Management of VMs is important to provide personalized virtual execution environments for every user. By putting the users' tasks into VMs, personalized environments tailored to the users' needs can be provided. Furthermore, users will find a homogeneous environment independent from the layer of resources on which the jobs will be executed. Any software a user has installed in a VM prepared for the computational job execution will be available.

Data Transfer. Data transfer mechanisms are responsible not only for making the users' VMs available on the target resources but also for ensuring the availability of the job's input data that is needed to execute the computational job successfully. It is important that data is routed to and from the destination in a reliable manner such that a user has access to her data after (and even during) job execution independent from the layer of resources on which the job is executed.

Security. It is crucial to shield users from each other in a shared environment e.g., when the computational job leaves the user's local workstation and is executed on remote resources. Using virtualization technology, every user is put into a VM and cannot spy on other users' data outside

of this VM. Moreover, metadata[44] will also be secured using VMs because every user is only able to see the process list of the VM she is using. Every user can only watch her own processes but cannot see any other processes executed concurrently in other VMs on the same physical hardware. Besides virtualization technology, data encryption must be put in place to prevent the interception of sensitive data from third parties.

3.2 Design

Based on the remarks made above and the requirements defined in the last chapter, components are proposed in this section that allow users to build a system that matches these requirements.

Although there has been rapid development in Cloud computing in recent years, many open research issues still need to be resolved [83, 19, 2, 253]. The 2010 Cloud Computing Report of the European Commission (EC) [99] identifies several technological issues in different research areas:

> To the *technological* aspects belong in particular issues related to (1) scale and elastic scalability, which is not only currently restricted to horizontal scale out, but also inefficient as it tends to resource over usage due to limited scale down capabilities [...]. (2) Trust, security and privacy always pose issues in any internet provided service [...]. (3) Handling data in clouds is still complicated – in particular as data size and diversity grows, pure replication is no viable approach, leading to consistency and efficiency issues. [...] (4) Programming models are currently not aligned to highly scalable applications and thus do not exploit the capabilities of clouds [...]. Along the same line, developers, providers and users should be able to control and restrict distribution and scaling behavior. This relates to (5) systems development and management which is currently still executed mostly manually, thus contributing to substantial efficiency and bottleneck issues.

One fundamental problem is the absence of well-defined interfaces to facilitate data and job exchange between Clouds that belong to different providers. Although Amazon offers standardized Web service interfaces, each user is responsible for exporting her data and importing it somewhere else to use it. Furthermore, Cloud resource providers such as Amazon offer on-demand provisioning of resources but do not offer a scheduling mechanism of any kind.

[44] Metadata of a process in an OS consists information like the owner of the process and the used resources.

This is suitable for providing services on Cloud resources, but not for using Cloud resources as compute nodes under the control of a scheduler. However, a user can install a scheduler onto her Cloud resources, though every user is individually responsible for scheduling jobs to the rented resources, which is not a viable solution for companies that want to enable their employees to use Grid and Cloud resources in a simple and efficient manner.

In the PT-Grid project, a cooperation between a software provider offering plasma physics simulation software and commercial users is examined. Up until now, the company has only been able to offer (very limited) in-house computing resources to its customers and is looking for a dynamically scalable solution to provide as much computing power to its customers as needed. Due to the drawbacks of Amazon EC2 with regard to computational performance for certain types of applications [238, 153], the company decided to use nationally available Grid resources for its simulation tasks, though it is open to using Cloud resources in the future. There is a strong interest in integrating the Grid and the Cloud in an on-demand manner and using the same VMs in both environments. The company aims to provide readily configured and easy-to-use execution environments to its costumers enabling them to work within their personal environments on different Grid sites (in different administrative domains), thus, the proposed system must fulfill these requirements.

In traditional cluster environments, the local administrator is responsible for operating the cluster to avoid installing software that endangers the local infrastructure or data within the system. In most cases, the administrator will not install any software for which she cannot guarantee. Even if she is willing to install custom software for users, the source code must be audited to make sure that there are no malicious operations. Obviously, code audition is only possible if the source code is available to the local administrator. While this is possible if open source software or software developed by local users is used, it is usually impossible when commercial software comes into play.

One possible way to overcome this problem is to separate each user's software from the underlying infrastructure by executing it in VMs. In each VM, a user can install the software needed, and even if this software turns out to be malicious, only the user's data stored in this VM is endangered. Furthermore, one can not spy outside a VM, making it impossible to track other users' actions and processes running in other VMs or natively on the physical nodes. There is no need to open the software to anyone because only the given VM's user is able to access the installed software and data. This enables the use of commercial software in Grid and cluster environments without requiring support from the local administrator.

This also means a shift of administrative tasks from the local system administrator to the user since every user is responsible for her own virtual machines. In academic environments, users may be able to administer VMs on their own,

but when commercial users come into play, support is needed. Software vendors may not only make their software available to their users, but may also provide readily installed VMs. Providing comprehensive VM provisioning tools to users and software companies is crucial for the wide acceptance and usage of the Grid or the Cloud.

Since the Grid connects HPC resources on different locations, users from one site are allowed to execute their jobs on any other site within the Grid network. In conjunction with personalized VMs, the system must ensure that every user has access to her own VMs on every Grid site.

One basic idea of Grid computing is to make resources available to every user Grid-wide (e.g., for scalability reasons), so every user should also be able to manage her VMs on every site. Providing distributed maintenance tools also prevents bottlenecks by having alternatives available if particular instances of the management tool are used heavily or even jammed by other users. Whenever the management tool detects that no resources are available to boot a VM when users log in to manage their VMs, they are redirected transparently to more idle instances of the management tool. This enables users to work with their VMs even if local resources are congested by other users.

3.2.1 Image Creation Station (ICS)

To offer users the possibility of managing VMs on their own, the ICS has been developed [208, 67]. The ICS is a software component that offers its users the ability to set up VMs on-demand. In conjunction with the XGE [66, 208], these VMs can be used in traditional Cluster Resource Managers (CRMs) for job execution. Neither the ICS nor the XGE require a Grid computing environment to work, but they extend traditional cluster batch schedulers (e.g. Torque/Maui and Oracle Grid Engine) to handle VMs that are transparent to the users submitting jobs. By using Grid middlewares such as Globus Toolkit 4 [72, 90] that interface with local CRMs, these components (and virtualization technology in general) can also be used in a Grid without the need to modify the existing Grid architecture.

The ICS is a stand-alone software component and enables its users to store multiple VMs. This allows users to create different execution environments for different jobs or to have the same software available in different versions – installed in different VMs to avoid side effects. The ICS is typically installed by the resource site's administrator on a publicly accessible host. No software must be installed on the users' computers in order to use the ICS; the simplest way is to use ICS' web frontend, which can be accessed via a normal web browser.

Figure 3.4 provides a more detailed view of the architecture of the ICS. The ICS is divided into several parts. The ICSd is the daemon providing the business logic. Functionality to fulfill the main tasks (shown below in Figure 3.10)

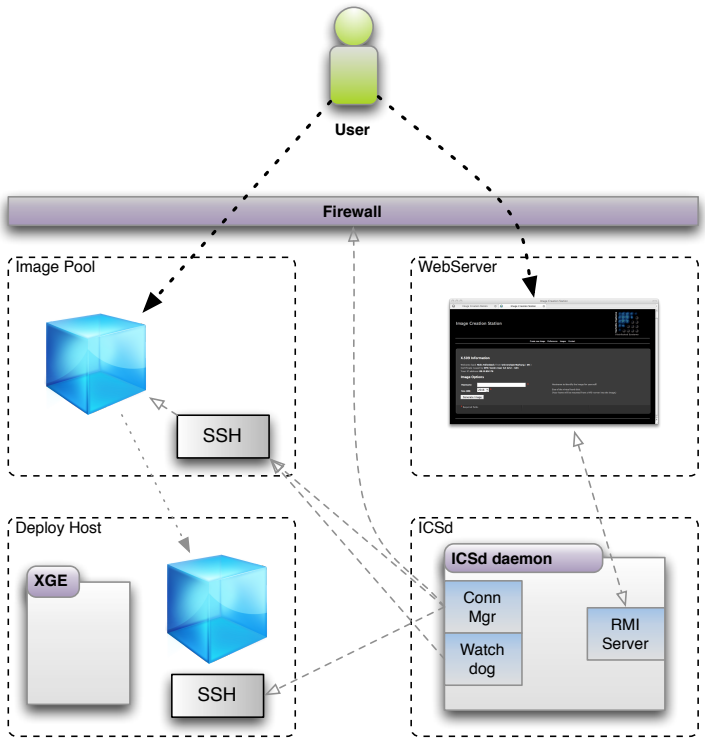


Figure 3.4: The ICS is divided into different parts that do not necessarily need to be installed on the same physical resource.

is located in this part of the ICS. It exposes its functionality via an "internal" Remote Method Invocation (RMI) interface that is used by the web page frontend but can also be used by a web or Grid service deployed to the local middleware headnode, allowing fully automated creation and management of VMs e.g., from BPEL workflows [8, 59, 55]. This interface also provides methods that allow automatic installation and removal of software packages. Another, "external" RMI interface exposes functionality to other ICS instances installed on remote sites i.e., providing methods to support inter-site VM image distribution.

VMs created or started by users for maintenance reasons will be booted on the image pool. This pool must reside on a computer that provides Xen virtualization technology [14] used to execute the VMs. It can be placed in a DMZ to ensure security of the site's infrastructure. This is the place where users can modify their VMs or install the software they need to run their jobs. Every user can log in to her VMs as privileged user, able to modify nearly everything within the VM (which may result in an unusable VM in the worst case). Paravirtualization technology is used for performance reasons; users cannot choose which kernel the VM should use. This ensures that the created VMs will be compatible with the underlying software stack used on the computing nodes and keeps the ICS simple to use. On the other hand, this may be a restriction in rare cases if software that should be installed into the VM requires a specific kernel version. When booting a VM on the image pool, dynamic port forwarding can be set up optionally on the outer firewall, allowing external users to access their VMs on the image pool by opening network ports and activating port forwarding mechanisms. The image pool is only used to execute VMs for

management reasons e.g., if a user wants to install new software or perform security updates. Therefore, not many resources need to be assigned to the active VMs.[45] Furthermore, VMs can share CPU cores when running on the image pool. Resource sharing is commonly used in scenarios in which several physical servers are consolidated on a single physical machine hosting several VMs but should be avoided in the area of High Performance Computing (HPC) because of the fact that VMs can influence other running VMs in a negative manner when relying on shared CPU cores.

Finally, after customizing their VMs, users need to deploy their VMs to the deploy host. The deploy host is a storage location shared with the XGE that is used to hand over VMs to the cluster network, which usually cannot be reached from the outside. After deploying a VM to the computing nodes in the private network, a user can run jobs on the nodes scheduled by the local CRM i.e., by submitting a job to the CRM's headnode or by submitting a Grid job to the site's Grid middleware or a meta scheduler.

3.2.1.1 Components

In this section, the different components of the ICS are presented in detail. Their functionality as well as design decisions will be discussed. Without focusing on technical details, the conceptual functionality will be introduced. First, the methods provided by a frontend will be exemplarily shown by presenting the web frontend that is delivered as a web application for the Tomcat server. After that, the ICSd will be presented, which receives the method calls from the frontend via the RMI interface. New VMs will be created by the ICSd on the image pool, which will be presented afterwards. The last component, the deploy host, will come into play when images are deployed to the XGE to be used for job execution.

3.2.1.1.1 Frontend The first component with which a user communicates is the frontend. As shown in Figure 3.4, one possibility is to provide a frontend as a web application for the Apache Tomcat container. The frontend does not need sophisticated built-in business logic because it only exposes functionality provided by the internal ICS RMI interface. That allows rapid development of other ICS frontends e.g., a stand-alone application running on the user's workstation.

As stated earlier, no additional software must be installed on the user's computer to use the web frontend, rather a standard web browser can be used to access the ICS web page. The only prerequisite that must be met is that the x.509 certificate must be imported to the browser used to access the ICS website. The Tomcat container is configured in such a way that the user's browser

[45] Nevertheless, users must be able to perform needed tasks like software compilation processes on the VMs started via the ICS.

must present the certificate which is then checked by Tomcat. Access is only granted if the certificate is valid. It is crucial to use a highly secure authentication mechanism at this point because a security flaw would permit malicious users to access sensitive data from others e.g., market competitors. Basing authentication on passwords chosen by users is a bad idea for public systems that hold confidential data [148], therefore the system has been developed based on a certificate infrastructure which is already used in the D-Grid or by Amazon. Thus, the ICS can be integrated seamlessly into a Grid or Cloud environment.

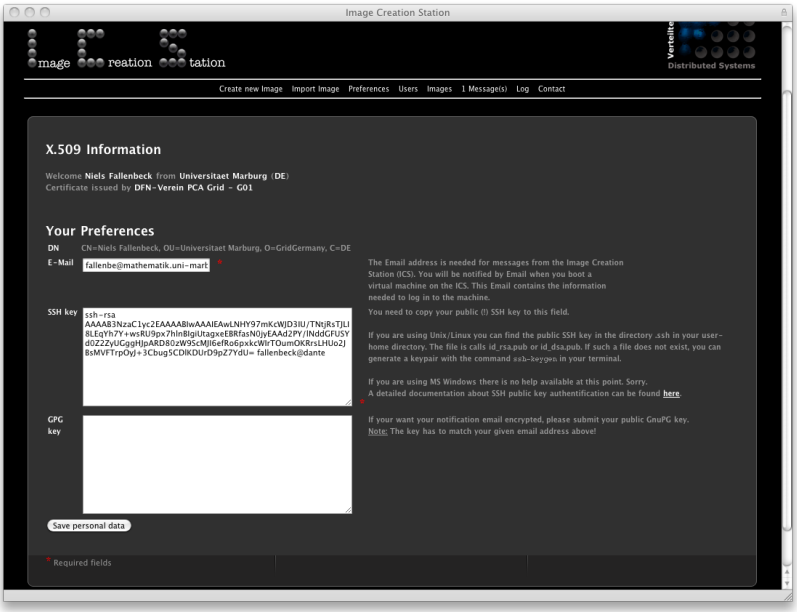


Figure 3.5: The ICS needs some basic preferences of each user before it can be used to create and manage VMs.

When the user logs in the first time, she must provide basic settings required for use of the ICS. Figure 3.5 shows the preference dialogue. Some information is extracted from the certificate used during authentication i.e., the user's Distinguished Name (DN). In addition, the user must provide an email address, which is used to send notifications when a VM has booted or the site's administrator announced updates to the ICS users. In addition, a public SSH key must be provided, which is used to authenticate a user when she tries to log in to her running VM. If enabled by the ICS administrator, a user may also provide her GNU Privacy Guard (GPG) public key, which will be used by the ICS to encrypt all emails sent to the user. Sending emails can be disabled by the administrator, but an internal messaging system will notify the user of recent changes in the web interface.

After submitting the necessary information, the user can start to create new VMs. There are three basic ways to create VMs on the ICS. A user can start with a very basic VM image that has only the OS installed. If the user needs any additional software, she must install it herself. Depending on the OS version installed into the VM image, the user can install software using mechanisms of that particular OS e.g., package managers provided by most Linux distributions, or by installing binary packages or compiling the software on her own. Alternatively, users can derive new images from existing ones

that have been provided by other ICS users, so called “image providers”. The local ICS administrator can authorize particular users to provide images. In most cases, these image providers are experienced users or software vendors that prepare images for normal users, who do not know much about OS and server administration. Since VM images that have been derived from these readily configured images can be used instantly for job execution, users can circumvent the time-consuming process of software installation, which is an important advantage for commercial users.

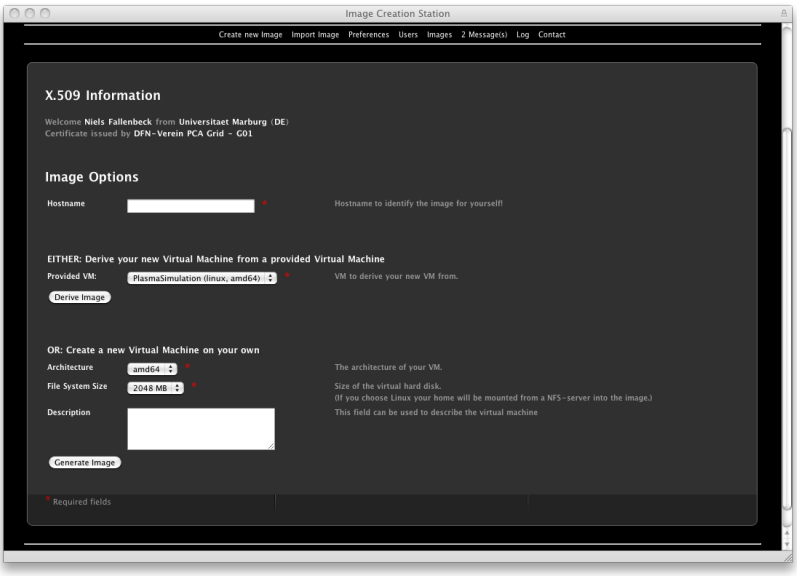


Figure 3.6: ICS users can create new VMs either from a very basic OS installation on their own or derive a new VM image from an existing one.

In Figure 3.6, the web page is shown that provides this function. Regardless of which method is used, the user must specify the hostname of the new VM image. Then, she can choose one of the provided images from which her new VM should be derived, if there are any provided images. If no provided images exist or the user decides to start with a basic VM image, she can choose the architecture of the OS. On x86-based systems, users can choose between a 32-bit OS (i386) or a modern 64-bit OS (amd64). The supported architectures can be defined by the local ICS administrator. Furthermore, the size of the image must be specified. The user can also give a more detailed and longer description of the VM, which is helpful, especially if the image should be provided to other users.

The third possibility for creating a new VM image on the ICS is to import an existing image from the user’s local virtualization tool. Therefore, the ICS supports an importation method for Virtual Machine disk (vmdk) files, a format which was initially developed by VMware but is now an open source format [234].

Figure 3.7 shows the screen which allows users to import existing vmdk images to the ICS. A user only has to specify the hostname of the new VM and the existing vmdk image which will be uploaded to the ICS. Depending on the size of the disk image and the network connection between the user’s

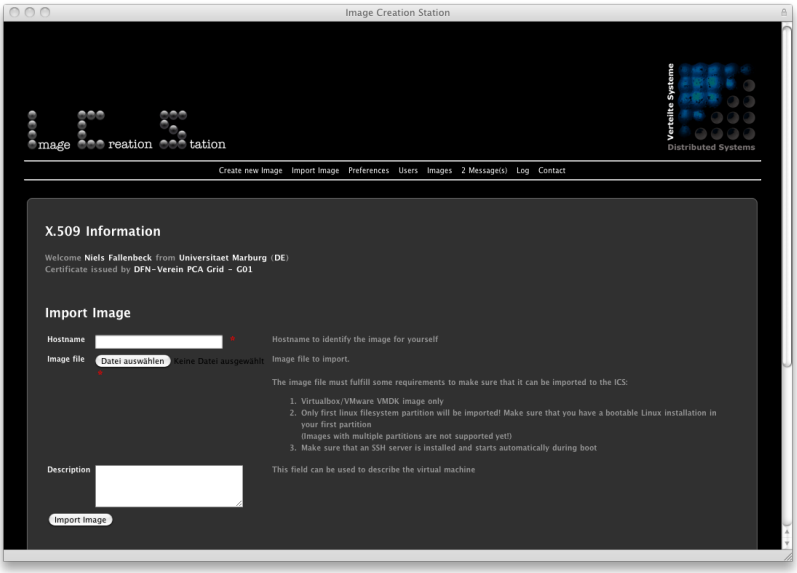


Figure 3.7: Existing vmdk images can be imported from the user's workstation to the ICS using the page shown of the web application.

workstation and the ICS host, this can be a time-consuming process. On the other hand, the user does not usually need to modify the imported image if all software has already been installed. Besides these two mandatory settings, a longer description can be given to describe the VM image in detail.

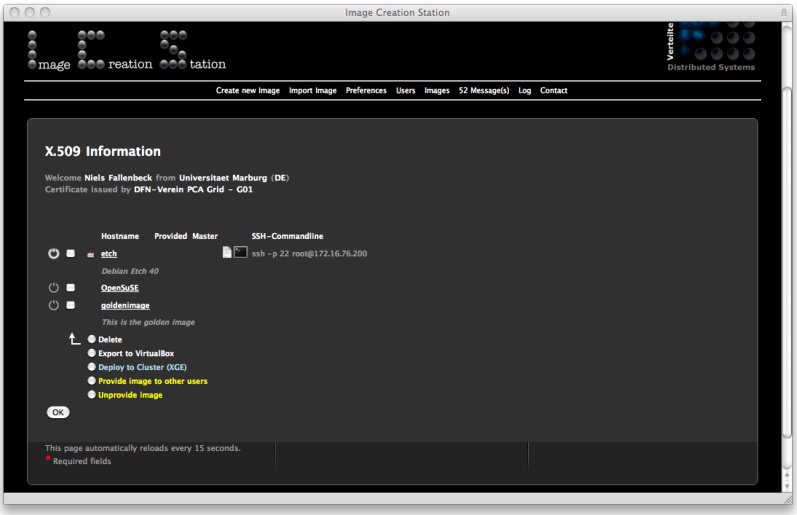


Figure 3.8: The ICS web application provides an overview of existing VM images and their state. If a VM is running, the user can directly log in to this VM.

After creating one or more VM images, a user can start or stop VMs based on the different images. Figure 3.8 shows the web application's page that provides an overview of the user's existing VM images. Furthermore, the state of each VM image is shown, if it is either active or inactive. To modify a disk image, it must first be activated. Then, a VM is started with this image, allowing the user to log in with her SSH key. A user can also use the web application to log in to her VM. Thus, the application provides a SSH applet which can be started for each running VM. The applet is shown in Figure 3.9.

In addition, the web application provides detailed information about each VM image such as the UUID or the revision of each image. The revision expresses the number of changes made to the image since its creation. Besides

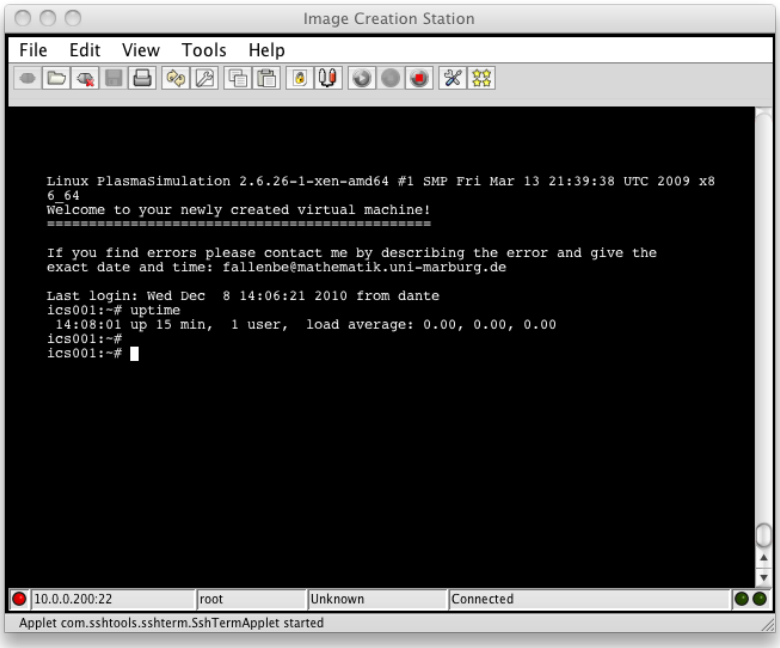


Figure 3.9: The ICS web application has a built-in SSH applet which can be used to connect to a user's running VMs.

this information, the architecture of the OS installed is presented as well as its file system size and type and the OS type and version. The user can find information from the ICS-internal accounting system like the date of the image creation and time required. This information is crucial in commercial environments in which users are billed for resource consumption.

All of the methods described above and the information provided by the web application is available via the internal RMI interface. The web application does not contain any sophisticated business logic but hands over user input directly to the ICSd and displays the results received in a suitable (readable) manner.

As shown, the frontend component (and the appropriate RMI interface) has been designed to meet requirements R1 – R5 of the requirements catalog, which address an easy and flexible way of creating and managing VM images.

3.2.1.1.2 ICSd As mentioned above, the ICSd is the core component of the ICS providing the business logic. Figure 3.10 depicts the the three main tasks of the ICS.

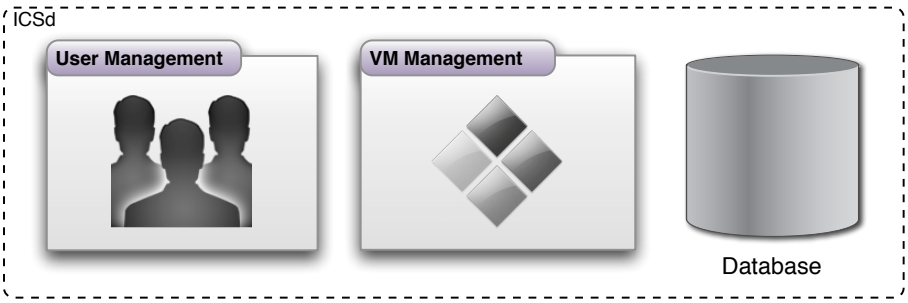


Figure 3.10: The ICS has three main tasks: manage the users, manage the VMs (create, modify and share), and store VMs and users' settings in a reliable and persistent manner.

First, the ICS has to deal with Grid users that are identified by their x.509 certificates. Every user can register herself on every ICS. In a distributed environment, many ICS instances can be active at the same time, and each instance must track each user's actions and manage VMs belonging to that particular user. VM management is the second main task and the most prominent part, as "image" is part of the ICS' name. Managing VMs encompasses numerous activities, such as creating images, enabling any desired software installation to the different VM images, and modifying OS installation and configuration. Furthermore, because users are not bound to one ICS but can use any ICS available in a system, VM images must be distributed and synchronized between all ICS instances. Moreover, image distribution is the key enabler for using user-specific virtualized execution environments in a system. If users create a VM image to use it for job execution, the system must ensure that the given images are available on every resource site potentially being used for job execution. The third task, which is not as visible as the first two parts, is the management of information and data. It relies on a local database to ensure reliable and persistent storage of the information. The database is also used to store the accounting data that is needed for billing the users in commercial environments. Although the databases of different ICS instances are independent and cannot be seen as a single distributed database, every database provides functionality to synchronize with the remote databases to provide a homogeneous system for VM management and configuration.

The ICSd can be executed on any platform supported by Oracle's Java implementation. In contrast to the image pool, the system hosting the ICSd must not meet any special requirements. The database used by the ICSd operates on a single file and is provided with the ICSd package, it does not require an additional Database Management System (DBMS) for operation. Earlier versions of the ICS were based on MySQL [164]. MySQL is more efficient than the SQLite [218] database which is used by newer versions of the ICSd, but experience showed that SQLite is sufficient for normal operation.

Since the ICSd is publicly accessible through the RMI interfaces, it has been realized in Java, which is believed to be a secure programming language [93, 80] in sense of security leaks that could be used to gain access to the underlying hardware. The basic concept of Java is to run programs not natively on the hardware but inside a virtual sandbox called JVM. Moreover, buffer overflows as possible in C will not occur in Java due to built-in security mechanisms like boundary checks. Nevertheless, Java has been proved to provide good performance and is also used in HPC environments [145].

The ICSd can be subdivided into several packages, each of which embraces specific functions. Figure 3.11 depicts most of the packages which can be found in the daemon.

Accounting. This package provides the functions necessary to track the users'

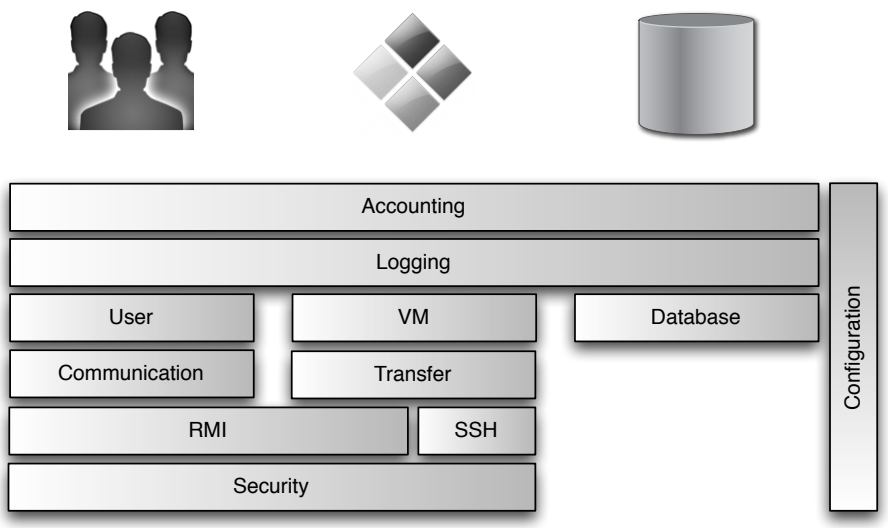


Figure 3.11: The ICSd is subdivided into several packages which can be associated to at least one of the main tasks described in Figure 3.10.

actions on the ICS. Accounting mainly has to deal with two events: the execution of a VM and the size of its image. Executions of VMs can be accounted easily because the ICS logs the runtime of a particular VM. Moreover, the resources used by that particular VM are specified in a configuration file; thus, resource consumption can be easily derived from this information. Accounting events regarding the VM include import or creation and deletion of the VM image as well as the process of exporting the image to the user. The duration of VM image creation is determined by the ICS by simply calculating the time from the beginning to the end of the creation process. To ensure consistency, a VM object in the database must not be removed when the user deletes the appropriate VM image from the ICS because every accounting entry refers to such an object in the database. Deleting the VM object would render all accounting entries for that particular VM useless.

Logging. The logging package is responsible for keeping track of the ICSd's actions. As usual, logging is the key resource providing information in the case of an error. Usually the logfile is detailed enough to figure out why a problem has occurred (lack of disk space, missing binary, etc.). The logging facility not only writes output to a logfile (if desired) but also provides a ring buffer which can be accessed by the RMI interface. That provides ICS administrators with detailed information about the last actions of the ICS via the used frontend.

User. The user package not only defines objects that represent the ICS users but also provides management functionality for these objects. It is used to create new user objects and to map certificates to ICS users.

VM. This package provides similar functionality with regards to VM management as the user package provides for managing users. Nevertheless, VM objects are more complex than user objects e.g., VM objects con-

tain at least one file that points to the disk image used by the given VM. Each VM object can be identified by a unique ID, which ensures that VMs with the same hostname located on different ICS instances can be distinguished from each other. Furthermore, there are two additional sub-packages. The software sub-package deals with installation of packages provided by the OS' package management system; the backend sub-package is used to map abstract methods to specific operations on the file system. There are several supported backends that can be used by the ICS i.e., a backend supporting Xen and another backend supporting VirtualBox. Obviously, the VM image creation process within the Xen backend differs significantly from the image creation process of VirtualBox.

Database. This package encapsulates database access in methods that are used by other packages. It is responsible for storing VM and user objects to the database and loading them during the ICS startup procedure. The database itself is a SQLite database file, which is stored on the local file system of the host running the ICSd process. It does not require an additional DBMS to be installed and can be used without any configuration.

Communication. This package provides functionality to communicate with the users. The ICS' internal messaging system is provided and a mail sub-package provides functionality to send (and encrypt) emails. Methods in this package are usually called to notify users e.g., when a new image was successfully created or a VM has become available after boot.

Transfer. All multi-site image transfer support is located in this package. Decisions whether or not to transfer a particular image are made here after an update notification has been received via the RMI interface.

RMI. This package mainly provides two different RMI interfaces and their implementation. One interface is referred to as the external interface, the second interface is referred to as the internal interface. The external interface is used to communicate with other ICS instances running on remote sites. It provides functionality to acquire information about VM images and to initiate image transfers. The internal RMI interface is used by the frontends and the service interfaces. It provides the necessary functionality to use the ICS locally. Both interfaces have been separated for security reasons; while the external RMI interface must be publicly accessible, access to the internal interface can be restricted to a particular IP address within the network.

SSH. Every access to the file system (i.e., in methods from the backend package) uses the functionality of this package. The SSH manager located in this package is responsible for managing connections to the remote

machines i.e., the image pool. Each of these connections provides functions to execute commands on the remote location while keeping track of these actions and their duration (which is needed to provide accounting information). Furthermore, access to the standard output and standard error stream of the executed process is provided as well as its exit code.

Security. Everything dealing with security can be found within this package. It provides methods to parse x.509 certificates and to extract data from them. The firewall sub-package is responsible for performing firewall modifications during runtime e.g., to set up port forwarding in the firewall allowing SSH access to the running VMs from an external location. Comparable to the concept of different backends found in the VM package, different firewalls are supported. At the moment, a backend to modify iptables firewalls [184] exists as well as a backend which is used in the dummy operation mode of the ICS or if firewall modifications are turned off by the administrator.

Configuration. This package affects all of the packages mentioned earlier in the sense that the ICS administrator can configure the ICS. As mentioned in the security package description, the administrator can define whether or not to modify the firewall when a VM is booted. Providing the general configuration of the ICS, each backend needs its own specific configuration e.g., specifying which OS kernel should be used to boot a VM. Therefore, this package could not be assigned to a specific topic.

Having introduced the vast majority of packages that can be found in the ICSd, few packages have yet to be mentioned. These packages provide functions to ease the programming process and to simplify maintainability of the software.

3.2.1.1.3 Image Pool The image pool is the location where new VM images are created and VMs are started. In contrast to the host that executes the ICSd, this host must be run with Linux that provide the Xen hypervisor. In small setups, the ICSd can be run on the image pool (which has been tested extensively during the development of the ICS). Nevertheless, the image pool has been designed to be independent from the ICSd host.

There are few requirements the host providing the image pool must fulfill. Most importantly, the host must have a Xen hypervisor installed to provide the virtualization environment used by the ICS. Second, Xen's tools package must be installed, which provides scripts to set up new VMs easily. The ICS' image creation process relies on these scripts which are written in Python. Finally, the image pool must be accessible via SSH by the ICS. As described earlier,

every command is executed via SSH to guarantee the platform independence of the ICSd component.

3.2.1.1.4 Deploy Host In short, the deploy host describes a storage area which is accessible by the XGE. The deploy host is used to hand over VM images created by the ICS to the XGE. The most simple setup is a shared storage accessible to both the ICS and the XGE. The deploy host must provide write access to the ICS and read access to the XGE. Moreover, it may reside on the same host as the image pool but usually does not.

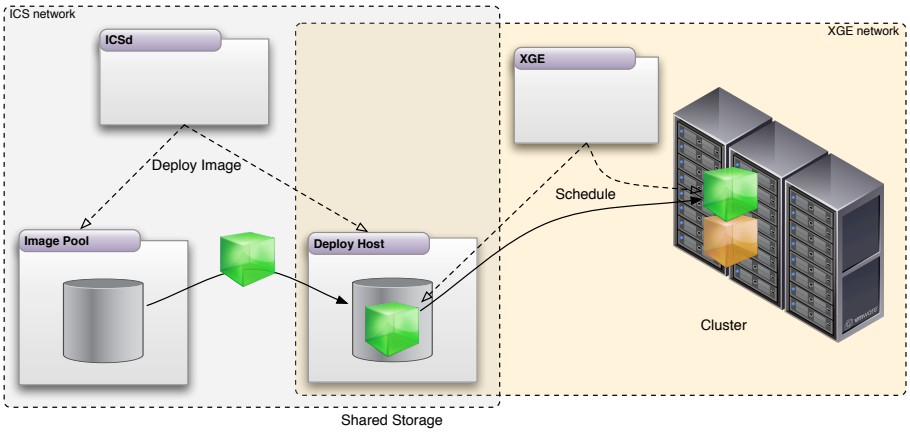


Figure 3.12: Simple architecture to ensure VM image transfer between the ICS and the XGE.

Figure 3.12 shows the simplest architecture possible for transferring a VM image created with the ICS to the XGE. When the ICS starts the deployment process, the image is copied to the deploy host, a shared storage which is also accessible by the XGE. As soon as the copy process has finished, the VM image can be used immediately by the XGE and on the XGE-managed resources for job execution.

Having the advantage of being very simple and quick and easy to set up, this architecture suffers from two major problems. First, because the deploy host is in the ICS network as well as in the XGE network, both networks could not be divided in a secure manner. Instead, no external hosts should gain access to any resources in the XGE network since it contains the cluster computing resources and the cluster users' data as well as infrastructure services needed for operating the cluster. If externally accessible nodes are successfully attacked by a malicious user, they can be used to grievously disturb the normal operation or even to gain access to data belonging to other users stored somewhere within this network.

Being suitable for academic environments which do not need to have strong security mechanisms in place, the system must also be suitable for commercial environments that force a strong separation between the resources located inside of a company and the publicly accessible hosts that are usually located in a Demilitarized Zone (DMZ).

A more sophisticated and secure solution is presented in Figure 3.13. Instead

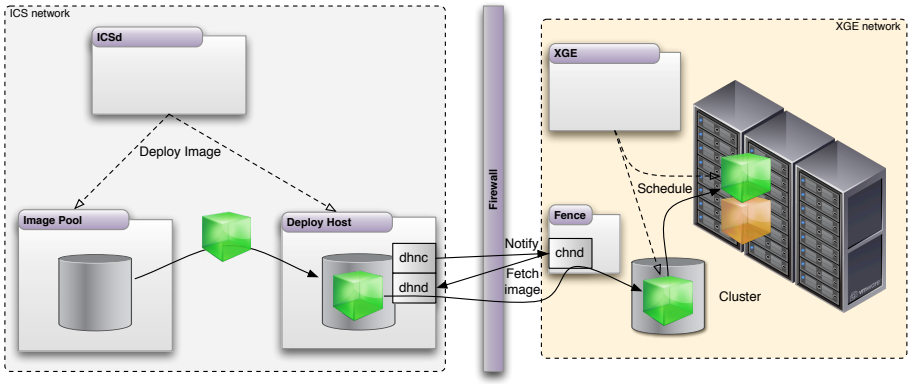


Figure 3.13: Secure architecture to ensure VM image transfer between the ICS and the XGE through a firewall using Fence.

of a shared storage area which is used to hand over VM images from ICS to XGE, a component has been developed called Fence [198]. In this architecture, the nodes that host the ICS installation are located in a DMZ.

To prevent external intruders from accessing the cluster hardware by exploiting weaknesses in the ICS or middleware components, the network is divided into two separate subnets, the border network and the cluster network. The DMZ guards both networks with a firewall configured to the specific needs of the network in question. The border firewall filters connections from the Internet and denies unwanted connections to all machines within the DMZ. However, since Grid middlewares, for example, require a large number of open ports to function correctly and efficiently [236], and a large number of fluctuating users need to access the Grid, the border firewall is relatively open. Therefore, the Grid headnode is located in the DMZ. Furthermore, since the ICSd host and the Tomcat container running the ICS web frontend are accessible to the public, these nodes should also be placed in the DMZ. The inner firewall guards the cluster network and prevents direct connections to the cluster subnet. To protect the cluster network, the inner firewall is very strict and only allows a single, specially designed cluster connector to pass through, it does not allow any interactive sessions to pass into the cluster network. The cluster resides in the cluster network and consists of a cluster headnode and a set of worker nodes. To transfer data from the deploy host located in the DMZ to the XGE headnode located in the secured network, three additional components are used:

dhnc. The *DMZ Head Node Client* provides the communication between the deploy host and Fence on the cluster headnode.

chnd. The *Cluster Head Node Daemon* represents the interface between the services on the deploy host and Fence on the cluster headnode.

dhnd. The *DMZ Head Node Daemon* is the second service on the deploy host. It is a background task, serving requests from chnd.

The dhnc monitors the location on the deploy host. When a new VM image

is deployed to this location, the dhnc notifies the chnd. For security reasons, chnd is listening at a single port that must be opened in the firewall. Once a notification has been received, the chnd itself establishes a connection to the dhnd. Thereafter, the data is transferred through the established data channel between chnd and dhnd. This channel can also be secured by putting network encryption mechanisms into place. To maximize security for the resource located in the "inner" network, the connection used to transfer the data is established from the inner network to the DMZ. Therefore, the firewall can be configured in a restrictive manner and needs only to provide one open port to be used to notify the chnd.

3.2.1.1.5 Firewall The firewall is the component that divides the ICS network from the outer world. For security reasons it is recommended to keep only a minimum number of ports open to the public. On the other hand, at least one open port is needed to connect to running VMs from the outside. The ICS itself does not need many open ports. When providing a web frontend to the users, at least one additional port must be opened serving the web sites. A second port is needed if the ICS should be used in a multi-site environment. The different ICS instances use this port to connect to and share data with each other. These ports are known in advance because of the fact that web sites are always served via HTTP on Port 80 or HTTPS on Port 443. The standard port used for RMI registries is 1099. Once it is determined how to setup the ICS, these ports can be opened permanently in the firewall and its setting does not need to be changed over time.

Open ports are also needed to connect to VMs running on the image pool from the outer network. One possibility is to open a number of ports in a given range which are assigned statically to particular IP addresses. Whenever a VM is started with a certain Internet Protocol (IP) address, it can be accessed by connecting to a certain port. A second possibility is to assign IP addresses to the VMs that are located in a DMZ where no firewall exists. This solution is simple since it does not require any configuration. However, it is also very insecure because every VM is completely exposed to the public.

For security reasons, a third possibility was chosen. Initially, no additional ports are open on the firewall. If a VM starts, a port will be chosen randomly from a given range (which can easily exceed 60000 possibilities[46]) and assigned to the VM. To make it accessible to the public, a port forwarding rule will be applied to the firewall which forwards the random port from the outside to the VM's standard SSH port. When a VM shuts down, the appropriate port forwarding rule will be deleted from the firewall.

This approach has the advantage that the port used for access is unknown

[46] Hiding a port within an enormous number of possibilities is a widely used security paradigm and referred to as "security through obscurity" [70, 169].

until it is chosen and assigned during a VM's startup procedure. Moreover, restarting a formerly started VM will most likely result in a change of the port used. SSH is a standard service widely used in the internet that provides secure access to computers. Its standard Port 22 is subject to port scans which are performed to check the existence of a computer system and test it in order to find vulnerabilities and to gain access to the particular system. Moving the SSH connections away from the standard port will minimize the risk of being successfully attacked.

Port forwarding is only necessary if the VMs started on the ICS image host and the users that want to connect to them are located in different subnets and the hosts in these subnets cannot communicate directly with each other. The ICS administrator can specify if the ICS should modify the firewall or not.

3.2.1.2 Creating VM Images

As the name ICS indicates, one of its main tasks is to create VM images. Based on the system configuration and the administrator's decisions, a user has different possibilities for creating a new VM image. Three of the methods have been briefly introduced above, however, there is, in fact, another method which has to be mentioned here.

3.2.1.2.1 From Scratch Using this method would result in a very basic VM. Installation packages are downloaded directly from the servers of the provider of a Linux distribution as shown in Figure 3.14.

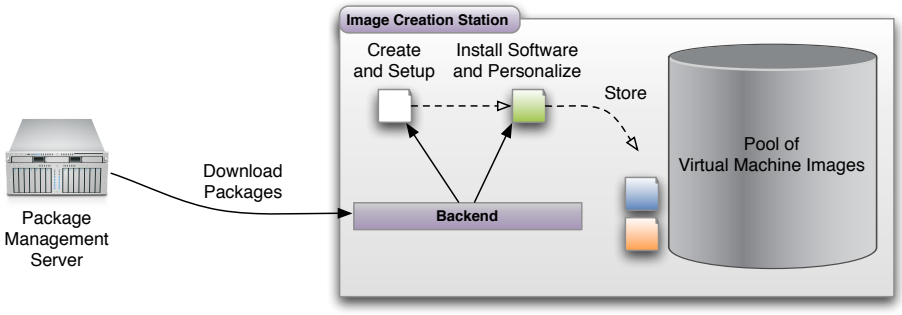


Figure 3.14: When creating a new VM image from scratch, the basic packages are downloaded from the package management servers of the particular Linux distribution.

The ICS administrator has the possibility of specifying additional software packages as well as files that will be copied to the newly created VM in a *role script*. This script resides on the image pool and is executed once the basic software installation of the packages received from the Linux distributor has been completed. It can be used to provide a basic configuration for the site's infrastructure, such as shared network storage or network configurations, by copying the contents of a skeleton directory to the newly created VM image. Moreover, the administrator can specify additional software packages to be installed afterwards e.g., a specific editor that should be provided to all users.

Because of the fact that the software will be directly fetched from the provider's

servers, actual versions of the software packages will be installed by using this method. On the other hand, although packages that have been downloaded are cached on the local machine, this method is the slowest possible method for creating a new image.

3.2.1.2.2 From Golden Image To speed up the image creation process, the administrator can provide a golden image. This method has not been mentioned earlier due to the fact that it will be automatically preferred over the preceding method if an appropriate golden image exists. The method is shown in Figure 3.15.

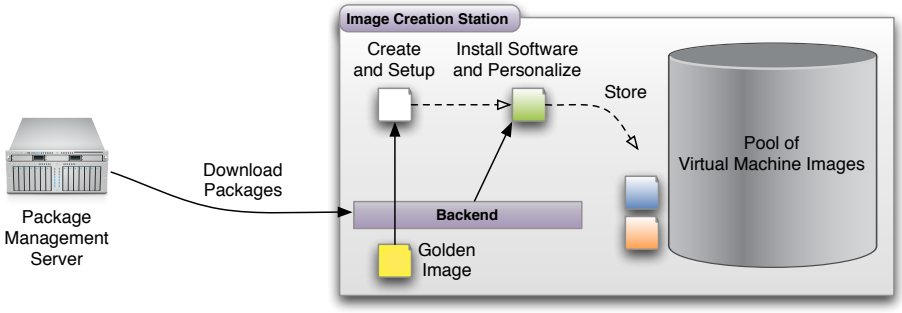


Figure 3.15: When a golden image is used, the contents of the golden image file are used as data source to create new images. It might also contain additional and readily configured software.

This method is comparable to the preceding method but will use a locally available tar archive which holds the contents of the new VM image. This allows administrators to put the software that is to be additionally installed directly into this archive and gives them the opportunity to make more sophisticated configurations. The contents of this archive will simply be copied to the new disk image created for the user. Afterwards, the *role script* will be executed, ensuring that packages and files specified within will be available on the user's VM.

If the ICS enables users to create VM images with different architectures (e.g., 32 bit or 64 bit) and different OS types (e.g., OpenSuSE or Debian Linux), a golden image must be provided for every architecture and distribution containing a software stack that matches the chosen architecture and OS type. It is possible to provide a golden image only for a subset of possible choices, a golden image is automatically used if present. If no golden image exists for a particular combination of architecture and OS type, the backend will automatically fall back to the method "from scratch" and download the software packages needed from the distribution's package management server.

Using a golden image speeds up the image creation process and allows administrators to provide additional software to all users. Nevertheless, since the golden image is the base image for all users, it should be kept small.

3.2.1.2.3 Derive from Existing VM Image Using the former image creation methods, the resulting image is a basic VM image containing only the

software needed to boot. If users need a specific software, they have to install it on their own. Figure 3.16 shows the possibility of deriving new VM images from existing ones provided by other users.

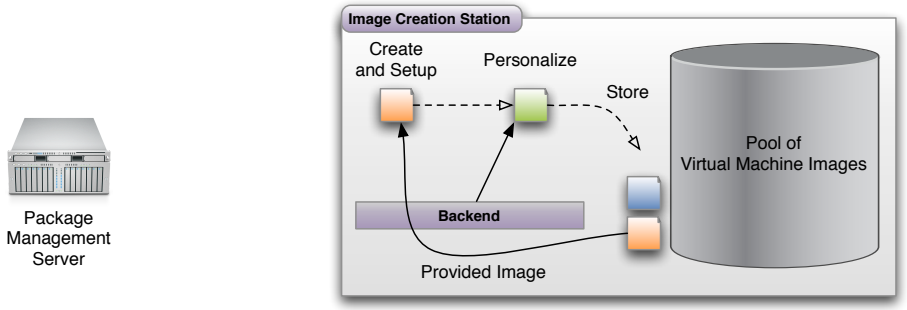


Figure 3.16: By deriving a new VM image from an existing image provided by another user, new VM images can contain a complex software stack. This allows even inexperienced users to easily use complex software without needing to set it up themselves.

Users providing images to other users must be authorized by the ICS administrator. In contrast to the two methods described above, this method is also suitable for users who do not have any knowledge in setting up servers and applications on their own. Typically, image providers are users with experience in system administration or in depth knowledge of the software they provide to other users e.g., software vendors.

To raise the level of security, the ICS administrator can prevent users from creating new images from scratch or from a golden image and force them to derive images from existing ones. Derived images rely on the images provided by image providers. By using multi-layer technology, which will be described in Section 3.2.1.3, images will be updated if the image provider updates the image she provided. This ensures that security problems will be patched in a fast and efficient manner. Providing this method, the administrative burden is shifted from possibly unexperienced users to experienced users and minimizes time between the creation of a VM image and its utilization by users who probably only wish to use a particular software product.

The approach for offering readily configured VM images to users is comparable with the approach of virtual appliances presented earlier in Section 2.4.2.

3.2.1.2.4 Import from Local Virtualization Solution If a user has downloaded a virtual appliance to her computer or has created and set up a VM using a Desktop virtualization software such as VMware Workstation or VirtualBox, she might want to use this particular VM in the Grid or the Cloud. Therefore, the ICS provides an import mechanism as presented in Figure 3.17.

Some customers already have readily configured VMs in their local environment, possibly used for executing their jobs on local systems or to test software installations. To avoid forcing users to set up existing VMs a second time, the ICS provides an import mechanism. The other way round, customers ask to export VMs from the ICS to their local virtualization environment. Although exporting and (especially) importing images into the widely accepted virtual

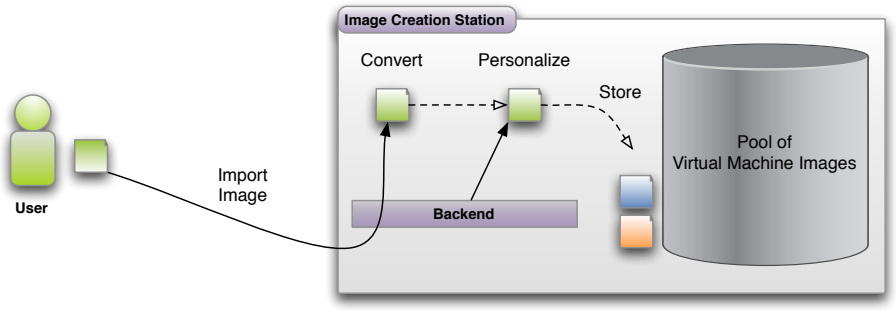


Figure 3.17: With the ability to import existing VM images to the ICS, a user can run VM images that she has already created or downloaded in the Grid or Cloud.

machine disk format vmdk [234] is a very complex process, it saves the customers time and eases the use of the ICS and the distributed system itself. The ICS supports the vmdk disk format for importing and exporting VM images, which is also used in the Open Virtualization Format (OVF) format [50]:

> The Open Virtualization Format (OVF) Specification describes an open, secure, portable, efficient and extensible format for the packaging and distribution of software to be run in virtual machines.

The first proposal for OVF was submitted to the DMTF in 2007, its latest specification 1.1.0 was published in 2010. The format has been designed to be independent from a given VMM or a software vendor and is supported by many virtualization products such as VMware, VirtualBox, Red Hat Enterprise Virtualization and even IBM z series. The OVF format consists of a directory that contains the disk image(s) in vmdk format and additional metadata information. An Open Virtualization Format Archive (OVA) file is a tar archive which contains the OVF folder.

By supporting disk images in vmdk format, the ICS is compatible with the majority of commercial virtualization products and supports the exchange of images between these solutions and the Grid or Cloud system.

Regardless of the method used to create a new VM image for a user, some actions are performed to personalize this image. As mentioned earlier, ICS users must provide a public SSH key. This key is copied into the new image and will be used to authenticate the particular user. Furthermore, by using SSH public key authentication, security does not rely on passwords chosen by the user, that could be rather weak [148]. A more complex solution suitable for environments that provide a certificate infrastructure involves a Grid Security Infrastructure (GSI)-OpenSSH[47] which is a modified version of OpenSSH that adds support for GSI authentication. This allows users to use the x.509 certificates that are used for authentication to the ICS also for logging into the running VMs.

[47] `http://grid.ncsa.illinois.edu/ssh/`

3.2.1.3 Multi-Layered VM Images

Environments that contain a large number of high performance computing resources are valuable targets for attacks. Once an attacker gains access to the compute nodes, she can use them for Distributed Denial of Service (DDoS) attacks because resources used for providing Cloud and Grid services usually are connected to the internet with high-bandwidth network links. DDoS attacks use a large number of computers that are controlled remotely and could very well be scattered around the world. The aim of a DDoS attack is to overload the target system, which then cannot answer the requests of its intended users. One technique to orchestrate these computers are botnets. A botnet consists of computers that have been infected by worms, trojans or direct attacks, which run a task in the background that connects to a remote resource that is used to orchestrate the infected computers' actions. In the recent past, many DDoS attacks have taken place targeting companies that discontinue support of the WikiLeaks platform. Attackers were successful in breaking down the websites or even the internal systems of companies like Mastercard, Visa and others.

Although particular nodes within a Cloud provider's network, that take part in a DDoS attack do not harm the provider's infrastructure, its reputation could be damaged. Moreover, if hacked nodes are used to send spam emails, the provider's range of IP addresses can get blacklisted by external mail servers and customers would lose the opportunity to run mail servers on these Cloud resources. The only way to solve this problem for customers is to change the provider in order to run their mail servers reliably, which usually requires a good deal of effort to set up their servers at a different location.

Experience shows that users of Cloud resources do not care about the security of their resources once their software is running.[48] On the other hand, Cloud providers have to ensure a secure and reliable operation of their site without relying on the patch management strategies of their customers. In 2008, a serious bug was discovered relating to the OpenSSL implementation of Debian Linux [138]. Due to an error in OpenSSL implementation, affected systems generated predictable random numbers used i.e., for creation of public/private key pairs. The bug existed for 2 years and nearly every machine set up during this time using a Debian-based distribution (e.g., Debian Linux, Ubuntu, etc.) was vulnerable. Services like SSH and SSL/TLS were affected as they are based on a private/public key pair, but attackers could easily predict any key pairs created on infected machines [247].

In order to close critical security bugs quickly, resource site administrators should be able to apply software updates to all machines on their own in a short amount of time. Garfinkel and Rosenblum explained why patch management is challenging in environments using virtualization technology [83, p. 2].

[48] Maybe this is where the witticism "Never change a running system" comes from.

> While conventional networks can rapidly "anneal" into a known good configuration state, with many transient machines getting the network to converge to a "known state" can be nearly impossible.
>
> For example, when worms hit conventional networks they will typically infect all vulnerable machines fairly quickly. Once this happens, administrators can usually identify which machines are infected quite easily, then cleanup infected machines and patch them to prevent re-infection, rapidly bringing the network back into a steady state.
>
> In an unregulated virtual environment, such a steady state is often never reached. Infected machines appear briefly, infect other machines, and disappear before they can be detected, their owner identified, etc. Vulnerable machines appear briefly and either become infected or reappear in a vulnerable state at a later time. Also, new and potentially vulnerable virtual machines are created on an ongoing basis, due to copying, sharing, etc.

Resource sites administrators must be able to create a "known state" once a crucial security vulnerability has been detected. One solution would be to prevent customers from accessing the VMs, boot the VMs one after another, and start the patch process. This method could potentially interrupt a site's service for a long time, depending on the number of VMs existing on the site because every machine has to be booted in order to apply the given patch. Even in small environments without commercial users this method becomes impractical. To prevent this, the solution must be able to apply software updates to even a large number of VM images within a short amount of time, without needing to interrupt the system's normal operation.

This can be achieved by providing a multi-layered disk image instead of a single one that contains all data. A layered file system is a virtual file system that comprises from more than one individual file system (layer) using a COW solution like UnionFS [176] or AUFS [10]. The term multi-layered expresses the possibility of including three or more layers into that file system in a flexible way and can be used in different scenarios. Finally, the term root file system expresses the use of the virtual file system itself as a root file system, in contrast to applying layers to individual folders.

There are different scenarios for using layered VM images, as shown in Figure 3.18. The user may set up the user specific part of a VM completely on her own, leading to a two-layered image (a). Alternatively, a software vendor (or one of its sales partners) may provide a layer containing one of its products (b). The layer could even include all of its products if the license management system used is able to restrict software access to customers with a valid license for that product. Provision of the layer also allows the vendor to keep the software up-to-date, as a service for its customers. Even in this case, the user may

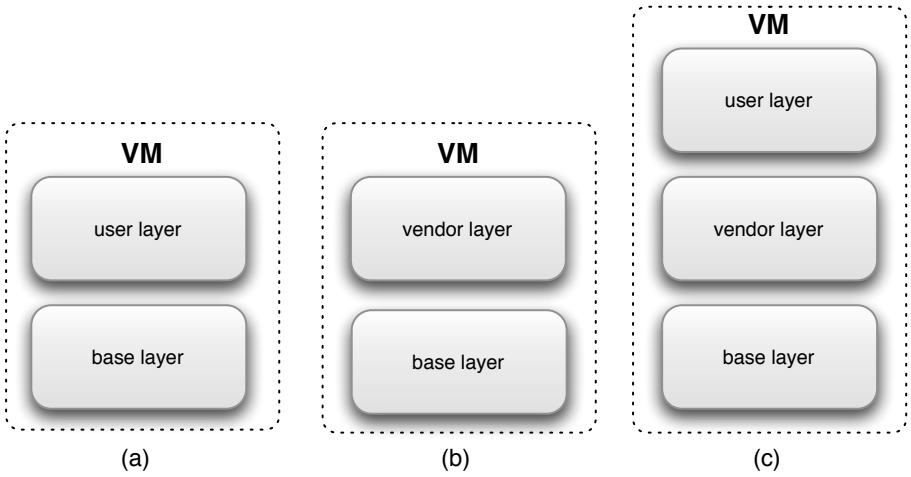


Figure 3.18: Scenarios for using layered VM images.

set up a custom layer on top of the vendor layer, containing extensions or other required tools (c). Nevertheless, other scenarios for using layered VM images might be possible and should be supported.

In environments used for batch job execution, a large number of similar jobs are submitted to a scheduler, where each job represents a part of the problem to be solved. A simulation process is typically divided into numerous independent tasks that will be executed concurrently on a number of compute nodes. These jobs are executed in multiple instances of the VM, and depending on the scheduler used, they are most likely executed consecutively. Thus, the user layer (as well as the base and possible vendor layers) should be cached locally on the individual nodes. To ensure that a cached layer is in a working state when it is used to form a root file system, it is best to make sure that it is not changed with respect to its original state. This can only be achieved by prohibiting write access to the layer during runtime.

3.2.1.3.1 Read-only Layer Access To ensure the reusability of layers cached at individual nodes, write access to these layers has to be prohibited during usage. Nevertheless, a running system needs write-access to the root file system. Thus, a single writable layer must be part of the multi-layered root file system. This can either be a ramdisk or a temporary layer stored locally on the node. The former approach is not suitable for HPC computing, where the users' applications likely consume much memory for their applications. Obviously, the latter approach also has a drawback: an empty layer has to be created before the VM can be booted.

All files not residing in the user's home directory will be lost after the machine is shut down. To allow the user to analyze errors or check the execution of her jobs, the logfiles have to be saved to a persistent storage area while the machine is shutting down. This also applies to the use of temporary layers instead of ramdisks because they are only kept during runtime and will eventually be reused for other VMs after having been completely overwritten

and reinitialized with a blank file system. This security precaution to prevent malicious users from trying to restore data from a temporary layer is done in the background. Multiple temporary layers exist that are used successively to avoid idle time while the layer is cleared.

3.2.1.3.2 Update Layers and Merging When a layer is updated, it has to be transferred to all of the sites using it. This involves marking all copies of that layer inside the caches of nodes as invalid, forcing the nodes to fetch the layer again. While the latter is usually a quick process because the connection between the nodes and the site's layer storage normally has a high bandwidth, the former implies transferring layers over wide area networks with lower bandwidth.

Instead of directly updating a layer, the layer can be mounted read-only inside a multi-layered root file system, together with a writable update layer on top. All changes, that occur during the update are then stored inside the update layer, which is much smaller than the entire layer that has been updated, because it only contains the changed files. Transferring the update layer to remote sites is thus a much faster process. At the remote site, the layer is updated using the contents of the update layer.

3.2.1.4 Exchanging VMs within Clouds and Grids

Using multi-layered images not only has the advantage of patching security flaws by updating a single layer, but it also results in small, user-specific layers containing only the software installed by the user. In multi-site environments, only the small user-layers have to be transferred between the sites instead of entire VM images, which are possibly much larger in size.

As mentioned earlier, users may be redirected to other ICS instances if the visited ICS does not have any resources left to fulfill the users' needs. By redirecting a user to another ICS, the system must make the VMs a user wants to modify available at the remote site. Every ICS gathers information about its users to minimize both data transfer and waiting time before a user can proceed to manage her VMs. When the (source) ICS decides to redirect a user to an external (target) ICS, it first queries the load of the target ICS to ensure that there are enough free resources to provide access for remote users. It continues querying other ICS instances until a target ICS is found that provides enough resources to external users. Then, the user will be redirected to this ICS. This decision is also stored in the ICS' local database. This allows the ICS to redirect a user the next time to an ICS instance which already contains the user's VM images. When the user logs into the target ICS, her data layers are automatically copied to this ICS when she updates the VMs on another ICS.

It should be noted that the load-based decision whether or not a user is redi-

rected to another Grid site does not affect any Grid scheduling decision because the ICS is not involved in scheduling decisions for job execution within a Grid. If a Grid job is scheduled to a particular site and the appropriate VM is not available at this site, the image transfer process is initiated independently of the ICS's load. The ICS fetches the VM image and deploys it instantly to the XGE. The load and the number of running VMs on the ICS host are only taken into consideration if the user wants to boot the VM on the dedicated ICS host for management reasons. Making the need to log into particular ICS instances independent from job scheduling decisions and the ability to execute jobs on local resources fulfills requirement R11.

As described earlier, every user can be identified by her certificate. This certificate contains a DN that is unique. The ICS web interface analyzes the certificate provided by the user's web browser to assign the performed actions to the user's account and to restrict access to data belonging to the given user. Since a user's Distinguished Name (DN) is unique on all sites, all data can be assigned to its owner throughout the whole system. Certificate infrastructures are common in numerous environments (e.g., certificates are also used in the Grid and by Amazon to manage access to users' VMs), so the ICS can easily be integrated into many different scenarios.

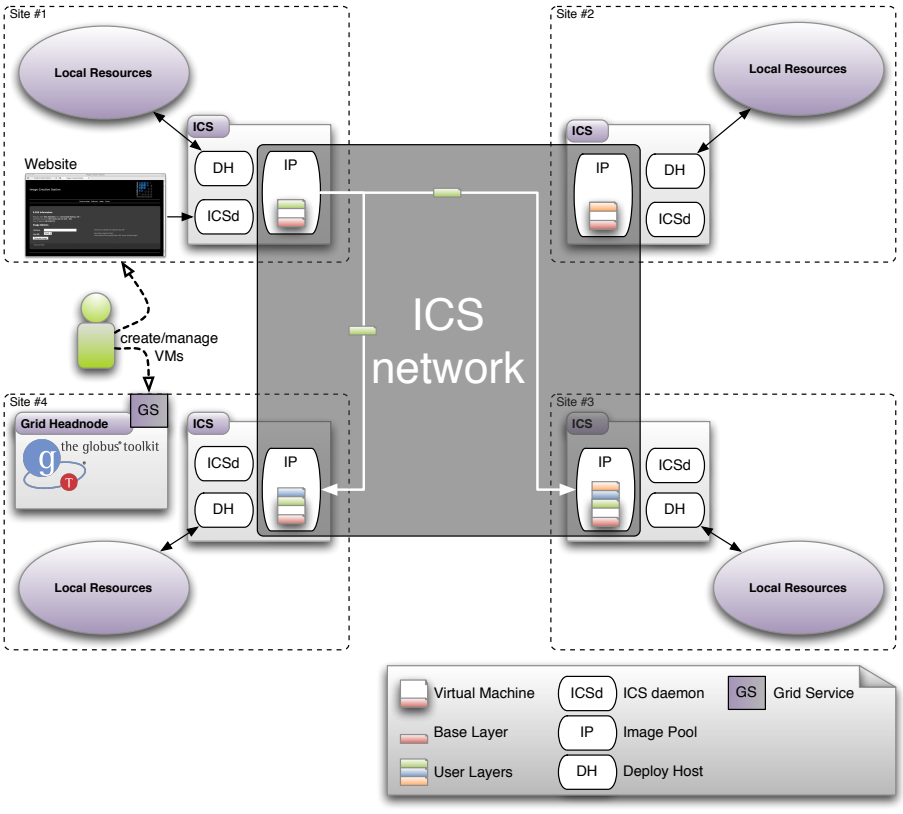


Figure 3.19: ICS instances on different sites are interconnected to share their users' data.

Figure 3.19 presents the architecture for moving a user's data between the ICS instances installed on the different sites. Copying entire VMs between the ICS instances would result in a huge amount of data being transferred from one site to another every time a user updates her VMs. This problem can be

minimized by using COW layers as described above [199], allowing system and user data to be divided into different layers. If the layer representing the basic system data is available on every site, only the user layer must be transferred to other locations when a user modifies his or her data within this layer. The size of the user layer depends on the software installed by the user.

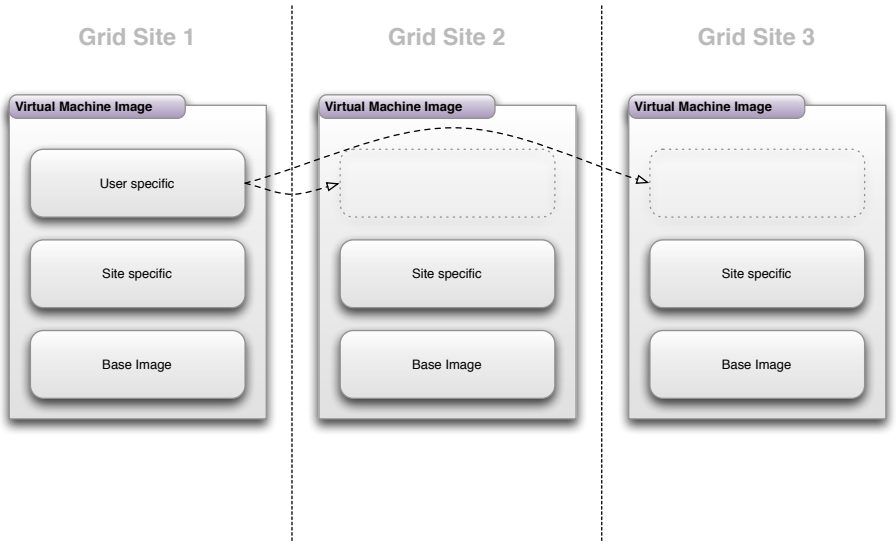


Figure 3.20: A VM's disk image is composed of different layers. By providing the Base Image *on every site, only the* User specific *layer must be copied. An optional* Site specific *layer might provide specific configuration files for each site.*

Figure 3.20 shows the underlying idea of using different layers for every VM. Every site must provide the *Basic Image* that holds the operating system and basic functionality. An optional layer called *Site-specific layer* might hold configuration files and software needed to provide a working environment on this site.

On top, the *User-specific layer* holds all software installed by the user. Moreover, all changes a user has made in the VMs are stored in that layer. This layer must be copied to sites where jobs are executed that belong to that user. It does not contain data changed or created during Grid job execution; it only contains data affected by the users' modifications on the ICS. Data generated during a Grid job execution, such as results or logfiles, must be stored on a network share within a cluster network of a particular Grid site. This data will never be available on the ICS, rather only via the Grid headnode.

Because users do not use every ICS instance available, user layers do not need to be copied to all available ICS instances, rather only to the ones regularly used by that particular user. A heuristic approach is used to decide which strategy is used to distribute a user's data. The local ICS administrator can configure it remotely, and updated layers will be fetched in advance for users who have logged into the ICS within a certain amount of time. This circumvents waiting times for users who frequently use more than one ICS to update their VMs because changes will already be available at the regularly used ICS instances when users log on to the system. The administrator can deactivate this function in order to minimize transfer overhead. This will cause waiting

times for these users because a remotely updated layer must be fetched before they can modify it on an ICS that does not have the layer available when the user logs in. In the following example, layers will be automatically prefetched for users who have logged on to the local ICS within the past 6 months:

Push Approach. Whenever a user shuts down a VM on an ICS, the updated user layer is transferred to all remote locations that this user has visited within the past 6 months e.g., if a user was redirected to another ICS earlier, her data will automatically be transferred to the target ICS when it is modified, providing an up-to-date image the next time the user is redirected to the target ICS.

Pull Approach. If a user has not used a given ICS for at least 6 months, updated user layers will not be copied to this ICS to prevent unnecessary data transfers. When a user is forwarded to an ICS that she has not used within at least this period or an ICS that she has never visited before, her data is fetched as soon as a VM has been chosen to be modified.

Because not only the ICS relies on actual versions of the VMs, the image transfer mechanism can also be used by other components to request specific VMs. As described earlier, the ICS provides two RMI interfaces. While the internal interface is used to access the ICS' VM image management capabilities, the external interface enables users to request a particular VM image and can be used by the XGE, for example. The ICS checks if the latest version of the requested image already exists on the image pool. If so, the image will be delivered immediately to the deploy host. If the latest version is not available or the image is not available at all on the image pool, the ICS immediately begins to fetch the latest version of the VM image from the appropriate remote ICS instance. After that, the image will be deployed to the deploy host. This functionality is fundamental in providing a solution that allows jobs to be executed that belong to users who do not have an account on the locally installed ICS instance. At the same time, it ensures the availability of the latest version of each VM image on the local computing resources. Consequently, requirement R6 is fulfilled.

3.2.1.4.1 BitTorrent Protocol The ICS can use BitTorrent's advanced network data transfer technology to transfer VM layers between the different sites. BitTorrent [37, 20] is a protocol for fast, efficient and decentralized distribution of files over a network. Every recipient downloading a file supplies this file (or parts of it) to other recipients also downloading the file. This reduces the overall costs in terms of network traffic, hardware costs and overall time needed to download the whole file.

Figure 3.21 shows the basic principle of BitTorrent. The node hosting the source file starts a *tracker* that coordinates the distribution of the file. Further-

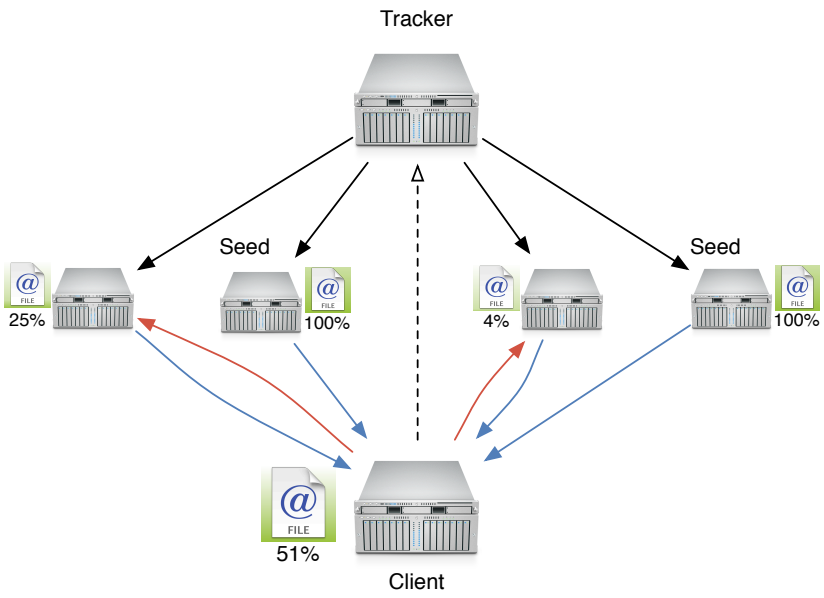


Figure 3.21: A BitTorrent tracker *organizes the download and keeps track of all nodes interested in a particular file. The* client *receives information from where (parts of) the file can be downloaded. Every client who downloads a file serves the already downloaded data to other clients. Computers that have completed the download and provide the complete file are called* seeds *[30].*

more, a file (a so called *torrent file*) containing metadata about the source file (the Uniform Resource Locator (URL) of the tracker) is generated and must be distributed to all clients (either actively with a push mechanism or passively with all clients downloading the torrent file from a web server). *Clients*, also called *peers*, connect to the tracker that tells them from which peers pieces of the file are available for download. A peer who shares parts or the entire file is called a *seeder*. Using this technique, sharing files between multiple peers benefits from high speed downloads and reduced transmission times compared to other techniques.

3.2.2 Dispatch Daemon (dispatchd)

When jobs should be scheduled to multiple sites, meta schedulers like GridWay [107, 108] can be used. Instead of submitting jobs to the headnode of a particular site, users submit their jobs directly to the meta scheduler. The meta scheduler gathers information about the state of all connected sites and schedules the pending jobs to the local headnode of one of the connected sites. To create an elastic system that can be used by commercial customers, such a meta scheduler must match some criteria.

Usually, several different resource sets are available to a meta scheduler. Since such schedulers were developed to connect different resources together, a meta scheduler, e.g., to be used within a company, must not only be able to use heterogeneous systems like a pool of desktop computers or a locally available cluster system which might be operated with different operating systems. It must also support administrators adding external resources like Grid oder Cloud computing resources without the need for restarting the scheduler itself, for a restart would result in aborting pending and currently running jobs, which would have to be rescheduled once the scheduler is running again. In the worst case, work already done is lost when the scheduler is restarted. This cannot be

tolerated because hundreds of users could be using the meta scheduler concurrently and thus lose their work.

Second, the meta scheduler must fit into the existing infrastructure to circumvent the need to adapt existing tools and to lower the burden of using the meta scheduler. This results in the analysis of already existing meta schedulers, which are commonly used in today's Grid computing community. A new solution should mimic widely used schedulers' interfaces to obtain a transparent integration into the existing computing landscape.

Third, commercial users need to influence scheduling decisions in order to prevent execution of computational tasks on particular resources for mainly two reasons. There could, for example, be legal restrictions on data that may not leave a geographical region. Or computational data could be sensitive and must not leave a company's in-house resources in order to minimize the ability of data theft or attacks against it. Existing schedulers usually do not allow users to influence a job scheduling decision. If users want to influence where their jobs are executed, they need to submit their computational jobs to the particular resource directly. Nevertheless, this approach forces users to decide themselves on which resource a particular job should be executed, even if several sites exist which could be used. Therefore, the system must enable users to specify a set of resources which can be used for computation of a particular job. Since this is a crucial requirement for commercial users, it can be found as R20 of the requirements catalog.

Last, since Cloud computing resources are easily available, a meta scheduler must be able to deal with these resources. In contrast to traditional resources, the number of nodes in the Cloud is not static. Traditional meta schedulers would not schedule jobs to a resource site whose nodes are already busy. A Cloud enabled meta scheduler must have knowledge that a particular site can provide additional (spare) compute nodes whenever new resources are needed. Meta schedulers must be extended to automatically acquire new resources from a Cloud provider as needed. To minimize computational costs, acquired resources must be shut down when they are no longer needed. Such functions reflect in R19 of the requirements catalog. In addition, as described in R18, the meta scheduler must avoid unnecessary utilization of costly external resources if spare resources exist which cost less to use.

As mentioned above, GridWay is a well-established meta scheduler e.g., used in the D-Grid and well-known to the users. It is connected to different resource sites and knows the state of the particular resources. On the other hand, it has many inferiorities that make it impossible to build an elastic system upon GridWay. It does not support the addition or removal of resources during runtime and it needs to be restarted every time the connected resources are altered. During the BEinGRID project[49] this problem seems to have been

[49] `http://www.beingrid.eu/`

resolved by the development of a GridWay plugin called *Broker GW-SLA* that can dynamically add resources to a GridWay instance during runtime [25]. It controls the internal state of GridWay and can add new resources based on the result of a SLA negotiation. Unfortunately, the authors of the software did not make the software (and their patch for GridWay) available.

Another reason why GridWay could not be used to build an elastic system is its inability to deal with Cloud providers. Scheduling decisions in GridWay are made based on the current state of the connected sites. If a site does not have resources free to execute a Grid job, GridWay would not schedule to this site even if new resources on that particular site could be required. Moreover, the attempt to minimize costs for external resource usage would ideally result in a system that have only the amount of resources available needed to execute the exact number of jobs scheduled to the system. As stated above, GridWay would not schedule additional jobs to a system that has reached its capacity.

Nevertheless, since GridWay is the de-facto standard in the Grid community and has matured in recent years, it should be extended in order to provide well-known interfaces to the users and to enable users to use already existing tools, such as for job management tasks. Moreover, GridWay also provides several mechanisms which are crucial for distributed job execution. It ensures the reliable and correct transfer of each job's input data to the remote sites where the job is executed later as well as the transfer of the job's output data back to the GridWay host once a job has been completed. Obviously, this is a basic requirement stated in R8. Additionally, GridWay also supports transferring intermediate results and checkpointing files back to the user during job execution as required in R13. This allows users to obtain information about their jobs' proceedings without needing to explicitly log into the remote resources executing the job, as this information is available directly at the GridWay host. If GridWay uses a local CRM for job execution, this behavior must be supported by the CRM at hand, such as Omnivore, which supports periodical data transfer during job execution.

To provide the needed functionality missing in GridWay, it must be extended by either developing a patch to add this functionality directly to the meta scheduler or by developing an additional component that provides this functionality. Since GridWay's source code is scattered on 303 files with roughly 80000 lines of (mostly uncommented) code,[50] an additional component has been developed that extends GridWay's functionality. To match the requirements presented above, this extension was designed to be transparent to the user.

Instead of submitting jobs to the GridWay meta scheduler, users submit their jobs to a component called dispatchd. Since it acts as a wrapper, it provides a command line interface that behaves exactly like GridWay. All of its command

[50] These values are pertinent to GridWay Version 5.6.1, which was used for developing the elastic system.

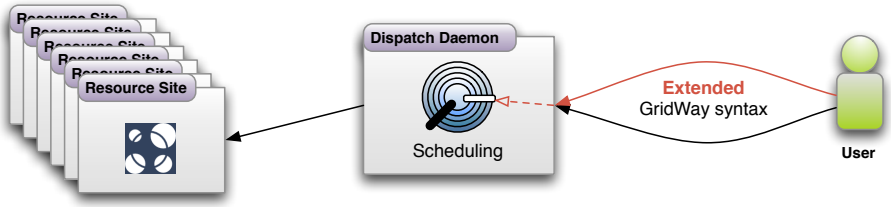


Figure 3.22: The dispatchd provides users with an extended GridWay syntax. In addition to traditional GridWay commands, extended commands can be used to influence the scheduling decision.

line options are fully supported and simply forwarded to one of the connected GridWay schedulers after the given command has been parsed by dispatchd and a decision has been made which particular GridWay instance known to dispatchd will be used. Using this approach, additional command line options can be made available to users, which allow them to influence scheduling decisions. This functionality is shown in Figure 3.22. As described, dispatchd is connected to several resource sites and each of them exposes its functionality via its own GridWay interface. Using traditional GridWay schedulers as interface to the resource sites has many advantages over connecting directly to the headnodes of the resource sites. GridWay has numerous connectors for different Grid middlewares, such as Globus, which have been tested can be reused. Furthermore, GridWay also supports the Omnivore Peer-To-Peer scheduler [102, 54], which can be used to connect resource sets whose participants are dynamically connected and leave the network, e.g. Desktop Grids. By supporting this variety of different local CRMs, requirement R7 is satisfied.

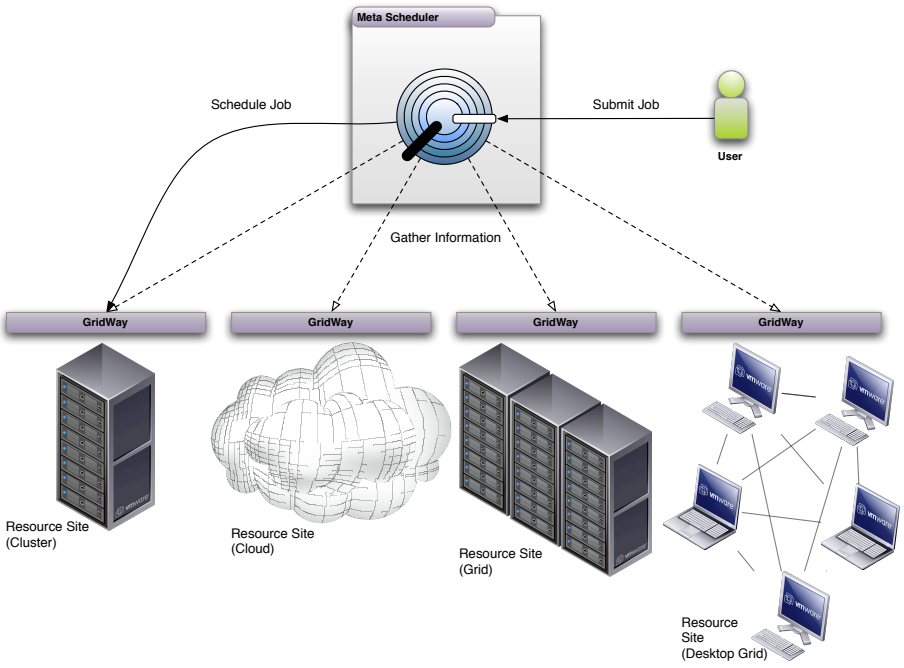


Figure 3.23: The Dispatch Daemon (dispatchd) is connected to several resource sites which provide a GridWay interface to the public. The daemon gathers information about each site's state and decides where to schedule the jobs submitted by the users.

Figure 3.23 shows the architecture of a system that uses dispatchd. Information is gathered from the connected sites, e.g., including the number of nodes and their current load, for example. Additionally, dispatchd contains information regarding whether or not a resource site is "elastic." An elastic site can

provide more resources than are currently available. It even knows maximum number of nodes every site can provide. If a site is marked as elastic, the maximum number of nodes will be taken into account when making scheduling decisions instead of the number of nodes available at the moment. This enables dispatchd to schedule jobs to resource sites even if it seems that no resources are available for job execution at the moment as required by R19. In respect to requirement R18, dispatchd first tries to schedule jobs to idle resources of the lowest layer available. Cloud computing resources will be only used if resources on lower layers can not satisfy the dispatchd's needs.

The characteristics of every resource site are specified in a configuration file. Each site is represented by a configuration file placed in a specific directory. This directory is regularly checked and all configuration files are parsed. If a configuration file has changed since the last check, the dispatchd configuration is updated. Every resource can be enabled and disabled via its configuration file when dispatchd is running. This strategy allows users to easily add or remove resource sites without needing to restart the whole system. Nevertheless, active jobs scheduled to a resource site which is removed during runtime will be rescheduled to resources on the remaining sites.

To make this component transparent to users, all command line arguments supported by GridWay are accepted. The dispatchd analyzes the commands performed by the users to figure out how many hosts a job needs to be executed. By providing the same commands as GridWay, users can do their work in the usual manner without needing to deal with a new software interface or even without noticing that they are not talking directly to GridWay. If needed, additional command line options can be used to influence the scheduling decision. Referring to Figure 3.1 on page 89, users can specify the last layer on which the resources may be located that are allowed to execute a given job. For example, by specifying Layer 2 as the last layer allowed, it can be ensured that the job will not be executed on resources outside of the company. As requested in R20, users can influence the scheduling decision of particular jobs.

Moreover, dispatchd also meets requirements R8 and R13. Because dispatchd uses the basic functionality provided by GridWay, GridWay's data transfer mechanisms, which have been described above, are also available when using dispatchd. This also applies to intermediate results and checkpoint files.

3.2.3 Request Daemon (reqd)

The resource sets found on lower layers[51] are usually static: the number of compute nodes in a cluster is fixed. Cloud resource providers such as Amazon provide a virtually unlimited number of resources, but the Cloud customer is responsible for acquiring and starting these resources on her own.

The Request Daemon (reqd) is a component that can request new compute

[51] The lower the layer, the nearer to the user are the resources located on that layer.

nodes depending on a given site's load. It is linked to a GridWay scheduling instance that periodically provides information about its state. For example, GridWay periodically checks the length of its waiting queue that contains all pending jobs that are to be scheduled on available resources. In addition, the length of this queue is sent to the reqd running on the same host. The waiting queue's length represents the current system state, showing the actual needed resources which are not available for computational tasks at the moment because if free resources were available, jobs would have been scheduled to these resources.

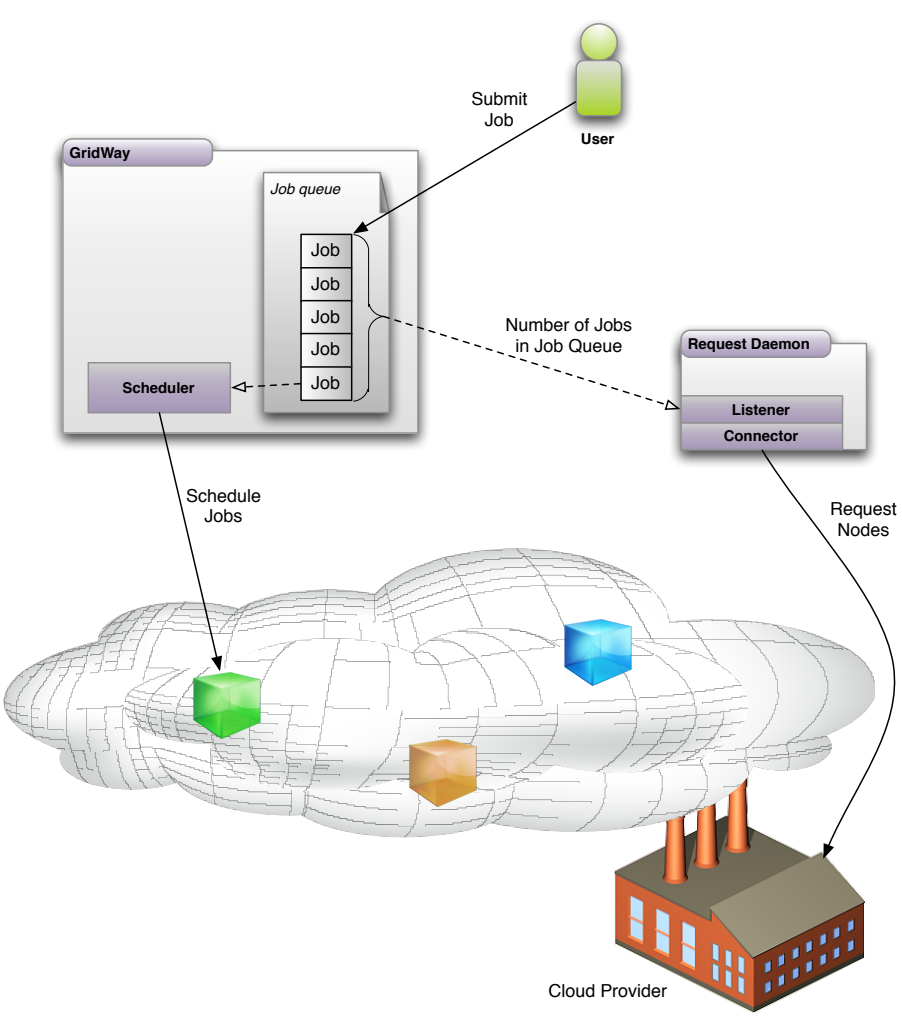


Figure 3.24: The GridWay meta scheduler sends the length of its job queue periodically to the Request Daemon (reqd). Depending on this information the decision is made whether or not to request new resources from the Cloud resource provider.

Figure 3.24 depicts how these components are connected. Whenever a user or the dispatchd submits a computational job to GridWay, this job will be put into its job queue. If resources are available for job execution, the job will be handed over to the local CRM and scheduled to these free resources. If no resources are available, the job will remain in that queue until resources become available. When dealing with Cloud resources, new resources can be acquired when needed. The reqd decides, based on the information provided by the meta scheduler and the local reqd configuration, when new resources are requested from the Cloud provider. If the new resources become available,

they have to be registered to the local CRM. Once completed, the CRM will update its state and schedule the pending jobs to these spare resources. If the request for new resources is declined for some reason (e.g., the Cloud provider can not fulfill the request at a certain time) or if fewer resources than requested are provided, GridWay and the local CRM will automatically adapt to the new situation; jobs will only be scheduled to the available resources, pending jobs will remain in the job queue.

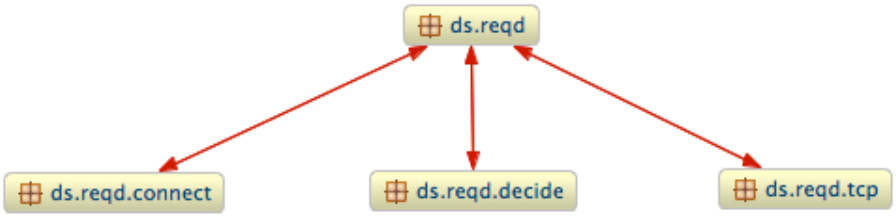


Figure 3.25: The Request Daemon (reqd) is designed modularly to support different resource providers and different strategies to decide when new resources should be acquired.

The reqd is designed modularly and allows the use of different resource providers and different strategies to decide how many nodes should be acquired from the Cloud provider. Figure 3.25 shows the three main parts of reqd. Beginning from the right, the `tcp` package contains the Listener module, which waits for the state information of the local GridWay. This module is used by the GridWay patch to announce the slots needed to handle the pending jobs. If a request is received by this module, it is handed over to the `decide` package, which uses a particular strategy to decide if and how many new resources should be acquired from the backend. Strategies can easily be added by every site administrator to match the particular environment. If new resources are about to be acquired, the command to acquire resources is forwarded to the `connect` package, which contains the interface to the particular infrastructure management interface used on the site. Like the strategy, this interface can be created or modified easily by the site administrator.

The reqd provides a standard strategy which is fairly simple. Every incoming request for new resources is stored in an internal data structure. New nodes will only be acquired if a certain number of requests were received within a certain amount time. The strategy used acquires new nodes if 10 resource requests are received within 5 minutes. The number of new resources to acquire is specified by calculating the mean of requested nodes. Since this is a very simple strategy, administrators can implement their own strategies easily.

The connector is responsible for communicating with the infrastructure management interface, which enables users to request new resources or shut existing resources down. Examples of such an interface include the Amazon interface for EC2 or the provided interface for IBM Tivoli Service Automation Manager (TivSAM). Comparable to Amazon EC2, TivSAM can provide new resources on-demand. The reqd was developed in the Biz2Grid project in conjunction with IBM and its functionality has been successfully proven in a

prototypical environment. Again, administrators can easily develop their own connectors allowing the interaction of reqd with the individual local infrastructure middleware.

Resource Probing

Since each running resource in the Cloud costs money, the number of running instances must be kept minimal. The reqd can not only request the Cloud provider to activate new instances but is also able to request it to shut down unused instances. One indication that idle resources exist is an empty job queue, indicating that there are enough resources to execute new jobs immediately. That might (but does not necessarily) mean that there are some free (running) instances that could be shut down without any impact on the system's overall performance. While it is easy to decide when to acquire new resources, it is difficult to decide which instances to shut down.

Meta schedulers like GridWay do not have any knowledge about the site's particular compute nodes, because the CRM only provides aggregated information. This information is suitable for deciding whether or not to schedule jobs to the CRM but does not provide enough information to decide which hosts are superfluous and can therefore be shut down. An additional component has been developed which provides the ability to probe computing resources in a flexible and adaptive manner.

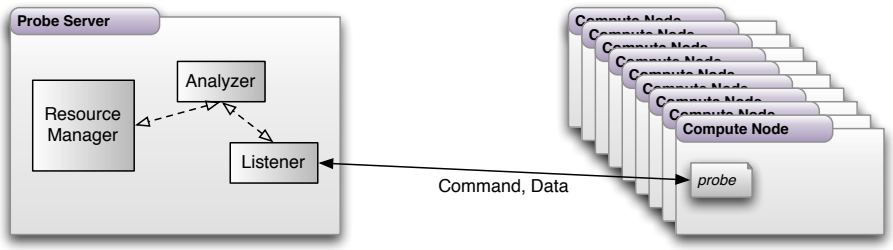


Figure 3.26: The probe server allows for easy analysis of the state of compute resources in order to decide whether or not they are idle and could be shut down.

The probe server is a component that can be run on the same host as reqd. This resource has two main goals: first, it is periodically queried by a script running on the compute resources to fetch a command, which gets executed on the nodes. The result of this command is sent back to the probe server to be analyzed. Second, the state of each compute node can be queried to decide which nodes to shut down. A network socket facilitates the communication between the probe server and its clients.

The administrator can specify a custom command and a condition stating that a particular resource is idle if this condition is met. Moreover, the administrator can modify the command sent to the compute nodes during runtime, making it possible to flexibly react on any changes. Nevertheless, providing a command is optional. The administrator can also use her own program that reports any desired measurement to the probe server. By setting an appropriate condition, the probe server is able to deal with any input. One benefit of specifying a

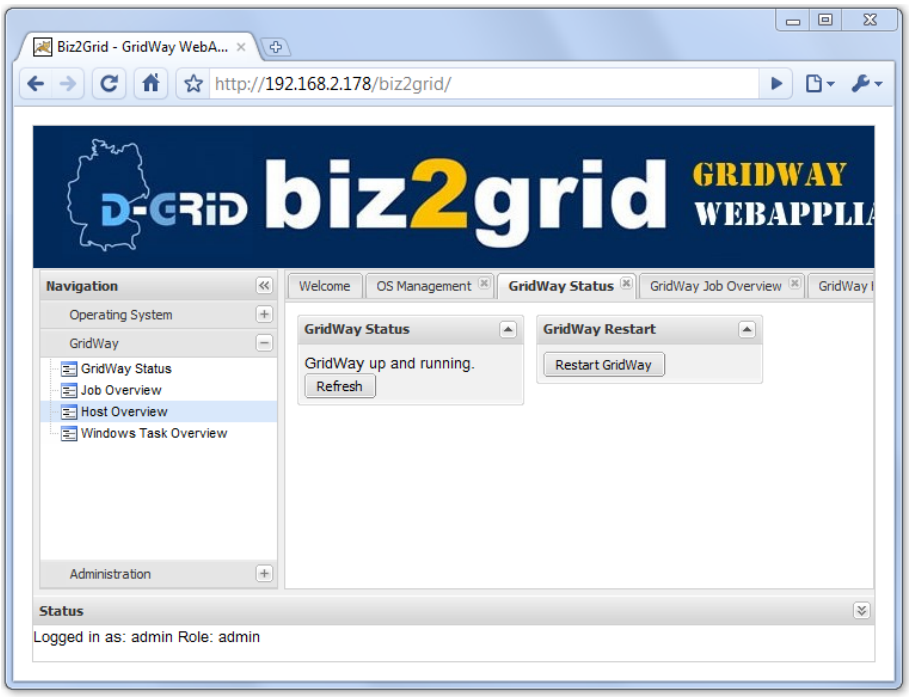


Figure 3.27: The web appliance provides management functionality and information about scheduled jobs. Running jobs can be observed and users can obtain information about the system's state. If problems arise, administrators are able to reset the system.

command on the probe server is that OS built-in commands can easily be used to determine a node's state. For example, a simple shell script can be used to query the load of a Linux compute node. By providing such an approach, every administrator can choose the desired method to probe compute nodes by providing maximal flexibility, i.e. this solution can deal with any desired OS installed on the compute nodes.

Moreover, the probe server tracks the state of all nodes sending requests. It can request a certain resource's state and returns if a resource is idle or not. Therefore, the probe server knows the state of the entire system and can decide if idle resources exist and which resources can be shut down, meeting requirement R19.

3.2.4 Web Appliance

Supporting all commands provided by the GridWay interface, dispatchd can be used with several tools originally developed for use with GridWay. During the Biz2Grid project, a transparent and user friendly administration user interface was developed which enables a system's user not only to track their jobs in the system but to abort already running jobs. Using this tool, system administrators can gather information about the connected resources, perform management tasks, and even restart the whole system.

The frontend, which is depicted in Figure 3.27, is based on the Google Web Toolkit (GWT).[52] The application was developed to work with every version of GridWay and can be also used with dispatchd to provide aggregated infor-

[52] `http://code.google.com/webtoolkit/`

mation about the whole system. It was designed to fulfill many of the commercial customer's needs. The requirements catalog in Table 2.1 showed that customers care about their jobs and would like to have information regarding where their jobs are executed. Moreover, they would like to be able to influence jobs and even abort them. While it provides information for normal users, administrators can perform additional tasks on a system. In the following, the functionality provided is presented.

1. The web application provides users with a status overview of submitted and already executed jobs. Therefore, R12 is fulfilled.

2. It allows users to abort submitted jobs regardless on which site the particular jobs are executed, fulfilling R15.

3. Administrators can display an overview of all of the worker nodes available to the scheduler, which fulfills requirement R16.

4. If an error occurred or the system needs to be restarted e.g., for maintenance reasons, administrators can use the web appliance to reset individual hosts or even the entire system, which has been described by requirement R14.

A powerful tool for system administrators, the developed web appliance also allows business users with no experience in distributed computing systems to control their jobs in the distributed environment.

3.2.5 End-To-End Data Encryption

Requirement R10 demands that the users' data has to be stored in an encrypted form even within the VMs, to aggravate possible attacks on the users' sensitive data. Because data encryption and decryption results in an computational overhead, users themselves should be able to decide whether or not particular data should be encrypted. Therefore, a tool has been developed that allows users to encrypt data before transferring it to the VM.

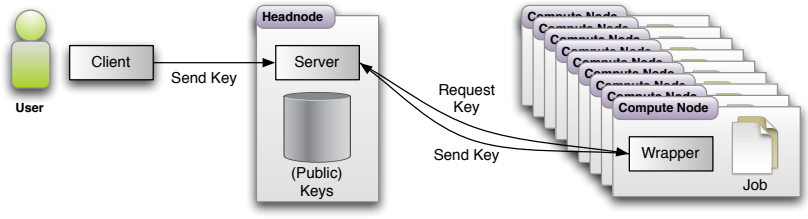


Figure 3.28: To ensure secure end-to-end data encryption, the architecture contains a client running on the user's computer, a server component running on the resource site's headnode, and a wrapper inside each VM.

The architecture of the proposed solution is shown in Figure 3.28. Three main components are involved to ensure a secure end-to-end data encryption.

Client. The client runs on the user's machine and provides a GUI to manage the user's keys and choose which files to encrypt. It uses a TLS-secured connection based on x.509 certificate authentication to commu-

nicate with the server in order to obtain keys for decryption of encrypted job results.

Server. The server component is installed on the resource site's headnode and manages the key pairs required to secure symmetric keys used for the encryption and decryption of data during job execution. It also provides the symmetric keys to the wrapper running in the VMs on the compute nodes.

Wrapper. The wrapper script is needed to request the symmetric key from the server to decrypt the data before job execution, and to encrypt the result once the job execution has finished. This script ensures that the data is encrypted when it is stored in the distributed system and is only decrypted during job execution.

The client provides a GUI which allows the user to easily select which file(s) to encrypt. If a directory is chosen, it gets compressed. The user can create a new symmetric key or choose an existing key which is used to encrypt the data. Moreover, this key is encrypted with the server's public key. Thereafter, the client connects to the server, which is installed on the headnode of the particular resource site. This connection is encrypted and the user's x.509 certificate is used for authentication. Using this encrypted connection, the encrypted symmetric key is sent to the server.

The encrypted data itself can either be put into one of the user's VM images or it can be sent to the host on which the user submits her job. Instead of starting the job directly, a wrapper script is used. Once the job has been scheduled by the CRM, this wrapper script is executed. Before executing the real job script, it connects to the server, authenticates with a nonce[53] and downloads the symmetric key that is needed to decrypt the job input data using a Secure Sockets Layer (SSL) connection. Once finished, the wrapper connects to the server and requests a new symmetric key. The server generates this key and sends it to the wrapper, which uses it to encrypt the job result data. It contains an expiration date which renders the key unusable after a specified amount of time.

A user is only able to access this data before the key expires. Therefore, the client connects to the server, providing its public key, which is used to encrypt the symmetric key that has been used to encrypt the result data. This encrypted key is sent back to the client and can then be used to decrypt the result data.

A local administrator needs to provide strong security mechanisms to prevent data leakage of the server component due to its exposed role in the proposed architecture.

[53] A nonce ("number used once") is a randomly generated password issued in an authentication protocol which can only be used once e.g., to prevent replay attacks [92, 223].

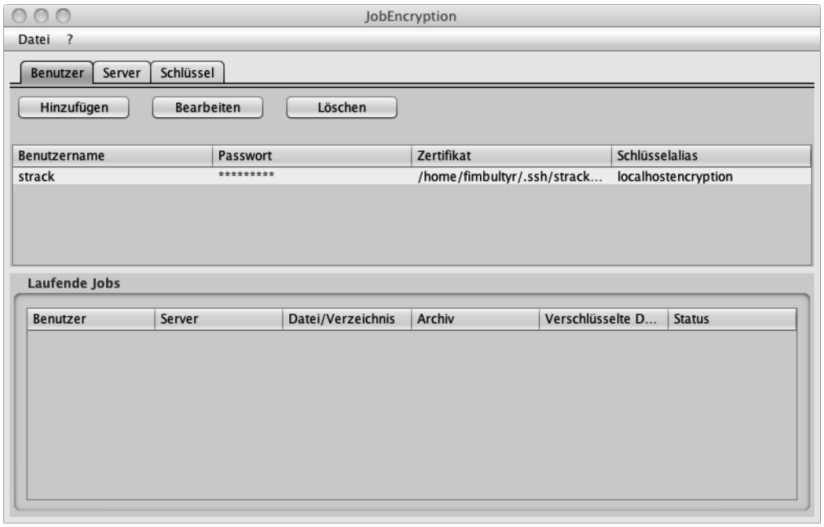


Figure 3.29: The frontend in which users can encrypt their data. They must create a certificate which is used for data encryption. Moreover, an overview over running jobs is provided.

The GUI of the client is shown in Figure 3.29. It is used to manage the different keys which can be used for data encryption. The GUI is clear and provides the end-to-end encryption functionality in an easy manner to enable even unexperienced users to use end-to-end encryption in publicly distributed systems. This component provides all of the functionality described in requirement R10.

3.2.6 Secure Inter-site Network Communication

In traditional Grids, one user's jobs can be executed on different locations. Unlike serial jobs which run independently from other jobs, parallel jobs require a working network connection between all participating hosts for data and message transfer e.g., by using MPI.

In addition, allowing access to particular worker nodes from external locations allows a system to host services e.g., by running a Web service container on Grid resources. In contrast to batch job scheduling, which does not usually require access to the compute nodes from the outside, service-oriented offers are only possible if users can access the given hosts.

The basic idea of this approach is to assign external IP addresses to the worker nodes instead of private addresses that can be only used in a private network.

An overview of the proposed architecture is shown in Figure 3.30. The meta scheduler, which is responsible for job scheduling decisions, handles the distribution of the job to a number of chosen sites where the headnode of the Grid middleware is installed. Every headnode handles the local scheduling in conjunction with the local CRM. At this point, the XGEs on all sites are invoked and coordinate the distribution of the Xen VMs to the worker nodes. The firewall solution, as proposed in the following sections, as well as inter-VM communication channels are set up. Once the VMs are ready on all sites, job execution begins and the user is able to communicate with all assigned VMs.

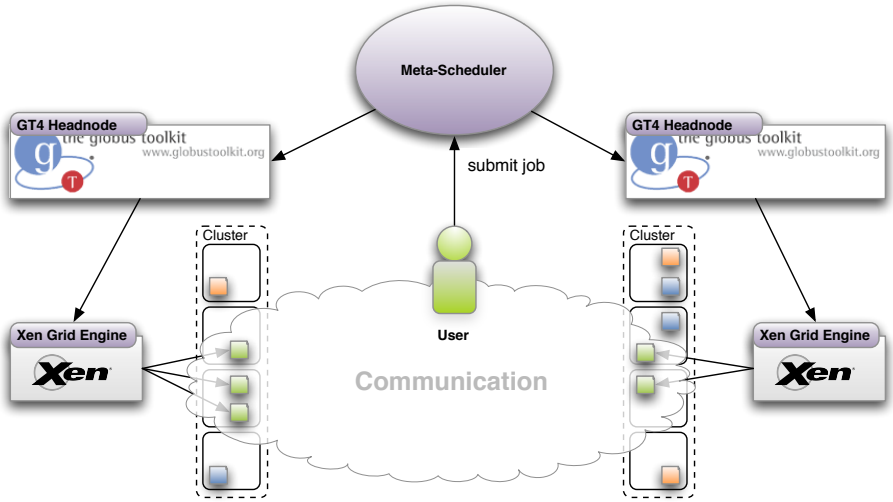


Figure 3.30: When running a parallel job across several sites, the participating hosts must be able to communicate which each other. Therefore, all of these hosts must be put into a "communication group" that allows data transfer and message exchange among the different sites. By accessing this group, the user has access to intermediate results and computational data even during job execution.

3.2.6.1 Secure Infrastructure Communication

An important issue is to ensure secure communication between the virtual worker nodes and the infrastructure services like shared storage (accessible via NFS or Samba), automatic IP address configuration (DHCP), or host name resolution (DNS), as described below.

3.2.6.1.1 Access to Shared Storage If the shared storage is only accessible from local, private IP addresses, devices with external IP addresses cannot access it. In the case where all virtual worker nodes have public IP addresses, there are three possible solutions:

Only allow access from known IP addresses. This is the easiest solution and would be sufficient in most cases. However, this solution does not offer flexibility in case of address range changes, new nodes etc. Furthermore, it creates additional complexity in multi-site setups.

Allow access on an application/job basis. Every application run has one or more VMs assigned with dynamically assigned IP address(es). Based on these addresses, access is granted on the same dynamic basis. This approach is more complex than the previous solution, but increases flexibility and scalability.

Place VMs and the storage server inside a dynamic, virtual VLAN. A virtual VLAN is the same as a physical VLAN, except that so-called virtual switches are used instead of real ones. The VDE switches of the Virtual Square Project [41, 42] that tries to interconnect different virtualized entities can be used to achieve this goal. Virtual Square even supports an encrypted virtual cable connection to interconnect the switches.

Due to the scalability benefits, the second solution was chosen. The solution guarantees that all VMs running on a particular cluster site only have access to their assigned storage.

Multi-site shared storage is not possible due to known problems with exporting file systems over network borders. This drawback can be circumvented by assigning a dynamic VPN between all participating sites.

Every VM has a dynamically assigned IP address that is fixed for the duration of a job. After the associated application finishes, the address is released. On each participating site, a DHCP server is set up to distribute the IP addresses. To prevent abuse of the service, DHCP requests are not allowed to pass through the routers guarding the network. Thus, all Grid nodes with a known MAC address can request an IP address. Nodes with unknown MAC addresses will not receive an IP address. Because all users are superusers inside their VMs, they could set a new MAC address and try to get another IP address from the DHCP. The presented solution prevents this using a fine-grained MAC address filter installed in dom0 of the virtual worker node. The filter knows the legal MAC addresses of the running VMs and thus only allows DHCP requests if the valid MAC address is used. A properly chosen lease time ensures that no connection between the DHCP server and the VMs is necessary once the application is started. With this setup, automatic address configuration for nodes on all sites can be guaranteed.

One problem arises when the presented setup is used: the management hassles introduced with multiple DHCP servers on several sites. To provide reliable IP addresses for the VMs, all DHCP server need to be synchronized. This could be accomplished by using a central resource that manages DHCP configuration. This configuration is automatically distributed to all participating sites. Because the number of VMs on one site is limited (the offered resources can only handle a certain number of VMs effectively), the distribution interval is not critical. It could run on a daily base.

Assigning host names to VMs is a necessary step to ensure identification for users and services (e.g., the resource manager, Globus Toolkit, monitoring software). Host name resolution is done by a DNS resolver installed on each site. After a VM is started, only connections to the well known DNS resolvers on the local site are allowed; connections to the DNS port on other machines are forbidden.

3.2.6.2 Dynamic Network Security

Dynamic firewall rules enforce network security and protection for the virtual worker nodes on dom0. A default XML template is used to create firewall rules for a particular application belonging to a user. It represents a secure default setting for average users, i.e., an uninterrupted workflow is possible without endangering other components. Based on this template, the user can apply her

own rules. The basic template allows access to the LDAP, NFS and HTTP services by default and limits the network speed (this could be a default value or a value given by a SLA.)

3.2.6.2.1 Firewalls To actually protect the network with the generated rules, the XGE has to deploy the script to the privileged dom0s on the virtual worker nodes.

The XGE contains a list of VMs on which each application is executed. The list is obtained from the local Grid engine that decides on which hosts (VMs and corresponding dom0s) the jobs will be executed. The generated firewall rules are now copied to all of these machines. Afterwards, the rules are deployed and become active. After the computation is finished, the XGE is also responsible for the removal of the deployed firewall rules. The XGE removes only the rules belonging to a certain application and user by managing a unique mapping between application, user and firewall rules.

It is important to install the firewall rules on dom0 of the host machine, not on the virtual machine itself. All users are superusers inside their VMs, so they could easily remove the rules and thus bypass the protection. To enforce the protection offered by these rules, every connection that is not explicitly allowed is denied. In a starting state, a VM is not permitted to open outgoing connections. When the template rules are first applied, connections to local services are allowed.

3.2.6.2.2 Traffic Limitations To ensure that network connection bandwidth is not misused, the traffic of VMs can be limited. A default rate for all traffic is defined by the base template. Furthermore, it is possible to define limits for all types of connections based on a single port or port ranges, protocols or sets of hosts (e.g., the VM interconnection rate is higher than the connection to hosts outside this set).

3.2.6.2.3 Protection against Denial of Service (DoS) Attacks Besides preventing bandwidth abuse from internal users, DoS protection is another threat the solution has to deal with. However, a complete and bullet-proof protection against this type of attack is not available. If the attackers' bandwidth and the number of attack nodes is high enough, nearly every infrastructure could be spammed with unwanted traffic. To mitigate the risk of DoS attacks, some on-by-default countermeasures are put in place:

- Dynamic traffic limitation of Internet Control Message Protocol (ICMP) messages to broadcast and multicast destinations from outside connections to prevent so-called *smurf* attacks [32].
- A limit of 100 incoming TCP SYN-packets per seconds ensures a good

SYN-flood protection. The protection is bucked-based, so it does not affect performance if the number of SYN-packets is lower than the specified threshold.

- Incoming connections to ports that are likely to get scanned or abused (e.g., OpenSSH) are limited.

3.2.6.2.4 Information Exchange To handle multi-site applications, a permanent communication channel between all running XGEs is needed. The purpose of the channel is to exchange information that needs to be present on all involved XGEs. Strong cryptography and authentication ensures that this information stays private and cannot be intercepted by malicious entities.

3.2.6.3 Inter-Node Communication

By default, the communication between the worker nodes is not encrypted. This is mainly due to overhead concerns. Encrypting the communication using a common cryptographic protocol produces CPU load, and the transfer rate is decreased. Nevertheless, setting up an encrypted connection between all worker nodes could be accomplished with a small overhead. All worker nodes are equipped with a SSH public/private key pair by default. This key pair is generated during creation time and is unique for every VM. After the VMs are deployed to the worker nodes, every VM contains the same key pair, so password-less access between all VMs is possible. OpenSSH can be used to setup an encrypted tunnel. Since Version 4.3, OpenSSH can also be used to setup virtual private networks. With the built-in scripting support, a tunnel could be setup automatically during startup time or thereafter by issuing certain commands.

Providing inter-site communication between VMs that work on the same job has been formulated in requirement R9. Meeting this requirement makes the execution of large-scale parallel computational jobs possible that would exceed the amount of local resources available on a single site. Besides that, the job's owner can access her data during job execution regardless of from the particular resources sites executing (parts of) the job; therefore requirement R17 is also satisfied.

3.3 Summary

In accordance with each of the requirements stated in Section 2.3.2, a system architecture has been proposed called the *Elastic Onion*. Different resources located around the user's local workstation are available to execute computational tasks on the user's behalf. Although the system, which spans around the user, decides autonomously where a particular job will be executed and transparently performs the necessary operations for successfully executing the job,

it is user-centric and respects attributes like the privacy settings (which may prevent the external execution of a particular computational job).

After providing an overview of the proposed system, its components have been presented. The ICS is the component that provides creation and management functionality to the users. It can provide different user interfaces and handle different virtualization backends. Therefore, it can be seamlessly integrated into existing infrastructures.

It must be clarified that the ICS is a component designed for VM management tasks. It enables users to maintain VM images independent from the location of the particular ICS instance a user decided to use.

To ensure job scheduling, the Dispatch Daemon (dispatchd) connects resource sites together providing the user with a single point of contact. It aggregates information about the connected sites and uses it to make scheduling decisions. In contrast to existing Grid meta schedulers, it respects the users' decisions not to use particular resources. It is based on proven components that have been in distributed environments for a long time.

The Request Daemon (reqd) enables the dispatchd to deal with Cloud resources by autonomously acquiring and shutting down resources managed by different infrastructure backends. The modular design allows the user to easily develop new backends and integrate even proprietary dynamic infrastructure management tools into the system.

A web frontend has also been introduced providing an interface that displays the system's state and information about running jobs. It can be accessed easily by every user to manage jobs scheduled to the system. This interface can also be used by administrators and enables them to perform management tasks in an easy and comfortable manner.

Another component has been introduced that provides end-to-end data encryption. A user can decide which data should be encrypted before putting it into a VM image. This guarantees data security even if data is stored on a shared storage used concurrently with other users.

Finally, an architecture has been presented that allows users to execute a parallel jobs on distributed sites by ensuring reliable network communication among the different sites and thus overcoming traditional limitations of cluster computing and Grid computing that focused on job batch scheduling. The proposed architecture takes several security threats into account and provides mechanisms to counter them.

As shown in Table 3.1, the presented components were designed in order to fulfill requirements presented in the requirements catalog in Table 2.1 in the previous chapter.

The Image Creation Station (ICS) presented in Section 3.2.1 was designed to provide an easy-to-use interface to create and modify VMs, fulfilling require-

Table 3.1: All requirements presented in Table 2.1 are fulfilled by the components presented in this chapter.

Component	Requirements
Image Creation Station (ICS)	R1, R2, R3, R4, R5, R6, R11
Dispatch Daemon (dispatchd)	R7, R8, R13, R20
Request Daemon (reqd)	R18, R19
Inter-site communication	R9, R17
Data encryption tool	R10
Web appliance	R12, R14, R15, R16

ments R1 – R5. The requirements regarding automatic VM image distribution for job execution (requirements R6 and R11) are met by the VM image exchange mechanism presented in Section 3.2.1.4.

The Dispatch Daemon (dispatchd) presented in Section 3.2.2 was designed to enable the use of a great variety of different computing resources (requirement R7). The utilization of the GridWay meta scheduler as a building block of dispatchd guarantees the fulfillment of requirements R8 and R13. Moreover, the scheduling mechanism and the Request Daemon (reqd) were de-signed to minimize computational costs and therefore fulfill requirements R18 and R19. In addition, by enabling the user to influence job scheduling decisions, dispatchd also fulfills requirement R20.

Communication requirements of the different computational nodes as expressed in requirements R9 and R17 are fulfilled by providing inter-node communication capabilities as described in Section 3.2.6.3, while data encryption as requested by requirement R10 is addressed by the data encryption tool presented in Section 3.2.5.

Finally, the web application presented in Section 3.2.4 provides information about the distributed system as well as the functionality to modify running computational tasks , therefore fulfilling requirements R12, R14, R15 and R16.

It has been shown that all requirements can be fulfilled by the proposed architecture whose components are based on well-known and proven technologies used in Grid and Cloud computing environments nowadays. In addition, new tools have been developed to ensure VM management and data transfer in distributed environments, pushing the border forward to a flexible and secure computing landscape beyond today's Grid and Cloud architectures.

4 Implementation

"All parts should go together without forcing. You must remember that the parts you are reassembling were disassembled by you. Therefore, if you can't get them together again, there must be a reason.
By all means, do not use a hammer."

IBM Maintenance Manual, 1925

4.1 Introduction

This chapter provides details regarding the implementation of the different components. First, basic functionality of the ICS will be shown. Beginning with the frontend, details of the ICSd's implementation will be provided. Thereafter, the process of creating VM images on the image pool will be shown and the process of deploying images will be discussed. The use of multi-layered images will be described briefly before focusing on the inter-site support of the ICS. In that area, the algorithm used to decide whether or not to transfer images between different locations will be presented.

The following section provides details about the implementation of the Dispatch Daemon (dispatchd), which is responsible for scheduling computational jobs among different sites like Desktop Grids or even Cloud sites. The last section of this chapter focuses on the Request Daemon (reqd) used to dynamically acquire additional resources from a site's backend depending on the cur-

rent load profile and to shut down unused resources in order to save energy and minimize computational costs.

4.2 Image Creation Station (ICS)

The ICS is the basic component that users use to create and manage images. It must be ensured that the ICS can be integrated into existing environments without much effort. Because it was developed in the D-Grid, it provides techniques to fit easily into existing Grid environments using a x.509 certificate infrastructure for authorization purposes. Nevertheless, the ICS has been designed modularly and can theoretically be used in other environments. If a site's administrator wishes to use the ICS in her environment, she can use an existing backend or provide her own backend that matches the given virtualization technology to get started. If needed, a frontend can be developed according to the site's needs using the ICS' RMI interface. One frontend is provided with the ICS, which is tailored for environments using x.509 certificates such as academic Grid environments or Amazon EC2.

4.2.1 Frontend

The frontend exposes the functionality of the ICS to the public. Besides authorization, the frontend does not hold any business logic necessary for creating and managing images but uses the RMI interface provided by the ICSd core component as described in Section 3.2.1.1.1.

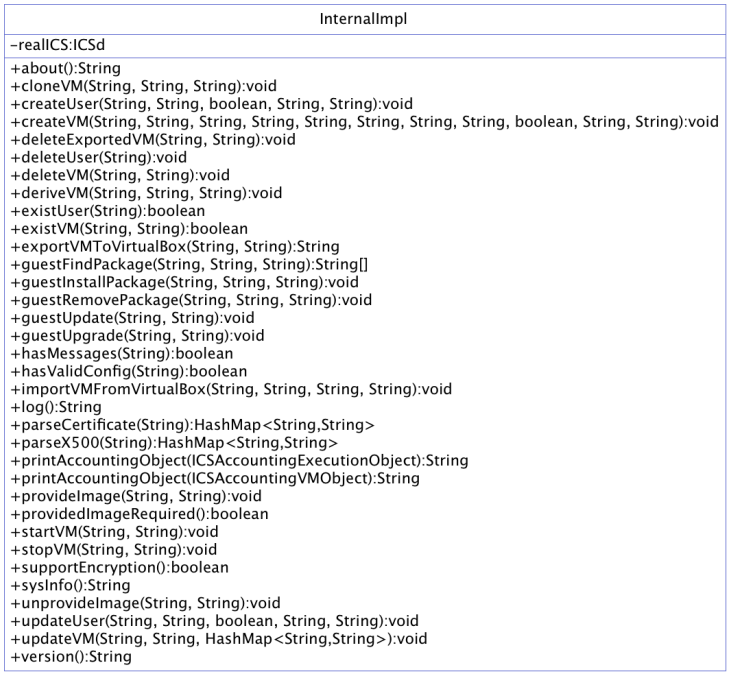


Figure 4.1: The ICS provides a RMI interface that can be used by frontends. This interface exposes a couple of methods to use the functionality of the ICS.

Figure 4.1 shows the methods provided by this RMI interface. This interface is made available either via a new RMI registry, which is started by the ICS during the startup process itself, or by registering the interface at an already

Listing 4.1: Configuration of Tomcat connectors for the ICS web frontend.

```
<Connector port="443" maxHttpHeaderSize="8192" maxThreads="150"
    minSpareThreads="25" maxSpareThreads="75"
    enableLookups="false" disableUploadTimeout="true"
    acceptCount="100" scheme="https" secure="true"
    clientAuth="true" sslProtocol="TLS" />

<Connector port="80" maxHttpHeaderSize="8192"
    maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
    enableLookups="false" redirectPort="8443" acceptCount="100"
    connectionTimeout="20000" disableUploadTimeout="true" />
```

existing RMI registry. The frontend can connect to this registry and use the provided methods. It is worth noting that the interface does not provide any authentication methods.[54] The frontend itself must ensure that only authenticated users can access the provided functionality. On the other hand, every method that deals with user-specific actions requires a unique string which is mapped to a user object in the ICS. Based on that string, method calls are assigned to the particular users. Therefore, the RMI registry should be kept secure, e.g. by using SSL encryption and authentication. In infrastructures like the D-Grid which already use x.509 certificates, the distinguished name of the users' certificates can be used. Certificates can also be used to authenticate users, but this functionality must be provided by the frontend itself.

The use of a RMI interface allows for easy provisioning of different frontends. For example, a provider using the ICS to provide VM management functionality to its customers might build a dedicated application which can be installed on the customers' computers and connects securely to the ICS instance. Another possibility is to use a website to provide functionality to the users without needing to install any additional software. The latter has been implemented as a web application for the Apache Tomcat web service container. It is used in the D-Grid environment to provide German Grid users the ability to manage their VMs themselves. Since every Grid user's Distinguished Name (DN) stored in the certificate is unique, she can be identified on every ICS instance running in the D-Grid. This is a basic requirement to enable inter-site VM support between several ICS instances running on different Grid sites.

Before focusing on the implementation details of this particular ICS frontend, the necessary Tomcat configuration is presented. An administrator planning to provide this frontend must put an authentication mechanism in place allowing only Grid users to access the web application. The Tomcat configuration can easily be configured to provide such an authentication mechanism.

The Tomcat server hosting the web frontend must provide at least a connector for a SSL-secured connection. Optionally, it can also provide a connector

[54] The RMI interface provides a method to parse x.509 certificates which can be used by the frontends in the authentication process but this method does not check the validity of the certificates.

Listing 4.2: The web application performs several checks before functionality is delivered, such as determining if the user's browser presents a valid certificate. Functionality depends on the permissions granted to the user by the ICS administrator.

```
<%@ page language="java" contentType="text/html; charset=ISO
    -8859-1" pageEncoding="ISO-8859-1"%>
<%@ page import = "de.fb12.ics.rmi.Internal" %>
[...]
<%@ include file="etc/config.jsp" %>
<%
String url = "rmi://" + ICSRMIHost + ":" + ICSRMIPort + "/" +
    ICSRMIServiceName;
[...]
Internal rm = (Internal) Naming.lookup(url);

if (request.isSecure()) {
    Object o = request.getAttribute("javax.servlet.request.
        X509Certificate");
    X509Certificate certs[] = (X509Certificate[])o;
    X509Certificate cert = certs[0];
    HashMap<String, String> userCert = rm.parseCertificate(cert
        .getSubjectDN().getName());
    String DN = userCert.get("dn");
    [...]
    exist = rm.existUser(DN);
    if (exist)
        hasValidConfig = rm.hasValidConfig(DN);
    if (hasValidConfig) {
        isSuperAdmin = rm.isSuperAdmin(DN);
        isImageProvider = rm.isImageProvider(DN);
    }
%>
```

for a normal, unencrypted HyperText Transfer Protocol (HTTP) connection. An excerpt of the Tomcat configuration file `server.xml` is shown in Listing 4.1. The first connector provides an SSL-encrypted HTTP connection on Port 443 which is the standard port for the HTTPS protocol. In contrast to the standard configuration example, `clientAuth` must be set to `true`. This rejects connections from browsers that do not present a valid x.509 certificate. Therefore, the certificate must be imported to each user's browser. Although specifying a second connector that accepts unencrypted connections on Port 80 (which is the standard port for HTTP), the user is forced to use the first connector to perform VM management operations; thus, the web application checks if the incoming connection is encrypted. Nevertheless, providing a second connector on Port 80 allows the use of an SSH applet from within the web interface. That allows users to log in directly via SSH to their VMs even if no SSH client is installed on their workstation (e.g., on a standard Windows installation).

To check a certificate's validity, Tomcat needs the public certificate(s) from the Certificate Authority (CA) that issued the user's certificate. These certificates must be added to a Java trust store. Java provides a keytool which can be used to create a trust store and to add the public certificates in order to facilitate the validation of the users' certificates.[55]

[55] The documentation on the ICS website provides a chapter that deals with the correct setup of a Tomcat server.

Listing 4.3: The web application uses several methods provided by the ICS RMI interface when the user creates a new VM image.

```
1  if (request.getParameter("button").equals("Derive Image")) {
2      String vHostname = request.getParameter("hostname");
3      String deriveFrom = request.getParameter("deriveFrom");
4      try {
5          rm.deriveVM(DN, deriveFrom, vHostname);
6          internalMessages += "Virtual machine \"" + vHostname +
               "\" successfully created.";
7      } catch (Exception ice) { [...] }
8  }
9  else {
10     String vVMType = "xen";
11     String vHostname = request.getParameter("hostname");
12     String vArch = request.getParameter("architecture");
13     String vOS = "linux";
14     String vDist = request.getParameter("distribution");
15     String vFS = "ext3";
16     String vFSSize = request.getParameter("sizeDropDown");
17     boolean vSwap = false;
18     String vSwapSize = "0";
19     String description = request.getParameter("description");
20     try {
21         rm.createVM(DN, vVMType, vHostname, vArch, vOS, vDist,
               vFS, vFSSize, vSwap, vSwapSize, description);
22         internalMessages += "Virtual Machine \"" + vHostname +
               "\" successfully created.";
23     } catch (Exception ice) { [...] }
24 }
```

Listing 4.2 shows a code snippet executed by the web application when the user connects to the ICS web frontend. After loading the configuration file of the web application (line 4), the connection to the ICS RMI registry is established (line 6 and 8). Tomcat checks if the connection is encrypted before delivering any functionality (line 10). If the connection is insecure, the user is redirected to the web pages via an encrypted connection.

Next, the user's certificate provided by the browser is parsed (line 11 et seq.). Tomcat extracts the DN, which will be used to identify the user for all subsequent actions. Tomcat must check if the user already has a valid configuration (line 19). Before being able to create VM images, a user must specify her email address as well as her public SSH key.[56] In addition, the web application checks if the user has additional attributes stored in the ICS database. If she is "super admin," the web application provides additional functionality to manage the users of the local ICS while only users marked as "image providers" are able to provide their images to other users.

To demonstrate the utilization of the RMI interface used by the web application, the image creation process is shown in Figure 4.3. As described earlier, there are different ways for users to create a new VM image: They can derive the new image from an existing one provided by another user or they can start from scratch by creating a new basic image to install their software. A screenshot of the web page providing the frontend is shown in Figure 3.6 on page 100.

[56] For more information regarding the user's configuration see Section 3.2.1.1.1 on page 98.

The if-condition in line 1 tests if the user would like to derive or to create a new image. If a new image should be derived, a new hostname must be specified (line 2) as well as the source image (line 3). Thereafter, the application calls the appropriate method provided by the RMI interface (line 5). Besides the arguments specified above, the user's DN is also used to identify the owner of the new image.

When creating a new image from scratch, the method call is more complex due to the fact that more information is needed characterizing the image to be created. Although the method provided by the RMI interface can handle numerous attributes, the web application only provides a few of them to circumvent confusion and provide an interface which can be used easily even by inexperienced users. In the context of the D-Grid, several attributes are fixed in order to ensure that the VM image can be executed in this environment e.g., Linux must be provided as OS (line 13). The type of VMs depends on the virtualization solution used and is fixed to Xen in this example (line 10). Moreover, swap space is deactivated during execution of the VMs on the ICS (lines 17 and 18). Nevertheless, swap space can be provided during job execution when VMs are executed on the compute nodes. The hostname, the Linux distribution, the architecture, the image size and the description of the VMs involved are extracted from the data provided by the user via the fields of the web page. In addition to the user's DN, these settings are provided to the `createVM()` method of the ICSd.

Aside from ensuring reliable user authentication, no business logic is implemented in the frontend. The ICSd checks data provided to the ICS methods and throws exceptions if data is invalid or an error occurred.

4.2.2 ICSd

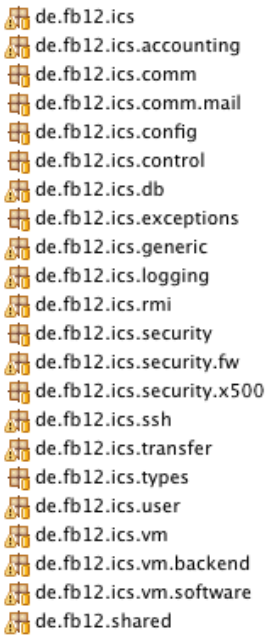


Figure 4.2: The ICS source code is divided into several packages.

The ICSd is the central component that contains the business logic. All dynamic information such as users' settings, the state of VMs, and history and log entries that can be used for accounting and billing purposes are stored in a local SQLite database. ICSd exposes its functionality via RMI interfaces, as described earlier, which can be used by the web interface, a Grid service or any other kind of frontend.

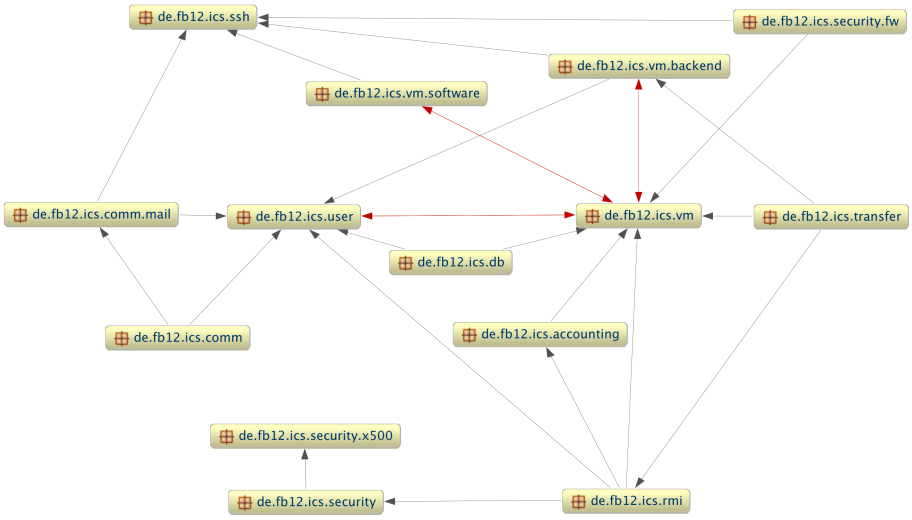


Figure 4.3: Interdependencies of the software packages found in the ICS project.

The ICSd is written entirely in Java and can run on several platforms supported by Oracle's Java Runtime Environment (JRE). Figure 4.2 shows the different packages into which the source code is divided. The interdependencies of the main packages are shown in Figure 4.3.

Although the ICSd can operate on any platform, the creation and management of VMs is more limited. Every action related to the creation, management or modification of VMs is performed on the image pool, which must provide Xen virtualization technology. The ICSd connects via SSH to this particular host to create, boot and shut down VMs as needed.

When a user deploys one of her VMs to the cluster, the VM will be shut down on the image pool and copied to the deploy host. As described in Section 3.2.1.1.4, there are different ways to transfer an image to the secured cluster network. From the deploy host, VM images are distributed to the nodes of the local cluster network [197].

ICSd, the image pool and the deploy host can reside on the same physical hardware, but for security reasons, it is recommended to install the image pool on a dedicated physical computer residing in a DMZ in order to secure the site's infrastructure. There are additional security mechanisms in place to prevent unauthorized utilization of the computing resources. First, every user is identifiable by her certificate, which is also needed to access the ICS. To log in to her VMs, a SSH key is needed which prevents users from abusing a VM. Second, every VM can be secured by a set of (external) firewall rules [196], which cannot be modified by the owner of the given VM. These firewall rules

Listing 4.4: The method used to connect to remote resources. For authentication on remote hosts, each SSH connection provides the ICS' private key.

```
1  protected boolean connect() {
2      this.thisConnection=new Connection(this.rHost, this.rPort);
3      try {
4          this.thisConnection.connect();
5          if (this.thisConnection.authenticateWithPublicKey(this.
               user, this.sshPK, null)) {
6              this.thisSession = this.thisConnection.openSession
                   ();
7              this.connected = true;
8          }
9          else {
10             this.log.info("ERROR: Could not authenticate with
                   public key.");
11             this.close();
12         }
13     }
14     catch (Exception e) {
15         this.connected = false;
16         this.close();
17     }
18
19     return this.connected;
20 }
```

prevent network traffic to other VMs when booted on the ICS image pool but allow access to the external sites necessary for software installation.

SSH Connections

Every action on the image pool and the deploy host is performed using an SSH connection to the remote machines. The ICSd uses the Trilead Java SSH2 library.[57] Based on the library, SSH connections and management functionality have been implemented to easily allow access to remote locations from the ICSd's classes.

Listing 4.4 displays the method used to connect to a remote location. The remote host (`rHost`) and the port used to connect (`rPort`) are specified in the constructor of each SSHConnection object. In line 5, the name of the user, which is used to perform the remote operations on the remote host (usually the privileged user root), and the appropriate private SSH key are used for authentication on the remote host. The method which implements the authentication process is provided by the Trilead library.

Once connected, each SSHConnection object provides a method for executing commands on the remote host. This method is displayed in Listing 4.5.

The method awaits the command to be executed remotely and will return an array of strings, which contain both the standard and the error output stream of the executed command. To ensure that no old output is returned, the SSHConnection object reinitializes the array at the beginning (lines 3 and 4). Before executing the command, it checks if the ICSd is connected to the remote host (line 6 et seq.). If the connection is disconnected, a reconnect will be per-

[57] `http://www.trilead.com/SSH_Library/`

Listing 4.5: The key functionality of a SSHConnection object is to execute commands on the remote host. Besides simply executing the command, additional information regarding this command is stored in the object e.g., needed time is stored to provide accounting and billing functionality.

```
public String[] exec(String newCmd) {
    this.cmd = newCmd;
    for (int i = 0; i < 2; i++)
        this.output[i] = "";

    if (!this.isConnected()) {
        if (!this.reconnect()) {
            this.output[1] = "Could not connect to remote host:
                 " + this.user + "@" + this.rHost + ":" + this.
                rPort;
            return this.output;
        }
    }

    Date endDate = null;
    Date startDate = new Date();
    try {
        this.thisSession = this.thisConnection.openSession();
        this.thisSession.execCommand(this.cmd);
        this.thisSession.waitForCondition(32, (long) 0);
        endDate = new Date();
        this.exitCode = this.thisSession.getExitStatus();
        this.output[0] += this.getStreamAsString(this.
            thisSession.getStdout());
        this.output[1] += this.getStreamAsString(this.
            thisSession.getStderr());
        this.thisSession.close();
    }
    catch (IOException e) {
        endDate = new Date();
        this.output[1] += "ERROR: Could not execute \"" + this.
            cmd + "\" on " + this.user + "@" + this.rHost + ":"
             + this.rPort + "\n";
        this.output[1] += "IOException: " + DataHandler.
            exceptionToString(e);
    }
    catch (NullPointerException e) {
        endDate = new Date();
        this.output[1] += "ERROR: Could not execute \"" + this.
            cmd + "\" on " + this.user + "@" + this.rHost + ":"
             + this.rPort + "\n";
        this.output[1] += "NullPointerException: " +
            DataHandler.exceptionToString(e);
    }

    this.runTime = ((float) (endDate.getTime() - startDate.
        getTime())) / 1000;
    this.aggregatedRunTime += this.runTime;
    return output;
}
```

Listing 4.6: The SSHManager stores SSHConnections created earlier. Whenever an SSH connection to a remote host is needed, existing connections will be reused. To provide dedicated connections for particular tasks, each connection can be tagged allowing several connections to one host at the same time.

```
1  public SSHConnection get(String remoteHost) {
2      return this.get(remoteHost, null);
3  }
4
5  public SSHConnection get(String remoteHost, String tag) {
6      String key = remoteHost;
7      if (null != tag)
8          key += "#" + tag;
9
10     if (activeConnections.containsKey(key))
11         return activeConnections.get(key);
12     else
13         return this.newConnection(remoteHost, defaultUser,
               sshPrivateKeyFile, tag);
14 }
15
16 private SSHConnection newConnection(String remoteHost, String
       userName, File sshPrivateKey) {
17     return this.newConnection(remoteHost, userName,
           sshPrivateKey, null);
18 }
19
20 private SSHConnection newConnection(String remoteHost, String
       userName, File sshPrivateKey, String tag) {
21     SSHConnection conn = new SSHConnection(remoteHost, 22,
           userName, sshPrivateKey);
22     String key = remoteHost;
23     if (tag != null)
24         key += "#" + tag;
25     this.activeConnections.put(key, conn);
26     return conn;
27 }
```

formed before a new session is opened for the command execution (line 16). Sessions are provided by the Trilead library to separate different executions using the same SSH connection. Thereafter, the command is executed (line 17). Directly after receiving the exit code of the command (line 18) the SSHConnection object reads and stores the output streams in the string array (lines 21 and 22). Finally, the session is closed (line 23). The runtime of the command is calculated using timestamps created directly before (line 14) and after (line 19) job execution, in line 36, the time is stored in the SSHConnection object (line 37) and the output array is returned.

To avoid having to create connections, every time a command is executed, created connections are cached and will be managed by the ICS' SSHManager.

The methods used to create and deliver SSH connections to other components of ICSd are displayed in Listing 4.6. Any component needing an SSH connection to a remote host calls the SSHManager's `get()` method, which awaits only the remote host as an argument. If desired, a tag can be specified to reserve a particular SSH connection for special purposes e.g., the watchdog component, described in Section 4.2.3.8, which monitors the VMs on the image pool uses a dedicated SSH connection to avoid interferences with other commands.

This method creates the key that identifies the particular SSHConnection object from the provided arguments. The object searches for the key in a list containing all existing SSH connections (line 10). If the particular connection exists, it will be returned (line 11), otherwise a method is called which creates (and returns) a new connection (line 13). This method creates a new SSHConnection object (line 21) with the settings either provided by the methods's caller or the configuration file. A key is created from the remote host address and the tag (lines 22 – 24), and the connection is stored in the local list (line 25). The newly created connection is then returned to the caller (line 26).

The proposed design is suitable for minimizing the number of open SSH connections to a remote host and allows users to reuse connections. By avoiding creating new connections every time a command is executed, the time required for the connection and authentication process is saved. In addition, this solution also allows for the creation of connections that are used exclusively for particular tasks by specifying custom tags. This allows connections to the same host to be separated, thus avoiding side effects when executing tasks concurrently.

4.2.3 Creating and Managing VM Images

As described earlier, the ICSd provides several mechanisms for creating VM images. Images will be created and started on the image pool, which must provide a virtualization layer supported by the ICS. This section provides an overview of the specific mechanisms found within the ICSd.

Due to the performance of para-virtualization technology, Xen had been chosen as the preferred virtualization technology. Comparable to KVM, Xen uses hardware virtualization support if available but can, in contrast to KVM, also run on older hardware that does not provide hardware virtualization support.

4.2.3.1 Create Images from Scratch

The first method of creating images is to create them from scratch. That means that only a minimal amount of software will be installed to provide the user with a basic VM. By using this method, software packages will be fetched directly from the provider of the OS distribution e.g., from Debian package servers. Ensuring the use of up-to-date software, this method can be time-consuming because every software package about to be installed has to be downloaded via the network. To speed things up, once downloaded packages are cached on the image pool, they can be reused.

Listing 4.7 shows the method used to create VM images either from scratch or from a golden image. The method requires a VM object containing all of the image's settings. These settings are necessary for the image creation process because all backend methods use information from the VM object for command execution. Several checks are performed before the image creation

Listing 4.7: The method of the ICS backend which is used to create images either from scratch or from a golden image.

```
public VM createDiskImage(VM vm) throws
    ICSOperationFailedException {
    vm.set("type", "xen");
    User owner = vm.getOwner();
    [...] // check settings
    String installMethod;
    String installSource;
    if (this.goldenImageExists(dist, arch)) {
        installMethod = "tar";
        installSource = getGoldenImageName(dist, arch);
    }
    else    installMethod = "debootstrap";
    [...]
    String cmd = "xen-create-image";
    cmd += " --force";
    cmd += " --hostname " + vm.get("hostname");
    cmd += " --size " + vm.get("fs_size") + "Mb";
    cmd += " --fs " + fsType;
    cmd += " --" + swapSettings;
    cmd += " --dist " + vm.get("os_version");
    cmd += " --mirror " + super.backendConfig.getProperty("
        MIRROR_DEBIAN");
    cmd += " --arch " + vm.get("arch");
    cmd += " --dir " + super.backendConfig.getProperty("
        XEN_VM_IMAGE_POOL_LOCATION") + "/" + owner.get("id");
    cmd += " --image=sparse";
    cmd += " --install-method=" + installMethod;
    if (installSource.trim().length() > 0)
        cmd += " --install-source=" + installSource;
    cmd = cmd.trim();
    cmd += " --dhcp";
    cmd += " --role=ics.role";
    cmd += " --role-args=" + this.skeletonDir;

    SSHConnection conn =super.host.sshManager.get(this.vmPool);
    conn.exec(cmd);
    int ec = conn.getExitCode();

    new VMFile(vm, this.getFSPath(vm) + "/disk.img");
    vm.set("creationtime", conn.getAggregatedRunTime());
    this.postprocess(vm);
    return vm;
}
```

process is started. Besides installing new images from scratch, this installation method can also create new VM images from a golden image. To choose the installation method, the ICS backend checks for the existence of a golden image that matches the chosen distribution and architecture of the desired VM image in line 7. On modern CPUs supporting 64 bit, both 32 bit and 64 bit architectures are supported. If a golden image exists, it will automatically be used as installation source, otherwise the installation method is set to `debootstrap`, a binary which uses downloaded packages from the provider as an installation source to set up a new VM image.

The command executed on the image pool is generated using the specified settings. When using Xen, the image creation process relies on the script `xen-create-image` provided by the `xen-tools` package. It provides an easy way to set up VM images and is highly configurable by a number of command line attributes. The command is build in lines 13 – 30 of the listing and executed in line 33.

Since many of the arguments are self-explanatory, only the `image` argument in line 23 should be explained here in detail. Xen supports two kinds of image files. When using full images, the space of the image file is assigned in the moment of image creation. If the user decides to create a disk image with a size of 2048 MB, this space is immediately assigned to the image file, which consumes 2048 MB on the file system. Since the image file must be created at the beginning, this can take some time depending on its size. In contrast, sparse images do not consume the whole 2048 MB from the beginning, though they are also dubbed growing images. Only the space consumed by data within the image's file system is allocated on the local hard disk. Since only a small image must be created at the beginning, the image creation process is rather quick. On the other hand, file creation operations within the disk image might be slower if new space must be allocated during runtime. The ICS uses sparse images in order to save space on the image pool's hard disk.

Once created, the disk image is registered as a VMFile object in the ICSd. This is necessary for the multi-site support which is described later.

The SSHConnection object used to execute commands also tracks the time needed for execution on the remote hosts. The time is stored in the VM object and can be used to account resource usage and bill the VM's owner.

At this point, a VM image has been created and contains a basic OS and some site-specific configuration settings performed by the role script.[58] If a golden image was used to create the new image, changes are also available that have been performed on the golden image by the ICS administrator. Before the image can be used with the ICS, additional data must be transferred to the image. Therefore, the `postprocess()` method is called, which awaits the VM object of the newly created image.

[58] The role script will be explained in detail in Section 4.2.3.1.1.

Listing 4.8: During post-processing, changes are applied to the VM image necessary for operation with the ICS.

```
1  public boolean postprocess(VM vm) throws
       ICSOperationFailedException {
2      [...]
3
4      String sshKey = ("" +  vm.getOwner().get("sshkey")).replace
           ("\n", "").trim();
5
6      String tmpMountPoint = super.backendConfig.getProperty("TMP
           ") + "/ICS_" + vm.getUniqueVMName() + "/";
7      cmd = "rm -rf " + tmpMountPoint + "; mkdir -p " +
           tmpMountPoint;
8      [...]
9
10     cmd = "mount -t " + vm.get("fs_type") + " -o loop " + vm.
           getFile().getFileName() + " " + tmpMountPoint;
11     [...]
12
13     cmd = "mkdir -p " + tmpMountPoint + "/root/.ssh";
14     [...]
15
16     cmd = "echo \"" + sshKey + "\" >> " + tmpMountPoint + "/
           root/.ssh/authorized_keys";
17     [...]
18
19     cmd = "echo \"" + super.host.getPublicSSHKey() + "\" >> " +
            tmpMountPoint + "/root/.ssh/authorized_keys";
20     [...]
21
22     cmd = "sed -i.ics '/^1:.*:respawn:/s/tty1/hvc0/' " +
           tmpMountPoint + "/etc/inittab";
23     [...]
24
25     cmd = "umount " + tmpMountPoint;
26     [...]
27
28     float vmTime = Float.parseFloat("" + vm.get("creationtime")
           ) + conn.getAggregatedRunTime();
29     vm.set("creationtime", vmTime);
30     [...]
31 }
```

To keep Listing 4.8 short, only the commands are shown which are executed to modify the image file. First, the user's public SSH key is read from the owner's user object in line 4. This key must be specified by the user in her preferences e.g., on the preferences screen of the web application. It is used to authenticate the user when she logs into the particular VM via SSH.

Because of the fact that some modifications within the image must be performed, the ICSd must be able to access, store and modify data within its file system. First, the ICSd creates a temporary mount point (line 6 – 7). If this mount point already exists, it will be deleted before the VM image is mounted (line 10). To enable the owner to log into her VM, the ICSd creates a directory holding the SSH configuration (called `.ssh`) for the privileged user (line 13) and the image owner's public SSH key is stored in the file `authorized_keys` (line 16). The SSH server reads this file during the authentication process when users try to connect to the VM. In addition to the owners's public key, the public key of the ICS is also stored to that file (line 19). This allows software installation later via the commands provided by the RMI interface. It must be mentioned that both public SSH keys are appended to the file `authorized_keys` and do not overwrite an already existing file. That allows previous users of this image to log in with their (private) SSH key e.g., if the image has been derived from an existing one. This issue is discussed in Section 4.2.3.3

When using Xen on Debian, the `inittab` file must be modified in order to provide the user with a console (line 22). This update is made just before the VM image is unmounted (line 25) from the temporary mount point. Again, the time needed to perform the operations within this method is added to the time needed for image creation itself to provide an accounting and billing mechanism.

4.2.3.1.1 Role Script The role script mentioned above is a script that is executed by the `xen-create-image` command. It can be used to install additional packages to a newly created VM image or to perform other desired actions. The ICS provides a role script which copies the contents of a given directory, the skeleton directory, to each created VM and ensures that a SSH server is installed.

Listing 4.9 shows the role script which is provided by the ICS. One may notice that this script awaits two arguments, the mount point of the VM disk image (line 4) and the location of the skeleton directory (line 5), in contrast to the command in line 35 of Listing 4.7, in which only one argument is passed to the script. This is due to the fact that `xen-create-image` passes the mount point of the image automatically as the first argument to every role script.

First, the script calls the script `common.sh` which is provided by the Xen tools, too. This scripts provides commands to easily install Debian packages

Listing 4.9: The role script provided by the ICS which copies the contents of the skeleton directory to the image and installs an SSH server.

```
1  #!/bin/sh
2  # Role script used by the ICS when new images are created.
3  # Copyright (c) 2010 by Niels Fallenbeck, University of Marburg
4  MOUNTPOINT=$1
5  SKEL_DIR=$2
6
7  #
8  #  Source our common functions - this will let us install a
       Debian package.
9  #
10 if [ -e /usr/lib/xen-tools/common.sh ]; then
11     . /usr/lib/xen-tools/common.sh
12 else
13     echo "Installation problem: /usr/lib/xen-tools/common.sh
           not found or not executable"
14 fi
15
16 if [ -d ${SKEL_DIR} ]; then
17     # Copy the skeleton-dir to the new virtual machine
18     cp -r ${SKEL_DIR}/* ${MOUNTPOINT}
19 fi
20
21 #  Install the packages
22 installDebianPackage ${MOUNTPOINT} ssh
23 installDebianPackage ${MOUNTPOINT} udev
```

Listing 4.10: Methods to decide whether or not a golden image exists which can be used as installation source.

```
1  public String getGoldenImageName(String distName, String arch)
       {
2      return super.backendConfig.getProperty("
           XEN_VM_GOLDEN_IMAGE_POOL_LOCATION") + "/golden_" +
           distName + "_" + arch + ".tar";
3  }
4
5  public boolean goldenImageExists(String distName, String arch)
       {
6      String goldenFileName = getGoldenImageName(distName, arch);
7
8      SSHConnection conn = host.sshManager.get(this.vmPool);
9      String cmd = "ls " + goldenFileName;
10     String[] output = conn.exec(cmd);
11     return output[0].trim().length() > 0;
12 }
```

to the newly created image. In this script, SSH (line 22) and `udev` (line 23) are installed to make sure that the VM can be accessed via SSH once it has been started on the image pool. The role script is also responsible for copying the contents of the skeleton directory to the image (line 18).

4.2.3.2 Create Images from a Golden Image

Creating a new VM image from a golden image instead of downloading all software packages from the distribution provider relies on the same methods shown in Listings 4.7 and 4.8. As described above, the given method automatically chooses a golden image as installation source if one exists.

Whenever the method `goldenImageExists()` is called, the ICSd checks whether or not an appropriate golden image file exists. This allows adminis-

Listing 4.11: This methods are responsible for derive a new VM image from an existing one.

```
1  public String getFSPath(VM vm) {
2      User owner = vm.getOwner();
3      return super.backendConfig.getProperty("
           XEN_VM_IMAGE_POOL_LOCATION") + "/" + owner.get("id") +
           "/domains/" + vm.get("hostname") + "/";
4  }
5
6  public boolean derive(VM originalVM, VM newVM) throws
       ICSOperationFailedException {
7      SSHConnection conn =super.host.sshManager.get(this.vmPool);
8      String cmd = "";
9      String errorMsg = "";
10     int ec = 0;
11
12     String srcDir = this.getFSPath(originalVM);
13     String targetDir = this.getFSPath(newVM);
14     cmd = "cp -r " + srcDir + " " + targetDir;
15     conn.exec(cmd);
16     ec += conn.getExitCode();
17
18     new VMFile(newVM, this.getFSPath(newVM) + "/disk.img");
19
20     float vmTime = Float.parseFloat("" + newVM.get("
           creationtime"));
21     newVM.set("creationtime", vmTime + conn.
           getAggregatedRunTime());
22
23     return this.postprocess(newVM);
24 }
```

trators to provide a new golden image or to delete an existing golden image (e.g., if problems occur with software provided by the image) during runtime without needing to restart the ICS. Listing 4.10 shows the methods used to perform the necessary checks. Several golden images can be exist, one for each combination of the OS distribution and its architecture. The location of the golden images is specified in the ICS configuration file. The resulting absolute path to the file in the file system of the image pool is returned by the function `getGoldenImageName()`. A standard `ls` command is performed (line 9) and its output is parsed to decide if a golden image exists (line 11).

4.2.3.3 Derive Images from Existing Ones

If users do not wish or are not allowed to create basic VM images, new images can be derived from preconfigured existing images. As stated earlier, that is a very comfortable way for software providers to create and configure VM images that contain a particular software product and offer them to other users. Moreover, this method can be used in scenarios in which inexperienced users need to create virtual execution environments. Shifting the administrative burden from inexperienced users to other users with more knowledge of system administration increases the security within the system. That is the reason why an ICS administrator might even force users to derive images from existing ones and reserve the right to create basic images..

The methods used to derive images are showed in Listing 4.11. While the first method is used to determine where an existing VM resides on the local hard disk, which depends on the ICS backend's configuration, the VM's owner and its hostname, the second method derives the new image by simply copying it to the new location (line 14). After registering the new image file as VMFile object in the ICSd (line 18), the time needed for derivation is stored within the VM object (line 21) and the post processing method (already presented in Listing 4.8) is called for the new image (line 23).

As described earlier, during post processing, existing public SSH keys are not overwritten by new ones, allowing the original owner to log into derived images (see Listing 4.8, lines 16 and 19). What looks like a security flaw is also an advantage when problems occur. If users run into problems caused by the software installed by the image provider, the provider herself can log into the faulty machine, figure out the problem and even fix it. The potential for misuse is small because of the fact that image providers must be authorized by the local ICS administrator. This minimizes the problem of malicious users providing VM images to gain access to the other users' data. Users that derive images from existing ones usually have a trust-based or contractual relationship to the image provider. If this behavior is not desired by the new owners, they can manually remove the image provider's public SSH key from the `authorized_keys` file.

4.2.3.4 Import Images

Another way to create a new VM image is to import a locally existing VM e.g., from a desktop virtualization software such as VirtualBox. This is helpful in environments in which VMs already exist and should be used in the Cloud or the Grid. Therefore, the ICS must be able to deal with external VMs and must provide an importation method that can deal with a commonly available and widely accepted file format. The vmdk format was chosen because it is spreading even in commercial virtualization products and because of its use in the Open Virtualization Format (OVF). This ensures compatibility with VMware products as well as with VirtualBox, qemu and others.

Since Xen, which does not support the vmdk file format, is used as VMM on the image pool, both importation and exportation methods are quite complex. For the conversion itself, software tools provided by VirtualBox and qemu must be used.

Listings 4.12 and 4.13 display the method that is used to convert a vmdk disk file to a raw disk file which can be used with Xen. Regardless of how a vmdk file has been uploaded, it is first copied to a temporary directory (line 10) that has been created for this importation and will deleted afterwards (line 8). The uploaded file is cloned using the `VBoxManage` tool provided by VirtualBox (line 12 and 13). The clone operation is used to uncompress the imported vmdk

Listing 4.12: This method is used to convert an uploaded image in vmdk format to the raw format supported by Xen. If possible, an additional file system check is performed to ensure the successful importation of the file.

```
1  public String importImage(String vmdkAbsFileName) throws
       ICSOperationFailedException {
2      String cmd;
3      File vFile = new File(vmdkAbsFileName);
4      String vFileName = vFile.getName();
5      SSHConnection conn = super.host.sshManager.get(super.
           backendConfig.getProperty("VIRTUALBOX_HOST"));
6      String workingDir = super.backendConfig.getProperty("
           VIRTUALBOX_DIR") + "/import/" + vFileName;
7      [...]
8      cmd = "rm -rf " + workingDir + "; mkdir -p " + workingDir;
9      [...]
10     cmd = "cp " + vmdkAbsFileName + " " + workingDir + "/step0.
           " + suffix;
11     [...]
12     cmd = super.backendConfig.getProperty("
           VIRTUALBOX_BIN_VBOXMANAGE");
13     cmd += " clonehd " + workingDir + "/step0." + suffix + " "
           + workingDir + "/step1." + suffix;
14     [...]
15     cmd  = super.backendConfig.getProperty("
           VIRTUALBOX_BIN_QEMUIMG");
16     cmd += " convert";
17     cmd += " -f " + suffix + " " + workingDir + "/step1." +
           suffix;
18     cmd += " -O raw " + workingDir + "/step2.raw";
19     [...]
20     String fsType = super.getFileSystem(conn,workingDir + "/
           step2.raw");
21     if (fsType != null) {
22         cmd = "mv " + workingDir + "/step2.raw " + workingDir +
                "/step3.img";
23         [...]
24     }
25     else {
26         [...]
27         cmd = "fdisk -lu " + workingDir + "/step2.raw";
28         conn.exec(cmd);
29         if (conn.getExitCode() != 0)
30             throw new ICSOperationFailedException("Could not
                   extract partition information from raw file: "
                   + conn.getErrorOutput());
31         [...]
32         String[] partInfoArray = conn.getOutput().split("\n");
33         String regEx = "^.*Units.*sectors.*$";
34         int mySectors = 0;
35         for (int i = 0; i < partInfoArray.length; i++)
36             if (partInfoArray[i].trim().matches(regEx)) {
37                 mySectors = i;
38                 break;
39             }
40         if (mySectors == 0)
41             throw new ICSOperationFailedException("Could not
                   parse sector information");
42         String sectorData = partInfoArray[mySectors];
43         String[] sectorArray=sectorData.trim().split("[ \t]+");
44         String sectorSize=sectorArray[sectorArray.length - 2];
```

Listing 4.13: This method is used to convert an uploaded image in vmdk format to the raw format supported by Xen. If possible, an additional file system check is performed to ensure the successful importation of the file. (cont.)

```
        regEx = "^.*step2.raw.*Linux$";
        int myPartition = 0;
        for (int i = 0; i < partInfoArray.length; i++)
            if (partInfoArray[i].trim().matches(regEx)) {
                myPartition = i;
                break;
            }
        if (myPartition == 0)
            throw new ICSOperationFailedException("Could not
                parse partition information");
        String partData = partInfoArray[myPartition];
        partData = partData.replace('*', ' ');
        String[] partDataArray = partData.trim().split("[ \t]+"
            );
        int start  = Integer.parseInt(partDataArray[1]);
        int end    = Integer.parseInt(partDataArray[2]);
        int length = end - start + 1;
        cmd  = "dd ";
        cmd += " if=" + workingDir + "/step2.raw";
        cmd += " of=" + workingDir + "/step3.img";
        cmd += " bs=" + sectorSize;
        cmd += " skip=" + start;
        cmd += " count=" + length;
        conn.exec(cmd);
        [...]
        fsType = super.getFileSystem(conn, workingDir + "/step3
            .img");
    }
    cmd = "which fsck." + fsType;
    conn.exec(cmd);
    if (conn.getExitCode() == 0) {
        cmd  = "fsck." + fsType;
        cmd += " -fy";
        cmd += " " + workingDir + "/step3.img";
        conn.exec(cmd);
        [...]
    }
    String imgFileName = super.backendConfig.getProperty("
        VIRTUALBOX_DIR") + "/import/" + vFileName.substring(0,
        vFileName.lastIndexOf('.')) + ".img";
    cmd = "mv " + workingDir + "/step3.img " + imgFileName;
    [...]
    return imgFileName;
}
```

Listing 4.14: The method used to determine the type of the file system of a given VM image.

```
1  public String getFileSystem(SSHConnection conn, String
       localPathToDiskImage) {
2      String f = (new File(localPathToDiskImage)).getName();
3      String cmd = "file " + localPathToDiskImage;
4      conn.exec(cmd);
5      [...]
6      String output = conn.getOutput();
7      if (output.matches("^.*" + f + ".*ext2.*$"))
8          return "ext2";
9      if (output.matches("^.*" + f + ".*ext3.*$"))
10         return "ext3";
11     if (output.matches("^.*" + f + ".*ext4.*$"))
12         return "ext4";
13     if (output.matches("^.*" + f + ".*XFS.*$"))
14         return "xfs";
15     if (output.matches("^.*" + f + ".*ReiserFS.*$"))
16         return "reiserfs";
17     if (output.matches("^.*" + f + ".*swap.*$"))
18         return "swap";
19     return null;
20 }
```

file in order to ensure that it can be handled by `qemu-img`. This command is used to convert the disk image from vmdk to raw (line 15 et seq.).

Once the image has been converted to a raw image, its file system type is analyzed by a function provided by the backend's superclass (line 20). This method is depicted in Listing 4.14.

This method uses the `file` command of the Linux OS to get information about a particular file (line 3). Afterwards, it uses a regular expression to extract file system information (line 7 et seq.) which is returned by the method.

If a file system could be determined using this method, the converted file contains a single partition without a partition table and can be processed directly (line 21). If the file contains a partition table (and possibly more than one file system partition), the output of `file` differs such that the presented method returns null. In that case, the partition table must be extracted from the file using `fdisk`. The command in line 27 of Listing 4.12 returns all necessary information. Its output might look like the following:

```
Disk step2.raw: 0 MB, 0 bytes
255 heads, 63 sectors/track, 0 cylinders, total 0 sectors
Units = sectors of 1 * 512 = 512 bytes
Disk identifier: 0x000b2fb8

Device      Boot Start     End      Blocks    Id  System
step2.raw1  *         63  3887729  1943833+  83  Linux
step2.raw2       3887730  4192964   152617+   5  Extended
step2.raw5       3887793  4192964   152586   82  Linux swap
```

The file `step2.raw` contains one data partition and one swap partition which is located in an extended area in the image. This output must be parsed

Listing 4.15: This method is used to create a new VM by using a previously imported disk image.

```
1  public VM createFromDiskImage(VM vm, String absFileName) throws
       ICSOperationFailedException {
2      SSHConnection conn = super.host.sshManager.get(this.vmPool)
           ;
3      String cmd = "";
4      int ec = 1;
5      User owner = vm.getOwner();
6      [...]
7
8      String targetDir = super.backendConfig.getProperty("
           XEN_VM_IMAGE_POOL_LOCATION") + "/" + owner.get("id") +
           "/domains/" + vm.get("hostname");
9      String targetFileName = targetDir + "/disk.img";
10     cmd = "mkdir -p " + targetDir;
11     cmd = "cp " + absFileName + " " + targetFileName;
12     [...]
13     new VMFile(vm, this.getFSPath(vm) + "/disk.img");
14     float vmTime = Float.parseFloat("" + vm.get("creationtime")
           ) + conn.getAggregatedRunTime();
15     vm.set("creationtime", vmTime);
16     [...]
17     this.postprocess(vm);
18     return vm;
19 }
```

to extract the partition to be imported. First, the sector size of this virtual disk is determined (lines 32 – 44). Afterwards the partitions are scanned to find a suitable Linux partition (lines 1 – 9 of Listing 4.13). If a partition is found, start and end sectors are determined by parsing this information from the `fdisk` output (lines 10 – 14). In addition, the length of the partition is calculated (line 15). This information is needed to extract the single Linux partition from the uploaded file. The `dd` utility is used to copy the partition into a new file (lines 16 – 22). Finally, the file system type is determined (line 24) using the method presented earlier.

At this point, a file containing a single data partition exists and the file system of this partition is known to the ICS. If possible, the ICSd performs a file system check to guarantee the file system's integrity. Therefore, it checks if the `fsck` tool supports the given file system type. That can easily be checked because the `fsck` tool provides different binaries for each supported file system type called `fsck.<fstype>`. If an appropriate binary file (line 28) exists, the check will be performed and found errors will be corrected automatically (lines 29 – 32). Finally, the successfully imported file is moved to the image directory and can now be assigned to a VM object (line 36).

Listing 4.15 shows the method which is used to assign an imported file to a VM. A VM object must be specified describing the new VM and the file name which is returned by the function shown in Listing 4.12. The owner is extracted from the VM object in line 5. Afterwards, the file is copied to the new destination directory in the file system which holds all of the ICS users' VM images (line 8 – 11). Moreover, the file is registered as VMFile object (line

13). Again, the time needed is stored in the VM object (line 14 – 15) before the post process method (Listing 4.8) is called to finish the image creation process.

4.2.3.5 Export Images

To prevent the data lock-in effect, an export function must be also provided to allow users to download VM images in the vmdk file format to their local infrastructure e.g., to backup a particular VM or to use it on the local system. Although the ICS has to deal again with different disk formats, exporting a VM image is much simpler than importing it.

Listing 4.16 shows the function responsible for converting a raw Xen disk image to a vmdk file which can be used with other virtualization technologies. The method awaits the VM object to which the disk image belongs. After determining the correct backend to handle the particular file (line 2) as well as the different file names needed, the ICSd creates a temporary directory and uses it to provide the exported image to the user (lines 7 and 12). It copies the original disk image to that directory (line 14) to avoid problems with the original ICS VM image if the export process fails for some reason. Then, it creates a temporary mount point (line 16) and mounts the image (line 18). The changes applied to the `inittab` during the post process after image creation or importation must be reverted (line 20) to ensure a normal operation on the user's virtualization infrastructure. After unmounting the image and removing the temporary mount point, the disk image will be converted from the raw format to the vmdk file format using the VirtualBox tool `VBoxManage` (lines 30 – 35).

The method returns the path to the exported image which then can be transferred to the user, e.g. by downloading it via the frontend.

4.2.3.6 Start Virtual Machines

After creating or importing a new VM image, users probably want to modify data or install software. Therefore, VMs must be started (and shut down afterwards) on the image pool and access to the running VMs must be provided.

The method for starting a VM on the image pool is presented in Listing 4.17. First, an IP address is obtained using the `assignAddress()` method provided by the backend's superclass (line 3). Furthermore, a port is determined which is used for the SSH connection to that machine (line 6 – 9). Next, the command is created respecting the settings in the backend's configuration e.g., the kernel and the ramdisk to be used (line 12 – 13) and the main memory assigned to VMs during startup (line 14). Optionally, the ICS administrator can specify extra options that are appended to the command (line 24 – 25). This can be used to hand over additional parameters to the Xen kernel during the boot process.

If the boot process of the VM has been successfully started (line 31), the ICS

Listing 4.16: This method exports an ICS VM image by converting it to the vmdk file format.

```
1  public String exportImage(VM originalVM) throws
       ICSOperationFailedException {
2      Backend originalBackend = Backend.getBackend("" +
           originalVM.get("type"));
3      SSHConnection conn = super.host.sshManager.get(super.
           backendConfig.getProperty("VIRTUALBOX_HOST"));
4      File originalFiles = originalBackend.localPath(originalVM);
5      File originalDiskImage = new File(originalFiles.
           getAbsolutePath() + "/disk.img");
6      [...]
7      String targetDir = this.getExportedImageDir(originalVM);
8      String tmpMountDir = targetDir + "/tmpmount";
9      String sourceFileName = targetDir + "/step0.img";
10     String targetFileName = this.getExportedImageName(
           originalVM);
11
12     cmd = "mkdir -p " + targetDir;
13     [...]
14     cmd = "cp " + originalDiskImage.getAbsolutePath() + " " +
           sourceFileName;
15     [...]
16     cmd = "mkdir -p " + tmpMountDir;
17     [...]
18     cmd = "mount -t ext3 -o loop " + sourceFileName + " " +
           tmpMountDir;
19     [...]
20     cmd = "sed -i.xen '/^1:2345:respawn/s/hvc0/tty1/' " +
           tmpMountDir + "/etc/inittab";
21     [...]
22     cmd = "umount " + tmpMountDir;
23     [...]
24     cmd = "umount -lf " + tmpMountDir;
25     [...]
26     cmd = "rm -rf " + tmpMountDir;
27     [...]
28     cmd = "rm " + targetFileName;
29     [...]
30     cmd = super.backendConfig.getProperty("
           VIRTUALBOX_BIN_VBOXMANAGE");
31     cmd += " convertfromraw";
32     cmd += " --format " + super.backendConfig.getProperty("
           VIRTUALBOX_IMAGE_FORMAT");
33     cmd += " " + sourceFileName;
34     cmd += " " + targetFileName;
35     conn.exec(cmd);
36     [...]
37     return targetFileName;
38 }
```

Listing 4.17: This method starts a user's VM on the ICS image pool and provides access to the running VM.

```
1  public boolean start(VM vm) throws ICSOperationFailedException
       {
2      // assign an address to the virtual machine
3      HashMap imageAddress = super.assignAddress(vm);
4      [...]
5
6      int lower = Integer.parseInt(super.backendConfig.
           getProperty("XEN_VM_PORT_LOWER"));
7      int upper = Integer.parseInt(super.backendConfig.
           getProperty("XEN_VM_PORT_UPPER"));
8      int port = super.getFreePort(lower, upper);
9      vm.set("port", port);
10
11     String cmd  = "xm create /dev/null";
12     cmd += " kernel='" + super.backendConfig.getProperty("
           XEN_VM_KERNEL") + "'";
13     cmd += " ramdisk='" + super.backendConfig.getProperty("
           XEN_VM_INITRD") + "'";
14     cmd += " memory='" + super.backendConfig.getProperty("
           XEN_VM_MEMORY") + "'";
15     cmd += " root='" + super.backendConfig.getProperty("
           XEN_VM_ROOT") + "1 ro'";
16     cmd += " disk='file:/" + this.getFSPath(vm) + "/disk.img,"
           + super.backendConfig.getProperty("XEN_VM_ROOTDEVICE")
           + "1,w'";
17     cmd += " name='" + vm.getUniqueVMName() + "'";
18     cmd += " vif='mac=" + vm.get("mac") + "'";
19     cmd += " dhcp='dhcp'";
20     cmd += " on_poweroff='destroy'";
21     cmd += " on_reboot='restart'";
22     cmd += " on_crash='restart'";
23
24     if ((super.backendConfig.containsKey("XEN_EXTRAS")) && (
           super.backendConfig.getProperty("XEN_EXTRAS").trim().
           length() > 0))
25         cmd += " extra='" + super.backendConfig.getProperty("
               XEN_EXTRAS") + "'";
26     [...]
27
28     SSHConnection conn = super.host.sshManager.get(this.vmPool)
           ;
29     conn.exec(cmd);
30     int ec = conn.getExitCode();
31     if (ec == 0) {
32         host.fw.activateSSHForward(vm);
33         vm.set("running", "true");
34     }
35     [...]
36     return ec == 0;
37 }
```

Listing 4.18: This method shuts down a running VM on the ICS image pool.

```
1  public boolean stop(VM vm) throws ICSOperationFailedException {
2      if (!running(vm)) return true;
3      boolean exit = false;
4      int tries = 5;
5      int destroyAfter = 3;
6      SSHConnection conn = super.host.sshManager.get(this.vmPool)
           ;
7      String cmd = "";
8      while (!exit && (tries > 0)) {
9          tries--;
10         if (destroyAfter <= 0)
11             cmd = "xm destroy " + vm.getUniqueVMName();
12         else
13             cmd = "xm shutdown -w " + vm.getUniqueVMName();
14         conn.exec(cmd);
15         if (!running(vm))    exit = true;
16         else                 destroyAfter--;
17     }
18     [...]
19     return !this.running(vm);
20 }
```

Listing 4.19: This method checks if a VM is running on the image pool.

```
1  public boolean running(VM vm) {
2      String cmd = "xm list " + vm.getUniqueVMName() + " | grep -
           i " + vm.getUniqueVMName() + " | wc -l";
3      SSHConnection conn = host.sshManager.get(this.vmPool);
4      conn.exec(cmd);
5      return conn.getOutput().trim().equals("1");
6  }
```

calls the method to add a port forwarding rule to the outer firewall, if desired. This ensures that a user can connect to her running VM. Using a random port known only to the ICS user results in increased security. SSH's standard Port 22 is well known and widely used; therefore numerous port scans have taken place targeting that port and trying to find a security flaw to acquire access to the machine. Using a randomly-chosen non-standard port would minimize the chance of successful attacks against the VM.

4.2.3.7 Shut Down Virtual Machines

After users have applied the desired changes to their machines, the VMs need to be shut down. The method providing this functionality is displayed in Listing 4.18.

Before performing any operation, the method checks if the machine is running. If a VM could not be shut down for some reason on the first try, the method tries it 5 times (line 4) altogether. After 3 failed attempts (line 5), the VM is forced to shut down.

As long as the particular VM is active and less than 5 tries were performed to shut it down (line 8), the VM is either shut down normally (line 13) or forcefully (line 11). If the VM has stopped, the loop will exit (line 15) and the method returns whether or not the VM has halted (line 19).

To determine if a VM is running on the image pool, the method displayed in Listing 4.19 has been developed. It uses the `xm` command to query the VMM to determine if a particular VM is active by searching for the unique name of a VM and counting the lines of the command's output (line 2). The output of `xm list` might appear as follows.

```
Name              ID   Mem VCPUs   State   Time(s)
Domain-0           0  1712     2  r-----  46964.8
12_matlab          6   128     1  -b----    234.2
39_mc2             7   128     1  -b----    240.1
```

The unique name of a VM is the combination of the owner's ICS internal ID and the hostname of the VM. The method specifies the VM's unique name and appends it as an argument to `xm list`. As a result, output will only be displayed if a VM with the given unique name is active. Moreover, an additional check, using the `grep` command, parses the received output to extract only the single line containing the particular VM. If a VM is active, counting the output lines will result in 1, otherwise no output will be received and the number of lines counted will be 0 (line 5).

4.2.3.8 Watchdog

The watchdog component is executed periodically on the image pool and is responsible for monitoring the running VMs. This is necessary to ensure that the ICS notices when a VM has crashed or the user shut it down via SSH and not using the frontend. In a live system, the watchdog can also be used to shut down all VMs not belonging to the ICS to avoid resource bottlenecks and having a negative impact on running VMs of customers.

Listing 4.20 shows the watchdog's main loop which is executed periodically. The Xen manager tool `xm` is used to print out all running VMs. This information is processed by another method which returns a list of the VMs' hostnames currently running on the image pool (line 5). This list is provided as argument to a function that synchronizes the ICSd's known state with the actual state of the image pool. This method is displayed in Listing 4.21.

This method uses two sets of hostnames. The first set, displayed in line 2, contains all VMs which are marked as running on the ICSd i.e., all VMs that were started by the ICSd. The second set contains all VMs which were found running by the Watchdog command and were provided as an argument (line 4).

In the first part of this method, VMs are determined which are marked as running in the ICSd but are not active anymore on the image pool. As shown in Figure 4.4, this set of VMs is given by the intersection of the set containing all VMs marked as running (`vmmRunning`) and the set of actual running VMs (`remoteRunning`) calculated in lines 7 and 8:

Listing 4.20: The watchdog monitors the VMs on the image pool. It is used to react to changes and to ensure that the ICS knows the state of its VMs.

```
public void run() {
    while (true) {
        this.conn.exec("xm list --long");
        try {
            this.host.vmManager.synchronizeDomUs(this.
                parseXmOutput(this.conn.getOutput()));
        }
        catch (Exception e) { }

        try {
            Thread.sleep(this.toWait);
        }
        catch (Exception e) {}
    }
}

public ArrayList<String> parseXmOutput (String input) {
    StringTokenizer st = new StringTokenizer(input);
    ArrayList<String> activeVMs = new ArrayList<String>();

    while (st.hasMoreTokens()) {
        String token = st.nextToken();
        if (token.equals("(name") && st.hasMoreTokens()) {
            String domUname = st.nextToken();
            activeVMs.add(domUname.substring(0, domUname.length
                () - 1));
        }
    }
    activeVMs.remove("Domain-0");
    return activeVMs;
}
```

Listing 4.21: Method to synchronize the state known to the ICSd with the actual state of the VMs on the image pool.

```
public void synchronizeDomUs(ArrayList<String> domUs) throws
    ICSOperationFailedException {
    Set<String> vmmRunning = this.displayRunningVMs().keySet();
    Set<String> remoteRunning = new HashSet<String>();
    remoteRunning.addAll(domUs);

    // marked as active but not running
    Set<String> inactiveVMs = new HashSet<String>(vmmRunning);
    inactiveVMs.removeAll(remoteRunning);

    if (inactiveVMs.size() > 0) {
        Iterator<String> vmi = inactiveVMs.iterator();
        while (vmi.hasNext()) {
            String vmHostnameToShutdown = vmi.next();
            this.shutdown(vmHostnameToShutdown);
        }
    }

    // running but unknown to manager or marked as inactive
    if (this.strategyForUnknownVMs.equals("SHUTDOWN")) {
        Set<String> unknownVMs = new HashSet<String>(
            remoteRunning);
        unknownVMs.removeAll(vmmRunning);

        Iterator<String> vmu = unknownVMs.iterator();
        while (vmu.hasNext()) {
            String vmHostnameToShutdown = vmu.next();
            this.shutdownExternalVM(vmHostnameToShutdown);
        }
    }
}
```

Figure 4.4: A running VM in the system can belong to at least one set. Either it is known to the ICS and is marked as running or it is unknown to the ICS but running on the image pool. If a VM belongs to both sets, its state is well-known to the ICS.

$$\texttt{inactiveVMs} = \texttt{vmmRunning} \cap \texttt{remoteRunning}$$

All VMs in the set `inactiveVMs` have either crashed or been shut down normally by their owners. Assigned resources such as IP addresses or ports must be freed by calling the `shutdown()` method for each of these VMs (lines 10 – 16).

In the second part of this method, VMs are detected that are unknown to the ICS. If the administrator configured the ICSd to shut down all unknown VMs, a set of VMs is created which contains all VMs running on the image pool which are unknown to the local ICS instance. As shown in Figure 4.4, this set is given by the intersection of the set containing all running VMs on the image pool and the set of all VMs marked as running (lines 20 and 21):

$$\texttt{unknownVMs} = \texttt{remoteRunning} \cap \texttt{vmmRunning}$$

Since unknown VMs should be shut down, the ICSd will send a command to `xm` to halt each of the VMs in this set (line 26). As mentioned earlier, this might be a good idea in live systems in order to prevent running VMs not managed by the ICS from negatively impacting the VMs of customers that are running concurrently on the system. Nevertheless, in shared environments, the behavior of shutting down unknown VMs might be undesired and can be deactivated.

Listing 4.22: The methods used to change the settings of the outer firewall. The first method is used to activate a port forwarding rule, the second method is used to delete a rule from the firewall system.

```
1  public boolean activateSSHForward(VM vm) throws
       ICSOperationFailedException {
2      String srcHost = host.config.get("FW_HOST_EXTERNAL");
3      String srcPort = "" + vm.get("port");
4      String tgtHost = "" + vm.get("ip");
5      String tgtPort = "22";
6      [...]
7      super.addForwarder(srcHost + ":" + srcPort, tgtHost + ":" +
           tgtPort);
8      String iptablesCmd = "iptables -v -t nat -I PREROUTING 1 -j
           DNAT -p tcp -i " + super.fwInterface + " --dport " +
           srcPort + " --to-destination " + tgtHost + ":" +
           tgtPort;
9      super.fsSSH.exec(iptablesCmd);
10
11     vm.set("external_host", srcHost);
12     vm.set("external_port", srcPort);
13     [...]
14     return super.fsSSH.getExitCode() == 0;
15 }
16
17 public boolean deactivateSSHForward(VM vm) throws
       ICSOperationFailedException {
18     String srcHost = host.config.get("FW_HOST_EXTERNAL");
19     String srcPort = "" + vm.get("port");
20     String tgtHost = "" + vm.get("ip");
21     String tgtPort = "22";
22
23     String iptablesCmd = "iptables -v -t nat -D PREROUTING -j
           DNAT -p tcp -i " + super.fwInterface+ " --dport " +
           srcPort + " --to-destination " + tgtHost + ":" +
           tgtPort;
24     super.fsSSH.exec(iptablesCmd);
25
26     vm.del("external_host");
27     vm.del("external_port");
28     [...]
29     super.delForwarder(srcHost + ":" + srcPort);
30     [...]
31     return super.fsSSH.getExitCode() == 0;
32 }
```

4.2.3.9 Modify Firewall Rules

The ability to automatically add port forwarding rules to an external firewall is crucial for operating the ICS. Since the SSH port of a running VM is chosen randomly during the VM's startup process, it is not known in advance to the system or other – potentially malicious – users.

The administrator can disable port forwarding in environments in which users can directly connect to the VMs e.g., when the IPs of the users' workstations are in the same subnet as the IPs assigned to the VMs. The ICS provides functionality to modify `iptables`-based firewalls to ensure an automated management of port forwarding rules for the VMs.

Listing 4.22 shows the methods used to add and remove port forwarding rules. Both methods await the VM object. All information necessary to set or

remove the firewall rules is assigned to the particular VM object or specified in the ICS configuration.

When the ICS starts a VM, it adds a port forwarding rule to the firewall. To connect from the outside, the externally accessible host (line 2) and the port assigned to the VM (line 3) must be known. Moreover, the IP of the running VM on the image pool (line 4) and the port on which the SSH server of the VM listens must be specified. This is the standard SSH port.

First, the forwarding rule will be added to a locally administered list of the ICS which keeps track of all active forwarding rules (line 7). Second, the backend-specific command to be executed is generated in line 8. The listed command targets an `iptables`-based firewall.[59] The command will be executed on the firewall host (line 9). Finally, values are added to the VM object specifying how the particular VM can be accessed from the outside (lines 11 and 12). These settings can be displayed and used by the frontend to provide access to the running VM.

The method for deleting a port forwarding rule is comparable to the previous method. The settings needed to identify a given rule are determined (line 18 – 21) before the rule is deleted from the firewall (line 23 and 24) and the local list (line 29).

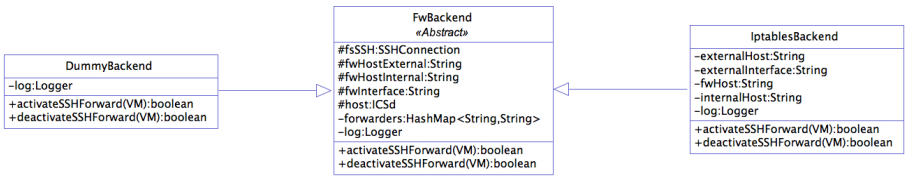


Figure 4.5: The ICS firewall package is designed modularly. New backends providing functionality for other firewalls can be easily developed.

Figure 4.5 shows the firewall package of the ICS. At the moment, the ICS only supports `iptables`-based firewalls. Nevertheless, the firewall package provides an abstract class that specifies methods for implementing in the backend classes. The abstract class also contains variables and methods that support the easy development of a the new backend e.g., the superclass provides a SSH connection which can be used to execute commands on the firewall host.

Besides the backend for `iptables`-based firewalls, the ICSd also provides a second backend, called DummyBackend. It is used if the ICS is executed in dummy mode[60] or if the ICS administrator has disabled the automatic port forwarding management, which is very useful during specialized fairs.

[59] The ICS itself is designed to be able to work with other firewalls as well, an appropriate firewall backend can be easily developed.

[60] The dummy mode has been developed to provide a mode for demonstrating the ICS' functionality without needing to set up an environment providing virtualization technology.

4.2.4 Deploy VM Images

When a particular VM image has been configured by its owner and should be used for job execution, it must be deployed to the deploy host. The deployment process is divided into two parts. First, the image must be transferred to the deploy host location and second, the XGE image configuration file must be modified to register the newly deployed image.

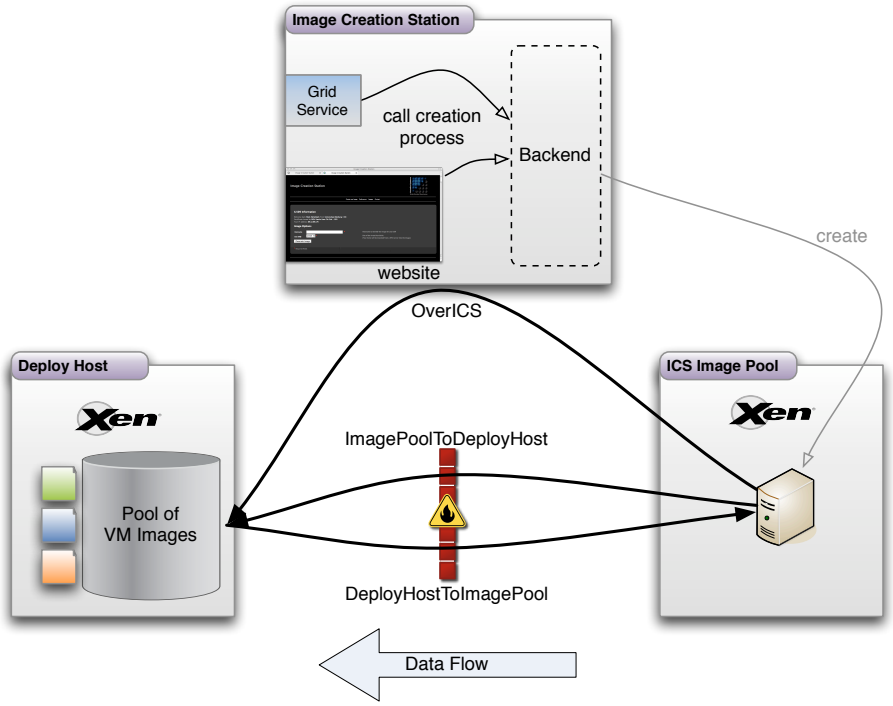


Figure 4.6: The ICS supports several mechanisms for VM image deployment from the image pool to the deploy host.

The complexity of the image transfer operation depends on the environment in which the ICS is installed. Figure 4.6 shows the image deployment methods supported by the ICS.

The simplest way of deploying a VM image is to perform a local copy operation if the image pool and the deploy host reside on the same machine. As clarified earlier, this is a most unlikely scenario due to security concerns. If these components reside on different hosts, more sophisticated deployment mechanisms must be used. Besides the local copy operation, the ICS supports the following deployment mechanisms:

ImagePoolToDeployHost. Using this method, the ICSd connects via SSH to the image pool and from there to the deploy host. For sparse images, `tar` is used to transfer the data from the image pool to the deploy host, whereby a secure SSH connection is used. In addition, the data can be compressed to speed up the deployment process and minimize the amount of data involved in the file transfer operation.

DeployHostToImagePool. A second possibility, which is very similar to the first one, is to connect to the deploy host via SSH and to initiate the

data transfer process from this host. Again, `tar` is used to transfer data over a secure SSH connection; data compression can also be used. This method can be used if the deploy host cannot be accessed via the image pool.

OverICS. If neither the image pool can be accessed via SSH by the deploy host nor the deploy host can be accessed via SSH by the image pool, this third method can be used. The ICS itself connects via SSH to both the image pool and the deploy host and transfers the data. This method is a fallback solution because at least the ICS must have access to both locations. Because of the fact that `tar` cannot be used in this scenario, using sparse images would not result in shorter transfer times in contrast to using one of the deployment mechanisms described above. Even if a sparse image with a maximum size of 2048 MB only consumes around 1 GB on the hard disk, the full 2048 MB would be transferred by this method. Moreover, data compression cannot be used.

Listing 4.23 shows the part of the code in which the deployment mechanism is specified. In line 2, the ICSd checks the source and the target of the deployment command. If they are the same, a local flag is set specifying that a local copy operation should be performed. This flag is checked in line 16. The local copy process has priority over the other deployment mechanisms and configuration settings regarding these mechanisms.

If this flag is not set, the configuration regarding the deployment mechanisms is read from the ICS configuration file. The setting `DEPLOY_DIRECTION` determines which one of the three deployment mechanisms described above will be used, while the setting `DEPLOY_COMPRESSION_METHOD` specifies whether or not data compression mechanisms should be used during the data transfer process. Data compression is not supported when using the *OverICS* deployment method.

If the *OverICS* method is used, the ICS connects to both the image pool and the deploy host via SSH, reads data from the image pool, and stores it on the deploy host. Secure File Transfer Protocol (SFTP) methods provided by the Trilead Java SSH2 library[61] are used. In contrast to *OverICS*, OS tools are used to transfer the data when using *ImagePoolToDeployHost* or *DeployHost-ToImagePool* mode. Here, the ICS connects to one host via SSH and connects from this host to the other host via SSH. The data transfer itself is performed using `tar`, a widespread software tool for Unix-based OSs. This tool supports both `gzip` and `bzip2` data compression, which can be activated if desired. Depending on the ICS configuration specified in the global configuration file, the commands will be created dynamically.

The creation of the commands is shown in Listing 4.24, which presents the

[61] `http://www.trilead.com/SSH_Library/`

Listing 4.23: The deployment mechanisms of the ICS rely on the environment in which the ICS is installed. If the image pool and the deploy host reside on the same host, a local copy process will be performed. If they reside on different hosts, the deployment mechanism used is determined based on the ICS configuration.

```
1  public SFTPConnection(SSHConnection sourceConnection, String
       sourceDir, SSHConnection targetConnection, String targetDir
       , Properties beConfig) {
2      if (sourceConnection.getTarget().equals(targetConnection.
           getTarget()))
3          this.remoteLocalCopy = true;
4
5      this.srcConn = sourceConnection;
6      this.targetConn = targetConnection;
7      this.backendConfig = beConfig;
8      [...]
9      this.result = this.transfer();
10     [...]
11 }
12
13 public boolean transfer() {
14     boolean result = true;
15     [...]
16     if (this.remoteLocalCopy)
17         result = remoteLocalCopy();
18     else {
19         String dir = "";
20         String comp = "";
21         if (backendConfig.containsKey("DEPLOY_DIRECTION"))
22             dir =backendConfig.getProperty("DEPLOY_DIRECTION");
23         else
24             return false;
25
26         if (backendConfig.containsKey("
               DEPLOY_COMPRESSION_METHOD"))
27             comp = backendConfig.getProperty("
                   DEPLOY_COMPRESSION_METHOD");
28         else
29             comp = "none";
30
31         if (dir.trim().equals("OverICS"))
32             result = this.sftpCopy(src.ls(sourceDir));
33         else
34             result = this.sshCopy(dir, comp);
35     }
36     [...]
37     return result;
38 }
```

Listing 4.24: When using either the ImagePoolToDeployHost *or the* DeployHostToImagePool *mode, the ICS connects to the remote host and from there to the other host. The data transfer itself will be performed by using the* `tar` *command, which also supports data compression mechanisms.*

```
1  private boolean sshCopy(String transferDirection, String
       compressionMethod) throws Exception {
2      SSHConnection conn = null;
3      String options = "fS";
4      if (compressionMethod.trim().equals("gzip"))
5          options += "z";
6      else if (compressionMethod.trim().equals("bzip2"))
7          options += "j";
8      String cmd = null;
9      if (transferDirection.trim().equals("ImagePoolToDeployHost"
           )) {
10         conn = this.srcConn;
11         cmd = "tar c" + options + " - --directory " + this.
               sourceDir + " . | ssh " + targetConn.getHostname()
               + " \"tar x" + options + " - --directory " + this.
               targetDir + "\"";
12     }
13     else if (transferDirection.trim().equals("
           DeployHostToImagePool")) {
14         conn = this.targetConn;
15         cmd = "ssh " + srcConn.getHostname() + " \"tar c" +
               options + " - --directory " + this.sourceDir + "
               .\" | tar x" + options + " - --directory " + this.
               targetDir;
16     }
17     else
18         return false;
19     conn.exec("mkdir -p " + this.targetDir);
20     conn.exec(cmd);
21     return (conn.getExitCode() == 0);
22 }
```

method called in line 34 of Listing 4.23. Depending on the transfer direction, data is either read from the local host and is then transferred to the tar instance on the remote host via SSH (line 11) or an SSH connection to the remote host is established to read the data via tar from the remote host and transfer it to the tar instance on the local host (line 15). The ICS configuration influences the different options provided to `tar`. To clarify the data transfer process, an exemplary command is shown:.

```
tar cfSz - --directory /ics/example . | \\
  ssh 10.0.0.2 "tar xfSz - --directory /ics/deployed/"
```

First, `tar` is started on the local node (the image pool) with options `cfSz`. Option `c` causes `tar` to create a new archive specified by the argument `f` (in fact, the archive name is set to "–" meaning that the data created by `tar` is written to the system's standard). The argument `S` forces `tar` to handle the read data as sparse images, i.e. only data is transferred that is used instead of transferring all of the VM's disk images even if only parts of them are used. The last option, `z`, enables `gzip` compression of the data stream. All files stored in the directory `/ics/example` are used as input files (`tar` changes to the directory specified by the `--directory` argument and reads all files: "`.`" matches all files in the current directory).

In the second part of the example, a SSH connection is established to the IP address 10.0.0.2 and the command specified between the quotation marks is executed. There is only one difference compared to the `tar` call in the first part of the command: Instead of using the `c` argument, `x` is used, causing `tar` to extract data that is read from the standard input ("–" as archive name as above). The extracted data will be stored in the directory specified by the `--directory` argument.

The first part of the example command reads data from the input files, creates a `tar` archive and compresses it with `gzip`. This data is written to the standard output stream which is piped (using "`|`") to the standard input of the second `tar` command. This command decompresses the data using `gzip`, extracts the `tar` archive and stores the received data at the specified location.

By handling all VM images as sparse images, only the data that is really consumed within the images is transferred to the remote host. Data compression is optional because it is a CPU intensive task. The administrator has to decide whether or not CPU cycles should be used to compress the data to be transferred. Using data compression can speed up the transfer process depending on the bandwidth of the network connection between the image pool and the deploy host. If data compression should be used, the administrator can choose between `gzip`, a widely used and fast compression algorithm, and `bzip2`, which is slower and more CPU intensive but usually creates better results in terms of the compression. No generic recommendation can be given since it depends heavily on the environment in which the ICS is used and the local administrator's habits.

4.2.5 Multi-Layered File System Images

The presented three-layer COW disk image architecture is based on AUFS [10] which is supported by Linux since Kernel Version 2.6.16 in 2006. The basic image hosts a Debian GNU/Linux stable based Linux distribution with a Xen-capable kernel. All tools needed to ensure seamless out-of-the-box operation are included, such as the entire Debian package management, standard compilers and interpreters, and a basic set of libraries including development headers. The basic image can run on every infrastructure supporting the Xen VMM. The site specific layer contains several additions needed to run the generic basic image on different sites. The changes to the basic image include special settings such as name resolution (DNS), IP configuration (DHCP-based, static addresses or peer-to-peer auto configuration) or user management (LDAP or NIS interfaces). Typically, the size of the site-specific layer is small, often between 10 KB and 1 MB. Finally, the user-specific layer contains all of the modifications made by the owner.

It should be mentioned that the implementation of COW file systems is transparent to the user. The user logs into the VM and modifies it as needed. As

shown in Figure 3.20, there are two important layers for the local administrator. While the software stack in the base image must be met by all sites linked together in the ICS collaborative network, every local administrator can put software and configuration files needed to run the VMs in the particular local network into the site-specific layer. This can include the local resource manager's configuration files as well as network NFS resources.

Since these settings are made by the local administrator, every user can focus on the VM modifications needed by his or her software and does not need to worry about COW or site-related details.

Considering the boot process of Linux, the Multi-Layered Root File System (MLRFS) must be built before the *init* process is executed to start the system. This allows the administrator to make arbitrary changes to the VMs, including changes to the service configuration or even the *init* process itself. Thus, building the root file system has to be done from inside the *initial ramdisk*, a minimal root file system that is responsible for mounting the real root file system. Inside the *initial ramdisk*, arbitrary scripts can be executed, as long as they are compatible with the restricted command set and shell.

The script must provide the following capabilities:

(1) Creating a root file system of an arbitrary number of layers

(2) Creating ramdisks to be used as layers

(3) Injecting files into the root file system

(4) Executing scripts (inside layers) before executing *init*

Capability (1) ensures the flexibility needed to ensure that the script can be used in the different scenarios of layered VMs defined above. The use of ramdisks (2) is a possible solution to the use of read-only layers, which is explained in detail below. Requirement (3) provides a simple way to install site-specific configuration files into the root file system. In combination with a ramdisk, the files can be put in a dynamically created layer with all files that are changed during runtime of the VM if the actual layers are mounted read-only. If configuration files do not provide enough possibilities for preparing a VM for execution at a specific Grid site, requirement (4) allows the site administrator to execute arbitrary scripts inside the VM before the *init* process is executed. This is necessary to change existing files instead of completely overwriting them. For example, fixing the Debian OpenSSH bug requires the administrator to exchange all keys created with the unpatched version. This includes exchanging the public key of the ICS, installed in the `authorized_keys` file of the root account during VM creation that needs to be replaced without touching other public keys most likely installed by the user.

4.2.6 Multi-Site ICS Support

Creating images on a single site is one main building block of a system combining several distributed resource sites. It is crucial to ensure automatic image exchange and synchronization between these sites enabling users to access their personalized VMs on every site. Therefore, the different ICS instances running on the distributed sites must interact and manage the exchange of VM images in a reliable and easy manner without the need for user interaction. This enables the system to adapt automatically to changes e.g., resulting from scheduling decisions.

4.2.6.1 Join the Collaborative Network

Whenever an ICS joins the collaborative network that connects the distributed ICS instances, it must propagate its state and obtain information from other ICS instances. To join the collaborative network, every instance needs the address of at least one other ICS already participating in this network. This address can be specified via the ICS configuration file or extracted from the Globus Monitoring and Discovery System (MDS) (if a Globus Toolkit headnode is available). Afterwards, the following actions will be performed:

1. The new ICS registers itself as a new network participant on the remote ICS and fetches the list of known ICS instances from the remote ICS.
2. It requests the current version information for all VMs of users that have logged in during the last 6 months. There are different possible states for every VM:
    - **no local changes, no remote changes** – Nothing must be done, since there were no updates during the period of a network split.
    - **no local changes, remote changes** – There were updates of this VM on the remote ICS, so the new version must be fetched from remote ICS instances.
    - **local changes, no remote changes** – There was an update on the local ICS during the period of a network split. Due to the fact that there were no remote changes, the local ICS can send an update notification to all known ICS instances.
    - **local changes, remote changes** – During the period of a network split, a user updated the particular VM on the local site and also on a remote ICS. Hence, there is a conflict that cannot automatically be solved. Therefore, the ICS renames the local VM and fetches the new version from a remote site. At this point, the user has both versions available and can decide whether or not to delete one of them. Even if this situation is very unlikely, data loss is impossible because both versions of a particular VM are saved.

Every time a user shuts down a VM on the ICS after performing modifications, a copy of the layer will be encrypted and a torrent file is generated by the ICS. The ICS uses BitTorrent's advanced network data transfer technology to transfer VM layers between different sites. A notification will be sent to all other known ICS instances containing the following information:

- **unique ID**
- **distinguished name** of the user who owns the VM
- **name** of the VM
- **modification time**
- **URL** of the torrent file that may be used to download the user layer

Every notification received from another ICS is stored in the local database. This prevents this ICS fetching the same file numerous times when being notified by different remote ICS instances. When receiving a notification regarding a particular user layer, the local torrent file is removed to prevent mixing up new and old files in the system.

4.2.6.2 Notification Management

Whenever an ICS instance receives a notification, the following algorithm is executed: Let I be all ICS instances in the system and V all existing virtual machines. $I \supset I_k, k \in I$ is the set of ICS instances known to the particular ICS k. Furthermore, $N = \{n_{s,v,t_n} / s \in I, v \in V\}$ describes all notifications in the system, each notification n_{s,v,t_n} can be identified by its sender s, its subject v and its timestamp t_n. Finally, the set $N \supset R_k, k \in I$ contains all notifications received by ICS k and $R_k \supset R_{k_{v,t}} = \{n_{s,v,t'} / s \in I, t' \geq t\}, v \in V$ containing all notifications regarding virtual machine v that have the same or a newer timestamp t_n than t.

```
for each k ∈ I do
  if ∃n_{s,v,t_n} ∈ R_k and R_{k_{v,t_n}} = ∅ then          (1)
    for each i ∈ I_k, i ≠ k, s do                           (2)
      send(i, n_{s,v,t_n})                                  (3)
```

The algorithm works as follows for every ICS k: Whenever a new notification is received regarding VM v that was never received before, and no newer notification regarding this VM exists (1) – this check is performed to avoid sending outdated notifications – this notification is forwarded (3) to all known ICS instances except the local ICS k and the sender s of the notification(2).

4.2.6.3 Image Transfer

Each ICS decides whether or not to fetch a particular user layer depending on the time that has passed since the layer's owner last logged in on this ICS. The local system administrator can configure this threshold to influence the behavior of the local ICS. If the ICS decides to prefetch the updated user layer, it connects to the given torrent URL and begins fetching the file.

If a user layer has not been prefetched, it must be transferred to the ICS when its owner logs in to modify it. As soon as a user decides which VM to work with (or is redirected to this ICS), the ICS immediately starts fetching the new file from the URL already known from an earlier update notification. Reflecting the mode of operation of BitTorrent, the download is faster when more remote sites have the needed version of the user layer available.

4.2.6.3.1 Encrypted Data Transfer If private data is transferred over an insecure link, it should be encrypted. However, encryption involves additional costs: The cryptographic computations produce CPU load on both the sender and receiver and take some time. BitTorrent traffic is commonly unencrypted but data traversing potentially insecure media like the internet should be encrypted to prevent data leakage and unauthorized modification during the data transfer. These costs are minimized by proactive encryption after a user's layer change. If a particular layer is requested by a remote ICS, it is already encrypted and the transfer process can start immediately.

Although the update notifications sent through the ICS collaborative network contain no sensitive information, the collaborative network as a whole can be secured using SSL/TLS encryption for notifications and data transfer.

4.3 Dispatch Daemon (dispatchd)

The dispatchd is responsible for scheduling jobs to remote resources. While it is comparable to GridWay in many ways, in contrast to GridWay, it also supports elastic sites with a dynamically changing number of compute nodes as well as the addition and removal of new resource sites during runtime.

As shown in Figure 3.23 on page 124, dispatchd employs existing GridWay schedulers to access the underlying resources of different resource sites. This decision has been made for two main reasons:

First, by using GridWay as interface for each resource site, the resources of each site can be addressed in an uniform manner. Moreover, dispatchd can provide an interface comparable to GridWay by forwarding received commands simply to the remote GridWay instance that has been chosen for job execution. Providing the same interface as GridWay enables users to utilize existing software tools developed for the use with GridWay. To provide extended func-

tionality, additional parameters can be considered e.g., to enable users to influence dispatchd's scheduling decisions. At the same time, GridWay's matured data transfer and authentication mechanisms can be used and do not need to be implemented by dispatchd again.

Second, GridWay is well-suited because of the fact that multiple Middleware Access Drivers (MADs) already exist allowing the utilization of a broad variety of resources. GridWay not only provides a MAD to schedule jobs to Globus, but it can also work with a Desktop Grid environment using Omnivore or execute jobs directly on the GridWay host.

Few requirements must be fulfilled by the remote sites in order to work with dispatchd. First, SSH access must be provided to allow dispatchd to log in and call the remote GridWay's `gwsubmit` script. Second, the user that submitted jobs to the local GridWay must be able to submit jobs under any user ID on the remote system. This can be achieved by providing `sudo` permissions to this user for the local GridWay's `gwsubmit` command. Finally, the ID of every user allowed to submit jobs to dispatchd must exist on the remote GridWay sites. This is needed to assign jobs to their owners.

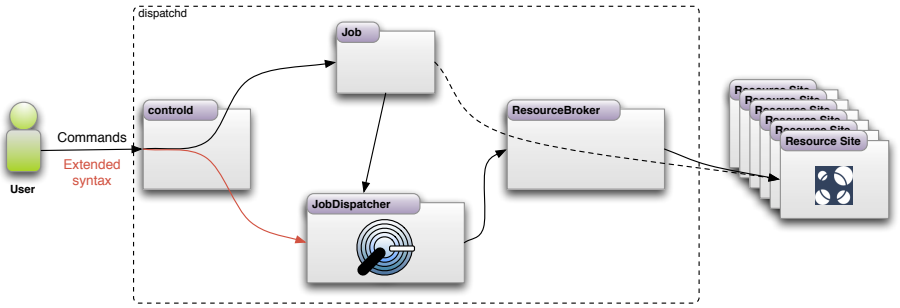


Figure 4.7: The Dispatch Daemon (dispatchd) provides several components.

The different components of dispatchd are shown in Figure 4.7. The interface provided to the user interprets the received command, creates a job object and stores the GridWay command. Extended arguments are parsed that can be specified by users to influence the decision regarding where a job will be executed. This decision is made by the JobDispatcher component and takes the extended arguments provided by the user and the job's characteristics and requirements into account. Once a decision is made, the job will be forwarded to the appropriate remote GridWay instance by the ResourceBroker component.

4.3.1 Manage Remote Resources

The resource broker's main runloop is shown in Listing 4.25. This runloop ensures that the remote resource site's configuration files are re-read periodically and that dispatchd recognizes any changes. This allows users to apply changes during runtime e.g., disable a remote site or update the number of compute nodes available. Listing 4.26 shows the method used to scan the configuration files, which is called in line 4 of the runloop.

Listing 4.25: The main runloop of dispatchd's resource broker component is rather simple. The resource site's configuration files are parsed and the list of available resources is cleaned up afterwards.

```
1  public void run() {
2      while (true) {
3          // scan directory
4          this.scanDir();
5
6          // cleanup list
7          this.cleanUpList();
8
9          [...]
10
11         // and sleep
12         try {
13             Thread.sleep(this.checkPeriod * 1000);
14         }
15         catch (InterruptedException e) { [...] }
16     }
17 }
```

Listing 4.26: Scanning configuration files for remote sites residing in dispatchd's resource directory.

```
1  public void scanDir() {
2      File[] resourcesDirContents =this.resourcesDir.listFiles();
3      for (int i = 0; i < resourcesDirContents.length; i++)
4          this.parseResourceFile(resourcesDirContents[i]);
5  }
```

For each file in the resource site configuration directory, a method is called which parses the particular file. This method is presented in Listing 4.27.

The file provided as an argument during the method call is read in line 4. Thereafter, several values are read from the configuration file in lines 8 – 19 which are processed afterwards.

An example of a configuration file is shown in Listing 4.28. Each resource is identified by a name, which can be chosen by the administrator (line 2). Moreover, a further description of a resource site can be provided if desired (line 3). In line 6, the resource can be enabled or disabled by a simple switch in the configuration file. This switch can be used to mark a resource as inactive. This can also be accomplished by deleting the particular resource's configuration file from the directory. The number of nodes available on this site is specified in line 7. By specifying a trust level, the user can influence the scheduling decision whether or not a resource is considered for the execution of a particular job. The lower the trust level, the more trusted the resource is by the administrator. An addition, a usage fee can be specified (line 9). To deal with elastic resource sites like Cloud resource sites, a particular resource set can be marked as scalable (line 12). If marked as scalable, the number of maximum nodes which can be acquired must be specified in line 13. In contrast to sites which are not scalable, the number of maximum nodes will be considered for scheduling decisions instead of the number of free nodes reported by the GridWay instance. The configuration details must be also provided that were used to connect to the remote host (lines 16 – 18). Finally, the command used to submit jobs on the remote host must be specified. It must be ensured that the job is executed with the same user ID used to submit the job to dispatchd.

Listing 4.27: Each resource site's configuration file is parsed and its values are stored in an internal resource list used for scheduling decisions.

```
public void parseResourceFile(File resourceFile) {
    Properties p = new Properties();
    try {
        p.load(new FileInputStream(resourceFile));
    }
    catch (Exception e) { [...] }
    [...]
    String name             = p.getProperty("name");
    String description      = p.getProperty("description");
    String enabledString    = p.getProperty("enabled");
    String nodesString      = p.getProperty("nodes");
    String trustLevelString = p.getProperty("trustlevel");
    String feeString        = p.getProperty("fee");
    String scalableString   = p.getProperty("scalable");
    String maxNodesString   = p.getProperty("maxnodes");
    String gwBinDir         = p.getProperty("gridway_bin_dir");
    String sshHost          = p.getProperty("host");
    String sshUser          = p.getProperty("user");
    String sshPortString    = p.getProperty("port");

    [...] // process values

    Resource thisResource = new Resource(name, description,
        enabled, nodes, trustLevel, fee, scalable, maxNodes,
        sshHost, sshUser, sshPort, gwBinDir);
    this.addResource(thisResource);
}
```

Listing 4.28: Example configuration file specifying a resource site to use with dispatchd.

```
# DESCRIPTION
name = Workgroup Server
description = Workgroup server located in rack C4

# RESOURCE ATTRIBUTES
enabled = true
nodes = 1
trustlevel = 1
fee = 0

# SCALABILITY
scalable = false
maxnodes = 1

# CONNECTION SETTINGS
host = 172.16.76.100
user = fallenbeck
port = 22

# GRIDWAY CONFIGURATION
gridway_submit_cmd = sudo -u %JOBOWNER% "/opt/gridway/5.6.1/bin
    /gwsubmit %JOBARGS%"
```

Listing 4.29: Example job template file describing a compute job.

```
NAME = Channel
EXECUTABLE  = Linux_2.6_channel
ARGUMENTS =
STDIN_FILE  = /dev/null
STDOUT_FILE = out.${JOB_ID}
STDERR_FILE = err.${JOB_ID}
INPUT_FILES = Linux_2.6_channel Linux_2.6_channel, energy.fl
    energy.fl, flow.nc flow.nc, inits inits, param param
OUTPUT_FILES = flow.nc flow.nc, status status, energy.fl energy
    .fl, inits inits
```

Listing 4.30: The `gwsubmit` script provided by the dispatchd determines the username of the caller to assign the job correctly.

```
#!/bin/sh
NC=`which nc`

USERNAME=`id -u -n ${UID}`
#USERNAME=`getent passwd ${UID} | cut -d : -f 1`

echo "SUBMIT --user ${USERNAME} $*" | ${NC} localhost 17952
```

This can be achieved by using `sudo` as shown in line 21 or directly using the `su` command if dispatchd connects to the remote site as privileged user root. However, for security reasons the use of `sudo` is strongly recommended. When executing the submit command on the remote host, `%JOBOWNER%` will be substituted by the username of the user which submitted the job to dispatchd and `%JOBARGS%` will be substituted by the command line options specified by the user. The dispatchd will remove extended arguments which are not supported by GridWay when the command is parsed.

The installation location of the GridWay instance must be also specified to access the local `gwsubmit` command used to submit the job to the remote GridWay instance.

4.3.2 Submit Jobs

To connect to remote sites, the same SSH methods are used as by the ICS that were introduced in Section 4.2.2. Since GridWay requires a job template file describing a computational task, at least this file must be transferred to the remote location. A sample file is shown in Listing 4.29

The computational task described by this job template file starts the binary file `Linux_2.6_channel`, which awaits several input files and creates several output files during computation. The input as well as the executable files must be copied to the remote location before the job is submitted to the remote GridWay instance. File transfer operations are also performed using the SSH function as described earlier. Once finished, output files are transferred back to the dispatchd host. For both input files and output files, the name of the source file and the target file name must be specified. This allows users to rename files during transfer to prevent that files from older jobs will be overwritten.

Listing 4.30 shows the `gwsubmit` script that dispatchd provides. It is used

instead of the real GridWay `gwsubmit` script and simply forwards all arguments to the local `dispatchd` listening on Port 17952. In addition, this script prepends the name of the user submitting the job. The command in line 4 returns the username in environments in which users are managed locally. When using LDAP, the alternative command in line 5 can be used to return the caller's username. A sample call of the `gwsubmit` script might appear as the following:

```
gwsubmit --trustlevel 2 -n 2 channel.jt
```

The `gwsubmit` script forwards the following string to the local dispatchd, which is parsed by dispatchd to consider (and remove) additional arguments from the GridWay submit command:

```
SUBMIT --user fallenbeck --trustlevel 2 -n 2 channel.jt
```

The username prepended by `gwsubmit` identifies which user has submitted the job. The switch `--trustlevel` is an extended argument which will be interpreted by dispatchd. In this case, the user requires that resources allowed to execute the command have a trust level of 2 (or less). The remainder of the command will be directly forwarded to the remote GridWay instance once the decision where to execute this job has been made:

```
sudo -u fallenbeck \\
  "/opt/gridway/5.6.1/bin/gwsubmit -n 2 channel.jt"
```

4.3.3 Schedule Jobs to Remote Resources

When dispatchd connects to the remote GridWay server to be used to execute the particular job, the job will be submitted using the username which has been determined by dispatchd's `gwsubmit` command. This ensures that each job can be attributed to a particular local user.

Deciding which remote resource should be used for a particular job is straightforward. The resource sites known to dispatchd are managed in an internal list and ordered according to their level of trust in ascending order. The set of resources taken into account to decide to which site a particular job will be submitted contains all remote sites known to dispatchd with a level of trust less than or equal to the level of trust specified by the user during job submission. If no level of trust was specified, all known resources are considered.

Beginning with the resources with the lowest level of trust,[62] dispatchd connects to the remote host and calls the `gwhost` script to obtain information about the resources' state. The output might look similar to this:

[62] It must be remembered the lower the level of trust a resource has, the more trustworthy it is.

```
HID PRIO OS    ARCH  MHZ %CPU MEM(F/T)     DISK(F/T) N(U/F/T) LRMS HOSTNAME
0   0    Linux  x86 2992    0  13/1009 418545/663247    0/2/2 Fork gw-hn
```

The command `gwhost` displays the state of the connected resources and provides information such as the installed OS, the available main memory and more. For dispatchd, the most important information is the column with the header `N(U/F/T)` which displays the number of used, free and total slots available. In this example, no resources are used and 2 slots are available for job execution. If the particular site has been marked as elastic in its configuration file, the maximum number of nodes which could be acquired is taken into account to decide if more resources could be made available on the remote site. If the site matches the job's needs, the job is submitted to the remote GridWay by the command specified in its configuration file and the scheduling process stops. If no matching resource could be found, the scheduling process will be repeated after a certain amount of time.

4.4 Request Daemon (reqd)

The reqd is responsible for acquiring resources from a resource site that provides a middleware that facilitates the provisioning of resources on-demand. New resources are acquired when available resources no longer sufficiently meet the customers' requirements. The reqd is the basic component that provides elasticity on resource sites that dispatchd will access.

4.4.1 Request Additional Resources

Therefore, the local GridWay component installed for each resource site has to be patched to periodically report the length of its waiting queue to the reqd. The patch only affects `gw_scheduler.c`, which holds the functionality to dispatch pending jobs to known compute nodes. The most important function containing the added functionality is shown in Listing 4.31. The first method sends the number of additionally needed resources to reqd if needed. To estimate the number of resources needed, the average number of slots (which represents the number of CPU cores) that the waiting jobs will need is calculated based on the jobs currently pending (line 6). Furthermore, this number is multiplied by the number of pending jobs (line 7) to estimate the number of slots needed to execute all waiting jobs. The difference between the slots needed to satisfy the pending jobs and the currently available slots (line 8) specifies the number of slots missing to meet the requirements for resources at this point in time. This number is sent to the reqd via a shell script named `request_slots.sh` that is provided with the patch.

The method shown in line 16 et seq. shows the main method of this class, which is executed when jobs from GridWay's waiting queue are scheduled to the available resources. For each job in this queue (line 20), the number of

Listing 4.31: Code in GridWay's scheduling component needed to report GridWay's current state to reqd.

```
static void gw_request_more_slots (gw_scheduler_t * sched, int
    num_free_slots, int num_needed_slots) {
    int num_pending_jobs, request;
    num_pending_jobs = sched->num_jobs;
    request = 0;
    if (num_pending_jobs > 0) {
        int mean_slots_per_job = num_needed_slots /
            num_pending_jobs;
        int needed_slots_for_pending_jobs = num_pending_jobs *
            mean_slots_per_job;
        request = needed_slots_for_pending_jobs -
            num_free_slots;
    }
    if (request > 0) {
        char external_cmd[255];
        snprintf(external_cmd, 254, "%s/bin/request_slots.sh %d
            ", getenv("GW_LOCATION"), request);
        system(external_cmd);
    }
} // gw_request_more_slots(gw_scheduler_t*, int, int)
static void gw_sched_dispatch (gw_scheduler_t * sched,
    gw_boolean_t reschedule, int * dispatched) {
    [...]
    b2g_free = 0;
    b2g_need = 0;
    for (i=0; (i<sched->num_jobs) && (*dispatched < max_dsp); i
        ++) {
        user = &(sched->users[sched->jobs[i].ua_id]);
        job  = &(sched->jobs[i]);
        running_user = user->running_jobs + user->dispatched;
        [...]
        for ( j=0 ; j < job->num_mhosts ; j++ ) {
            host = &(sched->hosts[job->mhosts[j].ha_id]);
            running_host = host->used_slots + host->dispatched;
            [...]
            free_slots = job->mhosts[j].slots - host->
                dispatched;
            b2g_free += free_slots;
            b2g_need += job->np;
            if ( free_slots >= job->np ) {
                *dispatched = *dispatched + 1;
                host->dispatched++;
                user->dispatched++;
                [...]
                gw_scheduler_job_del(sched,job->jid,1);
                [...]
            }
        }
    }
    gw_request_more_slots(sched, b2g_free, b2g_need);
} // gw_sched_dispatch (gw_scheduler_t*, gw_boolean_t, int)
```

Listing 4.32: The script `request_slots.sh`.

```
1 #!/bin/sh
2 echo "request $1" | nc localhost 32123
```

needed resources is determined (line 25 et seq.). This happens by iterating the list of hosts known to GridWay. If the number of free slots available on a certain host is larger than or equal to the number of slots needed to process the particular job (line 32 et seq.), it is dispatched to this particular host. Nevertheless, the number of free slots (before job dispatch) and the number of needed slots are stored.[63]

The external script `request_slots.sh` is used to hand over the information published by GridWay to the reqd. This script is a simple shell script which uses `netcat` (`nc`) to send the values given as command line arguments to a network port used by reqd to transfer the values.

Listing 4.32 shows this (very simple) shell script. It forwards its first input parameter (which represents the number of slots to be requested, see Listing 4.31, line 14) to the `reqd` listening on Port 32123 for requests. The command handed over to the reqd starts with the keyword `request` and the number of nodes needed. Besides querying new resources, reqd supports other commands as well. For example, the administrator can shut down the daemon using this interface[64] or view the reqd's current state.

```
fallenbeck@ma1244:~$ reqc show requests
Received Requests
=================
Sun Dec 19 14:01:17 CET 2010 1
Sun Dec 19 14:01:47 CET 2010 9
Sun Dec 19 14:01:19 CET 2010 6
Sun Dec 19 14:01:43 CET 2010 4
SUCCESS
```

To use reqd from the command line, `reqc` (reqd control) is available, a shell script comparable to the one shown in Listing 4.32. It simply forwards all input parameters to the reqd listener. The last line of output is either `SUCCESS` (if the command was successful) or `FAILED` (if the command could not be processed for some reason). In particular, a request for n new resources returns `SUCCESS` even if reqd decides not to acquire new resources from the provider.

Figure 4.8 shows the classes of the reqd and their interdependencies. It is worth noting that the main class is not shown in this picture in an effort to keep the figure clear. As shown earlier in Figure 3.25 on page 127, reqd is subdivided into three main parts. On the right hand side, the Listener is shown.

[63] The variables' prefix "b2g" refers to the Biz2Grid project in which reqd was initially developed.

[64] The ability to shut down reqd can be prevented by a configuration option and is disabled by default.

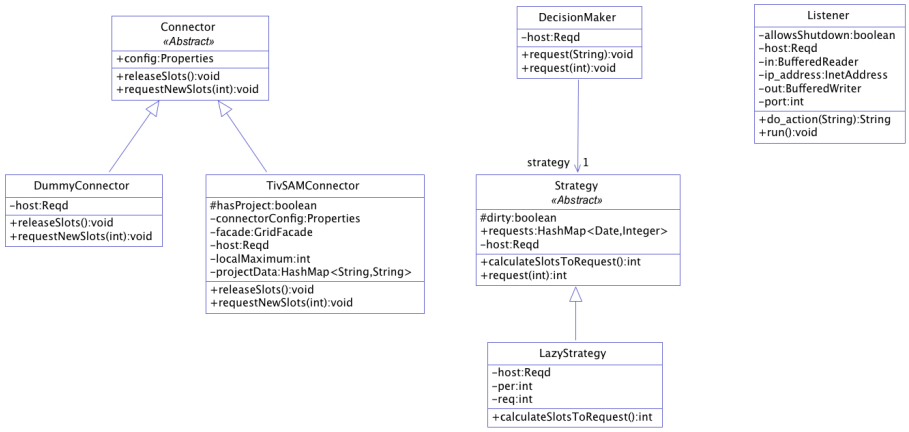


Figure 4.8: The classes of reqd and their dependencies.

For security reasons, it is normally bound to localhost and does not accept commands from anywhere but the local computer on which it is installed. If needed, the administrator can bind it to any other address allowing reqd to be installed on a host other than GridWay.

Commands received by the Listener class are either executed by the function `do_action()` (e.g., if reqd's state should be displayed or reqd should be shut down) or, in the case of a resource request, are handed over to the DecisionMaker. This class decides whether or not to acquire new resources from the site's backend. To calculate the number of new resources to acquire, DecisionMaker uses a Strategy. This abstract class provides basic functionality and defines the abstract method `calculateSlotsToRequest()` that must be implemented by every strategy. The reqd comes equipped with LazyStrategy, which acquires new nodes depending on the number of requests received within a certain amount of time. It only requests new nodes if 10 requests are received within 5 minutes to avoid acquiring new resources if only a short spike in demand for computational data occurs. If the load increases for a longer period of time, new resources can be provided quickly. The number of resources (slots) to request is the mean value calculated using the number of slots requested by each of the 10 requests.

Listing 4.33 shows how this decision is made. The first public method `calculateSlotsToRequest()` is called by the DecisionMaker and is required to return the number of resources required (i.e., 0 if no resources should be requested). To calculate this number, the request map is cleaned first (line 3). This method checks if the request map was modified since the last cleanup process (line 11) and deletes all requests that are too old to be considered by the strategy (in this case, every request older than 5 minutes will be removed) (line 12 et seq.). Once the request map has been cleaned, LazyStrategy checks the number of requests still in this map (LazyStrategy will only calculate the number of resources to be acquired if there are more than 10 requests left) (line 4). If so, the number of resources will be calculated (line 22

Listing 4.33: The three basic methods that are used by LazyStrategy to decide whether or not new resources should be acquired from the local site's infrastructure. Furthermore, the number of resources to request is calculated.

```
1  public int calculateSlotsToRequest() {
2      int request = 0;
3      this.cleanupRequests();
4      if (super.requests.size() >= this.req) {
5          request = this.numRequestSlots();
6          super.requests.clear();
7      }
8      return request;
9  }
10 private void cleanupRequests() {
11     if (dirty) {
12         Date deleteBefore = new Date((new Date()).getTime() - (
               this.per * 1000));
13         Iterator<Date> i = super.requests.keySet().iterator();
14         while (i.hasNext()) {
15             Date reqDate = i.next();
16             if (reqDate.before(deleteBefore))
17                 super.requests.remove(reqDate);
18         }
19         this.dirty = false;
20     }
21 }
22 private int numRequestSlots() {
23     int sum = 0;
24     Iterator<Integer> i = super.requests.values().iterator();
25     while (i.hasNext())
26         sum += i.next().intValue();
27     int roundedMean = (sum / super.requests.size());
28     return roundedMean;
29 }
```

et seq.). Thereafter, all requests will be deleted (line 6). Finally, the calculated number of resources to be requested is returned (line 8).

Once DecisionMaker decides that new resources should be acquired from the local infrastructure (i.e., if the strategy has returned a value unequal 0), DecisionMaker calls the method `requestNewSlots()` from the connector and provides the calculated value as an argument. This method as well as the method `releaseSlots()` is an abstract method that must be implemented by every class extending the abstract class connector i.e., every backend implementation. The reqd was initially developed to work with IBM TivSAM and provides a backend which has been successfully tested by the IBM R&D lab in Böblingen, Germany. Moreover, a DummyConnector is delivered to test the reqd if no TivSAM installation is available.

4.4.2 Shut Down Resources

To detect a shortage of available resources it is sufficient to monitor the length of GridWay's job queue. If jobs are not scheduled to resources but stay in the job queue, insufficient resources are available to satisfy the jobs' needs instantly. As shown above, different strategies can be used to decide whether new resources should be acquired from the local infrastructure management software.

Detecting the existence of unused resources that could be shut down in order

Listing 4.34: The probe configuration file of the probe server.

```
# This file specifies the command that will be executed on the
# compute nodes and used to decide whether or not resources
# can be shut down
PROBE_CMD=/usr/sbin/xm list | wc -l

# A probe is guessed to be idle if that condition matches
PROBE_COND_IDLE_CONN=<=
PROBE_COND_IDLE_VALUE=2
```

to save money and energy is more complicated i.e., an empty job queue does not necessarily mean that there are unused resources. Moreover, it is difficult to decide which resources are not being used and could be shut down. To achieve that functionality and provide a site administrator the flexibility even to use own metrics to decide whether or not a particular resource is used, the probe server has been developed. It has been designed to meet two major goals:

1. The probe server receives the values ascertained by the probes running on the compute nodes. The received values are analyzed in order to decide if a given condition is met.

2. The probe server provides a command which can be queried by the probes and is executed on the compute nodes. This command can be used to uniformly ascertain the resources state.

Listing 4.34 displays a probe configuration which can be used to decide if virtual machines are running on the compute nodes. This file allows administrators to specify a command which is used to measure the compute nodes. The result of this command is sent back to the server which checks if the specified condition is met in order to decide if a resource should be marked as idle. To specify the condition, the administrator can check if the received result equals the estimated value `PROBE_COND_IDLE_VALUE`. If the result (and the estimated value) is a number, additional comparators are supported which allow the probe server to test if the received result is less than, greater than or equal to the estimated value. The configuration file is periodically re-read allowing reconfiguration of the probe server without the needing to restart it.

In the given example, the Xen manager (`xm`) prints out all running VMs. The result might appear as follows:

```
Name                        ID   Mem VCPUs      State   Time(s)
Debian                      20   256     1     -b----  628103.1
Domain-0                     0  1603     2     r-----   21330.5
node02c0                    22    64     1     -b----      95.8
node02c1                    21    64     1     -b----     102.2
```

The number of lines of the output is counted and the result is sent back to the server. In the server, every host that sends a result which is less than or equal to 2 is marked as idle. The `PROBE_COND_IDLE_VALUE` needs to be set to 2

Listing 4.35: The probe script must be executed on the compute nodes. It queries the command from the probe server, executes it and sends the result back to the server.

```
1 #!/bin/sh
2 PROBESERVER=172.16.76.2
3 PROBEPORT=32124
4
5 cmd=`echo 'PROBE GETCOMMAND' | nc ${PROBESERVER} ${PROBEPORT}`
6 result=`eval $cmd`
7 echo "PROBE SENDRESULT ${result}" | nc ${PROBESERVER} ${
    PROBEPORT}
```

because there is one line containing the column headers and the dom0, which is always active even if there is no active VM that belongs to a user.

Allowing to specify a command in the server's configuration file prevents the administrator from configuring the probe script on each host. The probe script itself is quite simple to avoid the need for executing complex software on the compute nodes which would result in performance degradation of jobs running on the compute nodes.

Listing 4.35 displays a probe script that requests the command from the probe server (line 5), executes it (line 6) and returns the result to the server (line 7). Once this script has been correctly configured, the administrator can modify the command executed on every compute node simply by editing the command in the probe server's configuration file. Therefore, the measurement process can be adapted easily and quickly to consider changes in the system.

An administrator can use the provided probe script in conjunction with the probe server as well as her own software or monitoring software installed on the compute nodes. Moreover, the compute nodes do not need to be operated with a specific OS version since the administrator can send any result to the probe server as long as a condition can be posed to decide whether or not a particular resource is idle.

To obtain the state of a particular resource, the administrator or the reqd can query the probe server in a simple manner. Whenever a result of a probe is received, the server stores the hostname of the sender and the result of the condition automatically. If results are sent frequently to the server, accurate information about the system's state are available to reqd. If probes are rarely executed, interferences with running jobs can be minimized but information about the compute nodes' state stored in the server can be outdated.

4.5 Summary

In this chapter, several implementation details of the different components have been presented. Several ways of creating new VM images have been shown in detail as well as methods for importing and exporting VM disk images. Some components running in the background and ensuring a consistent state of the ICS like the Watchdog component have also been presented. In

addition, optimization details like the utilization of multi-layered file systems have been introduced.

Thereafter, the functionality needed to provide a system spanning across several geographically distributed sites was introduced. The ability to deal with different resource sets such as Desktop Grids or Cloud resources is the key component in creating a system that is able to link these different resources together and use them efficiently. Implementation details of the basic building blocks like multi-site support, the meta scheduler extension able to deal with heterogeneous and elastic resource sites, and the request component that adapts dynamically to the load profile of the local system have been presented.

The meta scheduler extension dispatchd can deal with different resource sites such as Desktop Grids and Cloud resources. It has been shown how users can submit jobs to dispatchd instead of GridWay and how the decision is made which resource site is used for job execution. Moreover, it has been shown how dispatchd deals with sites that can provide additional resources. To acquire resources automatically, reqd monitors the current state of a resource site. Implementation details have been presented as well as the algorithms used to decide when new resources are acquired and when running idle resources will be shut down.

Altogether, the components presented form a system as proposed in Chapter 3.

5 Experimental Results

"You can see they've gone from 50 instances of EC2 usage up to 3,500 instances of EC2 usage. It's completely impractical in your own data center over the course of three days to scale from 50 servers to 3,500 servers. Don't try this at home."

Jeff Bezos, CEO Amazon [106]

5.1 Introduction

To verify the functionality of the proposed system, numerous experiments have been performed. These experiments examine particular aspects of the system in many different areas. In order to be accepted on a larger scale, it is crucial that the system reacts quickly to its users' needs and that it can be used in a simple manner.

This chapter presents the different experiments which were performed as well as their results. In Section 5.2, the overall performance of VMs will be examined. As stated by many commercial project partners and even HPC system administrators, a small decrease in performance due to virtualization technology is acceptable for users if security is increased. Section 5.3 focuses on the ICS and its main task: creating and deleting images. The following section, Section 5.4, presents results of experiments related to the topic of transferring data. This is not only necessary in a multi-site scenario when VMs must be exchanged between different Grid sites but also when VMs need to be

deployed to the computational resources of a single site. Finally, Section 5.5 focuses on the scalability and the elasticity of the proposed solution.

5.2 Performance of VMs

A system's performance is crucial for customers. Hence, this section provides the results of some experiments performed to measure the performance of different virtualization solutions compared to native executions. First, the I/O performance of typical load profiles found in the Grid and on HPC resources is compared in different virtualization environments. Thereafter, the performance impact of using multi-layered disk images instead of single disk images is measured. Finally, since crucial for parallel job execution, the performance of MPI is measured in different scenarios.

5.2.1 I/O Performance

To examine the I/O performance, several measurements have been conducted to compare the performance of a native compute node with the performance of virtual machines of both Xen and VirtualBox. Table 5.1 presents the hardware and software equipment used to perform these measurements.

Table 5.1: System configuration of the node used for the I/O benchmarks.

Hardware:	Intel Core2 6600, Dual Core 2.4 GHz 2 GB main memory 250 GB hard disk
Software:	Debian GNU/Linux 6 "squeeze" Linux kernel 2.6.32 Xen 4.0.1 VirtualBox 4.0.4
VMs:	Debian GNU/Linux 6 "squeeze" Linux kernel 2.6.32 2 CPU cores 1.7 GB main memory 8 GB "sparse image"

The compute node was set up by using the most recent version of Debian GNU/Linux to ensure that the most current versions of the VMMs were available. Xen 4.0.1 was released on 25 August 2010 while VirtualBox 4.0.4 was released on 27 February 2011.

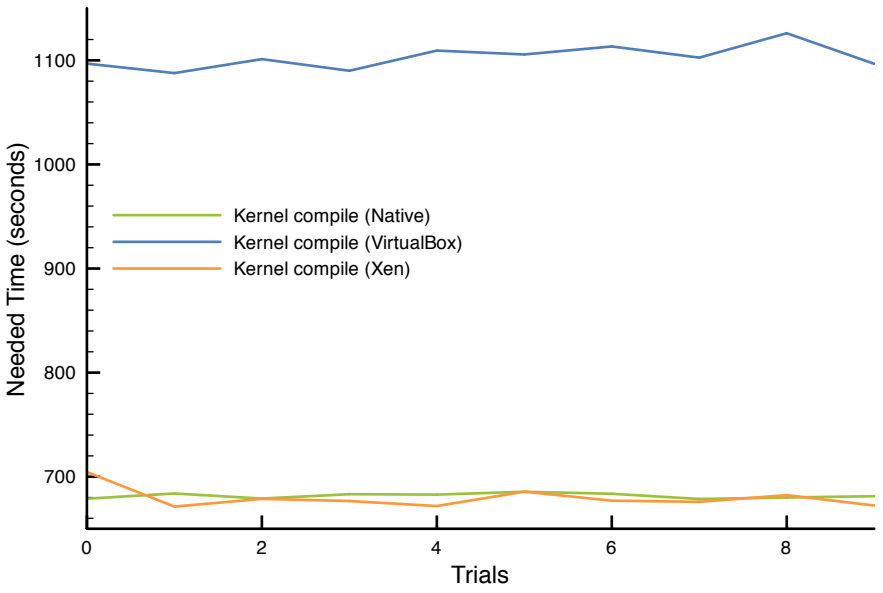
Compiling the Linux kernel is a widely-used method for testing the overall performance of a system. During the compilation process, file I/O takes place while the CPU compiles the kernel's C source code. A compilation process is a valid method of mimicing the workload of a scientific application. Ten compile processes have been performed to get a meaningful value. Second,

the `dbench` benchmark[65] was used to simulate file system load to measure the I/O throughput of data transfer. To calculate a robust mean, 350 measurements have been performed. The results are summarized in Table 5.2.

Table 5.2: Results of the I/O benchmarks.

		Native	Xen	Virtualbox
Kernel compile	Time	681.69 s	679.61 s	1102.94 s
	σ	2.37 s	9.34 s	10.81 s
dbench	Throughput	75.89 MB/s	272.39 MB/s	52.51 MB/s
	σ	3.23 MB/s	9.62 MB/s	3.49 MB/s

Figure 5.1: Time needed to compile a Linux kernel on the native machine and within a Xen and a VirtualBox VM.



As shown in Figure 5.1, compiling a Linux vanilla kernel Version 2.6.38 took 681.69 seconds on the average with a Standard Derivation (σ) of 2.37 seconds on a native machine. Compiling the Linux kernel within a Xen domU is 2 seconds faster: On the average, the compilation process needs 679.61 seconds to finish (σ = 9.34 seconds). This unexpected result is caused by Xen's sophisticated cache mechanisms and the fact that the para-virtualization approach used by Xen results in nearly no overhead. In contrast, the compilation process is much slower in a VirtualBox VM using a full virtualization approach: the kernel build process needs 1102.94 seconds to be completed (σ = 10.81 seconds).

Figure 5.2 displays the results of the `dbench` benchmark which shows similar characteristics. While reaching a throughput of 75.89 MB/s (σ = 3.23 MB/s) on the native machine, the throughput is even higher in the Xen VM (272.39 MB/s, σ = 9.62 MB/s), which is again caused by Xen's caching strategies. Again, VirtualBox cannot deliver a comparable throughput with 51.51 MB/s on the average (σ = 3.49 MB/s).

[65] `http://samba.org/ftp/tridge/dbench/`

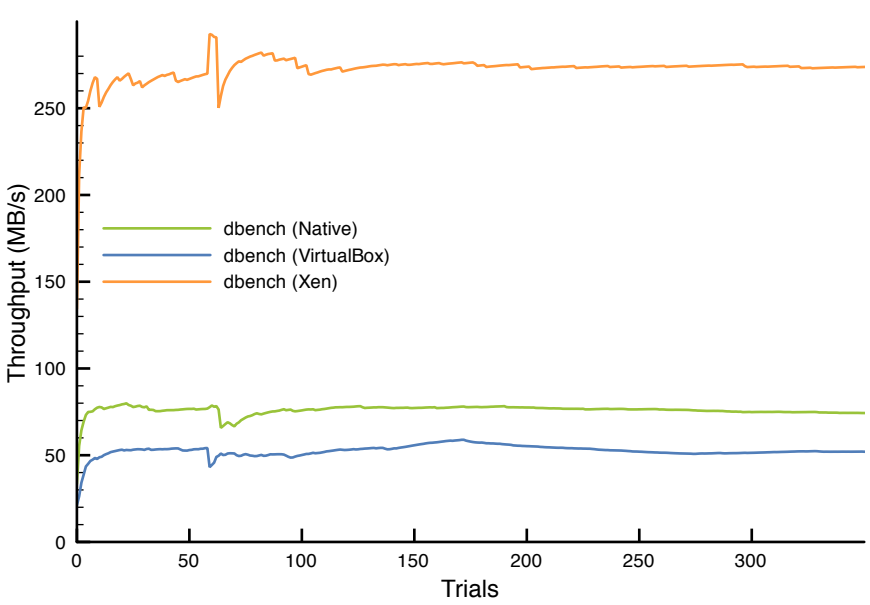


Figure 5.2: Throughput of the `dbench` benchmark executed on the native machine and within a Xen and a VirtualBox VM.

5.2.2 Network Performance

To examine the performance of the network interconnection between physical hosts and VMs, measurements were conducted between VMs on different physical hosts connected by a Gigabit Ethernet connection. In total, 3600 samples were recorded during a one-hour analysis of the network performance using `iperf`[66] as measurement tool. The physical machines were equipped with identical hardware as shown in Table 5.3:

Table 5.3: System configuration of the physical compute nodes used for network performance benchmarks and hosting the VMs.

Hardware:	AMD Opteron 2216 HE, 2x Dual Core 2.4 GHz 16 GB main memory 250 GB hard disk Gigabit Ethernet
Software:	Debian GNU/Linux 4 “etch” Linux kernel 2.6.18-6-xen-amd64 Xen 3.0.2
VMs:	Debian GNU/Linux 4 “etch” Linux kernel 2.6.18-6-xen-amd64 128 MB main memory 2 GB “sparse image”

Figure 5.3 displays the results of the measurements. The network bandwidth used is displayed on the horizontal axis while the vertical axis displays how many measurement points reached a particular bandwidth. It can be seen that

[66] `http://dast.nlanr.net/Projects/Iperf/`

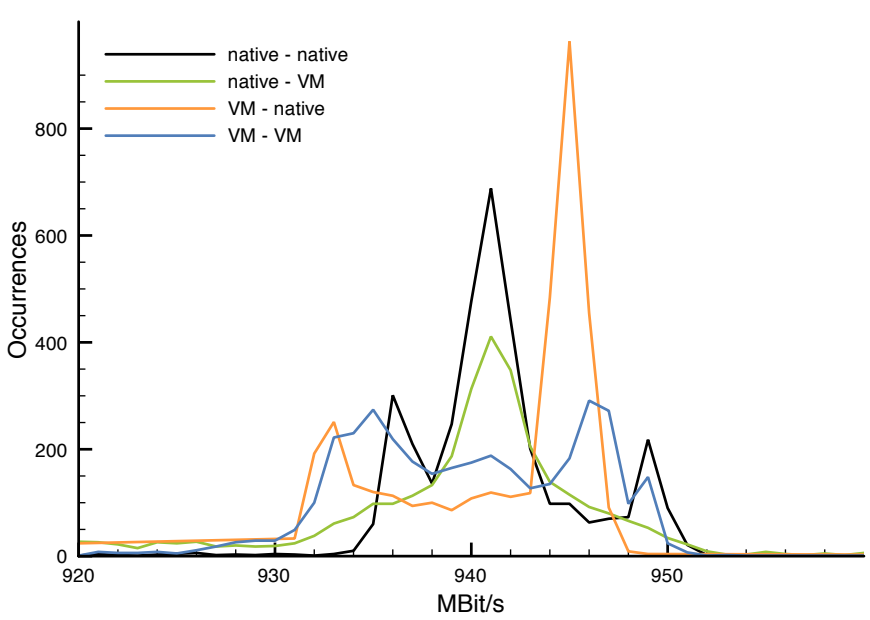


Figure 5.3: Network throughput between physical and virtual machines connected by the a Gigabit Ethernet network connection.

a bandwidth of 940 MBit/s has been measured at the most measurement points when transfer was measured between the two physical hosts.

Table 5.4: Network performance between physical and virtual hosts.

			Bandwidth	σ
native	–	**native**	940.4546214 MBit/s	10.00460088 MBit/s
native	–	**VM**	941.1294904 MBit/s	8.60848647 MBit/s
VM	–	**native**	932.7770045 MBit/s	22.52961662 MBit/s
VM	–	**VM**	938.6673163 MBit/s	15.80340526 MBit/s

Table 5.4 shows detailed information about the performed measurements. Between two native hosts, a network bandwidth of about 940.455 MBit/s could be reached (σ = 10 MBit/s). When measuring the performance between a native node and a VM hosted on the other machine, a throughput of 941.13 MBit/s was recorded (σ = 8.61 MBit/s). When transferring data from a VM to a physical host, a performance of 932.78 MBit/s could be reached (σ = 22.53 MBit/s). The performance was slightly higher when data was transferred between two VMs: data could be transferred with 938.67 MBit/s on the average (σ = 15.8 MBit/s) according to `iperf`. The slight differences in the measurements are due to the network traffic which takes place during the measurements. Nevertheless, there is nearly no difference in data transfer rates when using VMs instead of physical machines.

5.2.3 Multi-Layered VMs

The utilization of multi-layered disk images instead of single files can minimize both the amount of space needed to store many different VMs that belong to different users and the time needed to transfer images to remote sites. In the deployment process, the advantages of copying only the user layer to a remote

site are thwarted by the fact that every VM relies on a multi-layered file system. Using a COW file system as disk image produces additional costs when I/O intensive tasks are performed. Several measurements have been conducted in a VM with a main memory of 128 MB and two assigned CPU cores to investigate this overhead. The configurations include VMs with no layers as well as those with two or three layers as described in Section 3.2.1.3.

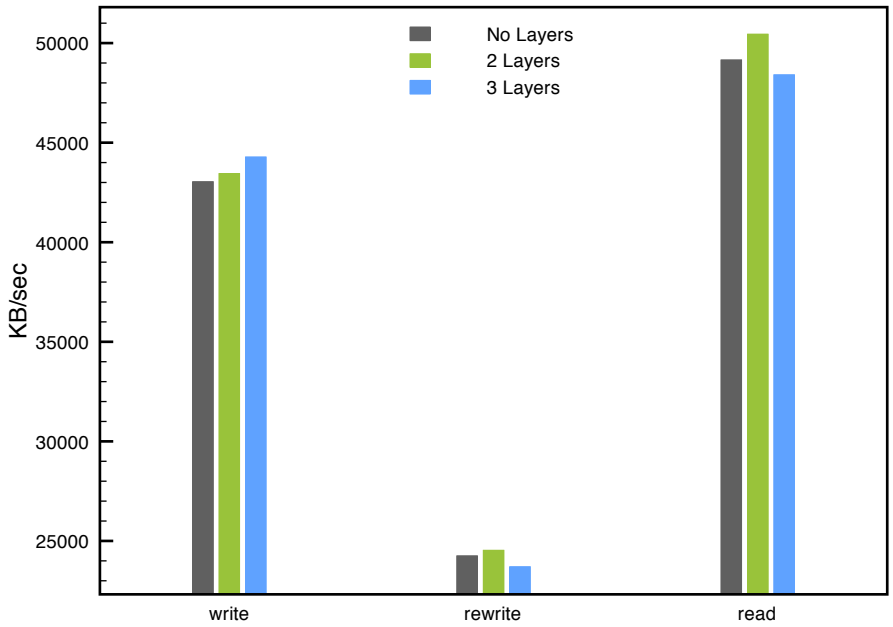


Figure 5.4: I/O performance measurements of the Multi-Layered Root File System (MLRFS) using the bonnie++ benchmark.

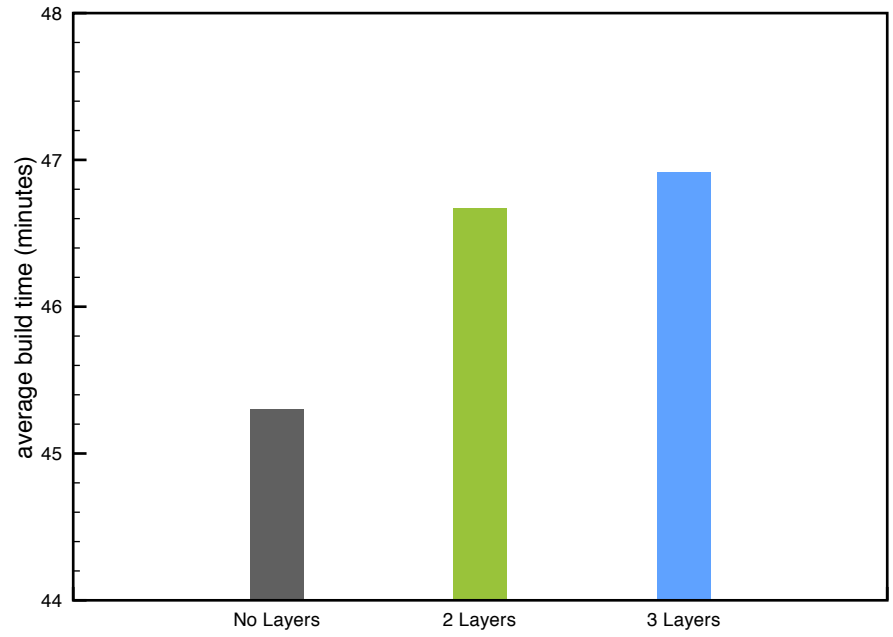


Figure 5.5: Performance measurements of the Multi-Layered Root File System (MLRFS) building a Linux kernel.

The first measurement was done using the `bonnie++` [38] benchmark, a well-known testing suite aimed to perform a number of file system related tests. The performance of the virtual hard disk has been examined when writing to the disk, when data is modified in place, which requires read-, seek- and

write-operations, and when data is read. For each test, `bonnie++` reports the number of kilobytes processed per second. The average throughput is shown in Figure 5.4. The performance of MLRFSs is comparable to the performance of the non-layered file systems, in some cases even better. The difference between the worst and best result is at most 4%. This met the expectations that COW file systems do not induce significant throughput degradations when large amounts of data are read or written.

In a second measurement, a Linux kernel was built to evaluate the behavior of AUFS when a large number of small files is read or written. In the layered case, the kernel source files were in layer $n-1$, while the output files were stored in layer n. A slightly degraded performance of the layered VMs can be expected because looking up a file requires checking all layers, and changing a file requires copying it to the writable layer first. The average build times are shown in Figure 5.5. The overhead of the layered VMs is below 3% in the layered cases.

It can be concluded that the introduction of the additional layers has a negative impact on performance. Due to the fact that most files of a regular job are written in the user's home directory that is remotely accessible, the overhead only comes into play in special circumstances.

5.2.4 MPI Performance

In addition to the analysis of the transfer performance, MPI performance measurements using the Intel MPI Benchmark (IMB)[67] have been conducted. The measurements were performed on 4 compute nodes equipped with identical hardware shown in Table 5.5.

Table 5.5: System configuration of the 4 compute nodes used for the MPI benchmarks.

Hardware:	AMD Opteron 2216 HE, 2x Dual Core 2.4 GHz 16 GB main memory 250 GB hard disk Gigabit Ethernet
Software:	Debian GNU/Linux 4 "etch" Linux kernel 2.6.18-6-xen-amd64 Xen 3.0.2
VMs:	Debian GNU/Linux 4 "etch" Linux kernel 2.6.18-6-xen-amd64 128 MB main memory 2 GB "sparse image"

Figure 5.6 shows the results of the IMB Sendrecv benchmark in which the processes form a periodic communication chain. Each process sends messages to the right neighbor and receives messages from the left neighbor in the

[67] `http://www.intel.com/software/imb/`

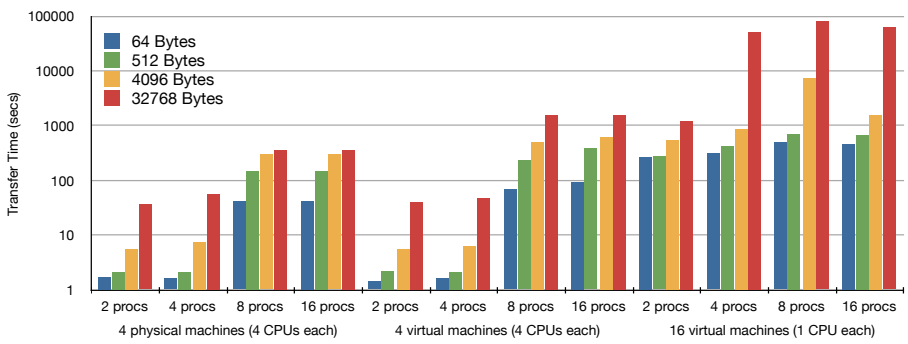


Figure 5.6: Runtimes of IMB Sendrecv benchmark on physical and virtual machines.

chain. First, the measurements were conducted on 4 physical machines with 4 physical CPU cores each. Then, measurements were performed with different message sizes within 4 VMs with 4 CPUs cores assigned to each VM. Finally, 16 VMs (4 VMs on each physical machine) with 1 CPU core assigned were used for these measurements.

It is evident that the MPI performance is best on the physical machines. The additional time between the measurements using 4 and 8 processes results from the fact that some communication was performed using the physical network between the processes. When conducting measurements with 4 or fewer processes, the communication takes place on one physical machine and does not have to use a network connection. When 4 VMs instead of 4 physical nodes were used, the performance was worse when a network link had to be used (for 8 processes or more). Here, the additional virtualization overhead comes into play, slowing down the network performance. This can be seen clearly when 16 VMs are used. Since each VM only had one CPU core assigned, MPI communication had to use the VM's network interface for every message. To gain the best MPI performance in a cluster, as few VMs as possible should be used with as many CPU cores assigned as possible. Other measurements performed by Youssef et al. [248] regarding MPI performance in virtualized environments used some memory management functions in Xen to achieve better performance in VMs than in physical machines in particular measurements.

5.2.5 Summary

This section has shown that the overhead using virtualization technology depends on the kind of computational tasks performed within the different VMs. It has been shown that Xen's hardware-assisted para-virtualization approach promises the least overhead compared with a full virtualization approach as used in VirtualBox. Due to this fact as well as Xen's ability to support older hardware that does not provide hardware-aided virtualization (and therefore is not supported by KVM), it has been chosen as basis for the presented approach.

The approach to use multi-layered file systems in order to save space on the compute nodes' hard disks and to minimize transfer time when VM images

are shared across multiple resource sites has been examined with regard to the overhead caused by the utilization of a stackable file system.

It has been shown that there is nearly no decrease in performance in data transfer rates when using VMs instead of physical machines that use a Gigabit Ethernet network interconnection. The performance degradation when communication is performed by the widely-used MPI library contradicts this finding. Measurements have been conducted using different configurations. While CPU-intensive computation is not really affected by the virtualization abstraction layer, the performance when performing intensive communication over network links is notably affected by the additional abstraction layer introduced by virtualization. Local CRMs providing the utilization of VMs as computing environments must ensure that maximal hardware possible is assigned to each of the VMs when executing parallel tasks in order to minimize communication over network links.

5.3 Image Creation Station (ICS)

The main task of the ICS is to enable its users to create and modify VMs themselves. It is crucial that VMs can be created in a fast and efficient manner. Moreover, these VMs must contain some basic configuration to use infrastructure services of a particular resource site such as network shares or gateways to the public network. Users should not only be able to create new VMs, but also to import VMs into the existing system. Moreover, being able to export VMs to the user's local resources is important to avoid the kind of data lock-in from which current Cloud systems are suffering. Finally, the ICS must also provide functionality to delete formerly created or imported VMs.

Table 5.6: System configuration of the computer hosting the ICS instance.

Hardware:	Intel Core2 Duo, 2x 2.4 GHz 2 GB main memory 250 GB hard disk Gigabit Ethernet
Software:	Debian GNU/Linux 5 "lenny" Linux kernel 2.6.26-2-xen-amd64 Xen 3.2 Java 1.6 Update 20 ICS 0.9.14
VMs:	Debian GNU/Linux 5 "lenny" Linux kernel 2.6.26-2-xen-amd64 128 MB main memory 2 GB "sparse image"

For each of the tests, an ICS instance installed on the computer system pre-

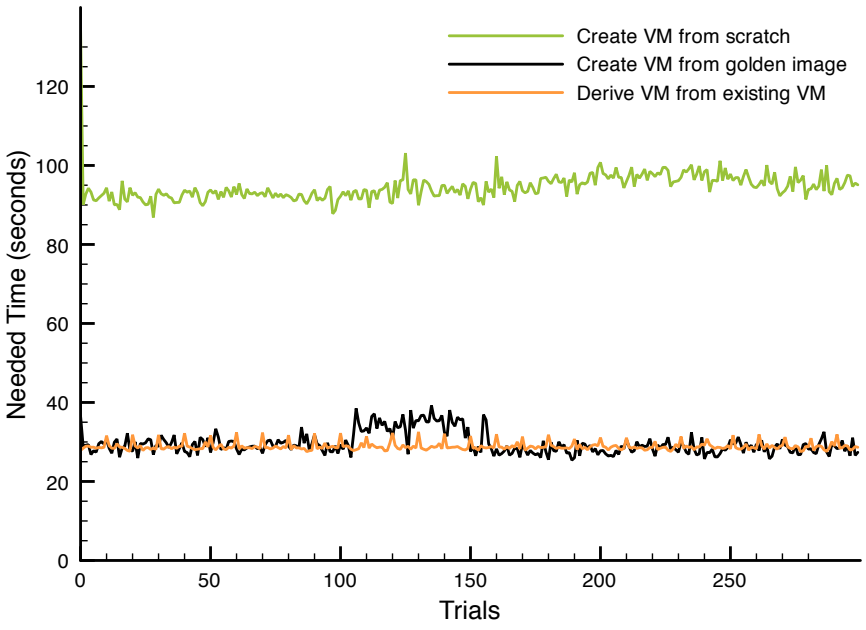
sented in Table 5.6 was used. It is notable that no multi-layered images were used to perform the tests since the multi-layered technology makes it impossible to create VMs from scratch. For the tests in this section, a basic Linux installation was performed which did not hold any user-specific data.[68] Additionally, the VMs' main memory size of 128 MB is negligible since the VMs were not started. Even if so, 128 MB of main memory would be enough for a basic Linux system in the VMs as used for testing.

The system was connected to the German research network (X-WiN) with a Gigabit Ethernet network link. Each of the different ICS measurements was repeated 300 times to calculate a robust mean. Moreover, by performing 300 VM management operations within a minimum amount of time has proven the system to be stable. During the testing period, the system did not suffer a single error.

5.3.1 Creating VMs

Before users can execute their first job, they need to create a VM as an execution environment. Figure 5.7 shows the time needed to create VMs with the ICS using the different image creation mechanisms.

Figure 5.7: Time needed to create VMs with the ICS using the different creation mechanisms.



First, a VM was created from scratch. That means that software packages used to install the OS to the VM were fetched from the official Debian FTP servers. Creating a VM this way took 94.53 seconds on the average (σ = 3.56 seconds). The first trial of this series of measurements took 134.69 seconds, which is a difference of 42.48%. The local caching mechanism of

[68] If multi-layered disk images would have been used, the user-specific layers would have stayed empty, so empty files would have been generated representing the user-specific layer. This would have lasted only fractions of a second.

`debootstrap` which is used to create the VM image caused this outlier. During the first creation process, the software packages were fetched from the public Debian FTP server. The caching mechanism stored the packages on the local hard disk of the system and reused the already downloaded packages in the following measurements. Ignoring this first outlier, the mean value dips to 94.39 seconds (σ = 2.7 seconds).

In the second series of measurements a golden image was used. A golden image can be seen as all of a VM's data packed into a single tarball. Instead of fetching the necessary software packages from a remote server, the needed data will be extracted from this tarball which is stored on the local hard disk of the system. This speeds up the VM creation process by a factor of 3. Creating a new VM from a golden image took 29.57 seconds on the average (σ = 2.8 seconds).

In the last series of measurements, the new VM was derived from an existing one that was provided by another user (an image provider). To derive a VM from an existing one, the image file of the source VM just needs to be copied. By using a sparse file with a (maximum) size of 2 GB, only existing data of the disk image is copied instead of an empty disk image which actually uses 2 GB on the hard disk. On the average, the copy process took 28.87 seconds (σ = 1.02 seconds) for a 2 GB sparse file with 431 MB of data.

Table 5.7: Time needed to create VMs with the ICS.

	From scratch		**Golden image**	**Derive from VM**
Time	94.53 s	94.39 s	29.57 s	28.87 s
σ	3.56 s	2.7 s	2.8 s	1.02 s

Table 5.7 shows that using a provided VM to derive new VMs speeds up the image creation process remarkably. Instead of needing around 94 seconds to create a new VM from scratch, creating a basic VM from either a golden image or by deriving the new VM from a provided VM would need less than 30 seconds. The difference between these two methods is that images derived from readily installed and configured VMs can contain a (complex) software stack matching special purposes while a golden image must not contain special software since it is used as basis for every VM, not derived from an existing one.

Amazon Elastic Compute Cloud (EC2) VM Creation

The aforementioned Amazon EC2 is probably the best-known example for IaaS Cloud computing offerings. Therefore, a number of measurements have been conducted to examine the time needed for VM creation. The measurements were conducted during the last week of March 2011. During the period of one week, one VM was requested every hour which results in 168 measurements that were used to calculate a robust mean. Moreover, by spreading the measurements over one week, changes in response time could be recognized.

Amazon VMs are characterized by different instance types. For the measurements, VMs were created using the Small Instance template which provides 1.7 GB of main memory, 1 virtual EC2 compute unit core[69], 160 GB of storage, a 32-bit architecture and moderate I/O performance. This instance type is the default template used to create new VMs.

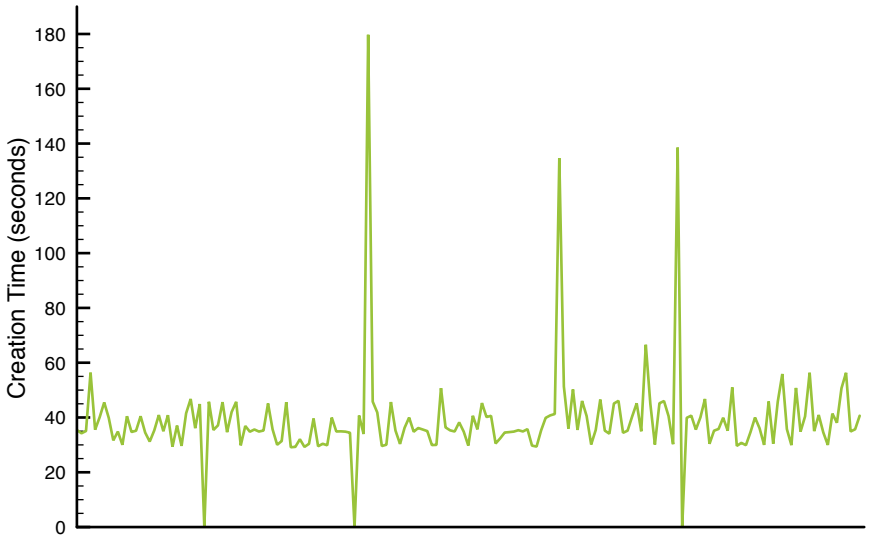


Figure 5.8: Time needed to create VMs on Amazon EC2 using the Small Instance template.

Figure 5.8 shows the results of the measurements. It can be seen that there is a large variance in the results. The image creation process ranges from 29.134 seconds to 179.728 seconds with six exceptional outliers: The VM creation process could not be satisfied three times and was very slow three other times.

Table 5.8: Time needed to create VMs on Amazon EC2 using the Small Instance template.

	VM creation
Time	39.071 s
σ	17.19 s

Table 5.8 provides the average of these measurements: 39.07 seconds after requesting a new VM on Amazon EC2, the user could access the machine (σ = 17.19 seconds). There was no particular time in which EC2 responses were exceptionally slow. The three eye-catching outliers occurred Sunday morning at 4 CEST (179.728 seconds), Monday evening at 22 CEST (134.63 seconds) and Wednesday at midnight CEST (138.534 seconds). The request could not be fulfilled on Friday afternoon at 15 CET, Sunday at midnight CET and Wednesday night at 1 CEST.[70]

Compared to the time needed to create a new VM image on the ICS, the proposed solution out-performed the Amazon Cloud service when using a golden

[69] One EC2 compute unit provides the equivalent performance of an 1.0 – 1.2 GHz AMD Opteron or Intel Xeon processor.

[70] Please note that two different time zones (CET and CEST) are given since the clock was changed on Sunday, March 27 at 2 a.m.

image (which took 29.57 seconds) or when the new VM image is derived from an existing one (which lasted 28.87 seconds) by around 25%. If no golden image is provided by the ICS administrator, Amazon's solution was 2.5 times faster than the ICS, which needed 94.53 seconds to create an image. Since a golden image can easily be provided by each administrator, the ICS can be seen as the better solution in terms of performance for VM image creation.

5.3.2 Importing and Exporting VMs

The importation process is divided into two parts. First, the disk image to be imported must be transferred to the computer hosting the ICS image pool. Second, the disk image must be converted from the vmdk format as used by the Open Virtualization Format (OVF) to Xen's raw image format. In addition, changes must be performed to the imported image to enable the VM to work with the site's infrastructure. The performance measurements focused on the second part of the importation process since the first part heavily depends on the user's network connection.

Instead of creating new VMs and installing the software needed, users may have local VMs that are already configured and ready to be used. The ICS allows users to import the disk images from their own system to the Grid. The other way round, VMs, that have been used successfully for job execution can be exported from the ICS to the user's local machine and can be executed with the local virtualization technology such as VMware or VirtualBox.

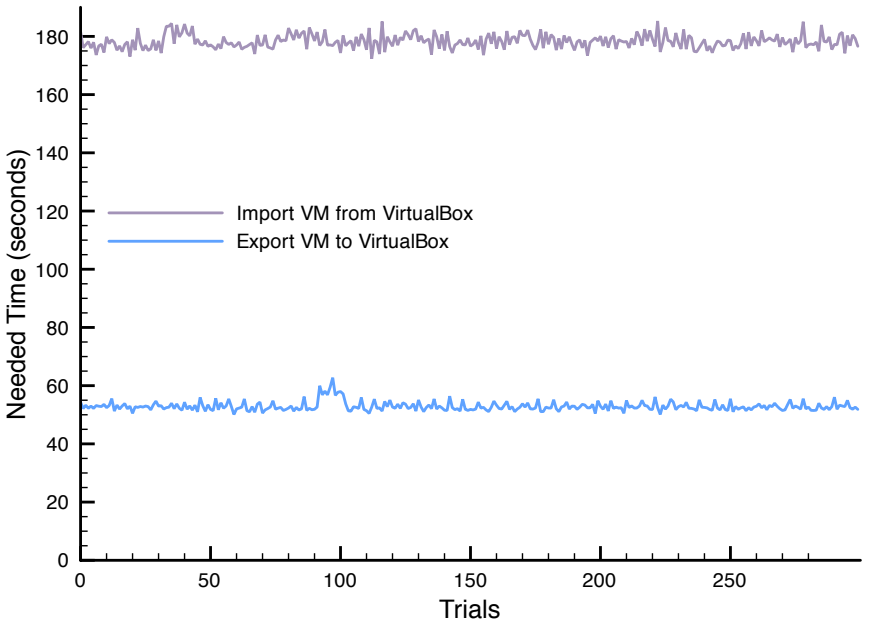


Figure 5.9: Time needed to import VMs to the ICS and to export VMs to the user.

Figure 5.9 shows the results of these measurements. Importing a VM from an existing disk image took 178.29 seconds on the average ($\sigma = 2.45$ seconds). This long amount of time is caused by the processes necessary to extract the

data from the vmdk file by parsing the vmdk's partition table, extracting the correct partition and converting it to the raw format. After that, a file system check is performed to ensure that all data has be converted successfully.

Compared to the importation process, exporting a VM disk image is much faster. On the average, conversion of a disk image from raw to vmdk format took 52.91 seconds (σ = 1.61 seconds). The fact that both VirtualBox and VMware can deal with vmdk files that do not contain a partition table makes it unnecessary to create a partition table and copy the data into an appropriate vmdk file, which significantly speeds up the export process. Table 5.9 summarizes the determined results.

Table 5.9: Time needed to import and export VMs.

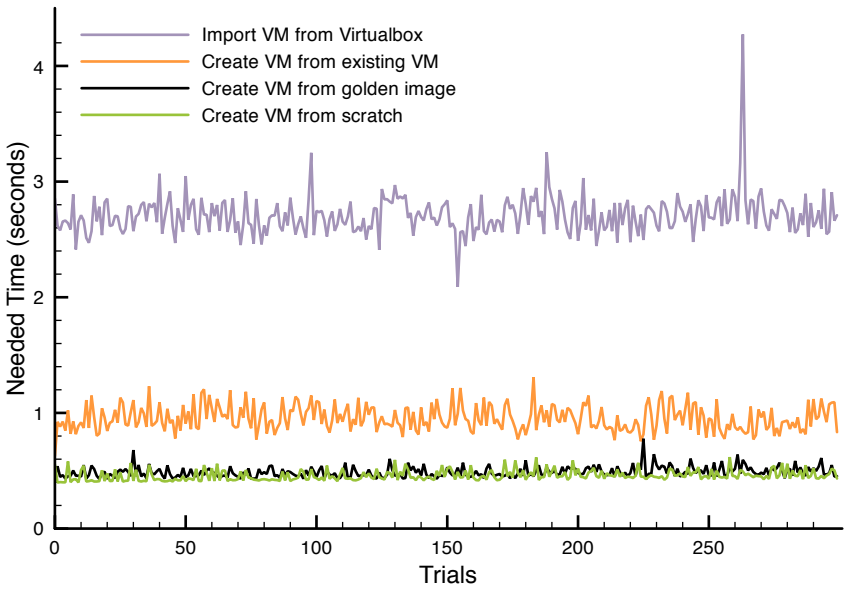
	Import VM	**Export VM**
Time	178.29 s	52.91 s
σ	2.45 s	1.61 s

Although importing and exporting a VM disk image took longer than creating a new VM, it can avoid having to set up VMs from scratch, which saves time if a complex software stack already exists in a VM on the customer's computers. As stated above, providing importation and exportation functionality supporting the wide-spread OVF format overcomes the data lock-in effect that exists in many current Cloud systems.

5.3.3 Removing VMs

Finally, functionality to delete VMs from the ICS is provided. Figure 5.10 shows the time needed to delete VMs that were created or imported at an earlier point in time.

Figure 5.10: Time needed to remove created or imported VMs from the ICS.



Time needed to delete a VM is obviously connected to the method used for

creating the VM. Deleting a VM that was created from scratch took 458.54 ms on the average (σ = 45.86 ms) which is slightly faster than deleting a VM that was created by using a golden image, which took 488.32 ms (σ = 46.4 ms). If the VM was derived from another VM, the deletion process took nearly twice as long, 954.13 ms on the average (σ = 110.53 ms). Deleting a previously imported VM lasted 2707.81 ms (σ = 168.79 ms). The values are summarized in Table 5.10.

Table 5.10: Time needed to remove VMs from the ICS.

	From scratch	**Golden image**	**Derived from VM**	**Imported**
Time	458.54 ms	488.32 ms	954.13 ms	2707.81 ms
σ	45.86 ms	46.4 ms	110.53 ms	168.79 ms

The differences between the deletion processes are highly unexpected since the processes do not differ from each other. A closer look at the logfile shows that the check if a VM is running, which is performed before deleting the files on the disk, took around 0.7 seconds if the VM was derived from another VM and about 1.5 seconds if the VM was imported. In contrast, if the VM was created from scratch or using a golden image, this check only took about 0.2 seconds. When dealing with imported VMs, an additional slowdown could be identified when removing a VM's files from the local hard disk: removing the files took around 1 second in contrast 0.1 – 0.2 seconds in the other cases. The latter can be explained by the size of the imported image file which is bigger than the sparse images used in the other cases, while the first decrease in speed is based on the behavior of Xen's `xm` command, though the exact details for this decrease could not be determined.

5.3.4 Summary

This chapter has shown measurements dealing with VM management. First, the image creation process of the ICS was examined. The comparison to Amazon's EC2 has shown that the ICS outperforms Amazon's solution if a golden image is provided by the system administrator or a new VM is derived from an existing image provided by another user.

In contrast to Amazon's solution, the ICS supports importing and exporting VM images using the widely supported vmdk format. This process is rather slow due to the image conversion process, which is necessary because Xen does not support the vmdk format but uses a raw disk format. Nevertheless, it is negligible when taking the time needed to set up a complete VM a second time into account, which may very time consuming if the VM contains a complex software stack. The importation and exportation of VM images not only allows the use of existing Grid- and Cloud-VMs but also allows users to exchange VM images between different virtualization infrastructures and even Cloud service providers.

Moreover, VM deletion is a rather quick process but shows some strange behavior caused by the Xen virtualization platform, which slows down image

deletion by a factor of 5 if the image was imported from the user's virtualization infrastructure.

5.4 Data Transfer

The speed of the network interconnections and the time needed to transfer a VM image between different resource sites is crucial for the system's overall performance. This chapter presents measurements that deal with this aspect of the system. Not only the data transfer strategies are measured but also the overhead resulting from encryption and decryption of the transferred data will be examined.

5.4.1 Data Transfer Strategies

To evaluate the performance of the different data transfer mechanisms, several measurements have been performed. The test environment consists of 10 machines with the identical hardware configuration shown in Table 5.11. The duration of the VM image distribution process to the 10 hosts was examined. A set of 100 measurements has been conducted in order to get a robust mean.

Table 5.11: System configuration of the 10 nodes used for the evaluation of data transfer mechanisms.

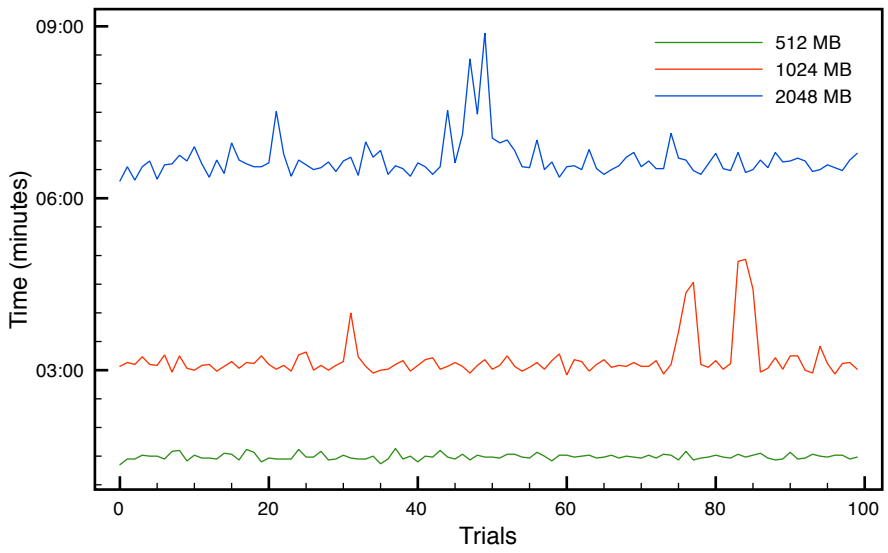
Hardware:	Intel Xeon E5420, 2x Quad Core 2.5 GHz 16 GB main memory 73 GB hard disk Gigabit Ethernet
Software:	Debian GNU/Linux 4 "etch" Linux kernel 2.6.18-6-xen-amd64 Xen 3.0.2

First, the time needed to transfer different images sizes using a traditional transfer mechanism was examined. In this case, data is transferred from one host to another simply via a network connection. The VM image file is distributed sequentially to the other hosts. The time needed to deploy images of different sizes is shown in Figure 5.11. It is notable that data encryption was not used during these measurements.

In Figure 5.12, the deployment times of BitTorrent are shown for transferring files of 512 MB, 1024 MB and 2048 MB. The host which holds the VM image file to distribute creates a torrent file which is read by the other hosts. These hosts concurrently begin to fetch the image from the source host and provide already downloaded data to other hosts. Therefore, a host can fetch data fragments from many different hosts which speeds up the data transfer process. This is why BitTorrent clearly outperforms the Unicast deployment method in this setup.

Table 5.12 shows the average transfer times for the different image sizes

Figure 5.11: Deployment times for different data sizes using Unicast.



and transfer methods. It is obvious that BitTorrent is the better method to transfer images between different hosts if there are several hosts that can act concurrently as data providers.

Table 5.12: Measured average durations of BitTorrent and Unicast deployment mechanisms and the time needed to create a BitTorrent file for a user layer of a particular size.

Image Size	Unicast	BitTorrent	BT creation
128 MB		13.6 s	1.655 s
256 MB		18.4 s	3.263 s
512 MB	89.4 s	29.0 s	6.434 s
1024 MB	191.1 s	62.3 s	12.796 s
2048 MB	401.0 s	120.9 s	25.903 s

While both Figures 5.11 and 5.12 provide only transfer times for images with a size of 512 MB, 1024 MB and 2048 MB, values for transferring smaller files (e.g., when using multi-layered file images) via BitTorrent can be found in this table. Nevertheless, it can be seen that the image deployment process of files with a size of 512 MB took as three times as long when using Unicast instead of BitTorrent (89.4 seconds when using Unicast compared with 29 seconds when using BitTorrent). The advantage of using BitTorrent instead of Unicast is even more evident the bigger the file to transfer is. When distributing files with a size of 1024 MB, only 62.3 seconds were needed when using BitTorrent instead of 191.1 seconds when using Unicast. In case of files with a size of 2048 MB, Unicast needed 3.3 times longer than BitTorrent (401 seconds compared to 120.9 seconds). As stated above, a torrent file needs to be generated for every distribution process when using BitTorrent, thus the time needed for the generation of the torrent file must be added to the actual transfer time. The generation time grows with the size of the file. Creating a torrent file for a 128 MB user layer took 1.7 seconds on the average, for a 256 MB user layer 3.3 seconds were needed. Generating the file for a 512 MB user layer took 6.4 seconds, for a 1024 MB user layer 12.8 seconds were needed, and to generate a

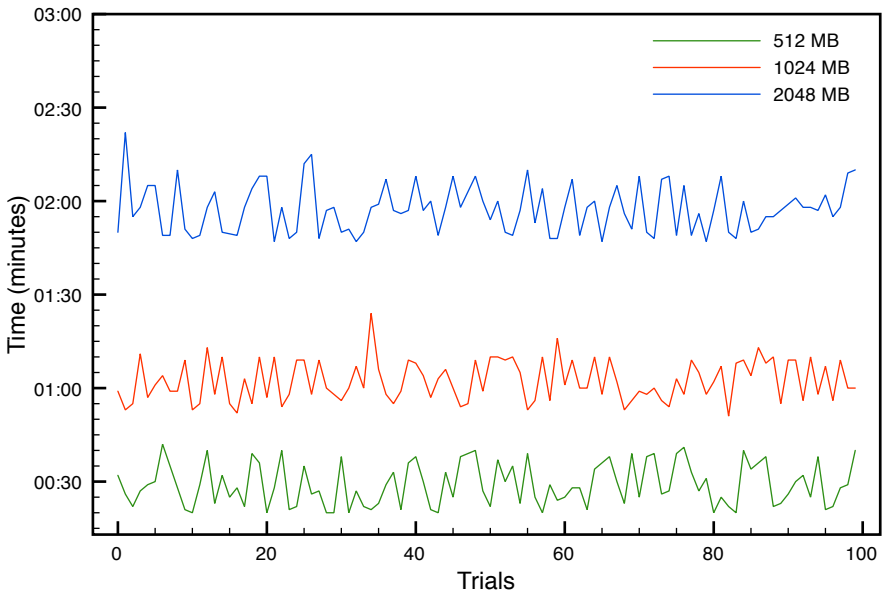


Figure 5.12: Deployment times for different data sizes using BitTorrent.

torrent file for a 2048 MB user layer 25.9 seconds were needed on the average. Thus, even when taking the generation time for the torrent file into account, BitTorrent clearly outperforms Unicast in a multi-site environment.

5.4.2 Inter-Site VM Image Exchange

On the other hand, BitTorrent suffers from its infamy as a technology widely used in file sharing networks. For this reason, many sites (e.g., universities) prevent the use of the ports needed to use BitTorrent. Therefore, image deployment measurements have been made using only Unicast data transfer technology. To evaluate the proposed approach in a public setting, a set of 50 measurements was conducted, transferring different layer sizes to another Grid site using a public network. Both Grid sites are operated by public universities interconnected by the German research network X-WiN that provides a 450 MBit/s network link. Because of the fact described above and the setup in which only two resource sites are connected, only unicast measurements can be conducted.[71] The hardware specifications of the Grid headnodes used to transfer the data is shown in Table 5.13.

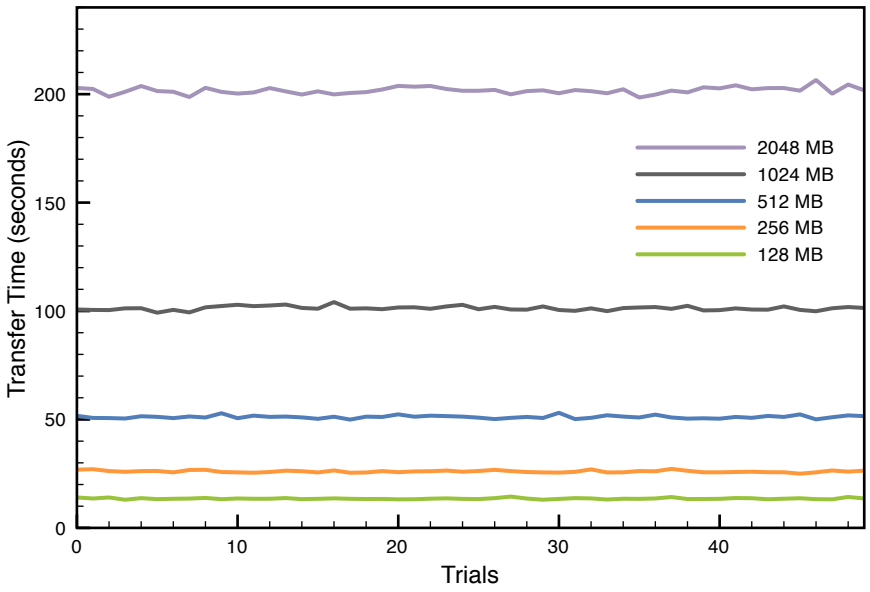
The results of the measurements are shown in Figure 5.13 and Table 5.14. A small 128 MB layer needed around 13.5 seconds to be transferred from one Grid site to the other. When transferring a layer that has twice the size (256 MB), nearly twice the time was needed (26 seconds). Transferring a 512 MB image took 51.15 seconds, while transferring a 1024 MB image took 101.3 seconds. The transfer of 2048 MB required 201.7 seconds, which corresponds to an average transfer rate of 10.15 MB/s (81.23 MBit/s).

[71] The advanced data transfer technology of BitTorrent is only useful in settings with more than two different hosts.

Table 5.13: System configuration of the headnodes used for the evaluation of data transfer mechanisms between the two Grid sites.

	Grid Site A	Grid Site B
Hardware:	AMD Opteron 2216 HE (2x Dual-Core 2.4 GHz) 16 GB main memory 250 GB hard disk Gigabit Ethernet	AMD Opteron 254 (2x 2.8 GHz) 4 GB main memory 80 GB hard disk Gigabit Ethernet
Software:	Debian GNU/Linux 4 “etch” Linux kernel 2.6.18	SuSE Linux Enterprise Linux Linux kernel 2.6.18

Figure 5.13: Transfer times of layers with different sizes between two public Grid sites using a X-WiN network connection.



5.4.3 Encryption and Decryption of Data

Since data encryption is useful when transferring data over potentially insecure network links like the internet, additional costs come into play for data encryption before the data transfer and data decryption afterwards.

Figure 5.14 shows the time needed to encrypt a plain data file using OpenSSL. To calculate a robust mean, 30 measurements were conducted. It can be seen that the time needed to encrypt a file increases arithmetically dependent on the size of that file. While a 128 MB file could be encrypted within 1.926 seconds on the average, encryption of a 256 MB file lasted 3.8 seconds. To encrypt a 512 MB file, 7.754 seconds were needed, encrypting a 1024 MB file took 16.259 seconds and encryption of a 2048 MB file took 33.459 seconds on the average.

Once the data transfer has finished, the received file must be decrypted on the target computer. Figure 5.15 shows the time needed to decrypt the different file sizes. Compared to the encryption times in Figure 5.14, the decryption process is slightly slower. To decrypt a 128 MB disk image, 2.03 seconds were needed

Table 5.14: Measured average values of Unicast deployment mechanism between two Grid sites connected with the X-WiN.

Image Size	Unicast
128 MB	13.50 s
256 MB	26.00 s
512 MB	51.15 s
1024 MB	101.3 s
2048 MB	201.7 s

Table 5.15: Hard- and software configuration used for the data encryption and decryption measurements.

Hardware:	AMD Opteron 2216 HE (2x Dual-Core 2.4 GHz) 16 GB main memory 250 GB hard disk Gigabit Ethernet
Software:	Debian GNU/Linux 4 “etch” Linux kernel 2.6.18-6-xen-amd64

on the average. To decrypt a 256 MB disk image, 4.076 seconds were needed while decryption of a 512 MB file took 8.47 seconds. If the disk image had 1024 MB, the decryption process needed 17.506 seconds to be completed, if the disk image had 2048 MB, decryption took 34.9 seconds.

When using data encryption, the time for encryption and decryption must be taken into account. The overhead results from the sum of the times needed to encrypt and decrypt a file of a particular size. Table 5.16 sums up the presented measurements and provides the overall time needed to put data encryption into place (overhead).

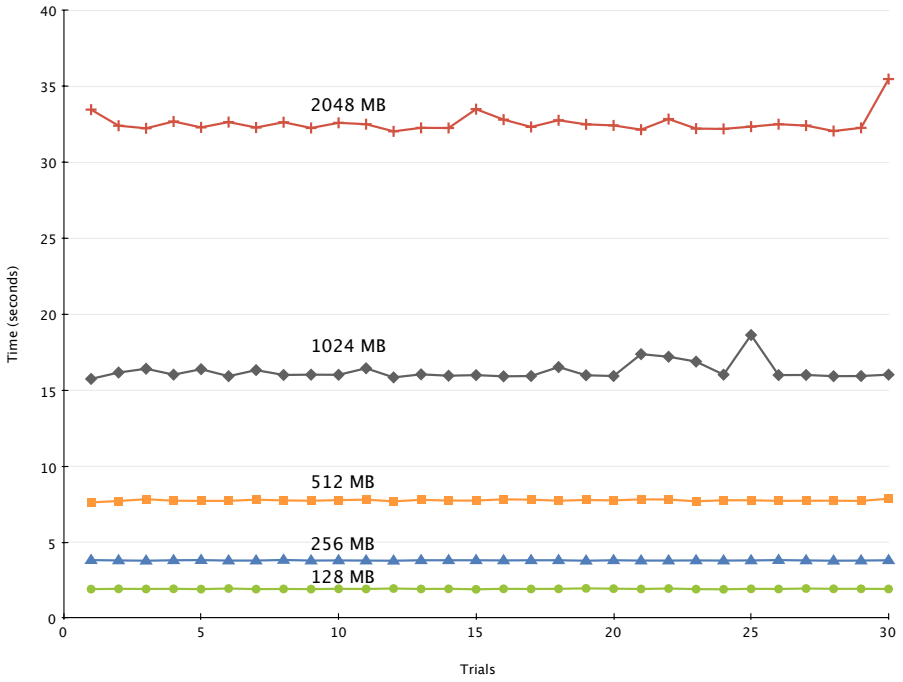
Table 5.16: Overview of the time needed to encrypt and decrypt files of different sizes.

Size of file	Encryption	Decryption	Overhead
128 MB	1.926 s	2.031 s	3.957 s
256 MB	3.800 s	4.076 s	7.876 s
512 MB	7.754 s	8.470 s	16.224 s
1024 MB	16.259 s	17.506 s	33.765 s
2048 MB	33.459 s	34.903 s	68.362 s

It is shown that it took an additional 4 seconds to securely encrypt a 128 MB data file during its copy process. When working with a 256 MB file, the overhead for encryption and decryption doubled almost to 7.9 seconds. The overhead when transferring a 512 MB file was 16.2 seconds while 33.8 seconds were needed when encrypting and decrypting a 1024 MB file. Finally, the cryptographic operations took 68.4 seconds when working with a 2048 MB disk image.

The green line presented in Figure 5.16 shows the dependency between the file size and the time needed to perform the cryptographic operations. The dashed light gray line shows the function $f(x) = 0.03125x$ to depict the almost linear increase of costs depending on the file size to be encrypted and decrypted.

Figure 5.14: Time needed to encrypt files of different sizes.



5.4.4 Firewalls

The utilization of firewalls is common in distributed systems to provide maximum security to the participating computers. Therefore, firewalls guard the incoming network traffic and reject unauthorized data transfer. Firewalls not only observe incoming network traffic but can also be used to prevent the utilization of particular (network) services establishing a connection from the secured part of the network to the public. As usual, there is a tradeoff between security and usability. If the firewall allows nearly every network traffic, the users are free to decide which services they would like to use. If the firewall is configured in a very restrictive way, users cannot use every software they may wish to use. On the other hand, the more restrictive a firewall is, the more secure the systems behind a firewall are.

As described in Section 3.2.1.1.5, firewalls are not only used to secure the users' VMs but also provide the possibility to forward incoming network traffic to another host in the network, allowing users to connect to their running VMs from the outside even if the VMs are not accessible directly from the outer network. Moreover, as described in Section 3.2.6, firewall rules can be deployed dynamically respecting the role of a user. If a particular user is untrusted by the local system administrator, network traffic can be restricted in order to prevent misuse of the provided resources. If a particular user is well known to the system's administrator, network traffic can be less restrictive even allowing particular users to access local network shares. For example, this decision can be made automatically based on the VO-membership of a particular user.

The local Grid headnode equipped with hardware and software shown in Ta-

Figure 5.15: Time needed to decrypt files of different sizes.

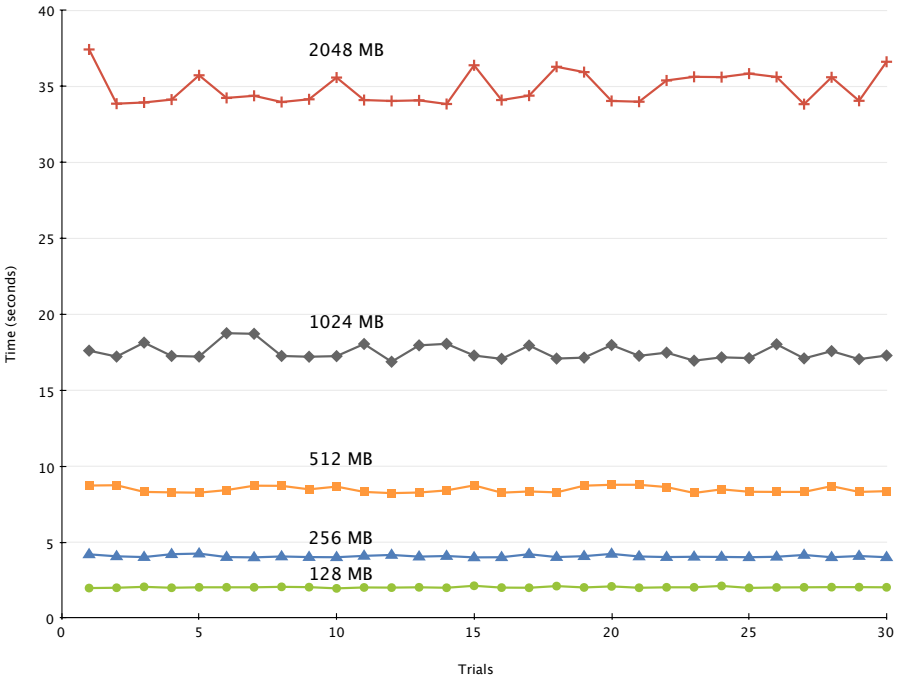
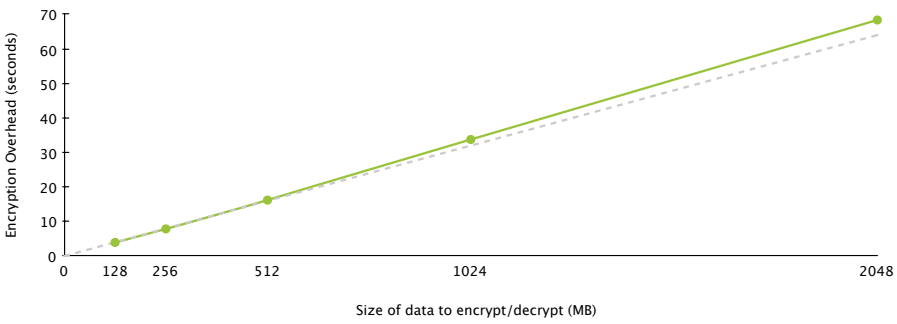


Figure 5.16: Time needed for data encryption and decryption grows linearly for files of different sizes.



ble 5.17 has been used as firewall host which provides firewall rules to prevent unauthorized access and to ensure accessibility of booted VMs on the ICS to their owners from the outside.

The impact on network performance of a different number of `iptables` firewall rules is shown in Figure 5.17. It can be seen that nearly no performance degradation can be registered when using less than 500 firewall rules. The ICS will apply one rule per active VM such that there can be many VMs booted before running into performance problems of the network, even if a number of additional static rules are applied to ensure the secure operation of the host. In fact, the Parallels Virtuozzo[72] developers recommend to not apply more than 200 to 300 iptables rules to their system.[73]

[72] `http://www.parallels.com/de/products/pvc46/`
[73] `http://download.swsoft.com/virtuozzo/virtuozzo4.0/docs/en/lin/VzLinuxUBCMgmt/18786.htm`

Table 5.17: Hardware and software configuration used for measurements of firewall rule deployment.

Hardware:	AMD Opteron 2216 HE (2x Dual-Core 2.4 GHz) 16 GB main memory 250 GB hard disk Gigabit Ethernet
Software:	Debian GNU/Linux 4 “etch” Linux kernel 2.6.18-6-xen-amd64

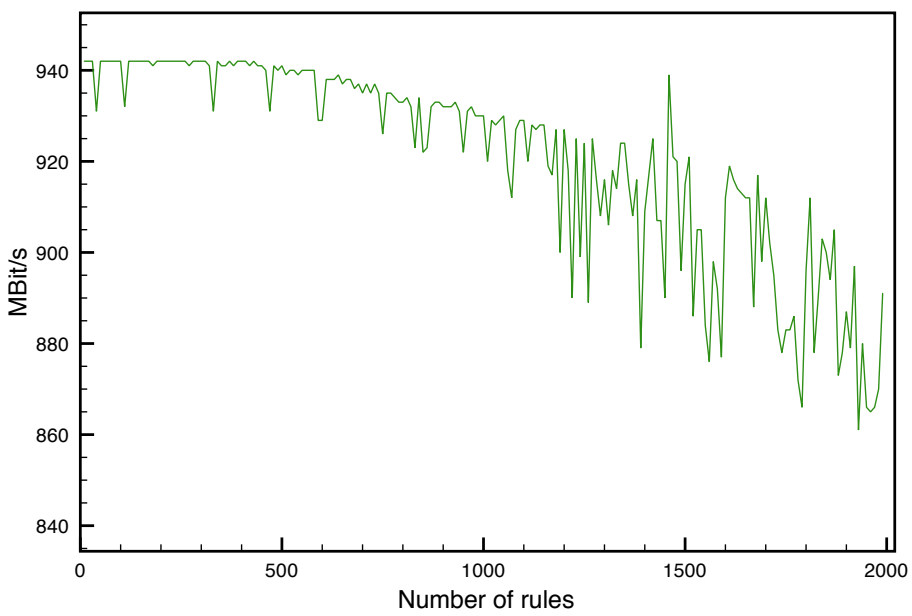


Figure 5.17: Network performance when using different numbers of firewall rules.

5.4.5 Summary

In this section, performance issues regarding VM and data transfer have been examined. First, data transfer strategies have been presented which can be used to deploy ICS VM images to local compute nodes or to the XGE. Besides a basic Unicast implementation, a more sophisticated implementation based on BitTorrent has been presented which clearly outperforms the Unicast deployment mechanism.

Thereafter, the network performance between two Grid sites connected by the X-WiN was examined. During these measurements, only the network performance between physical hosts connected by the research network was examined. Taking the measurements performed in Section 5.2.2 into account, it can be assumed that the use of virtualization technology does not negatively affect the data transfer rates between nodes connected with the X-WiN network. This allows users to access the X-WiN network with native performance when applying software installation on their VMs running on ICS instances of the different Grid sites.

Regarding security, the costs associated with encrypting and decrypting disk

images have been examined. It has been shown that the use of data encryption would result in an additional overhead which grows linearly depending on the particular VM image's size.

Finally, the degradation in network performance when using iptables firewall rules to ensure access to the VMs from the outside was examined. Nearly no overhead could be found when using less than 500 firewall rules, which is quite sufficient for the proposed solution.

5.5 Elasticity and Scalability

The proposed solution aims to provide a scalable system which can be used to connect existing (HPC) computing resources together and to provide a system that automatically adapts to its current load. Moreover, users can decide which resources are allowed to be used for executing their computational tasks. The measurements were performed on the hardware presented in Table 5.18.

Table 5.18: System configuration of the node used for the elasticity and scalability benchmarks.

Hardware:	Intel Core2 6600, Dual Core 2.4 GHz 2 GB main memory 250 GB hard disk
Software:	Debian GNU/Linux 6 "squeeze" Linux kernel 2.6.32

5.5.1 Time Needed to Scale

It is crucial that the system provides efficient scaling and scheduling mechanisms. To analyze the proposed solution, several measurements have been performed. A different number of jobs was submitted to dispatchd in order to examine its scheduling decisions. In order to focus only on the time needed for scaling and scheduling, simple jobs have been submitted that wait 60 seconds before quitting. Each of the measurements was repeated 50 times to calculate a robust mean.

Table 5.19: Time needed to submit different numbers of jobs to dispatchd.

Number of jobs	Time needed	σ
1	0.045909 s	0.040994 s
2	0.113866 s	0.293279 s
4	0.271751 s	0.112893 s
8	0.410093 s	0.279604 s
16	0.453746 s	0.279518 s
32	0.504081 s	0.002928 s
64	0.679984 s	0.014711 s
128	1.042323 s	0.013324 s

Table 5.19 shows the time needed for job submission to dispatchd. Submitting a single job took 0.05 seconds on the average (σ = 0.04 seconds) while

submitting 2 jobs took 0.11 seconds (σ = 0.29 seconds). Submitting 4 jobs took twice as long as submitting 2 jobs (0.27 seconds with σ = 0.11 seconds). Submitting 8 jobs took 0.41 seconds (σ = 0.28 seconds), submitting 16 jobs took only a little longer (0.45 seconds, σ = 0.28 seconds) and submitting 32 seconds took 0.5 seconds (σ = 0.003 seconds). Finally, submitting 64 jobs took 0.68 seconds (σ = 0.01 seconds) while submitting 128 jobs took 1.04 seconds (σ = 0.01 seconds). The time needed for job submission is independent on the kind of job submitted by the user. The dispatchd job submission procedure does not analyze the job template file, which is needed by GridWay later, but does only checks if command line arguments exist which specify the number of needed resources or which resources are allowed to be used.

It has been shown that job submission is fast enough to satisfy the users' needs even if a user submits a large number of jobs e.g., by using a shell script.

Table 5.20: Resource sets connected to dispatchd used for job submission.

	Server	**Cluster**	**Cloud**
Number of hosts	1	15	virtually unlimited
Number of cores	4	4	1
Network Bandwidth	100 MBit/s	100 MBit/s	100 MBit/s

Once submitted to dispatchd, its scheduler tries to detect resources suitable to match each job's needs. This scheduler loop runs every 10 seconds. The dispatchd used in this experiment was connected to three different resource sets shown in Table 5.20. As stated earlier in Section 3.2.2, each resource set is managed by a GridWay instance. These instances were hosted on hosts in the local network connected by a Fast Ethernet network connection. Since primitive jobs have been submitted to perform these tests, no data must be transferred to and from the hosts. Therefore, the network connection can be neglected.

Table 5.21: Time needed to submit computational tasks to remote resources by dispatchd.

	Server	**Cluster**	**Cloud**
Time needed	9.082942 s	9.749376 s	16.56238 s

Table 5.21 shows the time between job submission at the dispatchd host and the submission command performed by dispatchd on the GridWay host responsible for the particular resource. Since dispatchd tries to submit jobs on the lowest possible layer, these measurements have been performed by submitting 128 jobs to dispatchd such that the resources of all resource sets are employed for job execution. 9.08 seconds after the user's submission command, dispatchd submitted the job to the GridWay instance connected to the local workgroup server. Shortly after submitting 4 jobs to this GridWay instance (the server has 4 CPU cores available for job execution), dispatchd submitted the first job to the GridWay connected to the cluster. This happened 9.75 seconds after job submission to dispatchd and 0.67 seconds after dispatchd submitted the first job to the server. The next 60 jobs (the cluster has 15 hosts with 4 CPU cores each) were submitted to this GridWay instance before dispatchd scaled

out to the Cloud to meet the demands of the pending jobs. 16.56 seconds after job submission and 6.81 seconds after the first job submission to the cluster, dispatchd submitted the first job to the Cloud resource set.

The delay during the submission process is caused by dispatchd's internal scheduler runloop interval of 10 seconds. It can be reduced by decreasing the interval, which would result in a higher CPU load on the host running dispatchd. Nevertheless, the measurements have shown that dispatchd adapts quickly to job submission from users and quickly submits the jobs to the remote resource sets. With respect to long-running jobs, the scheduling overhead caused by dispatchd can be neglected.

5.5.2 Costs of Scale

The Communication to Computation Ratio (CCR) was proposed in Chapter 2.4.8, which is used to calculate the ratio of calculation costs and data transfer and storage costs. In order to calculate the real expenses of a computational task, at least three components must be taken into account: (1) costs for CPU usage as well as (2) data transfer and (3) storage costs.

Let $J = \{j_1, j_2, \ldots, j_n\}$ be a set of tasks that should be executed and let $r(j)$ be the function that returns the runtime of task j in hours. Moreover, let $n(j)$ be the function that returns the number of Computing Elements (CEs) needed to execute j_i and let c_{CE} define the costs of using one CE for one hour. The computational costs can be calculated by the following equation:

$$c_{calc} = c_{CE} \sum_{j \in J} n(j) \lceil r(j) \rceil \tag{1}$$

Let $I_j = \{i_{j,1}, i_{j,2}, \ldots, i_{j,k}\}$ be the set of input files that have to be transferred to the remote host and $O_j = \{o_{j,1}, o_{j,2}, \ldots, o_{j,l}\}$ be the set of output files that have to be transferred back to the user once task j is completed. Let $s(f), f \in I_j \cup O_j$ be the function that returns file f's size in gigabytes. Moreover, let c_{in} define the costs of transferring 1 GB of data to the remote resources and let c_{out} define the costs of transferring 1 GB back to the local workstation. Then the data transfer costs can easily be calculated by the following formula:

$$c_{transfer} = c_{in} \left\lceil \sum_{j \in J} \sum_{i_j \in I_j} s(i_j) \right\rceil + c_{out} \left\lceil \sum_{j \in J} \sum_{o_j \in O_j} s(o_j) \right\rceil \tag{2}$$

Finally, the calculation of storage costs is quite simple and needs not only to take input and output files into account but also temporary files created during task execution. Let $T_j = \{t_{j,1}, t_{j,2}, \ldots t_{j,m}\}$ be the set of files that are created temporarily during the execution of task j and $F_j = I_j \cup O_j \cup T_j$ the set of all files affected by execution of task j. Moreover, let c_s define the costs of storing one gigabyte for one month and let c_a define the costs for performing 1 million

data access operations (read and write). Moreover, let the function $a(j)$ return the number of necessary data access operations during task j's execution in millions. Assuming that every piece of data is deleted when a task is finished and the data is transferred back to the caller, the storage costs can be calculated using the following formula:

$$c_{store} = c_s \left\lceil \sum_{j \in J} \sum_{f \in F_j} s(f) \right\rceil + c_a \left\lceil \sum_{j \in J} \sum_{f \in F} a(f) \right\rceil \tag{3}$$

Therefore, the entire costs of executing a set of tasks on a particular resource site can be calculated by summing up the partial costs calculated by equations (1) – (3):

$$c = c_{calc} + c_{transfer} + c_{store} \tag{4}$$

Table 5.22: Calculation of costs for executing four sample jobs.

	Task 1	**Task 2**	**Task 3**	**Task 4**
Runtime	6 h	24 h	6 h	24 h
CEs	1	1	1	1
Size of input files	25 MB	25 MB	250 MB	250 MB
Size of output files	500 MB	500 MB	1 GB	1 GB
Size of temporary files	2 GB	2 GB	10 GB	10 GB
Data access operations	50'000	50'000	300'000	300'000
c_{calc}	\$0.51	\$2.04	\$0.51	\$2.04
$c_{transfer}$	\$0.10	\$0.10	\$0.10	\$0.10
c_{store}	\$0.30	\$0.30	\$2.10	\$2.10
c_j	\$0.91	\$2.44	\$2.71	\$4.24

Table 5.22 displays 4 sample tasks and presents the costs for executing each task. The calculation is based on the Amazon EC2 price list. Using a small standard instance in US East as used in Section 5.3.1, costs \$0.085 per hour. All incoming data transfer costs \$0.10 per GB while outgoing transfer costs \$0.15 per GB with the first GB free. Storage costs are \$0.10 per GB and month and accessing the storage costs \$0.10 per 1 million I/O requests.[74] Four sample tasks are presented differing in runtime or the amount of space needed during execution on the storage device.

One advantage of using EC2 is that the desired number of compute nodes can be acquired on demand. If a workload contains 100 jobs similar to Task 1's characteristics, the execution on a single workstation with one CPU core would last at least 600 hours. If the results must be available quickly, 100 EC2 instances can be acquired and used to execute the job during the next 6 hours. On the other hand, the user needs to pay around \$79.15 to get the results in time.

[74] `http://aws.amazon.com/ec2/pricing/`

5.5.3 Summary

This section has shown the ability of dispatchd to deal with dynamically changing load profiles that vary due job submissions from users. It has been shown that only a small overhead in job scheduling is caused by providing a dynamically scaling system to users who were previously forced to manually schedule their jobs to the desired resources. For typically long running computational jobs, these costs can be neglected.

Moreover, a cost function has been developed which can be used to estimate the costs of using a particular resource set. At the moment, dispatchd does not support specifying an amount of money which a user is willing to spend for job execution; however, this functionality can be easily implemented into dispatchd if the information needed for cost estimation is specified in the resource set's configuration file.

5.6 Summary

In this section, several tests have proven the functionality of the proposed system.

First, the performance of the chosen virtualization technology has been examined. It had been shown that the overhead caused by the additional virtualization layer differs depending on the workload characteristics, even if it is very minimal due to the para-virtualization approach used in Xen. During the runtime of the projects which formed the foundation of this thesis, several experts confirmed the necessity of the virtualization layer to ensure a strong user separation in multi-user computing systems.

It has been shown that the ICS can provide VMs faster than Amazon's EC2 and enable users to share these VMs automatically across different resource sites, a functionality that EC2 lacks. The time needed for automatic distribution of a VM image relies on the network links connecting the particular sites and the size of the VM image itself. Two different mechanisms have been presented to perform image distribution. Moreover, image distribution measurements using the German X-WiN network have been performed proving the usability of the presented approach in a Grid environment.

The performance impact of using firewalls in the system has also been examined. This is crucial when firewall rules are deployed dynamically e.g., in systems providing individually tailored security profiles to different users, depending on their level of trust specified by the local system administrator. To complete the measurements, deployment times of VM images to the XGE have been examined showing that the combination of ICS and XGE is a powerful combination that provides secure and personalized virtual execution environments in a multi-user computing system.

Finally, the responsiveness of the system has been proven by submitting a large number of jobs to dispatchd. It has been shown that the system reacts quickly to incoming jobs by dynamically making scheduling decisions and including all connected resource sets. In addition, a set of functions has been developed which allows to estimate the costs for executing computational tasks on a particular resource set. This set of functions could be used by dispatchd in the future to allow users to specify the maximum amount of money available for task execution.

6 Conclusions

"I think there's a world market for maybe five computers."
Thomas Watson, IBM chairman, 1943

6.1 Summary

Although Cloud computing has received a great deal of attraction in recent months, current Cloud infrastructures are often incompatible and hinder users from easily exchanging data and computations between different Cloud service providers. From an economic point of view, this might be beneficial for well-known providers, such as Amazon. However, it does not benefit users. Academically driven solutions such as Grid computing provide approaches that can be used to easily exchange data between different geographically distributed resources. Nevertheless, the administration overhead associated with Grid computing today is tremendous. Moreover, Grid users cannot easily install their desired software on the nodes used for computation. Therefore, the Grid has simply not been able to be used by end users up until today.

This thesis presents an approach to bring together the advantages of both Grid and Cloud computing and to overcome traditional limitations. On the one hand, virtualization technology used everywhere in the area of Cloud computing is brought to Grid computing. This not only results in a strong user separation accompanied by an increased level of security on the computational resources in the Grid, but also allows Grid users to install any desired software and even to choose the OS that should be installed on the resources available for computational tasks.

On the other hand, mechanisms have been put into place that enable users to exchange their virtualized execution environments – their VMs – among resources belonging to different administrative domains. Therefore, a component has been developed that supports importing and exporting VMs using a widely supported virtual disk format, which even allows users to use the VMs created in the system on their local workstations e.g., by using Desktop virtualization software such as VMware Workstation/Fusion or VirtualBox.

Therefore, several components have been developed:

Image Creation Station (ICS) The ICS is a component that not only enables users to create Virtual Machines (VMs) themselves but also provides mechanisms to manage VMs and even to import and export VM disk images.

During the image creation process, a user can choose between different methods. Users can start with a basic VM image containing only a minimal OS installation which allows them to install only the software they would like. In contrast, users experienced in system administration or software vendors with an installed software stack can provide configured VM images to other users. Deriving new VM images from provided images enables even inexperienced users or users who do not wish to spend any time with system installation and administration to easily create VMs on demand. Moreover, users can also import existing VMs e.g., from a locally installed desktop virtualization software such as VMware or VirtualBox. Besides that, existing VMs can also be exported from the ICS, allowing users to use their VMs within other virtualized infrastructures.

To perform management tasks, users can start VMs on the ICS. The ICS not only allows users to access their VMs by setting up appropriate firewall rules when needed but also provides Service interfaces to automatically perform software installation tasks e.g., from within BPEL workflows.

Finally, the ICS ensures system-wide availability of the users' VMs. Therefore, existing ICS instances in distributed systems are connected and distribute their users' VM images to every connected site. Besides ensuring the availability of each VM on every site, VM image synchronization allows users to access and manage their images with every ICS. This is useful in cases when a particular ICS instance becomes temporarily unusable e.g., due to hardware maintenance or when it is congested by other users.

Dispatch Daemon (dispatchd) The dispatchd is a transparent extension to the GridWay meta scheduler. GridWay is widely used in Grid environments to schedule computational tasks across multiple connected Grid

sites. Nevertheless, GridWay lacks some fundamental functions. For example, it is impossible to add or remove a particular Grid site to GridWay's managed resources without restarting the entire scheduler. This would require that all currently running jobs be rescheduled and lose already completed work. Moreover, users cannot influence GridWay's scheduling decisions to exclude particular resources from job execution, which is important if computational tasks process sensitive data which must not leave the company or a particular geographical region due to legal restrictions. However, dispatchd allows users to specify which resources can be used for task execution and allows administrators to add and remove sets of resources during runtime without interrupting job execution and scheduling.

Request Daemon (reqd) To deal with scalable resource sets like the resources provided by an infrastructure Cloud provider such as Amazon, dispatchd can be coupled with reqd. Instead of taking the current number of available nodes into account when making scheduling decisions, dispatchd will consider the maximum number of nodes which can be allocated. The allocation and deallocation of resources is handled by reqd, which examines the state of a particular GridWay scheduler to decide if additional resources should be acquired.

To decide whether or not resources should be acquired, reqd monitors the waiting queue of the GridWay instance. In contrast, the decision to deallocate active resources is more complex. To decide if a particular resource is idle and can be shut down securely, a server component was developed that interacts with particular resources to examine their state. Besides being coupled with existing monitoring systems like Ganglia or Nagios, the probe server can also use a simple shell script to dynamically query different metrics of the particular remote resource to analyze its current state. It can be reconfigured by the administrator during runtime without needing to be restarted. This allows a seamless integration into existing systems and minimizes the need to reconfigure an already existing software stack.

Bringing these technologies together allows for the concurrent use of resources from both worlds for the first time. It has been examined that Cloud computing resources focus on a different target than resources found in the area of Grid computing. While in a Grid HPC resources are usually connected, which provide a suitable environment in which to execute massively parallel computational tasks communicating via MPI, Cloud computing targets on the quick provisioning of additional resources that can also be used for computational tasks or to provide service offerings. By supporting the exchange of execution environments between both types of resources, a system can be built

that dynamically adapts to the workload and can provide virtually unlimited resources for executing computational tasks.

Instead of being limited to locally available computing resources, customers can now acquire the computational power needed to satisfy their needs on demand. The work presented in this thesis was embedded in several research projects with commercial partners belonging to different industrial sectors. Therefore, many different desires from these manifold environments influenced the design and the development of the proposed solution. This is why the system was designed generically and does not focus on one particular problem found in one particular branch of industry.

In addition, the system has been designed with respect to the administrators' desires. It fits seamlessly into existing landscapes by relying heavily on open standards and using widely accepted technologies. Moreover, the system is transparent to its users who might be keen in using the GridWay meta scheduler. The proposed solution extends the already provided functionality of existing components while preventing the introduction of additional unknown components to the users in order to not reduce the system's acceptance. From the administrator's viewpoint, the developed resources are easy to install and can easily be linked together with existing solutions such as a locally available Grid headnode installation. Relying on the GridWay meta scheduler which supports many different CRMs and which can be even used without having a CRM in place, the proposed solution is very flexible in terms of the supported computational resources. Whenever the available computational power becomes insufficient, new resources can easily be integrated into the system without the need for restarting it.

The prototypical implementation focuses on bringing well-tried components together ranging from traditional CRMs to data exchange mechanisms proven in Grid systems with thousands of users. In areas in which no sufficient solutions were available, either existing components have been combined to fulfill the needed requirements or new components have been developed to provide the missing functionality.

The resulting system increases the simplicity of utilization of the system and increases security at the same time.[75] Moreover, the system adapts automatically to the workload and can deal with a dynamically changing number of resources available for job execution. Respecting the benefits of Cloud computing, it can dynamically acquire additional resources whenever needed to fulfill its users' needs and to provide highly efficient job scheduling while minimizing computational costs by shutting down unused Cloud resources.

[75] This characteristic of the proposed solution is worth noting because usually a tradeoff exists between the security and the simplicity of use of a computer system: either a system is secure, but not usually very user-friendly, or a system is easy to use, but then often contains horrible security vulnerabilities.

6.2 Future Work

Of course, the proposed solution can be improved in many ways. As shown in Section 5.2.4, the utilization of virtualization technology causes a significant performance degradation when using network links for message passing. This overhead can be minimized by ensuring that a parallel job is distributed across a minimal amount of VMs. To minimize the number of VMs needed, the maximum possible amount of hardware must be assigned to each VM taking part in parallel job execution instead of using only one CPU core for each VM. The development of local CRMs must be pushed forward in order to ensure this functionality.

Moreover, dispatchd must be able to deal with cost limits specified by the user on a per-job basis. This can be easily achieved by specifying the particular resource's cost information in the configuration file which already exists. The computational costs of a job can be estimated by the set of functions presented in Section 5.5.2. Coupled with the selection of resources used for job execution, the system could ensure that the amount of money specified by the user will not be exceeded.

On the other hand, computational costs can be minimized by shutting down unused resources. As presented earlier, the system can shut down unused Cloud resources as soon as they are no longer in use. Many modern hardware components support power saving states which can be activated in times in which the particular resources are not used. Modern CPUs can reduce their speed if the system is idle in order to reduce energy consumption. Based on these technologies, unused resources could be put into power saving mode if no pending jobs exist. Moreover, entire systems could be shut down and reactivated when needed e.g., by using the Wake-on-LAN (WOL) standard.

Using virtualization technology to improve security in multi-user environments is widely accepted. Nevertheless, commercial customers have some worries about transferring their sensitive data in Grid or Cloud systems. This is mainly due to one reason: Even if data is stored encrypted to prevent unauthorized access and theft of data, it must be decrypted at some point in time when the job is executed on the remote machines. To prevent this, the idea of homomorphic encryption was developed [84, 204]. Providing a homomorphic encryption would allow user to process data without needing to decrypt it. The result calculated by an operation is also encrypted, but the decrypted result can be derived from the encrypted data because of the homomorphism. Homomorphic encryption seems to be a promising approach but has some disadvantages at the moment e.g., results gets "dirty" after some sequentially performed operations and cannot be used to derive the correct unencrypted result. Providing a robust homomorphic encryption could dramatically increase the utilization of public shared-user systems by commercial customers.

The Trusted Computing (TC) approach can be elaborated to provide a trust-relationship between the resource provider and the customer. The Trusted Platform Module (TPM) not only allows secure data encryption by using real Random Number Generators (RNGs) to create strong and secure keys, but also provides the ability to store these keys in a secure manner. Moreover, using the remote attestation ability, customers can prevent the processing of their data on untrusted systems of the resource provider. TPM can guarantee that a Trusted Computing Base (TCB) containing the software stack needed to execute the VM is in a trusted state and was not modified by malicious users or the administrator to get access to customers' data. To achieve this goal, existing middlewares must support TPM hardware and functionality must be implemented to provide the stated functionality.

Due to the fact that Cloud computing has received a great deal of attention recently and much effort has been invested in research in the area of Green computing, the solution proposed in this thesis provides a good foundation for future research work. Because Cloud computing is a new paradigm in computer science, many future challenges cannot be appraised today. It promises to be an exciting area of future research and seems to be able to provide approaches that allow users to overcome traditional limitations and enable a recognizable shift in the computing landscape. This thesis aimed at pushing technology one step further and dissolving the border between traditional computing landscapes and the promising approach of Cloud computing.

Acronyms

σ	Standard Derivation
AHM	All-Hands-Meeting
AI	Artificial Intelligence
API	Application Programming Interface
AWS	Amazon Web Services
BMBF	German Ministry of Education and Research
BIOS	Basic Input/Output System
BPEL	Business Process Execution Language
CA	Certificate Authority
CAPEX	Capital Expenditure
CE	Computing Element
CERN	European Organization for Nuclear Research
CEST	Central European Summer Time
CET	Central European Time
COW	Copy-on-Write
CRM	Cluster Resource Manager
CRTM	Core Root of Trust for Measurement
CPU	Central Processing Unit
D-Grid	German National Grid Initiative
DAAD	German Academic Exchange Service
DBMS	Database Management System
DDoS	Distributed Denial of Service
DHCP	Dynamic Host Configuration Protocol
DGI	D-Grid Integration Project
dispatchd	Dispatch Daemon
DMTF	Distributed Management Task Force
DMZ	Demilitarized Zone
DN	Distinguished Name
DNS	Domain Name System
dom0	Domain0
domU	DomainU

DoS	Denial of Service
EC	European Commission
EC2	Elastic Compute Cloud
EU	European Union
FS	File System
FTP	File Transfer Protocol
GIN	Grid Interoperability Now
GPG	GNU Privacy Guard
GRE	Generic Routing Encapsulation
GSI	Grid Security Infrastructure
GT4	Globus Toolkit 4
GUI	Graphical User Interface
GWT	Google Web Toolkit
G-WiN	German research network
HA	High-Availability
HaaS	Human as a Service
HIDS	Host Intrusion Detection System
HTT	HyperThreading Technology
HTTP	HyperText Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
HPC	High Performance Computing
HPL	High Performance Linpack
I/O	Input/Output
IaaS	Infrastructure as a Service
IBM	International Business Machines Corporation
IB	InfiniBand
ICMP	Internet Control Message Protocol
ICS	Image Creation Station
ICT	Information and Communications Technology
IDS	Intrusion Detection System
IMB	Intel MPI Benchmark
IOMMU	I/O Memory Management Unit
IP	Internet Protocol
IT	Information Technology
JaaS	Job as a Service
JNI	Java Native Interface
JRE	Java Runtime Environment
JVM	Java Virtual Machine
KVM	Kernel-based Virtual Machine
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LHC	Large Hadron Collider
LVM	Logical Volume Manager

MAC	Media Access Control
MAD	Middleware Access Driver
MDS	Monitoring and Discovery System
MLRFS	Multi-Layered Root File System
MMU	Memory Management Unit
MPI	Message Passing Interface
MVC	Model-View-Controller
NAS	Network-Attached Storage
NFS	Network File System
NGI	National Grid Initiative
NIC	Network Interface Card
NIDS	Network Intrusion Detection System
NIS	Network Information Service
NIST	National Institute of Standards and Technology
NPB	NAS Parallel Benchmarks
OASIS	Organization for the Advancement of Structured Information Standards
OGF	Open Grid Forum
OPEX	Operational Expenditure
OS	Operating System
OVA	Open Virtualization Format Archive
OVF	Open Virtualization Format
PaaS	Platform as a Service
PVFS	Parallel Virtual File System
QoS	Quality of Service
RAID	Redundant Array of Inexpensive Disks
reqd	Request Daemon
RMI	Remote Method Invocation
RNG	Random Number Generator
SaaS	Software as a Service
S3	Simple Storage Service
SFTP	Secure File Transfer Protocol
SGE	Sun Grid Engine
SIDS	Streaming Intrusion Detection System
SLA	Service Level Agreement
SME	Small and Medium Enterprises
SOA	Service Oriented Architecture
SSH	Secure Shell
SSL	Secure Sockets Layer
TCO	Total Cost of Ownership
TC	Trusted Computing
TCB	Trusted Computing Base
TCP	Trusted Computing Platform

TLS	Transport Layer Security
TPM	Trusted Platform Module
TivSAM	Tivoli Service Automation Manager
URL	Uniform Resource Locator
VC	Virtual Cluster
VLAN	Virtual Local Area Network
VM	Virtual Machine
vmdk	Virtual Machine disk
VMM	Virtual Machine Monitor
VO	Virtual Organization
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WOL	Wake-on-LAN
X-WiN	German research network
XGE	Xen Grid Engine
XML	eXtensible Markup Language

List of Figures

List of Tables

Listings

Bibliography

[1] *Security Guidance for Critical Areas of Focus in Cloud Computing*. Cloud Security Alliance, 2009.

[2] *Top Threats to Cloud Computing*. Cloud Security Alliance, 2010.

[3] David Abramson, Rajkumar Buyya, and Jonathan Giddy. A Computational Economy for Grid Gomputing and its Implementation in the Nimrod-G Resource Broker. *Future Generation Computer Systems*, 18(8):1061–1074, 2002.

[4] Marco Adinucci, Massimo Torquati, Marco Vanneschi, and Pierfrancesco Zuccato. The VirtuaLinux Storage Abstraction Layer for Efficient Virtual Clustering. *Processings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Computing (PDP 2008)*, pages 619–627, 2008.

[5] Jeannie Albrecht, Christopher Tuttle, Alex C. Snoeren, and Amin Vahdat. PlanetLab Application Management Using Plush. *ACM SIGOPS Operating Systems Review*, 40(1):33–40, 2006.

[6] Amazon.com. Amazon WebServices. `http://aws.amazon.com/`, April 2010.

[7] AMD. AMD Secure Virtual Machine Architecture Reference Manual. 2005.

[8] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business Process Execution Language for Web Services. 2003.

[9] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, and Randy Katz. Above the Clouds: A Berkeley View of Cloud Computing. *Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley*, 2009.

[10] AUFS Developers. Another UnionFS. http://aufs.sourceforge.net, 08 2010.

[11] Chang Bae, John Russell Lange, and Peter A. Dinda. Comparing Approaches to Virtualized Page Translation in Modern VMMs. *Technical Report NWU-EECS-10-07*, 2010.

[12] David H. Bailey, Eric Barszcz, John Thomas Barton, David S. Browning, Robert Leston Carter, Leonardo Dagum, Rod A. Fatoohi, Paul O. Frederickson, Tom A. Lasinski, Robert S. Schreiber, Horst D. Simon, V. K. Venkatakrishnan, and Sisira K. Weeratunga. The NAS Parallel Benchmarks. *International Journal of High Performance Computing Applications*, 5(3):63–73, 1991.

[13] Amnon Barak and Oren La'adan. The MOSIX Multicomputer Operating System for High Performance Cluster Computing. *Future Generations in Computer Systems*, 13(4 - 5):361 – 372, 1997.

[14] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 165–177, 2003.

[15] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. *Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC '05)*, pages 41–46, 2005.

[16] Werner Benger, Ian Foster, Jason Novotny, Edward Seidel, John Shalf, Warren Smith, and Paul Walker. Numerical relativity in a distributed environment. *Proceedings of the 9th SIAM Conference on Parallel Processing for Scientific Computing, Philadelphia*, 1999.

[17] Andreas Berl, Erol Gelenbe, Marco Di Girolamo, Giovanni Giuliani, Hermann De Meer, Minh Quan Dang, and Kostas Pentikousis. Energy-Efficient Cloud Computing. *The Computer Journal*, 53(7):1045–1051, 2010.

[18] Nikhil Bhatia and Jeffrey S. Vetter. Virtual Cluster Management with Xen. *Lecture Notes in Computer Science, Euro-Par 2007*, 4854:185–194, 2008.

[19] Ken Birman, Gregory Chockler, and Robbert van Renesse. Toward a Cloud Computing Research Agenda. *ACM SIGACT News*, 40(2), 2009.

[20] BitTorrent, Inc. BitTorrent. http://www.bittorrent.com/, 2008.

[21] Håvard K. F. Bjerke, Dimitar Shiyachki, Andreas Unterkircher, and Irfan Habib. Tools and Techniques for Managing Virtual Machine Images. *Euro-Par 2008 Workshops - Parallel Processing, Springer Lecture Notes in Computer Science*, 5415:3–12, 2009.

[22] Richard Boardman, Stephen Crouch, Hugo Mills, Steven Newhouse, and Juri Papay. Towards Grid Interoperability. *All Hands Meeting 2007, OMII-UK Workshop, 10 September - 13 September*, 2007.

[23] Rick Bradshaw, Narayan Desai, Timothy Freeman, and Katarzyna Keahey. A Scalable Approach to Deploying and Managing Appliances. *TeraGrid Conference*, 2007.

[24] Sharon Brunett, Dan Davis, Thomas Gottschalk, Paul Messina, and Carl Kesselman. Implementing Distributed Synthetic Forces Simulations in Metacomputing Environments. *Proceedings of the Seventh Heterogeneous Computing Workshop (HCW '98)*, 1998.

[25] Marta González Bugeiro, José Carlos Mouriño Gallego, Andrés Gómez Tato, Constantino Vázquez, Eduardo Huedo, Ignacio Martín Llorente, and Daniel A. Rodríguez Silva. Integration of SLAs with GridWay in BEinEIMRT project. *Proceedings of the 3rd Iberian Grid Infrastructure Conference (IBERGRID 2009)*, 2009.

[26] Rajkumar Buyya, David Abramson, and Jonathan Giddy. Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid. *Proceedings of the Fourth International Conference/Exhibition on High Performance Computing in Asia-Pacific Region*, 2000.

[27] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal. Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications, 2008 (HPCC '08)*, pages 5–13, 2008.

[28] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.

[29] Mike Cafarella and Doug Cutting. Building Nutch: Open Source Search. *Queue - Search Engines*, 2(2):54–61, 2004.

[30] Carmen Carmack. How BitTorrent Works. `http://computer.howstuffworks.com/bittorrent.htm/printable`, 03 2011.

[31] Philip H. Carns, Walter B. Ligon III, Robert B. Ross, and Rajeev Thakur. PVFS: A Parallel File System for Linux Clusters. *Proceedings of the 4th Annual Linux Showcase & Conference (ALS '00)*, 4, 2000.

[32] CERT Advisory CA-1998-01. Smurf IP Denial-of-Service Attacks. `http://www.cert.org/advisories/CA-1998-01.html`, 2008.

[33] Sivadon Chaisiri, Lee Bu-Sung, and Dusit Niyato. Optimal Virtual Machine Placement across Multiple Cloud Providers. *Proceedings of the IEEE Asia-Pacific Services Computing Conference (APSCC '09)*, pages 103–110, 2009.

[34] Bryan Clark, Todd Deshane, Eli Dow, Steven Evanchik, Matthew Finlayson, Jason Herne, and Jeanna Neefe Matthews. Xen and the Art of Repeated Research. *Proceedings of the Usenix Annual Technical Conference*, 2004.

[35] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live Migration of Virtual Machines. *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation*, 2:273 – 286, 2005.

[36] Arthur Charles Clarke. Hazards of Prophecy: The Failure of Imagination. *Profiles of the Future: An Enquiry into the Limits of the Possible*, 1962.

[37] Bram Cohen. Incentives Build Robustness in BitTorrent. *Workshop on Economics of Peer-to-Peer Systems (IPTPS)*, 2003.

[38] Russel Coker. bonnie++ Benchmark. `http://www.coker.com.au/bonnie++/`, 2009.

[39] Robert J. Creasy. The Origin of the VM/370 Time-Sharing System. *IBM Journal of Research and Development*, 25(5):483–490, 1981.

[40] John W. Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE Publications, Inc., 2008.

[41] Renzo Davoli. Virtual Square. *Proceedings of the First International Conference on Open Source Systems*, pages 76–81, 2005.

[42] Renzo Davoli and Michael Goldweber. Virtual Square (v^2) in computer science education. *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '05)*, 37(3):301–305, 2005.

[43] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, Mei-Hui Su, Karan Vahi, and Miron Livny. Pegasus:

Mapping Scientific Workflows onto the Grid. *Grid Computing, Lecture Notes in Computer Science*, 3165:131–140, 2004.

[44] Ewa Deelman, Gurmeet Singh, Miron Livny, G. Bruce Berriman, and John Good. The Cost of Doing Science on the Cloud: The Montage Example. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2008)*, 2008.

[45] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems. *Scientific Programming*, 13(3):219–237, 2005.

[46] Ruben Van den Bossche, Kurt Vanmechelen, and Jan Broeckhove. Cost-Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads. *Proceedings of the 3rd International Conference on Cloud Computing (CLOUD 2010)*, pages 228–235, 2010.

[47] Rob F. Van der Wijngaart. NAS Parallel Benchmarks Version 2.4. *NAS Technical Report NAS-02-007*, 2002.

[48] Todd Deshane, Zachary Shepherd, Jeanna Neefe Matthews, Muli Ben-Yehuda, Amit Shah, and Balaji Rao. Quantitative comparison of Xen and KVM. *Xen Summit*, 2008.

[49] Theo Dimitrakos, Josep Martrat, and Stefan Wesner, editors. *Service Oriented Infrastructures and Cloud Service Platforms for the Enterprise – A Selection of Common Capabilities Validated in Real-Life Business Trials by the BEinGRID Consortium*. Springer, 2010.

[50] Distributed Management Task Force, Inc. *Open Virtualization Format Specification 1.1.0*, 2010.

[51] Jack J. Dongarra. The LINPACK Benchmark: An Explanation. *Supercomputing, Springer Lecture Notes in Computer Science*, 297:456–474, 1988.

[52] Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet. The LINPACK Benchmark: past, present and future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820, 2003.

[53] John J. Donovan and Stuart E. Madnick. Hierarchical Approach to Computer System Integrity. *IBM Systems Journal*, pages 188–202, 1975.

[54] Kay Dörnemann, Jörg Prenzer, and Bernd Freisleben. A Peer-to-Peer Meta-Scheduler for Service-Oriented Grid Environments. *Proceedings*

of First International Conference on Networks for Grid Applications, ACM Press, 2007.

[55] Tim Dörnemann, Thomas Friese, Sergej Herdt, Ernst Juhnke, and Bernd Freisleben. Grid Workflow Modelling Using Grid-Specific BPEL Extensions. *German e-Science Conference*, pages 138–140, 2007.

[56] Tim Dörnemann, Ernst Juhnke, and Bernd Freisleben. On-Demand Resource Provisioning for BPEL Workflows Using Amazon's Elastic Compute Cloud. *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 140–147, 2009.

[57] Tim Dörnemann, Matthew Smith, Ernst Juhnke, and Bernd Freisleben. Secure Grid Micro-Workflows Using Virtual Workspaces. *Proceedings of the 34th Euromicro Conference on Software Engineering and Advanced Applications (SEAA '08)*, pages 119–126, 2008.

[58] Kathleen M. Eisenhardt. Building Theories From Case Study Research. In *The Academy of Management Review*, volume 14, pages 532–550, 1989.

[59] Wolfgang Emmerich, Ben Butchart, Liang Chen, Bruno Wassermann, and Sarah L. Price. Grid Service Orchestration Using the Business Process Execution Language (BPEL). *Journal of Grid Computing*, 3(3-4):283–304, 2005.

[60] Active Endpoint. Active BPEL Engine. `http://www.activebpel.org`, January 2011.

[61] Christian Engelmann, Stephen L. Scott, Hong Ong, Geoffroy Vallée, and Thomas Naughton. Configurable Virtualized System Environments for High Performance Computing. *Proceedings of the 1st Workshop on System-level Virtualization for High Performance Computing (HPCVirt '07), in Conjunction with the 2nd ACM SIGOPS European Conference on Computer Systems (EuroSys '07)*, 2007.

[62] Dietmar W. Erwin and David F. Snelling. UNICORE: A Grid Computing Environment. *Euro-Par 2001 Parallel Processing, Lecture Nots in Computer Science*, 2150:828–834, 2001.

[63] Thomas Fahringer, Alexandru Jugravu, Sabri Pllana, Radu Prodan, Clovis Seragiotto Jr., and Hong-Linh Truong. ASKALON: A Tool Set for Cluster and Grid Computing. *Concurrency and Computation: Practice and Experience, Special Issue: Grids and Web Services for e-Science*, 17(2-4):143–169, 2005.

[64] Thomas Fahringer, Radu Prodan, Rubing Duan, Francesco Nerieri, Stefan Podlipnig, Jun Qin, Mumtaz Siddiqui, Hong-Linh Truong, Alex

Villazón, and Marek Wieczorek. ASKALON: A Grid Application Development and Computing Environment. *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 122–131, 2005.

[65] Niels Fallenbeck. Scheduling von virtuellen Maschinen in Cluster-Umgebungen. Master's thesis, Philipps-University Marburg, Germany, 2007.

[66] Niels Fallenbeck, Hans-Joachim Picht, Matthew Smith, and Bernd Freisleben. Xen and the Art of Cluster Scheduling. *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing in conjuncture with the ACM/IEEE Conference on Supercomputing*, pages 4–11, 2006.

[67] Niels Fallenbeck, Matthias Schmidt, Roland Schwarzkopf, and Bernd Freisleben. Inter-Site Virtual Machine Image Transfer in Grids and Clouds. *Proceedings of the 2nd International ICST Conference on Cloud Computing (CloudComp 2010), Springer LNICST*, 2010.

[68] Dino Farinacci, Tony Li, Stan Hanks, David Meyer, and Paul Traina. Generic Routing Encapsulation (GRE). *RFC 2784*, 2000.

[69] Renato J. Figueiredo, Peter A. Dinda, and José A. B. Fortes. A Case For Grid Computing On Virtual Machines. *Proceedings 23rd International Conference on Distributed Computing Systems*, pages 550 – 559, 2003.

[70] Stephanie Forrest, Anil Somayaji, and David H. Ackley. Building Diverse Computer Systems. *6th Workshop on Hot Topics in Operating Systems (HotOS-VI)*, pages 67–72, 1997.

[71] Ian Foster. What is the Grid? - A Three Point Checklist. *GRIDtoday*, 1(6), 2002.

[72] Ian Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. *FIP International Conference on Network and Parallel Computing*, 4(21):513–520, 2006.

[73] Ian Foster, Timothy Freeman, Katarzyna Keahey, and Doug Scheftner. Virtual Clusters for Grid Communities. *Cluster Computing and Grid (CCGRID)*, 2006.

[74] Ian Foster and Carl Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of High Performance Computing Applications*, 11(2):115–128, 1997.

[75] Ian Foster and Carl Kesselman. The Grid: Blueprint for a New Computing Infrastructure. 1998.

[76] Ian Foster, Carl Kesselman, and Steven Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications*, 15(3), 2001.

[77] Timothy Freeman and Katarzyna Keahey. Flying Low: Simple Leases with Workspace Pilot. *Proceedings of the 14th international Euro-Par Conference on Parallel Processing (Euro-Par '08)*, pages 499–509, 2008.

[78] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*, 5:237–246, 2002.

[79] Thomas Friese, Matthew Smith, and Bernd Freisleben. Native Code Security for Grid Services. *8. Internationale Tagung Wirtschaftsinformatik (WI 2007)*, pages 513–530, 2007.

[80] J. Steven Fritzinger and Marianne Mueller. Java Security. *White Paper*, 1996.

[81] Fabrizio Gagliardi, Bob Jones, François Grey, Marc-Elian Bégin, and Matti Heikkurinen. Building an infrastructure for scientific Grid computing: status and goals of the EGEE project. *Philosophical Transactions of the Royal Society, Mathematical, Physical and Engineering Sciences*, 363:1729–1742, 2005.

[82] Simson L. Garfinkel and Hal Abelson, editors. *Architects of the Information Society: Thirty-Five Years of the Laboratory for Computer Science at MIT*. Massachusetts Institute of Technology, 1999.

[83] Tal Garfinkel and Mendel Rosenblum. When Virtual is Harder than Real: Security Challenges in Virtual Machine Based Computing Environments. *Proceedings of the 10th Workshop on Hot Topics in Operating Systems (HotOS-X)*, 10, 2005.

[84] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, 2009.

[85] Wolfgang Gentzsch. Sun Grid Engine: Towards Creating a Compute Power Grid. *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 35–36, 2001.

[86] Wolfgang Gentzsch. The Evolution of Oracle Grid Engine. `http://www.hpcinthecloud.com/blogs/The-Evolution-of-Oracle-Grid-Engine-102537499.html`, September 2010.

[87] Laura Gilbert, Jeff Tseng, Rhys Newman, Saeed Iqbal, Ronald Pepper, Onur Celebioglu, Jenwei Hsieh, and Marc Cobban. Performance Implications of Virtualization and Hyper-Threading on High Energy Physics Applications in a Grid Environment. *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS '05)*, 2005.

[88] Raffale Giordanelli and Carlo Mastroianni. The Cloud Computing Paradigm: Characteristics, Opportunities and Research Issues. *Technical Report RT-ICAR-CS-10-01*, 2010.

[89] Bryan Gleeson, Juha Heinanen, Arthur Lin, Grenville Armitage, and Andrew G. Malis. A Framework for IP Based Virtual Private Networks. *RFC 2764*, 2000.

[90] Globus Toolkit Developers. Globus Toolkit. `http://www.globus.org/toolkit/`, 2010.

[91] Robert Goldberg. *Architectural Principles for Virtual Computer Systems*. PhD thesis, Division of Engineering and Applied Physics Harvard University Cambridge Massachusetts, 1972.

[92] Li Gong. Variations on the Themes of Message Freshness and Replay. *Proceedings of the 6th IEEE Computer Security Foundations Workshop (CSFW)*, pages 131–136, 1993.

[93] James Gosling and Henry McGilton. The Java Language Environment. *Sun Microsystems Computer Company*, 1995.

[94] Otis Gospodnetic and Erik Hatcher. *Lucene in Action*. Manning Publications Co., 2004.

[95] Sven Graupner, Jim Pruyne, and Sharad Singhal. Making the Utility Data Center a Power Station for the Enterprise Grid. *Technical Report HPL-2003-53, Hewlett Packard Laboratories*, 2003.

[96] Ada Gravrilovska, Sanjay Kumar, Himanshu Raj, Karsten Schwan, Vishakha Gupta, Ripal Nathuji, Radhika Niranjan, Adit Ranadive, and Purav Saraiya. High-Performance Hypervisor Architectures: Virtualization in HPC Systems. *First Workshop on System-Level Virtualization for High Performance Computing (HPCVirt 2007), EuroSys*, 2007.

[97] Andrew S. Grimshaw and William A. Wulf. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40(1):39–45, 1997.

[98] Laura Grit, David Irwin, Varun Marupadi, Piyush Shivam, Aydan Yumerefendi, Jeffrey Chase, and Jeannie Albrecht. Harnessing Virtual Machine Resource Control for Job Management. *Proceedings of*

the First International Workshop on Virtualization Technology in Distributed Computing (VTDC), 2007.

[99] Expert Group. The Future Of Cloud Computing - Opportunities for European Cloud Computing beyond 2010. 2010.

[100] Peter H. Gum. System/370 Extended Architecture: Facilities for Virtual Machines. *IBM Journal of Research and Development*, 27(6):530–544, 1983.

[101] Katharina Haselhorst, Matthias Schmidt, Roland Schwarzkopf, Niels Fallenbeck, and Bernd Freisleben. Efficient Storage Synchronization for Live Migration in Cloud Infrastructures. *Proceedings of the 19th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP), IEEE Press*, pages 511–518, 2011.

[102] Michael Heidt, Tim Dörnemann, Kay Dörnemann, and Bernd Freisleben. Omnivore: Integration of Grid Meta-Scheduling and Peer-to-Peer Technologies. *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid '08)*, pages 316–323, 2008.

[103] Mark D. Hill. What is Scalability? *ACM SIGARCH Computer Architecture News*, 18(4):18 – 21, 1990.

[104] Oliver Hinz, Roman Beck, Bernd Skiera, and Wolfgang König, editors. *Grid Computing in der Finanzindustrie - Ein Herausgeberband des D-Grid-Projekts FinGrid*. E-Finance Lab, Goethe-Universität Frankfurt am Main, 2009.

[105] Michael Hohmuth, Michael Peter, Hermann Härtig, and Jonathan S. Shapiro. Reducing TCB size by using Untrusted Components: Small Kernels versus Virtual-Machine Monitors. *Proceedings of the 11th Workshop on ACM SIGOPS, European Workshop (EW '11)*, 2004.

[106] Jason Hsiao. Amazon.com CEO Jeff Bezos on Amazon.com. `http://animoto.com/blog/company/amazon-com-ceo-jeff-bezos-on-animoto/`, January 2011.

[107] Eduardo Huedo, Rubén Santiago Montero, and Ignacio Martín Llorente. A Framework for Adaptive Execution in Grids. *Software: Practice and Expertise*, 34(7):631–651, 2004.

[108] Eduardo Huedo, Rubén Santiago Montero, and Ignacio Martín Llorente. A modular Meta-scheduling Architecture for Interfacing with Pre-WS and WS Grid Resource Management Services. *Future Generation Computing Systems*, 23(3):252–261, 2007.

[109] Red Hat Inc. Red Hat Virtual Machine Manager. http://virt-manager.et.redhat.com, January 2011.

[110] D-Grid Initiative. http://www.d-grid.de/, 2011.

[111] David Irwin, Jeffrey Chase, Laura Grit, Aydan Yumerefendi, David Becker, and Kenneth G. Yocum. Sharing Networked Resources with Brokered Leases. *Proceedings of the Annual Technical Conference on USENIX (ATEC '06)*, 2006.

[112] Michael A. Iverson, Füsun Özgüner, and Gregory J. Follen. Run-Time Statistical Estimation of Task Execution Times for Heterogeneous Distributed Computing. *Proceedings of the High Performance Distributed Computing (HPDC '96)*, pages 263–270, 1996.

[113] Michael A. Iverson, Füsun Özgüner, and Lee Potter. Statistical Prediction of Task Execution Times through Analytic Benchmarking for Scheduling in a Heterogeneous Environment. *IEEE Transactions on Computers*, 48(12):1374–1379, 2002.

[114] Jeff Barr. Amazon EC2 Beta. http://aws.typepad.com/aws/2006/08/amazon_ec2_beta.html, 25. August 2006.

[115] Shantenu Jha, Andre Merzky, and Geoffrey Fox. Using Clouds to Provide Grids Higher-Levels of Abstraction and Explicit Support for Usage Modes. *Concurrency and Computation: Practice and Experience*, 2009.

[116] Laurence F. Johnsson, Alan Levine, Rachel S. Smith, and J. Troy Smythe. The Horizon Report: 2009 K-12 Edition. *The New Media Consortium*, 2009.

[117] Eric Jul, Henry Levy, Norman Hutchinson, and Andrew Black. Fine-grained Mobility in the Emerald System. *ACM Transactions on Computer Systems (TOCS)*, 6(1):109–133, 1988.

[118] Katarzyna Keahey, Karl Doering, and Ian Foster. From Sandbox to Playground: Dynamic Virtual Environments in the Grid. *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID '04)*, pages 34–42, 2004.

[119] Katarzyna Keahey, Renato J. Figueiredo, José A. B. Fortes, Timothy Freeman, and Mauricio Tsugawa. Science Clouds: Early Experiences in Cloud Computing for Scientific Applications. *Cloud Computing and its Applications (CCA-08)*, 2008.

[120] Katarzyna Keahey, Ian Foster, Timothy Freeman, and Xuehai Zhang. Virtual Workspaces: Achieving Quality of Service and Quality of Life

in the Grid. *Scientific Programming - Dynamic Grids and Worldwide Computing*, 13(4), 2005.

[121] Katarzyna Keahey, Ian Foster, Timothy Freeman, Xuehai Zhang, and Daniel Galron. Virtual Workspaces in the Grid. *Lecture Notes in Computer Science*, pages 22–34, 2005.

[122] Katarzyna Keahey and Timothy Freeman. Contextualization: Providing One-Click Virtual Clusters. *Proceedings of the Fourth IEEE International Conference on eScience (eScience '08)*, pages 301–308, 2008.

[123] Carl Kesselman and Ian Foster. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufman, San Mateo, 1998.

[124] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the Linux Virtual Machine Monitor. *Ottawa Linux Symposium*, pages 225–230, 2007.

[125] Nadir Kiyanclar. A Survey of Virtualization Techniques Focusing on Secure On-Demand Cluster Computing. *ACM CORR Technical Report cs.OS/0511010*, 2005.

[126] Wolfgang König, Roman Beck, Bernd Freisleben, Manfred Grauer, Michael Resch, Bernd Skiera, Helga Kaminski, Clemens Jochum, Oliver Schedletzky, Wolfgang Rau, Ulrich Wolf, and Wolfgang Decker. Financial Business Grid - Service Grid Architecture for the Financial Service Industry. 2007.

[127] Björn Könning, Christian Engelmann, Stephen L. Scott, and G. Al Geist. Virtualized Environments for the Harness High Performance Computing Workbench. *Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP '08)*, pages 133–140, 2008.

[128] Jürgen Krämer and Bernhard Seeger. PIPES – A Public Infrastructure for Processing and Exploring Streams. *ACM Special Interest Group on Management of Data (SIGMOD)*, 2004.

[129] H. Andrés Lagar-Cavilla, Joseph A. Whitney, Adin Scannell, Philip Patchin, Stephen M. Rumble, Eyal de Lara, Michael Brudno, and M. Satyanarayanan. SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing. *Proceedings of the 3rd European Conference on Computer Systems (EuroSys)*, 2009.

[130] Massimo Lamanna. The LHC computing grid project at CERN. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 534(1-2):1–6, 2004.

[131] Craig Larman and Victor R. Basili. Iterative and Incremental Development: A Brief History. *Computer*, 36(6):47–56, 2003.

[132] Erwin Laure, Claudio Grandi, Steve Fisher, Akos Frohner, Peter Kunszt, Aleš Křenek, Olle Mulmo, Fabrizio Pacini, Francesco Prelz, John White, Maite Barroso, Predrag Buncic, Rob Byrom, Linda Cornwall, Martin Craig, Alberto Di Meglio, Abdeslem Djaoui, Francesco Giacomini, Joni Hahkala, Frédéric Hemmer, Stephen Hicks, Ake Edlund, Alessandro Maraschini, Robin Middleton, Massimo Sgaravatto, Martijn Steenbakkers, John Walk, and Antony Wilson. Programming the Grid with gLite. *Computational Methods in Science and Technology*, 12:33–45, 2006.

[133] libvirt Developers. libvirt: The virtualization API. `http://libvirt.org`, January 2011.

[134] Jiuxing Liu, Wei Huang, Bulent Abali, and Dhabaleswar K. Panda. High Performance VMM-Bypass I/O in Virtual Machines. *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference (ATEC '06)*, 2006.

[135] Hans Löhr, Hari-Govind Ramasamy, Ahmad-Reza Sadeghi, Stefan Schulz, Matthias Schunter, and Christian Stüble. Enhancing Grid Security Using Trusted Virtualization. *Autonomic and Trusted Computing, Lecture Notes in Computer Science, Springer*, 4610:372–384, 2007.

[136] Wei Lu. Making your Workspace secure: Establishing Trust with VMs in the Grid. *SuperComputing 2005 Poster*, 2005.

[137] Peter Lucas, Joseph Ballay, and Ralph Lombreglia. The Wrong Cloud. *MAYA Design*, 2009.

[138] Luciano Bello. OpenSSL – predictable random number generator. `http://www.debian.org/security/2008/dsa-1571`, 2008.

[139] James Manyika. Google's View on the Future of Business: An Interview with CEO Eric Schmidt. *The McKinsey Quarterly*, 2008.

[140] Paul Marshall, Katarzyna Keahey, and Timothy Freeman. Elastic Site: Using Clouds to Elastically Extend Site Resources. *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID '10)*, 2010.

[141] Peter Mell and Tim Grance. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology*, 2009.

[142] Maged Michael, José E. Moreira, Doron Shiloach, and Robert W. Wisniewski. Scale-up x Scale-out: A Case Study using Nutch/Lucene. *Parallel and Distributed Processing Symposium (IPDPS), IEEE International*, 2007.

[143] Dejan S. Milojičić, Fred Douglis, Yves Paindaveine, Richard Wheeler, and Songnian Zhou. Process Migration. *ACM Computing Surveys (CSUR)*, 32(3):241 – 299, 2000.

[144] Rubén Santiago Montero, Rafael Moreno-Vozmediano, and Ignacio Martín Llorente. An Elasticity Model for High Throughput Computing Clusters. *Journal of Parallel and Distributed Computing*, page to appear, 2010.

[145] José E. Moreira, Samuel P. Midkiff, Manish Gupta, Pau V. Artigas, Marc Snir, and Richard D. Lawrence. Java Programming for High-Performance Numerical Computing. *IBM Systems Journal*, 39(1):21–56, 2000.

[146] Rafael Moreno-Vozmediano, Rubén Santiago Montero, and Ignacio Martín Llorente. Elastic Management of Cluster-based Services in the Cloud. *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds (ACDC '09)*, pages 19–24, 2009.

[147] Rafael Moreno-Vozmediano, Rubén Santiago Montero, and Ignacio Martín Llorente. Multi-Cloud Deployment of Computing Clusters for Loosely-Coupled MTC Applications. *IEEE Transactions on Parallel and Distributed Systems*, (99), 2010.

[148] Robert Morris and Ken Thompson. Password Security: A Case History. *Communications of the ACM*, 22(11), 1979.

[149] Derek Murray, Grzegorz Milos, and Steven Hand. Improving Xen Security through Disaggregation. *Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '08)*, pages 151–160, 2008.

[150] Michael Myers and Stephen Youndt. An Introduction to Hardware-Assisted Virtual Machine (HVM) Rootkits. *crucialsecurity.com*, 2007.

[151] Hidemoto Nakada, Takeshi Yokoi, Tadashi Ebara, Yusuke Tanimura, Hirotaka Ogawa, and Satoshi Sekiguchi. The Design and Implementation of a Virtual Cluster Management System. *Proceedings of 1st IEEE/IFIP International Workshop on End-to-end Virtualization and Grid Management*, pages 61–71, 2007.

[152] Susanta Nanda and Tzi-cker Chiueh. A Survey of Virtualization Technologies. *Techology Report, State University of New York at Stony Brook*, 2005.

[153] Jeffrey Napper and Paolo Bientinesi. Can Cloud Computing Reach The TOP500? *Proceedings of the combined Workshops on UnConventional High Performance Computing Workshop plus Memory Access Workshop (UCHPC-MAW '09)*, pages 17–20, 2009.

[154] Ripal Nathuji and Karsten Schwan. VPM Tokens: Virtual Machine-Aware Power Budgeting in Datacenters. *Proceedings of the 17th International Symposium on High Performance and Distributed Computing (HPDC '08)*, 2008.

[155] Gil Neiger, Amy Santoni, Felix Leung, Dion Rodgers, and Rich Uhlig. Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization. *Intel Technology Journal*, 2006.

[156] Barry Clifford Neuman. Scale in Distributed Systems. *Readings in Distributed Computing Systems, IEEE Computer Society Press*, pages 463–489, 1994.

[157] Nimbus Developers. Nimbus Open Source Toolkit. `http://www.nimbusproject.org/`, 2010.

[158] Hideo Nishimura, Naoya Maruyama, and Satoshi Matsuoka. Virtual Clusters on the Fly — Fast, Scalable, and Flexible Installation. *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, 2007.

[159] Novell, Inc. SuSE Studio. `http://susestudio.com/`, 2009.

[160] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. Eucalyptus: A Technical Report on an Elastic Utility Computing Architecture Linking Your Programs to Useful Systems. *UCSB Computer Science Technical Report Number 2008-10*, 2008.

[161] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. Eucalyptus: An Open-Source Cloud Computing Infrastructure. *Journal of Physics: Conference Series*, 180(1), 2009.

[162] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The Eucalyptus Open-source Cloud-computing System. *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09)*, pages 124–131, 2009.

[163] OpenNebula Developers. OpenNebula – The Open Source Toolkit for Cloud Computing. `http://www.opennebula.org`, 2010.

[164] Oracle. MySQL: The world's most popular open source database. http://www.mysql.com/, 2010.

[165] Simon Ostermann, Radu Prodan, and Thomas Fahringer. Extending Grids with Cloud Resource Management for Scientific Computing. *Proceedings of the 10th IEEE/ACM International Conference on Grid Computing*, pages 42–49, 2009.

[166] Pradeep Padala, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kenneth Salem. Adaptive Control of Virtualized Resources in Utility Computing Environments. *SIGOPS Oper. Syst. Rev.*, 41(3):289–302, 2007.

[167] Mayur R. Palankar, Adriana Iamnitchi, Matei Ripeanu, and Simson L. Garfinkel. Amazon S3 for Science Grids: A Viable Solution? *Proceedings of the 2008 International Workshop on Data-aware Distributed Computing*, pages 55–64, 2008.

[168] Philip M. Papadopoulos, Mason J. Katz, and Greg Bruno. NPACI Rocks: Tools and Techniques for easily deploying manageable Linux Clusters. *Concurrency and Computation: Practice & Experience*, 2003.

[169] Fabien A. P. Petitcolas, Ross J. Anderson, and Markus G. Kuhn. Information Hiding – A Survey. *Proceedings of the IEEE*, 87(7):1062–1078, 1999.

[170] Charles P. Pfleeger and Shari Lawrence Pfleeger. *Security in Computing*. Pearson Education, 3. edition, 2003.

[171] Hans-Joachim Picht. Utilizing Virtualization Technologies for Cluster and Grid Security. Master's thesis, Philipps-University Marburg, Germany, 2007.

[172] Gerald Popek and Robert Goldberg. Formal Requirements for Virtualizable Third Generation Architectures. *Proceedings of the Fourth ACM Symposium on Operating System Principles (SOSP '73)*, 17(7):412–421, 1974.

[173] Michael L. Powell and Barton P. Miller. Process Migration in DEMOS/MP. *ACM SIGOPS Operating Systems Review*, 17(5):110 – 119, 1983.

[174] Ian Pratt, Keir Fraser, Steven Hand, Christian Limpach, and Andrew Warfield. Xen 3.0 and the Art of Virtualization. *Proceedings of the Linux Symposium*, 2:65–77, 2005.

[175] Daniel Price and Andrew Tucker. Solaris Zones: Operating System Support for Consolidating Commercial Workloads. *Proceedings of the*

18th USENIX Conference on System Administration (LISA '04), pages 241–254, 2004.

[176] David Quigley, Josef Sipek, Charles P. Wright, and Erez Zadok. UnionFS: User- and Community-Oriented Development of a Unification File System. *Proceedings of the 2006 Linux Symposium*, 2:349–362, 2006.

[177] Himanshu Raj and Karsten Schwan. High Performance and Scalable I/O Virtualization via Self-Virtualized Devices. *Proceedings of the 16th international Symposium on High Performance Distributed Computing (HPDC '07)*, 2007.

[178] Darrell Reimer, Arun Thomas, Glenn Ammons, Todd Mummert, Bowen Alpern, and Vasanth Bala. Opening Black Boxes: Using Semantic Information to Combat Virtual Machine Image Sprawl. *Proceedings of the fourth ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments (VEE '08)*, 2008.

[179] Hans P. Reiser and Rüdiger Kapitza. VM-FIT: Supporting Intrusion Tolerance with Virtualisation Technology. *Proceedings of the 1st Workshop on Recent Advances on Intrusion-Tolerant Systems, in Conjunction with EuroSys 2007*, pages 18–22, 2007.

[180] John Scott Robin and Cynthia E. Irvine. Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor. *Proceedings of the 9th Conference on USENIX Security Symposium (SSYM '00)*, 9, 2000.

[181] Benny Rochwerger, David Breitgand, Eliezer Lavy, Alex Galis, Kenneth Nagin, Ignacio Martín Llorente, Rubén Santiago Montero, Yaron Wolfsthal, Erik Elmroth, Juan Cáceres, Muli Ben-Yehuda, Wolfgang Emmerich, and Fermín Galán. The RESERVOIR Model and Architecture for Open Federated Cloud Computing. *IBM Journal of Research and Development*, 53(4), 2009.

[182] Fernando Rodríguez, Felix Freitag, and Leandro Navarro. A Multiple Dimension Slotting Approach for Virtualized Resource Management. *Proceedings of the 1st Workshop on System-level Virtualization for High Performance Computing (HPCVirt) 2007, in conjunction with the 2nd ACM SIGOPS European Conference on Computer Systems (EuroSys)*, 2007.

[183] rPath Inc. rBuilder. `http://www.rpath.org`, January 2011.

[184] Rusty Russell and Harald Welte. Linux netfilter Hacking HOWTO. Technical report, 2000.

[185] Paul Ruth, Xuxian Jiang, Dongyan Xu, and Sebastien Goasguen. Virtual Distributed Environments in a Shared Infrastructure. *Computer*, 38(5):63–69, 2005.

[186] Joanna Rutkowska. Subverting Vista Kernel For Fun And Profit. *Black Hat Briefings*, 2006.

[187] Ritu Sabharwal and Julio Guijarro. Avalanche: Managing Deployments for Enterprise Scale Grids. *Proceedings of the 1st Bangalore Annual Compute Conference (Compute '08)*, 2008.

[188] Salesforce.com. The Force.com Multitenant Architecture. 2008.

[189] Salesforce.com. Force.com: A Comprehensive Look at the World's Premier Cloud-Computing Platform. 2009.

[190] Constantine P. Sapuntzakis, David Brumley, Ramesh Chandra, Nickolai Zeldovich, Jim Chow, Monica S. Lam, and Mendel Rosenblum. Virtual Appliances for Deploying and Maintaining Software. *Proceedings of the 17th USENIX Conference on System Administration (LISA '03)*, 2003.

[191] Constantine P. Sapuntzakis, Ramesh Chandra, Ben Pfaff, Jim Chow, Monica S. Lam, and Mendel Rosenblum. Optimizing the Migration of Virtual Computers. *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, 36(SI):377–390, 2002.

[192] Constantine P. Sapuntzakis and Monica S. Lam. Virtual Appliances in the Collective: A Road to Hassle-Free Computing. *Workshop on Hot Topics in Operating Systems*, 2003.

[193] Shane Schick. Five Ways of defining Cloud Computing. `http://www.itworldcanada.com/blogs/shane/2008/04/22/five-ways-of-defining-cloud-computing/48746/`, April 2008.

[194] Matthias Schmidt, Niels Fallenbeck, Kay Dörnemann, Roland Schwarzkopf, Tobias Pontz, Manfred Grauer, and Bernd Freisleben. *Aufbau einer virtualisierten Cluster-Umgebung*, pages 119–131. 2009.

[195] Matthias Schmidt, Niels Fallenbeck, Roland Schwarzkopf, and Bernd Freisleben. Virtualized Cluster Computing. In *Research Report High-Performance Computing in Hessen*, pages 147–148, 2010.

[196] Matthias Schmidt, Niels Fallenbeck, Matthew Smith, and Bernd Freisleben. Secure Service-Oriented Grid Computing with Public Virtual Worker Nodes. *Proceedings of 35th Euromicro Conference on Internet Technologies, Quality of Service and Applications (ITQSA)*, pages 555–562, 2009.

[197] Matthias Schmidt, Niels Fallenbeck, Matthew Smith, and Bernd Freisleben. Efficient Distribution of Virtual Machines for Cloud Computing. *Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2010), IEEE Press*, pages 567–574, 2010.

[198] Matthias Schmidt, Matthew Smith, Niels Fallenbeck, Hans-Joachim Picht, and Bernd Freisleben. Building a Demilitarized Zone with Data Encryption for Grid Environments. *Proceedings of First International Conference on Networks for Grid Applications*, pages 8–16, 2007.

[199] Roland Schwarzkopf, Matthias Schmidt, Niels Fallenbeck, and Bernd Freisleben. Multi-Layered Virtual Machines for Security Updates in Grid Environments. *Proceedings of 35th Euromicro Conference on Internet Technologies, Quality of Service and Applications (ITQSA)*, pages 563–570, 2009.

[200] Hingzhang Shan and John Shalf. Using IOR to analyze the I/O Performance for HPC Platforms. *eScholarship Repository, Lawrence Berkeley National Laboratory, University of California, University of California*, 2007.

[201] Tom Shanley. *InfiniBand Network Architecture*. MindShare, Inc., 2003.

[202] Michael Sheehan. Cloud Computing Expo: Introducing the Cloud Pyramid. `http://cloudcomputing.sys-con.com/node/609938`, August 2008.

[203] Larry Smarr and Charles E. Catlett. Metacomputing. *Communications of the ACM*, 35(6):44–52, 1992.

[204] Nigel P. Smart and Frederik Vercauteren. Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. *Public Key Cryptography (PKC 2010)*, 6056:420–443, 2010.

[205] Jim Smith and Ravi Nair. *Virtual Machines: Versatile Platforms for Systems and Processes*. Morgan Kaufmann, 2005.

[206] Matthew Smith, Thomas Friese, Michael Engel, Bernd Freisleben, Gregory A. Koenig, and William Yurcik. Security Issues in On-Demand Grid and Cluster Computing. *Sixth IEEE International Symposium on Cluster Computing and the Grid Workshops (CCGRIDW'06)*, pages 14–24, 2006.

[207] Matthew Smith, Thomas Friese, and Bernd Freisleben. Towards a Service-Oriented Ad Hoc Grid. *Proceedings of the Third International Symposium on Parallel and Distributed Computing/Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (ISPDC '04)*, pages 201–208, 2004.

[208] Matthew Smith, Matthias Schmidt, Niels Fallenbeck, Tim Dörnemann, Christian Schridde, and Bernd Freisleben. Secure On-demand Grid Computing. *Journal of Future Generation Computer Systems, Elsevier*, 25(3):315–325, 2009.

[209] Matthew Smith, Matthias Schmidt, Niels Fallenbeck, Christian Schridde, and Bernd Freisleben. Optimising Security Configurations with Service Level Agreements. *Proceedings of the 7th International Conference on Optimization: Techniques and Applications (ICOTA7)*, ICOTA:367–368, 2007.

[210] Matthew Smith, Christian Schridde, and Bernd Freisleben. Securing Stateful Grid Servers through Virtual Server Rotation. *Proceedings of the 17th Internation Symposium on High Performance Distributed Computing (HPDC '08), ACM Press*, pages 11–22, 2008.

[211] Matthew Smith, Fabian Schwarzer, Marian Harbach, Thomas Noll, and Bernd Freisleben. A Streaming Intrusion Detection System for Grid Computing Environments. *Proceedings of the 11th IEEE International Conference on High Performance Computing and Communications (HPCC '09)*, pages 44–51, 2009.

[212] Warren Smith, Ian Foster, and Valerie Taylor. Predicting Application Run Times Using Historical Information. *Job Scheduling Strategies for Parallel Processing, Springer Lecture Notes in Computer Science*, 1459:122–142, 1998.

[213] Stephen Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson. Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors. *SIGOPS Oper. Syst. Rev.*, 41(3):275–287, 2007.

[214] Ahmed Soror, Umar Minhas, Ashraf Aboulnaga, Kenneth Salem, Peter Kokosielis, and Sunil Kamath. Automatic Virtual Machine Configuration for Database Workloads. *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD '08)*, pages 953–966, 2008.

[215] Borja Sotomayor, Katarzyna Keahey, and Ian Foster. Combining Batch Execution and Leasing Using Virtual Machines. *Proceedings of the 17th International Symposium on High Performance Distributed Computing (HPDC '08)*, 2008.

[216] Borja Sotomayor, Rubén Santiago Montero, Ignacio Martín Llorente, and Ian Foster. Capacity Leasing in Cloud Systems using the OpenNebula Engine. *Workshop on Cloud Computing and its Applications (CCA08)*, 2008.

[217] Borja Sotomayor, Rubén Santiago Montero, Ignacio Martín Llorente, and Ian Foster. Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Internet Computing*, pages 14–22, 2009.

[218] SQLite developers. SQLite Home page. `http://www.sqlite.org/`, 2010.

[219] Christopher Strachey. Time-Sharing in Large Fast Computers. *Proceedings of the International Conference on Information Processing*, pages 336–341, 1959.

[220] Frederic Stumpf, Michael Benz, Martin Hermanowski, and Claudia Eckert. An Approach to a Trustworthy System Architecture Using Virtualization. *Autonomic and Trusted Computing, Lecture Notes in Computer Science*, 4610:191–202, 2007.

[221] Frederic Stumpf, Patrick Röder, and Claudia Eckert. An Architecture Providing Virtualization-Based Protection Mechanisms Against Insider Attacks. *Proceedings of the 8th International Conference on Information Security Applications (WISA '07)*, 4867:142–156, 2007.

[222] Jeremy Sugerman, Ganesh Venkitachalam, and Ben-Hong Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. *USENIX Annual Technical Conference*, 2001.

[223] Paul Syverson. A Taxonomy of Replay Attacks. *Proceedings of the 7th Computer Security Foundations Workshop (CSFW)*, pages 187–191, 1994.

[224] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: The Condor experience. *Concurrency and Computation: Practice and Experience, Special Issue: Grids and Web Services for e-Science*, 17(2-4):323–356, 2005.

[225] Marvin M. Theimer, Keith A. Lantz, and David R. Cheriton. Preemptable Remote Execution Facilities for the V-System. *ACM SIGOPS Operating Systems Review*, 19(5):2 – 12, 1985.

[226] Mauricio Tsugawa and José A. B. Fortes. A Virtual Network (ViNe) Architecture for Grid Computing. *Proceedings of the 20th International Parallel and Distributed Processing Symposium*, 2006.

[227] Mauricio Tsugawa and José A. B. Fortes. Characterizing User-Level Network Virtualization: Performance, Overheads and Limits. *Proceedings of the Fourth IEEE Conference on eScience*, pages 206–213, 2008.

[228] Rich Uhlig, Gil Neiger, Dion Rodgers, Amy Santoni, Fernando Martins, Andrew Anderson, Steven Bennett, Alain Kagi, Felix Leung, and Larry Smith. Intel Virtualization Technology. *Computer*, 38(5):48–56, 2005.

[229] Western Union. The Future Role of Western Union as a Nationwide Information Utility. *Strategic Plans*, 1965.

[230] Geoffroy Vallée, Thomas Naughton, Christian Engelmann, Hong Ong, and Stephen L. Scott. System-Level Virtualization for High Performance Computing. *Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP '08)*, pages 636–643, 2008.

[231] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A Break in the Clouds: Towards a Cloud Definition. *ACM SIGCOMM Computer Communication Review*, (39):1–6, 2009.

[232] Constantino Vázquez, Eduardo Huedo, Rubén Santiago Montero, and Ignacio Martín Llorente. On the Use of Clouds for Grid Resource Provisioning. *Future Generation Computer Systems, ELSEVIER*, 2010.

[233] Sam Verboven, Peter Hellinckx, Frans Arickx, and Jan Broeckhove. Runtime Prediction based Grid Scheduling of Parameter Sweep Jobs. *Proceedings of the 2008 IEEE Asia-Pacific Services Computing Conference (APSCC '08)*, pages 33–38, 2009.

[234] VMware developers. Virtual Disk Format 1.1. Technical report, 2007.

[235] Eugen Volk, Jochen Buchholz, Stefan Wesner, Daniela Koudela, Matthias Schmidt, Niels Fallenbeck, Roland Schwarzkopf, Bernd Freisleben, Götz Isenmann, Jürgen Schwitalla, Marc Lohrer, Erich Focht, and Andreas Jeutter. Towards Intelligent Management of Very Large Computing Systems. *Proceedings of Competence in High Performance Computing (CiHPC)*, 2010.

[236] Gian Luca Volpato and Christian Grimm. Recommendations for Static Firewall Configuration in D-Grid. Technical report, D-Grid Integrationsprojekt (DGI), 2007.

[237] Gregor von Laszewski, Joseph A. Insley, Ian Foster, John Bresnahan, Carl Kesselman, Mei Su, Marcus Thiebaux, Mark L. Rivers, Steve Wang, Brian Tieman, and Ian McNulty. Real-Time Analysis, Visualization, and Steering of Microtomography Experiments at Photon Sources. *Proceedings of the 9th SIAM Conference on Parallel Processing for Scientific Computing*, 1999.

[238] Edward Walker. Benchmarking Amazon EC2 for High-Performance Scientific Computing. *Login, Vol. 33, No. 5*, pages 18–23, 2008.

[239] Jinpeng Wei, Xiaolan Zhang, Glenn Ammons, Vasanth Bala, and Peng Ning. Managing Security of Virtual Machine Images in a Cloud Environment. *Proceedings of the 2009 ACM Workshop on Cloud Computing Security (CCSW '09)*, pages 91 – 96, 2009.

[240] Christof Weinhardt, Arun Anandasivam, Benjamin Blau, and Jochen Stößer. Business Models in the Service World. *IT Professional*, pages 36–41, 2009.

[241] Aaron Weiss. Computing in the Clouds. *netWorker*, 11(4):16–25, 2007.

[242] Andrew Whitaker, Richard Cox, Marianne Shaw, and Steven Gribble. Rethinking the Design of Virtual Machine Monitors. *Computer*, 38(5):57–62, 2005.

[243] Mark White, Bill Briggs, and Chris Weitz. Depth perception - A dozen technology trends shaping business and IT in 2010. *Deloitte Technology Trends 2010*, 2010.

[244] Matthew S. Wilson. Constructing and Managing Appliances for Cloud Deployments from Repositories of Reusable Components. *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing (HotCloud '09)*, 2009.

[245] David Isaac Wolinsky, Abhishek Agarwal, P. Oscar Boykin, Justin R. Davis, Arijit Ganguly, Vladimir Paramygin, Y. Peter Sheng, and Renato J. Figueiredo. On the Design of Virtual Machine Sandboxes for Distributed Computing in Wide-area Overlays of Virtual Workstations. *Proceedings of the Second International Workshop on Virtualization Technology in Distributed Computing(VTDC 2006)*, 2006.

[246] Ming Q Xu. Effective Metacomputing using LSF MultiCluster. *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 100–105, 2001.

[247] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. When Private Keys are Public: Results from the 2008 Debian OpenSSL Vulnerability. *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference (IMC '09)*, 2009.

[248] Lamia Youseff, Rich Wolski, Brent Gorda, and Chandra Krintz. Evaluating the Performance Impact of Xen on MPI and Process Execution for HPC Systems. *2nd International Workshop on Virtualization Technology in Distributed Computing (VTDC)*, 2006.

[249] Lamia Youseff, Rich Wolski, Brent Gorda, and Chandra Krintz. Paravirtualization For HPC Systems. *Workshop on XEN in HPC Cluster and Grid Computing Environments (XHPC) on the International Symposium on Parallel and Distributed Processing and Application (ISPA 2006)*, pages 474–486, 2006.

[250] Weikuan Yu and Jeffrey S. Vetter. Xen-Based HPC: A Parallel I/O Perspective. *Cluster Computing and the Grid*, pages 154–161, 2008.

[251] Yang Yu, Hariharan Kolam, Lap-chung Lam, and Tzi-cker Chiueh. Applications of a Feather-weight Virtual Machine. *Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '08)*, pages 171–180, 2008.

[252] Bin Zhang, Ehab Al-Shaer, Radha Jagadeesan, James Riely, and Corin Pitcher. Specifications of A High-level Conflict-Free Firewall Policy Language for Multi-domain Networks. *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies (SACMAT '07)*, pages 185–194, 2007.

[253] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud Computing: State-of-the-Art and Research Challenges. *Journal of Internet Services and Applications, Springer Press*, (1):7 – 18, 2010.

[254] Wenli Zhang, Mingyu Chen, and Jianping Fan. HPL Performance Prevision to Intending System Improvement. *Parallel and Distributed Processing and Applications, Springer Lecture Notes in Computer Science*, 3358:777–782, 2004.

[255] Xuehai Zhang, Katarzyna Keahey, Ian Foster, and Timothy Freeman. Virtual Cluster Workspaces for Grid Applications. *ANL Tech Report ANL/MCS-P1246-0405*, 2005.

Index

Curriculum Vitae

Niels Fallenbeck

Theresienstraße 46
80333 München
Deutschland

Telefon +49 (170) 4704400
eMail niels@fallenbeck.com

geboren 5. Dezember 1978 in Gießen
ledig

Ausbildung

05/2007 – 04/2011	Philipps-Universität Marburg Promotionsstudium: Informatik Promotion zum Dr. rer. nat. Abschlussnote: „Sehr gut”
10/2000 – 04/2007	Philipps-Universität Marburg Diplomstudium: Informatik Abschlussnote: „Sehr gut”

Berufserfahrung

seit 08/2011	Fraunhofer AISEC (Applied and Integrated Security) Research Fellow
05/2007 – 04/2011	Philipps-Universität Marburg Wissenschaftlicher Mitarbeiter Schwerpunkte: Cloud Computing, Grid Compting, Sicherheit, Skalierbarkeit
07/2002 – 04/2007	Eisenberg-Film GmbH Leitung IT-Administration
01/2003 – 09/2003	Bundeskongress Soziale Arbeit Beratung, Konzeption, Realisierung, Betreuung und Hosting der offiziellen Internet-Präsenz