Utility-based Allocation of Resources to Virtual Machines in Cloud Computing

Dissertation

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

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

Dorian Minarolli
geboren in Shkoder

Marburg, 2014
Hochschulkennziffer 1180

Vom Fachbereich Mathematik und Informatik der
Philipps-Universität Marburg als Dissertation am
14.11.2014
angenommen.

Erstgutachter: Prof. Dr. Bernd Freisleben, Philipps-Universität Marburg

Zweitgutachter: Prof. Dr. Betim Çiço, South East European University,
Tetovo, Macedonia.

Tag der Einreichung: *19.08.2014*.
Tag der mündlichen Prüfung: *14.11.2014*.

Zusammenfassung

In den letzten Jahren hat Cloud Computing eine hohe Popularität als neues Informationsverarbeitungsmodell erreicht, in dem Ressourcen elastisch nach Bedarf in einem *pay-as-you-go* Modus genutzt werden können. Ein wichtiges Ziel eines Cloud-Anbieters ist die dynamische Zuweisung von Ressourcen zu Virtuellen Maschinen (VMs) aufgrund der aktuellen Auslastung, um die Performanz von Anwendungen auf einem vereinbarten Service-Level-Agreement (SLA) Niveau zu halten, bei gleichzeitiger Reduktion der Kosten von Ressourcen. Das Problem besteht darin, einen adäquaten Kompromiss zwischen den beiden gegensätzlichen Zielen der Performanz von Anwendungen und den Kosten von Ressourcen zu finden. In dieser Dissertation werden Ansätze für die Erzielung eines solchen Kompromisses vorgestellt, die auf der Verwendung einer Nutzenfunktion für Performanz und Kosten basieren. Die vorgeschlagenen Lösungen allokieren Ressourcen zu VMs auf globaler Ebene eines Rechenzentrums und der lokalen Ebene realer Maschinen anhand der Optimierung der Nutzenfunktion. Die Nutzenfunktion, gegeben als Differenz zwischen Performanz und Kosten, repräsentiert den Profit des Cloud-Anbieters und bietet die Möglichkeit, den Performanz-Kosten Kompromiss in einer flexiblen und natürlichen Weise auszudrücken.

Für die globale Ebene wird eine zweistufige Zuteilungsstrategie der Ressourcen vorgestellt. In der ersten Stufe weisen Steuerungsinstanzen in den lokalen Rechnern dynamisch Ressourcen-Anteile an die VMs zu, um eine lokale Nutzenfunktion zu maximieren. In der zweiten Stufe gibt es eine globale Steuerungsinstanz, die VM Live-Migration zur Optimierung einer globalen Nutzenfunktion durchführt. Experimentelle Ergebnisse zeigen, dass die Optimierung der globalen Nutzenfunktion durch Änderung der Anzahl der realen Rechner aufgrund der Auslastung die Performanz von Anwendungen auf einem akzeptablen Niveau hält, bei gleichzeitiger Senkung der Kosten von Ressourcen.

Um mehrere Ressourcen auf lokaler Ebene zuzuteilen, wird ein Ansatz auf Grundlage der Rückkopplungssteuerungstheorie und der Optimierung der Nutzenfunktion vorgeschlagen. Hierdurch werden Anteile von mehreren Ressourcen, wie CPU, Speicher, Festplatte und Netzwerk I/O-Bandbreite, dynamisch an VMs zugewiesen. Zur Bewältigung der komplexen Nicht-Linearitäten, die in gemeinsam genutzten virtualisierten Infrastrukturen zwischen VM-Performanz und Ressourcenzuteilung existieren, wird eine Lösung vorgeschlagen, die Ressourcen zu VMs anhand einer Nutzenfunktion basierend auf Performanz und

Energieverbrauch zuteilt. Ein künstliches neuronales Netzwerk (KNN) wird verwendet, um ein Online-Modell der Beziehung zwischen VM-Ressourcenzuteilung und Anwendungsperformanz und ein weiteres Online-Modell der Beziehung zwischen VM-Ressourcenzuteilung und Energieverbrauch zu erstellen. Um hohe Optimierungszeiten im Falle einer erhöhten Anzahl von VMs zu vermeiden, wird ein verteilter Ressourcen-Manager vorgeschlagen. Dieser besteht aus mehreren KNNs, die jeweils für die Modellierung und Ressourcenzuweisung einer VM verantwortlich sind, aber Informationen mit anderen KNNs fr die Koordination der Ressourcenzuteilungen austauschen. Experimentelle Ergebnisse in simulierten und realistischen Umgebungen zeigen, dass die verteilten KNN Ressourcen-Manager bessere Performanz-Kosten Kompromisse erzielen als eine zentralisierte Version und eine verteilte unkoordinierte Version des Ressourcen-Managers.

Um die Problematik des Erstellens eines akkuraten Online-Anwendungsmodells verbunden mit langen Modelladaptionszeiten zu behandeln, wird ein Ansatz auf der Grundlage der Fuzzy-Steuerung vorgeschlagen. Er optimiert eine Nutzenfunktion basierend auf einer Hill Climbling Heuristik, die als Fuzzy-Regeln implementiert ist. Zur Vermeidung langer Optimierungszeiten im Fall einer erhöhten Anzahl von VMs wird ein Multi-Agenten Fuzzy-Regler entwickelt, bei dem jeder Agent, parallel zu anderen Agenten, seine eigene lokale Nutzenfunktion optimiert. Der Fuzzy-Steuerungsansatz eliminiert die Notwendigkeit, ein Modell vorab aufbauen und bietet eine robuste Lösung auch für verrauschte Messungen. Experimentelle Ergebnisse zeigen, dass die Multi-Agenten Fuzzy-Regler die Nutzenfunktion besser optimieren als eine zentrale Version eines Fuzzy-Reglers und ein aktueller adaptiver optimaler Steuerungsansatz, insbesondere bei einer erhöhten Anzahl von VMs.

Um einige der Probleme reaktiver VM-Ressourcenzuteilunsansätze zu behandeln, wird eine proaktive Ressourcenallokationslösung vorgeschlagen. Dieser Ansatz entscheidet über VM-Ressourcenzuweisungen basierend auf der Vorhersage des Ressourcenbedarfs mittels der maschinellen Lernmethode der Support Vector Machine (SVM). Um die Abhängigkeiten zwischen VMs der gleichen Multi-Tier-Anwendung zu modellieren, wird die Bedarfsprognose mehrerer kreuzkorrelierter Ressourcennutzungszeitreihen aller VMs einer mehrschichtigen Anwendung durchgeführt. Wie Experimente zeigen, führt dies zu einer verbesserten Vorhersagegenauigkeit und Anwendungsperformanz.

Abstract

In recent years, cloud computing has gained a wide spread use as a new computing model that offers elastic resources on demand, in a pay-as-you-go fashion. One important goal of a cloud provider is dynamic allocation of Virtual Machines (VMs) according to workload changes in order to keep application performance to Service Level Agreement (SLA) levels, while reducing resource costs. The problem is to find an adequate trade-off between the two conflicting objectives of application performance and resource costs. In this dissertation, resource allocation solutions for this trade-off are proposed by expressing application performance and resource costs in a utility function. The proposed solutions allocate VM resources at the global data center level and at the local physical machine level by optimizing the utility function. The utility function, given as the difference between performance and costs, represents the profit of the cloud provider and offers the possibility to capture in a flexible and natural way the performance-cost trade-off.

For global level resource allocation, a two-tier resource management solution is developed. In the first tier, local node controllers are located that dynamically allocate resource shares to VMs, so to maximize a local node utility function. In the second tier, there is a global controller that makes VM live migration decisions in order to maximize a global utility function. Experimental results show that optimizing the global utility function by changing the number of physical nodes according to workload maintains the performance at acceptable levels while reducing costs.

To allocate multiple resources at the local physical machine level, a solution based on feed-back control theory and utility function optimization is proposed. This dynamically allocates shares to multiple resources of VMs such as CPU, memory, disk and network I/O bandwidth. In addressing the complex non-linearities that exist in shared virtualized infrastructures between VM performance and resource allocations, a solution is proposed that allocates VM resources to optimize a utility function based on application performance and power modelling. An Artificial Neural Network (ANN) is used to build an online model of the relationships between VM resource allocations and application performance, and another one between VM resource allocations and physical machine power. To cope with large utility optimization times in the case of an increased number of VMs, a distributed resource manager is proposed. It consists of several ANNs, each responsible for modelling and resource allocation of one VM, while exchanging information with other ANNs for coordinating resource allocations. Experiments, in simulated and realistic environments, show that the distributed ANN resource manager achieves better performance-power trade-offs than a centralized version and a distributed non-coordinated resource manager.

To deal with the difficulty of building an accurate online application model and long model adaptation time, a solution that offers model-free resource management based on fuzzy control is proposed. It optimizes a utility function based on a hill-climbing search heuristic implemented as fuzzy rules. To cope with long utility optimization time in the case of an increased number of VMs, a multi-agent fuzzy controller is developed where each agent, in parallel with others, optimizes its own local utility function. The fuzzy control approach eliminates the need to build a model beforehand and provides a robust solution even for noisy measurements. Experimental results show that the multi-agent fuzzy controller performs better in terms of utility value than a centralized fuzzy control version and a state-of-the-art adaptive optimal control approach, especially for an increased number of VMs.

Finally, to address some of the problems of reactive VM resource allocation approaches, a proactive resource allocation solution is proposed. This approach decides on VM resource allocations based on resource demand prediction, using a machine learning technique called Support Vector Machine (SVM). To deal with interdependencies between VMs of the same multi-tier application, cross-correlation demand prediction of multiple resource usage time series of all VMs of the multi-tier application is applied. As experiments show, this results in improved prediction accuracy and application performance.

Erklärung

Ich versichere, dass ich meine Dissertation

Utility-based Allocation of Resources to Virtual Machines in Cloud 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.

Dorian Minarolli
Marburg, im August 2014

Acknowledgments

The work for this PhD dissertation has extended over the course of five years. This experience has proven to be intense, fulfilling and at the same time challenging, and it would not have been possible without the help and support of several people whom I would like to thank.

First of all, I would like to thank my supervisor, Prof. Dr. Bernd Freisleben, for his support and advice during this PhD study. He has guided me throughout my research and provided helpful comments on versions of this dissertation. I also thank him for the support given to secure my stay in Germany and for awarding me a research assistant position that supported me financially during the last years.

I would also like to thank Prof. Dr. Betim Çiço, who encouraged and supported me to pursue the doctoral studies at the University of Marburg.

A warm thank you goes to the former and present colleagues of the Distributed Systems Group at the University of Marburg for their support and advice. I want to thank Kay Dörnemann, Ernst Juhnke, Dr. Niels Fallenbeck, Afef Mdhaffar, Dr. Matthias Schmidt, Roland Schwarzkopf, Pablo Graubner, Dr. Tim Dörnemann, Timo Féret, Dr. Matthew Smith, Dr. Ralph Ewerth, Dominik Seiler, Matthias Leinweber, Lars Baumgärtner, Markus Mühling, Dr. Markus Mathes, Dr. Steffen Heinzl, Dr. Christian Schridde, and also the Faculty Secretary Mechthild Keßler for her support with administrative matters.

Last but not least, a special thank you goes to my family for the great moral support and encouragement during those challenging moments of my studies.

This PhD work is partly supported by the German Ministry of Education and Research (BMBF) (D-Grid Initiative and HPC Call) and the Albanian Government (Excellence Fund).

Contents

1 Introduction

1.1 Motivation

In recent years, cloud computing has gained a wide spread use and attention from both academia and commercial IT enterprises. This is a new computing paradigm that delivers on demand computing resources, from a shared pool of resources, by supporting the pay-as-you-go model. In this paradigm, cloud consumers can lease compute resource (e.g CPU) or storage (e.g disk space) as much as they need. On demand, they can later increase or decrease this amount and pay only for the amount of resources they consume. This paradigm is similar to the power grids that offer electricity as a utility and the payment is based only on the consumed amount. The resource can be offered in different layers of abstractions. The cloud is composed of three different models, (1) Software-as-a-Service (SaaS) that offers resources as software abstraction, (2) Platform-as-a-Service (PaaS) that offers resources as operating system services or programming frameworks and, (3) Infrastructure-as-a-Service (IaaS) that offers resources as raw computing power or storage. Among these three models, the most important and widely used is the IaaS. IaaS, with its most representative example Amazon Elastic Compute Cloud (EC2) [4], will also be the focus of this work. One of the main benefits of the IaaS is the reduced Total Cost of Ownership (TCO) for the cloud consumer. He does not need to make big investments in setting up a physical infrastructure but can rent the necessary resources from the cloud and if needed in the future, increase them. Moreover, the customer does not incur any expenses on managing the physical machines.

One of the enabling technologies of IaaS clouds is virtualization, provided by KVM [65], Xen [8] or VMware [112]. A virtualization technology offers the possibility to encapsulate all software, from the operating system to applications, in a container called a Virtual Machine (VM). Moreover, a software layer called Virtual Machine Monitor (VMM) (Hypervisor in Xen terminology), which manages resources to VMs, makes it possible to run several VMs on the same physical machine. The VMM plays for VMs the same role as an operating system kernel plays for application processes. VMM offers security and fault isolation between VMs running on top of it. Thus, VM technology makes it easier for IaaS to offer compute resources as VMs of different sizes which can be added or removed on demand.

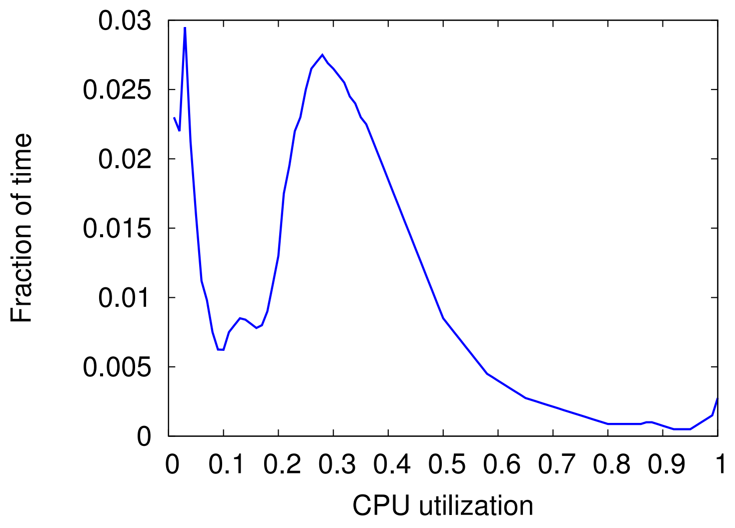


Figure 1.1: Average CPU utilization of more than 5000 servers during a period of six months (source: The Case for Energy-Proportional Computing [10]).

Besides its benefits to cloud consumers, virtualization technology also brings benefits to cloud providers. Due to the workload nature of Internet application servers, most are underutilized with an estimation of resource utilization of 15 to 20 percent [115]. Figure 1.1 shows the average CPU utilization of more than 5,000 servers at Google, during a six-month period. When inspected over longer time frames, the utilization tends to fluctuate between 10 and 50 percent [115]. The usual practice nowadays in data centers is over-provisioning of resources for the peak demand, in order to have acceptable performance even on high load. But since there is a big difference in resource utilization, between low and high load demand, most of the time there will be a waste of resources and high power costs. The static allocation of resources also increases the risk of application performance violations, especially when there is a burst in workload demand, as in the case of seasonal on-line offers or unexpected news events. This is also a non-efficient practice, as it gives some applications more resources than needed while letting some others starve for it.

On the other hand, virtualization technology offers the possibility to adjust resources according to the workload through two mechanism: a) *VM live migra-*

tion, and b) *dynamic resource share allocation*. VM live migration [22] makes it possible to transfer a running VM from one physical machine to another, with minimal interruption and down time. Dynamic resource sharing allocation makes it possible to give a certain resource share (e.g. CPU share) to a VM and change it later at run time. These mechanisms open up the possibility for VMs, with low server utilization and variable workload, to be consolidated at run time to fewer physical machines and dynamically adapt their resources according to the workload. This increases overall utilization, as well as reduces power, cooling and management costs of the cloud provider data center. Given that today's big data centers consist of hundreds, even thousands of servers, applying the above mechanism will result in huge savings in power and operating costs. Also, cloud consumers will benefit from virtualization technology mechanisms, since they will get the performance required with reduced costs. As a result, they will need to pay only for the minimum needed resources for the actual application workload demand.

On the other side, because of the huge management complexity of driving this mechanism at run time, it is impossible for a human administrator to deal with them. Thus, the need emerges to have automatic, high level decision-making techniques that manage VM resource allocation at run time, requiring little human intervention. These techniques must be able to allocate VM resources in a way as to keep application performance according to SLA levels, while increasing utilization, reducing power and operating costs of the cloud provider infrastructure. The focus of this work is devising automatic VM resource allocation techniques to manage the application performance and operating cost in cloud computing infrastructures.

1.2 Research Challenges

There are several challenges related to IaaS automatic VM resource allocation, which it is divided into two groups. In the first group, challenges relate to resource allocation at the global data center level through VM live migration, while in the second, they relate to dynamic resource allocation at the local physical machine level.

- **Global level VM resource allocation challenges**. VM live migration is used as a resource allocation mechanism for dynamically consolidating low utilization VMs on few physical machines. It can also be used for load balancing, in an overloaded situation, to spread the VMs on other physical machines in order to avoid performance level violations.

  There are several questions that need to be addressed here, such as: When to migrate a VM? Which VM to migrate? To what destination physical machine has to migrate to? In other terms, this is the general problem of finding the mapping between VMs to physical machines and dynamically changing it, as the workload changes at run time, in order to lower the number of physical machines and keep application performance to SLA levels. This can be seen as an instance of the bin-packing problem for packing VMs of different sizes into the smallest numbers of bins (physical machines). There are some proposals [122, 63, 70] that treat it as a bin-packing problem where the sizes of items to be packed are VM resource utilizations (e.g. CPU utilization) and the sizes of bins are resource capacities of physical machines. Since, in general, the bin-packing problem is NP-hard, these works use heuristic algorithms to find approximate solutions to the problem. The problem with these approaches is that they use low-level metrics, such as resource utilization, to decide when and which VM to migrate. But making migration decisions only on utilization metrics fails to provide an assurance on high-level performance metrics. This is firstly because the mapping between resource utilization and performance is not clear and changes with the workload, and secondly, because the performance interference the migrating VM can have with other VMs on the destination physical machine is unknown. What complicates things more is the fact of having multiple resources that makes it a multi-size bin-packing problem. Also, a data center can have heterogeneous physical machines of different resource capacities, so the bins are of different sizes. The degree of complication increases because as the workload changes, the sizes of VMs change, and thus the problem becomes multi-size bin packing with items to be packed elastically in size. This requires a combination of local VM resource allocation with global resource allocation through VM live migration. This combination is not yet adequately incorporated into existing state-of-art techniques.

These proposals are based on a rigid policy implemented on heuristic ordering algorithms, such as First-Fit-Decreasing (FFD), of only packing VMs to as few physical machines as possible which can be problematic, as a data center later may need to follow a load balancing policy of using as much physical machines as possible. This policy change requires completely changing the algorithm. What is required is a more flexible approach that can change according to data center policy changes. Existing heuristic algorithms also have difficulties simultaneously taking into account multiple conflicting criteria such as power consumption, performance metrics, cache interference, VM communication dependencies etc. An approach that makes live migration decisions based on higher-level metrics, such application performance or utility values, will be a more preferable and flexible choice. Basing VM resource allocations and live migration decisions on the optimization of a utility function would offer the required multiple criteria optimization and flexible allocation policy changes.

Another problem with existing approaches using linear and constraint programming optimization techniques is the long optimization time. This makes them impractical for real time resource allocation. What is required is a simple, lightweight and low overhead approach for utility function optimization.

Another issue is that heuristic algorithms should not find a mapping of VMs to physical machines that result in a complete rearrangement compared to the previous configuration. Even if the new mapping could be optimal, in terms of the number of physical machines, it may require a lot of VMs live migrations taking place at the same time. This can cause a long reconfiguration time and a big overhead for the network and the VM performance.

The last issue is the problem that can arise when live migration decisions are taken according the load. In the case when the load changes fast in the inverse directions, it can make a VM migrate forth and back continuously in ping-pong way. Thus the decision-making technique should take into consideration also this kind of stability issue.

- **Local level VM resource allocation challenges**. One approach to VM resource allocation in a physical machine is to set shares according to resource utilisation. For example, there are works [118, 49, 90] that use feed-back control theory approaches to keep resource utilisation to a certain level in order to keep performance to an acceptable level. They suppose that a certain utilisation level corresponds to a certain performance level. The problem here is that the mapping between a certain resource utilisation level and performance is hard to find. The mapping changes over time and is different for different workload intensities and workload mixes.

  A better approach should be to find a direct mapping or model between resource allocation shares and performance levels, which would create the possibility to directly manage performance and keep it according to SLA levels. There are works [130, 90] that adopt the control-theory approach to directly control the performance to a certain level. The problem with these approaches is that they use linear models of resource allocation performance mappings that do not work well in practice. This is because in general the performance is a non-linear function of resource allocations [67, 118]. The linear models capture the relationship between resources and performance only around the operating point, which can be approximated as linear, but if the workload changes to another operating point the model is no longer valid. There are some approaches that somehow cope with this by building adaptive linear models such as ARMA [89]. In this case, the model parameters are continuously updated on-line, but again, this remains a linear model and the adaptation time to the new operating point is very long.

  Another issue with control-theory approaches is the difficulty to simultaneously control different conflicting objectives, such as power and performance. A more flexible solution, also mentioned in the global resource allocations research challenges, is the use of utility functions that offers a natural way of optimizing across different objectives. One consequence of using optimization techniques for resource allocation is that, since it depends on the number of resource allocation combinations, increasing the number of VMs increases the optimization time as well. A promising approach is a distributed resource management solution where optimization is done for each VM separately therefore reducing time complexity. On the other side, this approach raises the need for coordination between different VM resource allocations in order to eliminate conflicting decisions, especially between different VMs of the same application.

Other works [28, 14, 75] make use of queuing theory to model non-linear resource performance relationships. However, they are very simple mathematical models unable to capture complex relationships between performance and resource allocation in shared virtualized environments. There are several reasons for this complexity but the most notable one is performance interference between VMs running on the same physical machine. This means that the performance of a VM does not depend only on the resource allocation given to it but also on the workload of other VMs and the resource allocation given to them. This interference comes as a result of several reasons. Virtualization technology in general offers fault and security isolation between VMs, but it does not provide performance isolation [66]. Although some resources can be partitioned (CPU and memory), other ones (shared caches and disk I/O bandwidth) cannot do so. The VMs will have contention for these resources resulting in unpredictable performance interference between them. In some virtualization technologies, such as Xen, a special VM called Dom0 responsible for doing I/O operations on behalf of other VMs exists. Even the Dom0 VM can be a source of contention [43] because an I/O intensive VM can deprive other VMs of using Dom0 for doing their I/O work. Another issue that increases the complexity of resource-performance relationships is the dependence of application performance on the ability to access at the same time multiple type of resources and the complex interdependencies between resources usages. This also places the need to have resource allocation techniques that take into account resource dependencies by allocating multiple resources simultaneously. The complex relationship between resources and performance makes it nearly impossible to built accurate mathematical models. Moreover, it makes it even more difficult to build off-line empirical models, due to the difficulty of creating all situations and workloads that can happen in a real environment.

One option that has the potential to create more robust and accurate models is to build on-line empirical models, using machine-learning techniques. However, this approach is not free of challenges. Firstly, since the model is trained on-line during application runtime, the training configuration samples applied should be representative of a wide range of values in order to build an accurate model. On the other hand, they should not violate application performance. This is related to the well-known trade-off between exploitation and exploration, where exploitation actions are taken from the learned policy or model and exploration actions are random to explore unknown regions. In order to reduce performance violations, the training time should also be low. Moreover, since the performance-resource relationship can change over time, the model should also have a low online adaptation time. A second challenge lies in the fact that some measured metrics, such as VM performance, are very noisy and can create difficulty to build accurate online models. To reduce the noise one should increase the control interval between training samples. However, doing so will increase online training time, which results in another trade-off needed to be taken into consideration. A third issue is that of delayed effects of resource allocation on application performance, where effects on the performance of a resource configuration are only shown after several control intervals. These delayed effects create difficulties in building accurate performance models.

Another issue is that resource allocation is usually done in a reactive way, meaning that the amount of resources allocated for the next interval is based on previous, already happened events. This implies that allocation decisions are often taken when SLA performance violations have already occurred. A better approach would be a proactive resource allocation that predicts resource utilization and allocates resources before it reaches a certain level, eliminating performance SLA violations. Usually, existing proactive resource allocation approaches make predictions of utilization time series separately for each resource. This has the drawback of not taking into account cross-correlation between time series of different resources. The latter is especially true in IaaS cloud environments running multi-tier applications, where there are interdependencies between VMs of the same application. A new approach that takes into account cross-correlation between resource of multiple VMs of the same application is needed. This would result in better prediction accuracy and resource allocation decisions.

1.3 Research Contributions

The main contributions of this dissertation are several approaches to automatically allocate VM resources that manage application performance and operating costs of an IaaS cloud. These approaches range from global optimization at the data center level (by VM live migration), to the physical machine level (by dynamic resource share allocation.):

- **Utility-based Virtual Machine Consolidation and Load Balancing**

  An approach for VM consolidation and load balancing based on optimizing a utility function that allows finding the right trade-off between performances and operating costs is presented. This approach combines control-theory based dynamic resource allocation at the physical machine level and a heuristic driven VM live migration at the cluster level. The first one maximizes a local utility function, while the second optimizes a global utility function. Local and global utility functions represent the profit derived from one physical machine and that from the entire cluster. Most of the works base their live migration decisions on fixed heuristics driven by VM utilization and threshold values. The novelty of this approach is the combination of local and global resource allocations as well as making migration decisions based on a higher-level metric, such as utility value. This is more important for the cloud provider as it represents its profit. The utility function is expressed as the difference between the utility of VMs and the cost of cloud provider infrastructure. The utility of a VM represents the monetary value the cloud consumer pays to the cloud provider for consuming a certain amount of resources. The cost represents the sum of the operating costs for every physical machine of the infrastructure. The SLA contract in this case charges based on the resources consumed and tries to guarantee them by maximizing the utility function. Compared to existing works this approach also offers more flexibility in changing the allocation policy. By doing so it makes it more aggressive in consolidating or balancing the load, by only changing the weight coefficients of VMs utilities or costs functions, as will be explained more deeply in Chapter 3. This approach also takes measures to reduce stability problems that can occur with VM live migrations.

- **Allocation of Multiple Types of Resources to Virtual Machines Using Control Theory**

  The next contribution is the development of a dynamic VM resource allocation technique on the local physical machine level. It addresses the allocation problem of multiple types of resource to VMs in order to optimize a utility function that represents the cloud provider's profit from one physical machine. The approach is based on applying feed-back control theory to control multiple resources such as CPU, memory, disk and network I/O. This keeps their utilizations to certain levels and the performance according to SLAs. It also gives different priorities to different VMs and in the case of contention, for any resource, it applies an algorithm that resolves it by allocating resources in such a way as to maximise a utility function. The utility function is expressed as the difference between the utility of VMs and the operating cost of one physical machine. The utility of a VM represents the monetary value the consumer pays to the provider for consuming a certain amount of resources and getting a certain performance level. The SLA contract, in this case, has the form of a utility function that charges on resource consumption and provided performance levels. The contract tries to guarantee both of them by applying feed-back control and utility function maximization. Since most of the works focus on allocating one or two resources, the novelty of this approach is considering multiple types of resources and developing a new form of SLA contract that takes into consideration both resources consumed and performance levels.

- **Virtual Machine Resource Allocation via Multi-Agent Fuzzy Control**

  A VM resource allocation approach that allocates CPU and memory resources of a physical machine through the optimization of a utility function is presented. The utility function expresses two conflicting objectives of performance and resource costs, and represents the provider's profit deriving from one physical machine. Utility function optimization is achieved through a fuzzy-controller, by using a hill-climbing search heuristic implemented through fuzzy-rules. Compared to other approaches, heuristic based fuzzy-control allocation does not require building beforehand an analytical or statistical model of the relationship between resource allocation and performance metrics. What is needed is just the specification of some fuzzy-rules for the hill-climbing algorithm. Another advantage of the fuzzy-control approach is that the amount of allocation that a VM resource can be increased is not a fixed step value, but can be adapted at run time according to the amount of utility change. The fuzziness also makes it possible to make adequate decisions even in the presence of noisy utility value measurements. To cope with long utility optimization time, as a result of an increased number of VMs, a multi-agent fuzzy controller is developed, where the global utility function is divided into local utility functions. The fuzzy-controller is divided into multiple agents, each responsible for resource allocation of one VM through optimization of its own local utility function.

- **Distributed Resource Allocation to Virtual Machines via Artificial Neural Networks**

  This approach allocates resources at local physical machine level. It optimizes a utility function expressing the trade off between the application performance and power consumption of the physical machine. The utility function represents the profit that the provider derives from one physical machine. The function depends on performance levels of VMs and power consumption of the physical machine. This means that we have a SLA contract model stating the application performance levels that should be kept by the cloud provider. The optimization is done by using an Artificial Neural Network (ANN) based model of the relationships between VM resource allocations, performance and power consumption of the physical machine. The inputs to the ANN model are VM resource allocations and the outputs are the performance and power levels of VMs. Firstly, this avoids dealing with low-level resource utilizations by managing directly the application performance. Secondly, since ANN are known for their good approximations of non-linear functions, it addresses the problem of capturing non-linear relationships, between performance and resource allocations, in virtualized environments. The ANN models are built on-line, avoiding the need to build an analytical model beforehand, thus making the approach independent of any specific application.

  There are two developed versions of the resource manager. In the centralized resource manager, two Multiple-Input Multiple-Output (MIMO) ANN based models are being used. The first model captures the relationship between resource allocations and performance metrics, and the other captures the relationship between resource allocations and physical machine power consumption. To cope with the increased number of VMs, which can result in an increased utility optimization time, a distributed version of resource manager is developed. The centralized manager is divided into several managers each responsible for resource allocation of one VM through optimization of its own local utility function.

- **Cross-Correlation Prediction for Virtual Machine Resource Allocation Using Support Vector Machines**

  An approach called Automatic Proactive Resource Allocation (APRA) is presented, which pro-actively allocates resources to VMs of multi-tier applications running across several physical machines of a cloud-computing infrastructure. This is achieved using a supervised machine learning technique for time series forecasting, namely Support Vector Machine (SVM), which predicts resource usage demand. Based on this prediction, VMs receive the minimum allocations to satisfy their demands, thus reducing resource costs. Given the interdependencies between VMs of a multi-tier application and correlated resource usage between multiple VMs of the same application, cross-correlation prediction is applied by simultaneously making predictions of multiple resources of the multi-tier application. This increases the prediction accuracy of resource usage and therefore improves the resource allocation to VMs, making it possible to mitigate SLA performance violations. Another advantage of this approach is that it is non-intrusive, since it makes allocation decisions based only on resource usage metrics measured outside VMs.

1.4 Publications

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

1. Dorian Minarolli and Bernd Freisleben. Utility-based Resource Allocation for Virtual Machines in Cloud Computing. In *Proceedings of 16th IEEE Symposium on Computers and Communications, ISCC 2011*, pp. 410–417, IEEE press, 2011.

2. Dorian Minarolli and Bernd Freisleben. Utility-driven Allocation of Multiple Types of Resources to Virtual Machines in Clouds. In *Proceedings of 13th IEEE Conference on Commerce and Enterprise Computing, CEC 2011*, pp. 137–144, IEEE press, 2011. (*Best Paper Award*)

3. Dorian Minarolli and Bernd Freisleben. Virtual Machine Resource Allocation in Cloud Computing via Multi-Agent Fuzzy Control. In *Proceedings*

of 3rd International Conference on Cloud and Green Computing, CGC 2013, pp. 188–194, IEEE press, 2013.

4. Dorian Minarolli and Bernd Freisleben. Distributed Resource Allocation to Virtual Machines via Artificial Neural Networks. In *Proceedings of 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2014*, pp. 490–499, IEEE press, 2014.

5. Dorian Minarolli and Bernd Freisleben. Cross-Correlation Prediction of Resource Demand for Virtual Machine Resource Allocation in Clouds. In *Proceedings of the 6th International Conference on Computational Intelligence, Communication Systems and Networks, CICSYN 2014*, pp. 119–124 , IEEE press, 2014.

1.5 Organization of this Dissertation

This dissertation is organized as follows:

Chapter 2 introduces topics that lay out the fundamentals for this work. This includes cloud computing, virtualization, supervised machine learning, feedback control systems, fuzzy-control systems and utility function optimization. This chapter closes with an overview of related work in the area of VM resource allocation in cloud computing.

Chapter 3 introduces the proposed resource allocation approach at the global data center level, based on VM live migration and utility function optimization. In this chapter the design, implementation and experimental evaluation of this approach are presented.

Chapter 4 presents the proposed resource allocation approach of multiple type resources at local the physical machine level, based on control-theory and utility optimization. Its design, implementation and experimental evaluation are presented.

Chapter 5 presents the proposed resource allocation approach at the local physical machine level, based on multi-agent fuzzy control, which allocates resources based on utility optimization using a hill-climbing search heuristic. Its design, implementation and experimental evaluation are introduced.

Chapter 6 presents the proposed resource allocation approach at the local physical machine level, optimizing a utility function using an ANN based performance and power modelling. A distributed resource management design including its implementation, simulated and realistic experimental set-up and evaluation are presented.

Chapter 7 presents the proposed proactive resource allocation approach at the global data center level called Automatic Proactive Resource Allocation (APRA). It is based on VM resource usage prediction, using support vector machines for time series forecasting. Its design, implementation, experimental set-up and evaluation are introduced.

Finally, Chapter 8 concludes the dissertation and discusses future work.

2
Fundamentals

2.1 Introduction

This chapter is divided into two parts and presents the background and definitions on which the developed VM resource allocation approaches are based. In the first part, the concepts of cloud computing, virtualization, supervised machine learning, feed-back control systems, fuzzy-control systems and utility function optimization are explained. For each concept, its main features, drawbacks, advantages and representative technological solutions are described.

In the second part of this chapter, work related to VM resource allocation approaches is presented. The related work is grouped in different sections, based on the main VM resource allocation mechanism used, such as VM live migration, vertical scaling VM resource allocation, horizontal scaling VM resource allocation, proactive VM resource allocation.

2.2 Definitions

This section gives the fundamental definitions and technological solutions this work is based upon. It starts in Subsection 2.2.1 with the concept of cloud computing. Then, it continues in Subsection 2.2.2 with virtualization technology, the underlying technology of cloud computing. In Subsection 2.2.3, two supervised machine learning techniques are introduced, namely Artificial Neural Networks and Support Vector Machines. In Subsection 2.2.4, feed-back control systems, their properties and how they can be designed to control computing systems, are presented. In Subsection 2.2.5, fuzzy-control systems together with their main design properties are introduced. Subsection 2.2.6 describes the concept of utility function optimization together with deterministic and heuristic algorithms for achieving it.

2.2.1 Cloud Computing

2.2.1.1 Historical View of Cloud Computing

Cloud computing, as a new computing model that offers several advantages compared to existing distributed models, has gained a wide spread use in recent years. Its advantages are: flexible resource management, reduced costs and economies of scale, easy fault management etc. Although a relatively recent computing paradigm, the idea behind cloud computing (in the form of utility computing) dates back to 1961 when John McCarthy, a computer scientist who influenced the early development of Artificial Intelligence, stated: *"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"*. In essence, offering computing resources as a service to consumers in the same way as the electricity company offers electricity, is the idea cloud computing implements. In this utility computing model, the consumer gets a computing service, on demand, on a pay-as-you-go fashion. This means that the consumer gets only the required demand of computing service and pays only for the service he has used.

It took a long time down the road of computing history for the vision of cloud to become a reality. In 1970s, through a powerful computer, mainframes offered shared computing resources in a centralized way, where users could connect through their dump terminals. Then came the area of small PC computers connected to local area networks, with the service offered through dedicated servers, isolated from each other. Later, this network based computing model was grown by the advent of the Internet, which made it possible to connect different networks into a world wide global network, where any user and in any part of the network could access the service. The Internet and the development of higher bandwidth network technologies allowed some early forms of utility computing such as email and search engine services.

Small and big Internet service providers started to offer hosting services for websites, removing the burden for maintaining the site for a leasing fee. Some early companies offering these remote services through Internet were Salesforce.com [101] and Amazon.com [3]. The term cloud computing emerged around 2006 through the advent of commercial offerings such as Amazons Elastic Compute Cloud (EC2) [4] and later Google App Engine [41]. As this new model of computing advanced, new forms of applications such as web2.0 based social media (e.g Facebook), service oriented workflows and new programming paradigms such as MapReduce [25] emerged. However, with the spread of the cloud new challenges seemed to arise. Among the most important issues that could be mentioned are: more complex system to be managed, problems of consumer lock-ins, security and privacy problems. Since cloud computing is very dynamic and rapidly changing, it is difficult to predict what its future will be. Certainly, in order to be a sustainable and successful technology, it will need the convergence of other fields such as service-oriented computing, autonomic computing, virtualization etc.

2.2.1.2 Definition and Advantages of Cloud Computing

The main question sailing in the vast discourse and marketing hype associated with the term cloud computing, is what exactly is cloud computing? There are many definitions of the cloud computing concept and no unique standard definition, which has occasionally created confusion within the IT industry. For example, Gartner [38], a research and advisory company in US has given the following definition:

"A style of computing in which scalable and elastic IT-enabled capabilities are delivered as a service to external customers using Internet technologies".

This definition emphasizes the scalable and elastic nature of cloud computing. Elastic, in this context, means that computing resources offered to consumers can shrink and expand according to the workload demand. Forrester Research [37] gives the following definition:

"A standardized IT capability (services, software, or infrastructure) delivered via Internet technologies in a pay-per-use, self-service way."

One of the definitions that is taking more and more acceptance from IT industry is that of National Institute of Standards and Technology (NIST) [86]:

"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".

Yet, another definition that captures all characteristic features of cloud computing is given in the following:

"Clouds are a large pool of easily usable and accessible virtualized resource (such as hardware, development platforms and/or services). These resources can be dynamically re-configured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs". [110]

Some of the unique features and advantages of cloud computing over existing computing models can be listed as follows:

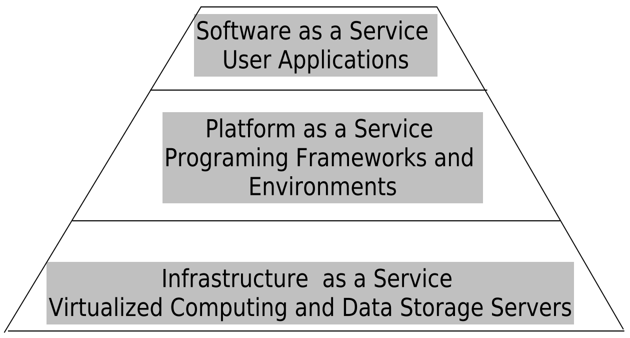

Software as a Service
User Applications
Platform as a Service
Programing Frameworks and Environments
Infrastructure as a Service
Virtualized Computing and Data Storage Servers

Figure 2.1: Abstraction layers of cloud computing

- Up-front reduction of infrastructure investment costs for new companies. To fulfill their early stage needs, companies do not need to invest on hardware resources but can lease virtualized resources from cloud providers.
- The availability of virtually infinite resources that can be provided on-demand in an elastic way. What this means is that resources can be requested and released as needed according to the workload variability. This eliminates wasted resources associated with over-provisioning and performance violations associated with under-provisioning as a result flash crowds.
- Pay-as-you-go billing model where cloud consumers are charged only for the cloud resources used in a fine-grained way, resulting in a more economically efficient way of using resources.
- Shifted operational costs, risks and infrastructure management complexities from cloud consumer to the cloud provider. This makes it possible for companies to focus on innovative Internet services and business models, rather than on infrastructures supporting them.
- Reduced power and operating costs for the cloud provider because of more efficient utilization of infrastructure resources as a result of shared multiplexing of virtual resources of different cloud consumers. This also results in lower cost offerings for could consumers.

2.2.1.3 Architecture of Cloud Computing

According to the layer of abstraction that computing service is offered, three types of delivery model in cloud computing can be identified: a) Software as a Service (SaaS), b) Platform as a Service (PaaS) and, c) Infrastructure as a Service (IaaS). This layered architecture is shown schematically in Figure 2.1 where each layer uses the service of the layer below.

- *Software as a Service* offers application software to the consumers over the Internet. The application software runs at the cloud provider severs and is accessed through some user interface that is usually a web browser. The consumer does not have any control on the software running platform or the amount of resources allocated to it. It only has control to just a few software customizations. Such a model is exemplified by providers such as Salesforce.com [101].

- *Platform as a Service* offers a platform and programming environment such as API framework and libraries, to develop application software. In this case, the platform together with the application run on cloud provider servers. The consumer does not have any control on the platform itself, but only on the application it is developing. Such a model is exemplified by providers such as Google App Engine [41] or Microsoft Windows Azure [79].
- *Infrastructure as a Service* offers raw computing resources such as compute and storage servers, situated in a cloud provider's data center, in the form of VMs over the Internet. The consumers can request these resources on demand by renting them for a limited period of time. In this model, the consumer has control over the platform and customization of the operating system, but does not have control over the physical hardware and amount of resources allocated to its VMs. A well-known example of an IaaS provider is Amazon EC2 [4].

In IaaS clouds, the following entities can be identified, as shown in Figure 2.2.

- *Cloud provider*: the entity that offers compute resources in the form of VMs, running on its own data center physical machines.
- *Cloud consumer*: the entity that leases resources in the form of VMs to host his own application services (e.g. web site) and pays the cloud provider for the usage of VMs or the performance levels it receives.
- *End user*: the entity that usually interacts through a web interface to application services owned by cloud consumer.
- *Service Level Agreement (SLA) contract*: is a contract between cloud provider and cloud consumer that states the levels of resources or performance metrics that should be guaranteed, the amount of charges for guaranteeing them and any monetary penalties for violating them.

Most of the current IaaS implementations apply an SLA contract that guarantees and charges only for consumed VM resources. The other form of SLA contract is the one that charges only for the performance levels achieved. In this dissertation both forms of contracts are used. However, since it offers the best solution for both cloud consumer and cloud provider [85], in this work mostly a contract that guarantees and charges only for application performance levels is used.

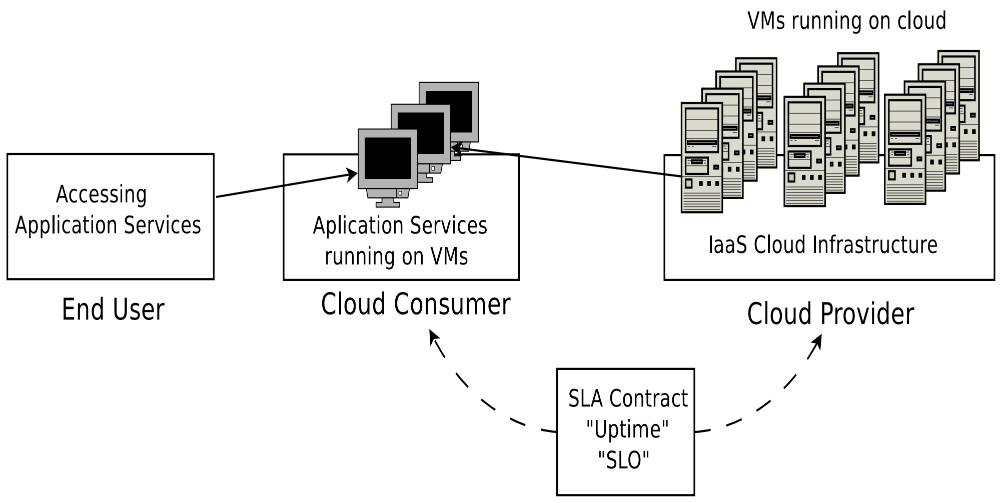

VMs running on cloud
Accessing Application Services
End User
Aplication Services running on VMs
Cloud Consumer
IaaS Cloud Infrastructure
Cloud Provider
SLA Contract "Uptime" "SLO"

Figure 2.2: IaaS cloud

IaaS cloud computing systems can be divided according to the deployment model supported. In the *public cloud* model, virtualized resources are offered on demand, in a pay-as-you-go fashion, to the general public consumers. In this case, the cloud provider is in charge of infrastructure operating costs while the cloud consumer pays only for the resources used. In the *private cloud* model, the virtualized resources are run on a private organization infrastructure and used for its own needs for better flexibility, reliability and resource usage efficiency. The *hybrid cloud* model combines the features of both the public and private cloud. In this model, an organization runs its own private cloud infrastructure but it is also connected and can lease additional resources from a public cloud, when additional capacity is needed to fulfill increased workload. This offers control over critical data and services without the need for capital investment on additional hardware resources.

2.2.2 Virtualization

One of the key enabling ingredients of the cloud computing model is the virtualization technology. This technology allows virtualizing and sharing computing resources in the form of VMs between different cloud consumer applications. Moreover, it provides the key advantageous features of cloud computing over other computing models such as elastic resource provisioning, efficient resources usage, performance and security isolation etc.

2.2.2.1 Virtualization Technologies

Virtualization is a general concept and can be applied in different software stack abstraction levels. For instance, the operating system offers a virtual machine in the form of API system calls to application programs. It virtualizes resources

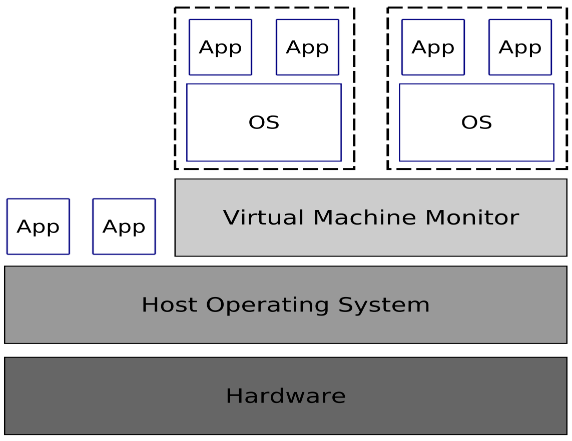


Figure 2.3: HostOS-based virtualization

such as CPU or memory and gives each process the illusion that it has the whole CPU or memory for itself. Another kind of virtualization is the so-called operating system level virtualization. This virtualization can encapsulate an operating environment in a container with its own root account and it can run several of them on top of the operating system kernel. Some examples of OS level virtualization are FreeBSDs chroot jails [62], Linux VServer [95] and OpenVZ [23].

Another important type of virtualization is the so-called machine virtualization. In this case, the whole operating system and its applications are encapsulated in a virtual machine that offers the illusion of a real physical machine. Some examples of this kind of virtualization are Kernel Virtual Machine (KVM) [65], Microsoft Virtual Server [77], Windows Virtual PC [78], User Mode Linux (UML) [27], VMware Workstation [114], VMware ESX server [113] and Xen Hypervisor [8]. Machine virtualization can be further divided into different kinds depending on how the virtualization is achieved:

- *HostOS-based*: Here, different VMs containing the whole operating system run on top of a virtualization layer called Virtual Machine Monitor (VMM) or Hypervisor, which (as shown in Figure 2.3) itself runs on top of an existing operating system called Host OS. A representative example of this approach is VMWare Workstation [114]. The VMM multiplexes VMs and controls the access of VMs to resources by relying on the Host OS, which provides some functionalities such as I/O device handling. One of the advantageous features of HostOS-based virtualization is an easy installation of VMM as a normal application and support for diverse types of I/O devices through the use of existing HostOS device drivers. The main disadvantage is the high performance overhead of I/O device data transfers since all I/O operations have to go through the HostOS layer.

- *Kernel-based*: In this approach, the operating system takes the role of

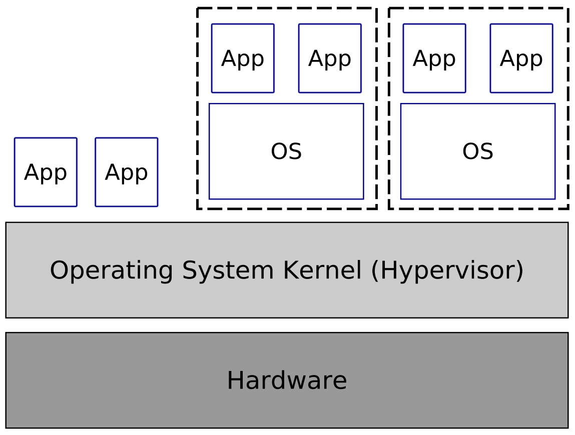


Figure 2.4: Kernel-based virtualization

Hypervisor to multiplex VMs that run as user space processes. This architecture is shown in Figure 2.4. Some of representative examples of this virtualization technology are UML [27] and KVM [65].

- *Hypervisor-based*: This is one of the main approaches of machine virtualization, where several VMs run on top of a software layer called a Virtual Machine Monitor (VMM) or Hypervisor, which itself runs directly on top of the hardware as shown in Figure 2.5. The VMM multiplexes VMs on physical resources by offering them a virtual hardware similar to the physical one. Some representative examples of this kind of virtualization are VMWare ESX server [113] and Xen [8]. These are called bare-metal Hypervisors since they run directly on hardware. One of the advantages of this approach is that it provides strong security and performance isolation between VMs and better reliability as a result of a small Hypervisor, which is only responsible for the main resource management tasks.

2.2.2.2 Hypervisor-based Virtualization

Since Hypervisor-based virtualization technology is the most used virtualization approach in IaaS clouds and the one this work is based upon, a more detailed analysis of this approach is presented starting with its initial historical roots.

2.2.2.3 Brief History of Virtual Machines Monitors

Virtual machine technology is not a new approach. It has been developed around the end of the 1960s by IBM, with the VM/370 Virtual Machine Monitor that permitted to run several VM operating systems on top of the physical

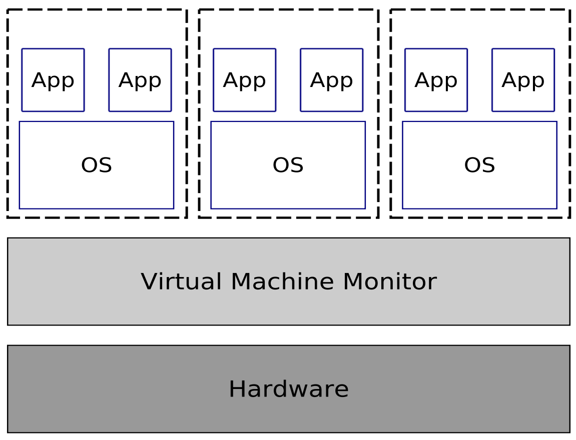


Figure 2.5: Hypervisor-based virtualization

hardware. The main motivation of using virtualization those days was efficient use of expensive mainframe computers where each user could get a portion of the physical hardware through its own VM. With the advent of the PC and low cost computers, where each user could afford to buy one, the need for virtualization-based sharing of expensive computers decreased. As a result, virtualization technology disappeared until the beginning of 1990s where researchers at Standford University started to show interest on virtualization as a solution to the problem of running existing operating systems on complex parallel machines. This was also the starting motivation when the VMware [112] company started to develop virtualization solutions for everyday PC computers. Since then, the industry and academia were showing more and more interest in virtualization as a solution to new emerging problems. Some of these problems are as the following: First, the proliferation of server machines often underutilized results in high power and management costs. Here, the virtualization offers the possibility to consolidate multiple servers in the form of VMs on fewer physical machines, lowering power and management costs. Second, operating systems are becoming more and more complex resulting in reliability and security problems for applications running on them. Here, virtualization offers the possibility to isolate VMs of different users where the crash or security break of one OS does not influence the other VMs or the entire system.

2.2.2.4 Approaches to Hypervisor-based Virtualization

The basic idea of Hypervisor-based Virtualization, at least with respect to CPU virtualization, is the following. The VM holding the operating system and application software is run in unprivileged CPU mode, while the Hypervisor is run in privileged CPU mode. Normally, the VM runs unprivileged instructions directly on the CPU, creating the illusion that the whole CPU is belonging to the VM. When the VM tries to run a privileged instruction, it is trapped at the Hypervisor that takes control and emulates the functionality of this instruction.

In this way, the Hypervisor can control hardware resources and isolate VMs from each other. This requires a virtualizable CPU, meaning it should support two execution modes and provide the possibility to trap to the Hypervisor. Popek and Goldberg [94] more formally defined when a CPU architecture is virtualizable. First, they divided the instruction set into two main groups:

- *Privileged instructions* are instructions that can be executed only in privileged CPU mode.
- *Sensitive instructions* are instructions that can change resource configuration information of VMs or that their behavior depends on resource configuration information.

They concluded with the condition that the CPU's sensitive instructions should be a subset of the privileged instructions in order for the CPU to be virtualizable. In other words, all sensitive instructions executed by some VM that try to change the resource state of another VM should be trapped to the Hypervisor. Only in this way the Hypervisor can have and retain control over the physical machine.

Unfortunately, recent CPU architectures such as Intel x86, do not obey to this condition and therefore are not virtualizable. This is because they have a set of instructions, especially those related to memory configuration, that do not trap if they run in unprivileged mode. For this reason, several solutions have been developed to enable virtualization technology on Intel x86 CPUs:

- *Full virtualization*: a technique developed by VMware, based on dynamic binary translation in which all virtualizable instructions are run directly on the CPU while non-virtualizable ones are translated on-the-fly to a new sequence of instructions that can be executed directly on the CPU and provide the required functionality. To lower the overhead of the translation, the sequence of instructions that are translated for the first time is cached and can be reused when it is required for the next translation. As a typical example of this approach serve VMware ESX and ESXi servers [113]. The advantage of this technique is that the operating system is not aware that it is running on a virtualized environment and does not need to be modified. This is because the translation is applied on the binary code at run time. Also, the translation overhead is not big as only a small percentage of instructions need to be translated while cached translations speed up the process.
- *Paravirtualization* is another important hypervisor based approach to support virtualization on non-virtualizable CPUs. In this approach, operating system kernel source code is modified by replacing non-virtualizable instructions with the so-called hyper-calls, made to the hypervisor. Thus, the operating system communicates directly with the hypervisor through the hyper-calls interface provided by it. The main representative example

of the paravirtualization approach is the Xen open source project [123] with its Xen hypervisor. The approach offers low virtualization overhead through efficient hyper-calls, eliminating overheads associated with trapping privileged instructions running in unprivileged mode. The main drawback, however, is that the operating system is aware that is virtualized and needs to be modified to be ported to the Hypervisor. This means that legacy operating systems cannot be run on virtualized hardware.

- *Hardware-assisted virtualization* approach is based on new generation CPUs such as Intel with its Virtualization Technology (VT-x) [55] and AMD with its AMD-V [1] that offer support for virtualization. These new CPU technologies offer a new level of privileged mode, different from the normal privileged and non-privileged mode, where the hypervisor can run. With these technologies, non-virtualizable instructions running outside this new privileged mode will trap to the hypervisor, which will emulate their functionality. Operating systems can be ported unmodified to the virtual environment although carrying with some virtualization overhead because of the trap to the hypervisor. Several virtualization solutions support hardware-assisted virtualization such as Xen, VMWare ESX and KVM.

2.2.2.5 Memory and Device I/O Virtualization

The main idea with respect to memory virtualization for different solutions is the following. In the same way that the VM operating system virtualizes the memory to its own processes by keeping a page table of memory mapping between virtual and physical addresses, the Hypervisor virtualizes machine physical memory by keeping the mapping between VM physical addresses and actual machine addresses allocated to the VM. To speed-up the address translation process, the Hypervisor keeps a shadow page table of direct mapping between VM process virtual addresses and the actual machine addresses, and uses Memory Management Unit (MMU) to make the translation at run time. If the VM operating system changes its own page table, the Hypervisor notices this and makes the corresponding change in the shadow page table in order to keep control of physical memory allocated to VMs.

With respect to device I/O virtualization, what generally happens in most of the virtualization solutions is that the Hypervisor provides virtual device interfaces to the VMs to submit their I/O requests. The Hypervisor then multiplexes different I/O requests coming from multiple VMs to the real I/O device. This implies that the Hypervisor must have somehow access to the real I/O device. One possible way to achieve this is employed by VMWare ESX server. In this case, the Hypervisor contains a small number of device drivers that permits it to directly access the real I/O devices. The drawback here is that it cannot support the variety of I/O devices existing today, especially on modern PC computers, since the Hypervisor code should be kept to minimum to have a reliable system. Another possibility adopted by Xen and VMware Workstation is for the Hypervisor to use device drivers, already existing on a host operating system, in order to access the real I/O devices. Since the host operating system is usually Windows or Linux that already have device drivers, the support for a variety of I/O devices becomes possible with the cost of increased I/O overhead, since another layer of indirection is introduced.

2.2.2.6 Virtual Machine Live Migration

One of the most important mechanisms used for resource allocation to VMs in IaaS clouds is VM live migration. This is the movement of a VM from one physical machine to another with negligible runtime interruption of running applications. Live migration of VMs allows flexible management of IaaS clouds firstly, by providing load-balancing capabilities by moving VMs from overloaded physical machines to under loaded ones. Secondly, it makes it possible to dynamically consolidate VMs to fewer physical machines, thus reducing power and operating costs. Since this is an important resource allocation mechanism also used in this dissertation, a brief overview of the VM live migration process is given. Since there are several techniques for achieving live migration, the explanation is limited to the so-called pre-copy technique [22] of live migration, widely used in practice through the Xen Hypervisor.

Two performance indicators of a live migration technique are *downtime* and *total migration time*. *Downtime* is the time during which the operating system and all applications running on it are suspended for the final transfer to destination physical machine. *Total migration time* is the time from the start of migration process until the VM is completely transferred. Pre-copy live migration technique has shown downtimes in the orders of milliseconds that cannot be noticed by users of applications services running on the VM. The main idea of pre-copy VM live migration technique is to transfer the memory image of a VM from source to destination host through a number of iterations, with a subset of memory pages transferred in each iteration. More concretely, the technique is implemented as a number of stages as follows [22]:

- *Reservation*: When a request for migration from one host to another is issued, a check whether free resources are available on the destination host is performed and if so, a reservation for a VM container is made.

- *Iterative pre-copy:* In the first iteration, all memory pages of the VM are transferred from source host to destination. In the next iterations, in order to keep memory consistency, only the pages that are changed during the previous transfer are transferred again. To keep track of changed pages, the *shadow page tables* are used in the Hypervisor, where all page address entries are marked as read only. When a page is written, a trap is issued to the Hypervisor setting a bit to a dirty pages bitmap indicating which page is updated. In each new iteration, the shadow page tables are recreated and the dirty pages bitmap is cleared.

- *Stop and copy:* In this stage, the VM operating system is suspended and the CPU together with remaining changed pages are transferred to the destination. At the end of this stage, there are two copies of the VM memory images, one at the source and the other at the destination host, meaning that if a failure happens the original copy can be reused.

- *Commitment:* The destination host notifies the source host that it has received the complete VM memory image. After that, the source host replies back and finally the VM can be discarded on the source host.

- *Activation:* The VM on the destination host is activated and some post migration administrative tasks such as the notification that the VM IP address has moved to a new host, are run.

2.2.3 Supervised Machine Learning

Machine learning is a sub-field of artificial intelligence that studies how intelligent agents can improve their performance by learning from past experience. This is needed since agents cannot be programmed at design time with algorithms that can predict all dynamic and changing situations and act accordantly. There are different forms of learning such as *unsupervised learning*, *supervised learning* and *reinforcement learning*. Since in this work we investigate and apply supervised machine learning for resource allocation to VMs, we review this kind of learning technique together with two widely used representatives, namely artificial neural networks and support vector machines.

The goal of supervised machine learning is the following. Given a set of samples in the form of input-output (x_i, y_i), where x_i is the input value (it can also be a vector of several values) and y_i is the output taken from an unknown function $f(x)$, the goal of supervised learning algorithm is to find a function $f'(x)$ that approximates the unknown function. The set of all samples is called the training set and the process of finding f' is called learning or training, while f' itself is called hypothesis. After the learning phase, given a new input value x_j that is taken outside of the training set, the approximate function f' should predict a value $f'(x_j)$ that is with sufficient accuracy near to the real value $f(x_j)$. When the learned function f' predicts new values with high accuracy, we say that it generalizes well. Generally, when the output of the predicted function takes finite discrete values we have a classification-learning problem. On the other hand, when the output takes any continues number we have a regression learning problem. In this dissertation, the focus is on regression learning of a function.

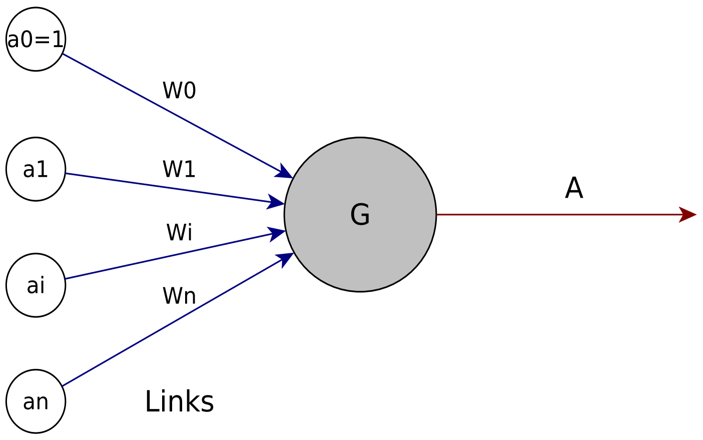


Figure 2.6: Neuron

2.2.3.1 Artificial Neural Networks

In this section, we first introduce the structure of artificial neural networks and then move on describing the learning algorithms used to train the artificial neural network. The basic building unit of an artificial neural network is the neuron given in Figure 2.6. It is sometimes called a perceptron and takes as input a number of links while it has one output link. Each input value a_i, before entering the neuron unit through its corresponding link, is multiplied by a weight w_i. There exists an input value a_0 that has the value of 1. At the entrance of the neuron unit, the sum of all input values multiplied by their weights is taken as below:

$$y = \sum_{i=1}^{n} a_i * w_i \tag{2.1}$$

Then, an activation function G is applied to get the output value of the neuron.

$$A = G(y) = G(\sum_{i=1}^{n} a_i * w_i) \tag{2.2}$$

The activation function can be of different types, but normally is a non-linear function such as the sigmoid function given in Figure 2.7. The non-linear activation functions make it possible for the artificial neural network to learn and approximate general non-linear functions.

Next, it is explained how different neurons can be connected to form a complete structure of the artificial neural network. One type of artificial neural network, applied in the work presented in this dissertation, is called feed-forward neural network. This is a structure composed of several layers of neurons where the

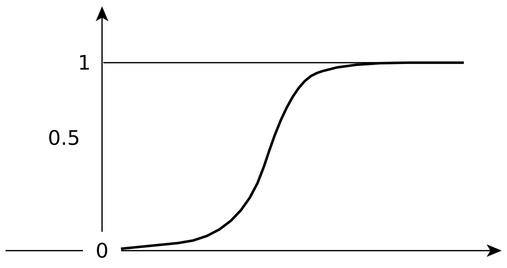


Figure 2.7: Sigmoid activation function

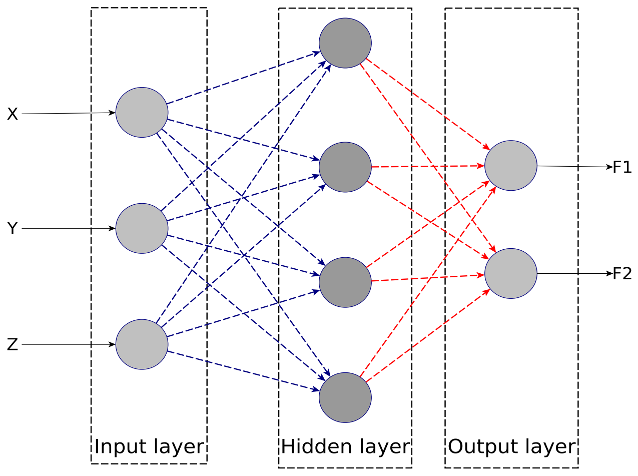


Figure 2.8: Feed-forward artificial neural network

neuron outputs of one layer go through links to the neuron inputs of the next layer. The structure of such an artificial neural network with three layers, an input layer, one hidden layer and an output layer is shown in Figure 2.8. This neural network structure has three inputs x, y, z connected to the input layer composed of three neurons and two outputs $F1$, $F2$ that are provided by the output layer composed of two neurons. The hidden layer is composed of four neurons and takes inputs from the input layer and sends its outputs to the output layer. The distinguished feature of feed-forward neural network is that the flow of values passing through the links of the network goes only in one direction from inputs to the outputs of the network. The artificial neural network shown in the Figure 2.8 can be used to learn two functions $F_1(x, y, z)$ and $F_2(x, y, z)$ both depending on three input variables x, y and z. To learn a function, the artificial neural network needs to undergo a training phase. This is provided with a number of samples of the form (x_i, y_i) where x_i is the input vector and y_i is the output vector of the unknown function to be learned. The goal of the training phase is to find the values of all the neural network weights in order to approximate, as accurately as possible, the unknown function. Since one of the well know learning algorithms for artificial neural network training is the back-propagation algorithm [6], it will be provided an explanation of this algorithm in the following.

The main idea of back-propagation learning is to evaluate the error between a desired output value and the output predicted by the artificial neural network. Based on this, the weights are updated in such a way as to minimize this error. Then, the error is re-evaluated and weights are updated again continuing for a number of iterations until we get a very small error. More concretely, let p_i be the predicted output vector of an artificial neural network for sample i from a set of n samples, and y_i be the real output vector for the same sample. The learning algorithm is based on the error function over all samples as defined below:

$$E = \frac{1}{2} \sum_{i=1}^{n} ||p_i - y_i||^2 \tag{2.3}$$

This is a function of neural network weights because of the way the neural network is structured by transforming the inputs to output values through link weights and node activation functions. The goal of the learning algorithm is to minimize the error function by finding the weights of neural network that give minimum value to function E. The basic error function optimization method used in back-propagation is the gradient descent applied in iterative steps. Because of the sigmoid activation function sometimes this optimization can get stuck to local minima, which is a known drawback of artificial neural network learning. Without going into much details, the idea is to calculate in every iteration the error function gradient with respect to each weight and use it to update the weights. We can define the gradient of error function E as below:

$$\bigtriangledown E = (\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ..., \frac{\partial E}{\partial w_n}) \tag{2.4}$$

The gradient $\frac{\partial E}{\partial w_i}$ is the derivative of function E with respect to w_i. Each weight

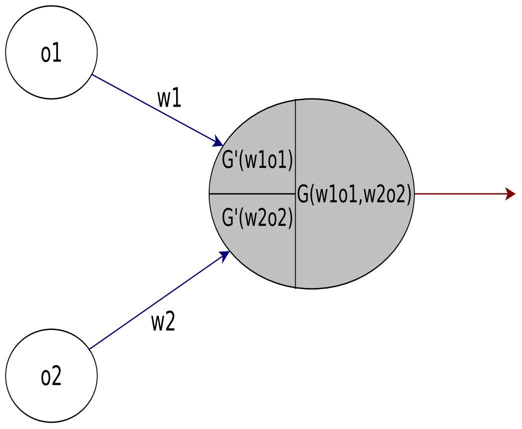


Figure 2.9: Perceptron division

w_i is updated by using the corresponding gradient and a learning coefficient α which determines by what amount the weight is updated. The update rule is given below:

$$w_i = w_i^{old} + \triangle w_i = w_i^{old} + (-\alpha \frac{\partial E}{\partial w_i}) \tag{2.5}$$

The learning algorithm continues for a number of iterations where in the first iteration, the neural network weights are set to random values. The algorithm ends when the error function reaches a small value. Each iteration can be divided into four steps:

- *Feed-forward step*: During this step, the information flows from input neurons to output neurons, through hidden layer. As each sample from the training set is presented to the inputs of the network, the outputs as well as partial derivatives of activation functions with respect to every weight are calculated for each neuron. The last calculation is needed for the calculation of partial derivatives of error function with respect to the weights in the next step. Each neuron can be imagined as divided in two parts. In the left part the partial derivatives of activation function at the input sample considered are saved and in the right, the activation function output for the same sample. More concretely, this division is shown in Figure 2.9 where o_1, o_2 are the activation function outputs of neurons that connect through input links to the neuron in question. $G(w_1o_1, w_2o_2)$ is the output of activation function G of the neuron in question for the inputs w_1o_1 and w_2o_2. $G'(w_1o_1)$ and $G'(w_2o_2)$ are partial derivatives of G with respect to the inputs w_1o_1 and w_2o_2. From $G'(w_io_i)$ we can easily derive the partial derivative with respect to weight w_i as $G'(w_i) = o_iG'(w_io_i)$ considering o_i as a constant.

- *Back-propagation to the output layer:* The general idea of this is starting from the error function E and going backward cumulatively calculating partial derivatives of error function based on saved partial derivatives

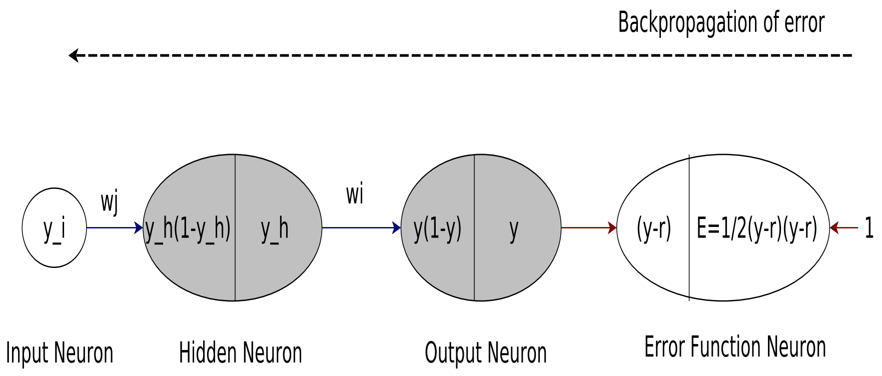


Figure 2.10: Back-propagation steps

(calculated in previous step) on each neuron. In Figure 2.10, it is shown how this step proceeds for one path of the neural network, connecting one input neuron to output neuron. This step requires the addition of another neuron called error function neuron that calculates the error function $E = 1/2(yr)(yr)$ and its derivative $E' = yr$, where y is the predicted output from the output neuron and r is the real output of one sample. The calculated derivative of error function with respect to weight w_i is given as:

$$\frac{\partial E}{\partial(w_i)} = \frac{\partial E}{\partial(w_i y_h)} * y_h \tag{2.6}$$

On the other hand, $\frac{\partial E}{\partial(w_i y_h)}$ is calculated based on multiplication of derivatives saved on the neurons as we move backwards as given by the rules of function transformation derivatives:

$$\frac{\partial E}{\partial(w_i y_h)} = y(1-y)(y-r) \tag{2.7}$$

Finally, we have the partial derivative or gradient with respect to w_i as follows:

$$\frac{\partial E}{\partial(w_i)} = \frac{\partial E}{\partial(w_i y_h)} * y_h = y(1-y)(y-r)y_h \tag{2.8}$$

- *Back-propagation to the hidden layer:* In this step, the partial derivative of error function $\frac{\partial E}{\partial(w_j)}$ with respect to w_j is calculated as follows:

$$\frac{\partial E}{\partial(w_j)} = \frac{\partial E}{\partial(w_j y_i)} * y_i \tag{2.9}$$

On the other hand, the $\frac{\partial E}{\partial(w_j y_i)}$ can be calculated by applying cumulatively the derivative rules backward as the following:

$$\frac{\partial E}{\partial(w_j y_i)} = y_h(1-y_h)w_i(\frac{\partial E}{\partial(w_i y_h)}) = y_h(1-y_h)w_i(y(1-y)(y-r)) \tag{2.10}$$

Finally, we can calculate $\frac{\partial E}{\partial(w_j)}$ as follows:

$$\frac{\partial E}{\partial(w_j)} = \frac{\partial E}{\partial(w_j y_i)} * y_i = y_h(1-y_h)w_i(y(1-y)(y-r))y_i \tag{2.11}$$

- *Weights update step:* In this step after all partial derivatives of error function with respect to every weight are calculated by the back-propagation steps then all the weights are updated at the same time by the update rule given in Equation 2.5.

The above steps are explained for one sample. In order to take into account all samples from the training set, the above partial derivatives are calculated for every sample separately and the update of each weight w_i is given as the sum of updates for all samples as follows:

$$\triangle w_i = \triangle w_i^1 + \triangle w_i^2 + ... + \triangle w_i^n \tag{2.12}$$

where $\triangle w_i^j$ corresponds to update value of weight i for sample j and $\triangle w_i$ is the final update over all samples in the current iteration. This method of applying the update to all weights after calculating the updates for all samples is called *batch learning*. An alternative method is *online learning* in which the weights are sequentially updated sample by sample. In this form of learning not all training samples are needed at the same time but can be applied when they come. This is suitable for situations when memory capacity is limited to hold all the samples.

One difficulty with artificial neural networks is deciding on their structure and number of neurons in hidden layers. The problem is that setting a higher than needed number of neurons can create a situation known as over-fitting. In this case, the network learns every sample providing a very specific function that will not be able to generalize and predict accurately unseen data. There are no golden rules in setting the number of neurons, but in practice, trying several networks and choosing the best one is the most used approach. More concretely, the training set is divided into a training set and a validation set. Then, different neural network structures with different number of neurons are trained on the training set and evaluated on the validation set. This evaluation estimates the generalization error that is the prediction error on unseen data. The neural network that provides the least generalization error is chosen as the best network structure for the training set and number of inputs in question. In practice, more advanced validation techniques exist to overcome over-fitting such as K-fold cross-validation.

2.2.3.2 Support Vector Machine

Support Vector Machine (SVM) is a machine learning technique based on statistical learning theory, which characterizes properties of learning machines to enable them to generalize well to unseen data [103]. Since in this work this technique is used for time series prediction, we give a brief overview of its main idea. SVM can be used for regression to estimate a function of the following form:

$$f(x) = \langle \upsilon, x \rangle + \beta \tag{2.13}$$

where x is the input vector, β is a threshold constant, υ is the weights vector and $\langle , \rangle$ is the dot product. In the case of a non-linear function it can be represented in the following form:

$$f(x) = \langle \upsilon, \psi(x) \rangle + \beta \tag{2.14}$$

where $\psi()$ is a function that maps x in the input space to a high dimension feature space in order to make $f()$ linear for applying linear regression. The goal of the SVM is to generate a function $f(x)$ that is as flat as possible by minimizing the norm $||\upsilon||^2$ and that has a deviation from the actual value y_i for all training samples at most ε. The constant ε is represented by empirical risk $R_e(f)$. Overall the goal is the minimization of what is called the regularized risk $R_r(f)$ given below:

$$R_r(f) = R_e(f) + \frac{\delta}{2}||\upsilon||^2 \tag{2.15}$$

where δ is called regularization constant and is used to reduce over-fitting. The empirical risk is given as:

$$R_e(f) = \frac{1}{N}\sum_{i=0}^{N-1} C(x_i, y_i, f(x_i)) \tag{2.16}$$

where C is a cost function that determine how to penalize errors, x_i is the sample input, y_i is the actual value of the predicted value $f(x_i)$. One of the most used cost functions is the ε-insensitive loss function given below:

$$C(x_i, y_i, f(x_i)) = \begin{cases} |y_i - f(x_i)| - \varepsilon & if\ |y_i - f(x_i)| \geq \varepsilon \\ 0 & otherwise. \end{cases} \tag{2.17}$$

Minimization of $R_r(f)$ is written in slightly different form below:

$$minimize : c\sum_{i=1}^{n} C(x_i, y_i, f(x_i)) + \frac{1}{2}||\upsilon||^2 \tag{2.18}$$

where the constant c includes the $1/N$ factor. This is an optimization problem and assuming a convex function it can be solved using the Lagrange multipliers method by formulating the dual optimization problem given below:

$$Maximize : -\frac{1}{2}\sum_{i,j=1}^{N}(a_i - \dot{a}_i)(a_j - \dot{a}_j)\langle x_i, x_j \rangle \tag{2.19}$$

$$-\varepsilon\sum_{i=1}^{N}(a_i - \dot{a}_i) + \sum_{i=1}^{N}\gamma_i(a_i - \dot{a}_i) \tag{2.20}$$

$$constrain : \sum_{i=1}^{N}(a_i - \dot{a}_i) = 0 : a_i, \dot{a}_i \in [0, C]. \tag{2.21}$$

There are several quadratic programming optimization algorithms for solving the above equation by finding the parameters a_i. One of the most used algorithms is the improved version of Sequential Minimal Optimization (SMO) [93] also used in this work for time series prediction. By getting the solution for the weights υ based on Karush-Kuhn-Tucker conditions we have:

$$\upsilon = \sum_{i=1}^{N} (a_i - \dot{a}_i) x_i \tag{2.22}$$

and the function $f(x)$ can be given as:

$$f(x) = \sum_{i=1}^{N} (a_i - \dot{a}_i) \langle x_i, x \rangle + \beta \tag{2.23}$$

In building the function $f(x)$, only those data points x_i that have non-zero Lagrange multipliers a_i are taken, which are only a small subset of the whole data set. These data points are called support vectors. As stated earlier, to deal with a non-linear function $f(x)$, the input space x_i is mapped to a feature space $\psi(x_i)$ that is linear by giving the below solution for the weights:

$$\upsilon = \sum_{i=1}^{N} (a_i - \dot{a}_i) \psi(x_i) \tag{2.24}$$

and the transformed function:

$$f(x) = \sum_{i=1}^{N} (a_i - \dot{a}_i) \langle \psi(x_i), \psi(x) \rangle + \beta \tag{2.25}$$

The term $k(x_i, x) = \langle \psi(x_i), \psi(x) \rangle$ is called the kernel function. Thus to estimate the function $f(x)$ it is sufficient to apply directly a kernel function without the need to know the transformation function $\psi(x)$. The kernel function should also satisfy the Mercers conditions. There are several kernels that satisfy it, in particular the polynomial kernel $k(x_i, x) = (\langle x_i, x \rangle + 1)^p$ with exponent $p = 1$.

2.2.4 Feed-back Control Systems

In this section, an overview will be given of feed-back control systems, their basic properties and the control theory that forms the mathematical foundation they are based on. Recently, these systems are investigated for controlling computing systems and managing resources in virtualized data centers and cloud computing infrastructures. They are mostly used to keep a certain quality of service metric to a desired level even in the presence of environmental changes and disturbances, by providing more reliable and robust computing systems. For instance, in a virtualized data center these systems are used to keep to desired levels certain application performance metrics such as throughput, response time, queue lengths, resource utilizations etc. As part of the resource

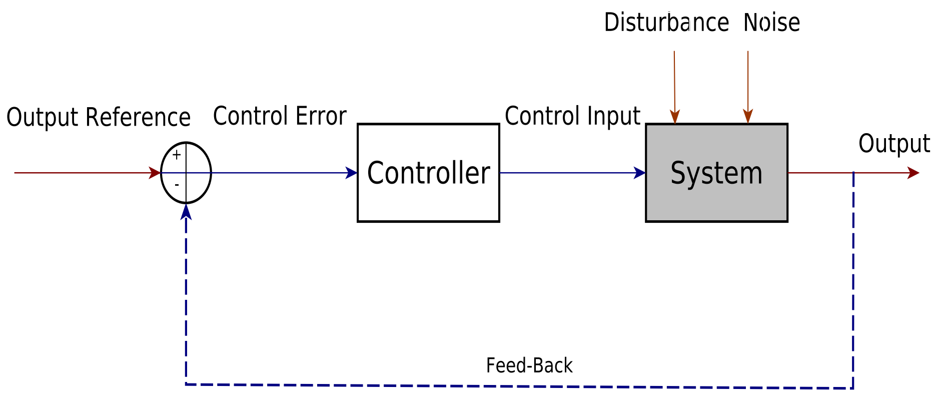


Figure 2.11: Feed-back control loop

allocation technique development, some of the decision making processes are based on feed-back control systems, thus in the following the basic theoretical principles of these systems are explained.

The aim of a feed-back control system is to keep a certain metric value that is the output of a controlled system to a desired reference value. It achieves this by applying a corrective action as input to the controlled system aiming to reduce the deviation of the output from the reference value. Schematically, a typical feed-back control system is composed of the elements shown in Figure 2.11.

- *System*: is the system to be controlled by keeping its output metric to a desired value given by output reference. Output is a metric that can be measured from outside the system while the output reference is given as input to the feed-back control system and can change over time. In computing systems the output metric can have different forms such as application response time, CPU utilization etc.

- *Controller*: is the feed-back controller that acts on the system through a control input in order to influence the output. Here, it is supposed that the system input and output signals are chosen such that the output is affected by the input change. In this way, if the output deviates from the reference value, it is possible to apply the input with the right amount and direction in order to bring the output close to the reference.

- *Feed-back*: is a path that sends back in a loop the measured system output for comparison to the output reference. This is why this feed-back controller system is called closed loop system. In this kind of system, the controller decides on the control input value based on the control error that is given as input to the controller. The control error is calculated as the difference between the output taken from the feed-back path and output reference. Based on the way the control input depends on control error, a variety of controllers such as proportional, integral etc, can be designed.

- *Disturbance and Noise*: Disturbance is an input signal to the system that cannot be controlled and can change the output of the system. One of the goals of the feed-back controller is to keep the system output to the desired value even in the presence of disturbances. In computing systems, disturbance can take various forms, such as administrative tasks that consume resources, workload changes etc. Noise is some random change of the output value coming from internal causes of the system or sensor measurement.

Feed-back control systems for controlling computing systems normally work in discrete time intervals. They generate control input for the next interval based on system output and control error on current and previous time intervals. In discrete time interval notation, any signal such as the output value is given as $y(k)$ or $y(k-1)$ representing the output in the current interval k and previous interval $k-1$. In this section principles of feed-back controllers for single input and output systems are discussed, upon which this work is based. These controllers called Single-Input Single-Output (SISO) systems are the most used in practice especially for their implementation simplicity. In real world environments, there are also systems that have multiple input and output signals called Multiple-Input Multiple-Output (MIMO) systems. These controllers are generally less used since they are more difficult to design and analyse.

There are certain properties, which are desirable for feed-back control systems to have in order to operate in a satisfactory manner. There exists a whole body of control-theory that gives theoretical grounds to help analyse and design controllers that satisfy the required properties. This means that if the theoretical assumptions are realistic enough in practice then there is the certainty that the control system will work reliably according to specifications. The desired properties of feed-back control systems are the following:

- *Stability*: This is one of most important property of feed-back control systems. A system is unstable if for a given input the output oscillates or even worse increases in a boundless way going to infinity. Instability can occur, when as the result of a controller error there can be a more than needed corrective input, leading to a greater error in the opposite direction further leading to an even greater corrective actions and producing oscillatory output. Obviously, stable control systems are desirable and control theory can help to design controllers that have this property.

- *Accuracy*: This property has to do with how closely system output converges to the desired reference value in a steady state. Also, this property is desirable since it decides how well the controller is able to meet the desired quality metrics. To quantify this property, generally the inverse of it is used, which is the inaccuracy represented by the control error in steady state.

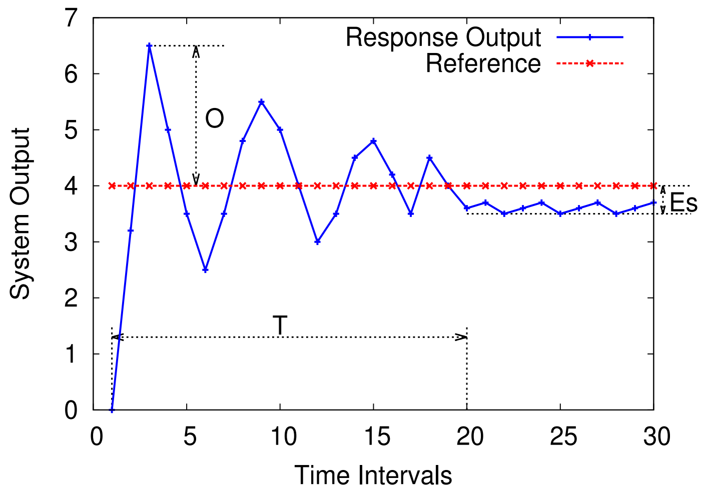


Figure 2.12: System output response

- *Response Time*: This is the time interval from the time the input changes until the output converges to the steady state value. It is desirable for the response time to be short in order for the system to respond quickly to input changes. This makes it possible for the system to adapt and control the output even in the presence of disturbances that quickly change over time.
- *Overshoot*: This is the maximum deviation of the output value from the output value in the steady state.

In Figure 2.12, the output and the properties of a stable feed-back controlled system after an input change is shown. After some time, the system output converges to steady state. For this system, T represent the response time, E_s is the steady state error that represent the inaccuracy of the controller and O represent the overshoot. There are different types of feed-back controllers, but in the following sections an overview of the most important ones is given.

2.2.4.1 Proportional Controller

This is the simplest of all controllers in which the control input generated by the controller depends proportionally to the control error as given below:

$$u(k) = K_p e(k) \tag{2.26}$$

where $e(k) = y(k) - y_{ref}$ is the control error that is the difference between measured output and desired reference. $u(k)$ is the control input and $K_p > 0$ is the controller gain. The intuition behind this controller is that if the value of error is large, then a large input action should be generated in order to bring the output back to the reference value. One problem is that it generates a non-zero steady state error all the time, meaning that the system output cannot converge exactly to the reference value. This phenomenon happens because in order for the system to generate an output, it will usually need a non-zero input, which will only happen when there is a non-zero control error. This means that there should always be a non-zero steady state error in order for the system to work and produce some output. This can be a problem in practice because the system is not able to keep the output to the desired value.

2.2.4.2 Integral Controller

In this type of controller, the control input does no longer depend on control error at current time interval, but on the integral of the error. In the case of a discrete time controller, the control input depends on the accumulative error over time as given below:

$$u(k) = K_i(e(k) + e(k-1) + ... + e(1) + e(0)) = K_i(e(k) + E_a) \tag{2.27}$$

where K_i is called the integral gain and E_a is cumulative error from interval 0 to interval $k-1$. The integral controller eliminates the problem of non-zero steady state error encountered in proportional control. Even if the error $e(k)$ in the current time interval becomes zero, because of a non-zero accumulative error, the control input remains non-zero and therefore the system output remains non-zero and equal to desired reference value. A more widely used controller is what is called Proportional Integral (PI) controller, which includes a proportional term and an integral term as shown below:

$$u(k) = K_p e(k) + K_i E_a \tag{2.28}$$

where K_p and K_i are proportional and integral controller gains respectively.

2.2.4.3 Controller Design

The main goal of the feed-back controller design is to decide on the controller structure and parameters in order to have a stable loop-back system and for the controller properties to have desired values such as satisfactory accuracy, low response time and overshoot. First, in the design process a model of relationship between system output and control input is found in a process known as system identification. In other physical systems, this model can be derived from the first principle approach based on known physical laws that govern the system. In computing systems, it is difficult to derive similar laws as a result the model is mostly derived from empirical experimentation. For PI controllers, usually a linear model of the first order of the form below is identified through a linear regression fitting process:

$$y(k) = ay(k-1) + bu(k-1) \tag{2.29}$$

In this model, the system output $y(k)$ in the current interval depends on system output and input in the previous interval. The coefficients a and b are

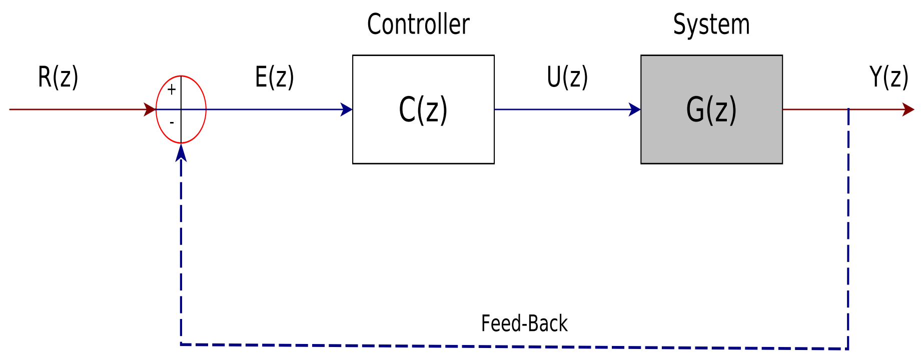


Figure 2.13: Feed-back loop transfer functions

estimated from the regression data fitting process. This model assumes that the relationship between system output and input is linear, which is not the general case for computing systems. However, this relationship is linear only near a small region around the operating point and the controller can be applied for this region. If the operating point changes, for example because of an application workload change, the system model and therefore the controller parameters should be changed to accommodate the new working regime. There are more advance techniques to deal with this issue such as gain scheduling and adaptive control.

One of the mathematical tools that control theory offers to make it easy to analyse and design feed-back controllers is the Z-transform. This is a representation of a signal or system model in frequency domain. Especially the Z-transform of a system model is called the transfer function of the system and represents the relationship between output and input in frequency domain as shown below:

$$G(z) = \frac{Y(z)}{U(z)} \tag{2.30}$$

where $Y(z)$ and $U(z)$ are the z-transforms of output and input signals respectively. With some transformations, for any system model given in the time domain as shown in Equation 2.29 we can find its z-transform $G(z)$ needed for further analyses. For the controller design, what we are most interested is the transfer function of the whole feed-back loop that gives the relationship between output $Y(z)$ and desired output reference $R(z)$:

$$F(z) = \frac{Y(z)}{R(z)} \tag{2.31}$$

The transfer function of the feed-back loop can be derived from transfer functions of its components by applying some function transformation rules. For example, for the feed-back loop shown in Figure 2.13 we can find the following transfer function:

$$F(z) = \frac{C(z)G(z)}{1 + C(z)G(z)} \tag{2.32}$$

where $C(z)$ is the transfer function of the feed-back controller and $G(z)$ is the transfer function of controlled system. For a PI controller with proportional gain K_p and integral gain K_i its transfer function is given below:

$$C(z) = \frac{(K_p + K_i)z - K_p}{z - 1} \tag{2.33}$$

After having a feed-back loop transfer function $F(z)$ it is important to find and analyse its poles, which are the values of variable z where the transfer function denominator becomes zero. The transfer function poles in general can be complex numbers and the position of them in the complex domain determine the behavior of the feed-back loop such as its stability, response time, accuracy etc. For example, a pole that has an absolute value greater than 1 represents an unstable feed-back loop and a pole that has an imaginary part gives oscillatory response. Basically, the controller design is reduced to finding the controller gains K_i and K_p such as the poles of the feed-back loop transfer function have desired properties of a stable, low response time, low overshoot and high accuracy system. Although there are several standard methods for achieving this, they are not going to be discussed further in this work.

2.2.5 Fuzzy Control Systems

Fuzzy control systems are based on a different control philosophy to control and automate a given process. While classical feed-back control systems are based on exact mathematics and laws, fuzzy control systems are based on fuzzy set mathematics. Its development started with the seminal work of Lotfi A. Zadeh [127] around 1960s. In the classical set theory any object can be or not be a member of a set based on a binary membership function. In the fuzzy set theory any object can be partially a member of a set based on a membership function which provides different degrees of certainty for being or not being part of the set.

Most of the real world engineering and computer system control problems are imprecise, highly dynamic and partially defined. This makes applying classic control theory approaches based on exact mathematics not suitable, because of the difficulty to capture and express decision-making solutions with deterministic control algorithms for problems highly partial and uncertain. The motivating factor for using fuzzy control systems is the fact that they handle well uncertainty and as a result offer simple and flexible solutions to highly complex and imprecise dynamic control problems. Fuzzy control systems also make it easier to incorporate human knowledge in the decision-making process in the form of fuzzy rules. Such an incorporation of human intelligence makes them intelligent and allows to reason like humans in solving partially defined and highly complex problems, where an exact control algorithm is not applicable. Another benefit of fuzzy-control systems is that for applying them usually any mathematical or empirical model of the controlled system is not needed.

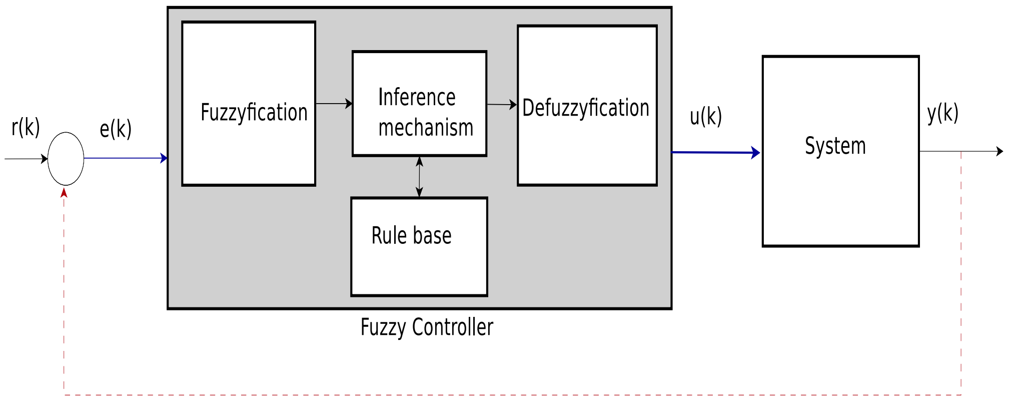


Figure 2.14: Fuzzy control system

This provides the possibility to control very complex non-linear systems that are very common in shared virtualized computing systems.

The general architecture of a fuzzy control system is shown in Figure 2.14. It is composed of four components: fuzzyfication, defuzzyfication, inference mechanism and the rule base. An explanation of each component is given in the following sections.

2.2.5.1 Fuzzyfication

To understand the fuzzyfication process, it is important to understand what it means for a variable x to be a member of a set. In the classical set theory, we can say for a variable x that it can be or not be a member of a set X. In the fuzzy set theory, a variable x can be a member of a set X only with a certain degree as decided by a membership function $\eta : x \to [0, 1]$. For example, a variable x can be member of a of set X with a degree given by $\eta(x) = 0.2$ and not a member of the set with degree 0.8. A variable x can also be member of different fuzzy sets with different degrees given by their respective membership functions. For example $\eta_A(x) = 0$, $\eta_B(x) = 0.75$, $\eta_C(x) = 0.4$ means that x is a member of set A with degree 0, a member of set B with degree 0.75 and a member of set C with degree 0.4. Membership functions can have different forms depending on the problem at hand and for the example above can be represented graphically as shown in Figure 2.15.

The process of converting an exact value of a variable to a fuzzy set with a certain degree through the membership function is called fuzzyfication. For a fuzzy control system the fuzzyfication component is responsible for the fuzzyfication of input variable values to fuzzy sets that internally are represented as linguistic variable values. For example, a particular input variable value can correspond with different degrees to four fuzzy sets represented by four linguistic variable values such as small, medium, large and very large.

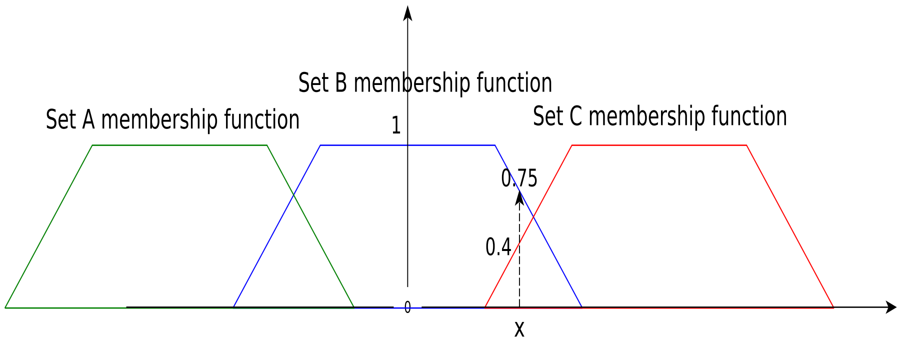


Figure 2.15: Membership functions for three fuzzy sets A, B and C

2.2.5.2 Rule Base

A rule base is a collection of IF-THEN type of rules that incorporate human knowledge on how to take decisions and control a given system. The rules are constructed based on linguistic variables that make it easy for human experts to describe decision-making heuristics in natural language. For the fuzzy control system shown in Figure 2.14 which is used to keep the system output $y(k)$ to the reference value $r(k)$ we can have the following linguistic variables. *"Error"* which is an input to the fuzzy controller and is the given as $e(k) = r(k) - y(k)$. *"ErrorDerivative"* which is also an input to the fuzzy controller, although not shown in figure, is given as $\frac{de(k)}{dk}$ and represents the rate of change of error. *"InputChange"* that is the output of the fuzzy controller or control input of the system and is given as the amount $\Delta u(k)$ to be added to existing input $u(k)$, in order to bring the output $y(k)$ near to the reference value $r(k)$. Each of the above linguistic variables can take linguistic values such as *"NegativeSmall"*, *"PositiveLarge"*, *"PositiveMedium"* etc. For example, *"NegativeSmall"* represents a value that is with a negative sign and has a small absolute value. The linguistic values represent different fuzzy sets and each fuzzy controller input value is assigned to different fuzzy sets with different degrees as decided by corresponding membership functions. One of the IF-THEN rules constructed for controlling the above system is the following:

$$IF \;\; Error \;\; is \;\; PositiveLarge \;\; AND \;\; ErrorDerivative \;\; is \;\; PositiveLarge \\ THEN \;\; InputChange \;\; is \;\; PositiveLarge \tag{2.34}$$

This rule is applicable in a situation when the system output is below reference value, since the error is *"PositiveLarge"* and it is moving rapidly away from it since the error derivative is *"PositiveLarge"*, meaning that the error is increasing. In this situation, to bring the system output to the reference value, the input should be increased with large value as decided by THEN clause. This assumes that the output increases proportionally with the input. Similar rules are created for all situations of the system. The inference mechanism decides which rules are applicable and activates them based on which fuzzy sets the input values correspond to. The result of inference mechanism are input changes to be applied to the system in the form of linguistic values as given by the THEN clause of the relevant rules.

2.2.5.3 Defuzzyfication

Since the inference mechanism can give as output several input change values as the result of several rules being activated, the question that remains is which rule's output should be the final decision of the fuzzy controller. The basic idea is to combine the linguistic value decisions of all the activated rules and generate one decision in the numerical form that can be applied as input to the system. This combination is the responsibility of the defuzzyfication component. There are several methods and formulas to make the above combination but a well known one is the *"centre of gravity"* method. According to this method, for each activated rule the degree that the rule is applicable is found. This is the minimum of the degree that each input value contained in the IF part of a rule belongs to the corresponding fuzzy set. Next, the area under the function $\mu\_i\_c(x)$ is found as follows:

$$S_i = \int \mu\_i\_c(x) \tag{2.35}$$

If we denote with $\mu\_i(x)$ the membership function of the fuzzy set corresponding to the output value of rule i, then $\mu\_i\_c(x)$ is the result of $\mu\_i(x)$ cut from the top at the degree that the rule i is applicable. In Figure 2.16 the darker area shows the function $\mu\_i\_c(x)$ as the result of the cut of $\mu\_i(x)$ at point 0.65. We denote with a_i the value in the centre of the $\mu\_i\_c(x)$ function. The final fuzzy controller output value $\Delta u(k)$ estimated by the *"centre of gravity"* method is given by the following formula:

$$\Delta u(k) = \frac{\sum_{i=1}^{n} a_i S_i}{\sum_{i=1}^{n} S_i} \tag{2.36}$$

where a_i and S_i is found for each activated rule i.

2.2.6 Utility Function Optimization

The key element for the VM resource allocation techniques developed in this dissertation is the utility function. The utility function provides the possibility to express two or more conflicting goals, such as performance and resource costs, in a single expression. Therefore, the VM resource allocation problem of optimizing these conflicting goals is converted into utility function optimization. Since function optimization is important for the developed resource allocation techniques, in this section an overview of several function optimization methods

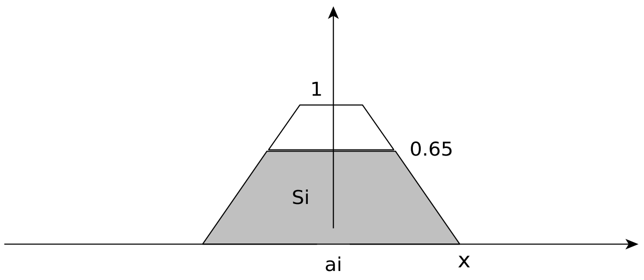


Figure 2.16: Membership functions cut from top at point 0.65

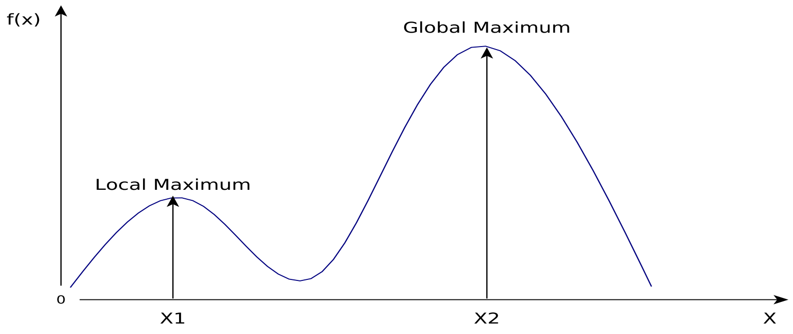


Figure 2.17: Local and global optimum

is given. Then, the focus is shifted on two optimization methods used in this work such as hill-climbing local search and genetic algorithms.

Function optimization problems are those that deal with finding the values of several variables of a function, so that the corresponding function value is optimal. Optimization here can mean maximization if the task is to find the maximum value of a function, or minimization if the task is to find the minimal value of it. The set of variable values that gives an optimal function value is called the optimal solution of the optimization problem. The set of all possible solutions to the optimization problem is called the search space, since the optimization is seen as a search for finding the optimal solution. If variable values should also satisfy some constraints besides being an optimal solution, the optimization problem becomes a constrained optimization problem. Depending on the optimization problem it can happen that the optimization function will have several solutions that have optimal values only for nearby regions. These are called local optima, while only one of them is called global optimum and represents the global optimal value relative to all solutions. To illustrate the idea, Figure 2.17 shows the function $f(x)$ of one variable x. It has one local maximum (optimization here is maximization of function) at point x_1 and one global maximum at point x_2. The existence of local optima makes it difficult for an optimization algorithm to determine if a solution is a global optimum or a local one, especially when the problem has a large search space.

For any computational problem and certainly for optimization problems we can talk about time complexity of an algorithm that solves the problem. The algorithmic time complexity is given by $\Theta(.)$ notation and gives the time needed to find a solution as the function of problem size. In the case of function optimization problems, the problem size is the number of function variables n that determine search space size. This is because increasing the number of variables increases the number of variable value combinations. An optimization algorithm having a time complexity $\Theta(n)$ for a certain problem means that the time needed to find an optimal solution is linear with problem size n. The class of problems that the best known deterministic algorithm has time complexity of $\Theta(n^k)$ are called polynomial time problems or class P problems and their solution time is a polynomial function of n with some constant k. Another class of problems are NP-complete problems for which any deterministic algorithm that finds the optimal solution in polynomial time is not yet known. These are called ”hard” problems since their solution times grow exponentially with the search space size, becoming intractable to solve in reasonable time.

One of the obvious and most naive type of optimization algorithm is what is called the *exhaustive search*. As the name implies, this algorithm works by trying all possible solutions of the search space and keeps the best one as the optimal solution. Although this algorithm is guaranteed to find the global optimum, as the complexity of search space increases the problem solution time of the algorithm becomes very large, making it infeasible for the applicability in practice. There are other methods to solve optimization problems and they can be divided in two categories: a) deterministic algorithms and b) heuristic algorithms.

2.2.6.1 Deterministic Algorithms

The basic idea of deterministic algorithms is that they search the solution space by starting with some deterministic solution and applying a deterministic rule to advance the solution toward the optimal one. Generally, these methods are applied only to optimization problems that are of specific forms and satisfy certain constraints. Most of them require optimization functions to be given in a specific analytical form. They are guaranteed to find an exact optimal solution if there is only one global optimum. However, they can get stuck on local optima if there are several local optimal solutions and given the wrong initial solution. They also provide the same solution (sometimes the wrong one) every time they are run, because of their deterministic nature, making them inflexible. Although there are specific cases when they are able to find the optimal solution efficiently, in general they have exponential time complexity. Some of the deterministic algorithms are listed in the following:

- *Greedy Local Search*: in which the solution found is modified to reach several neighbourhood solutions and the one that gives the greatest improvement is selected as the next solution and the search continues in its locality area. One of the representative local search algorithms that follow the above strategy is the hill-climbing search algorithm.

- *Dynamic Programming*: is an algorithmic design approach that permits to develop optimization algorithms. Algorithms based on this approach solve the optimization problem by dividing it into smaller sub-problems and providing the solution to the bigger problem in terms of a set of smaller problems. It can by applied in recursive form by saving solutions of sub-problems for later reuse, increasing the efficiency of the algorithm.

- *Branch and Bound*: is some kind of *exhaustive search* which tries different solutions but continuously divides the search space into different parts and eliminates those parts that do not contain the optimal solution. It continuously reduces the search space until the optimal solution is found.

- *Divide and Conquer*: is an approach that divides the optimization problem into sub-problems, recursively finds the solution of upper sub-problems through merging the solutions of their sub-problems. This continues until the solution of the main problem has been found.

- *Linear Programming*: is a mathematical optimization method that is applied when the function to be optimized and its constraints are in a linear form. The most used form of linear programming is the simplex algorithm. According to this algorithm, firstly some slack variables are introduced to convert constrained inequalities to equalities. Then, by manipulating the so-called base variables, it goes from one basic feasible solution to another until the optimal solution is found.

- *Gradient Method:* which requires that the function to be optimized is defined and differentiable. According to this method, the gradient $\nabla F(X_0) = (\frac{\partial F}{\partial x}, .. \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z})$ at the current solution point X_0 is calculated. The gradient gives the direction and the amount that each variable should be modified in order to move toward the optimum value. Each variable is modified according to the formula $x_0 = x_0 + \delta \frac{\partial F}{\partial x}$, which is the case for the variable x. After this modification, another solution point is reached and its gradient at this new point is calculated. This process continues until gradient $\nabla F = 0$, which means that the optimal solution is reached.

2.2.6.2 Heuristic Algorithms

Deterministic algorithms generally take exponential time to find an optimal solution for NP-complete optimization problems. To cope with this problem, other kind of methods, called heuristic algorithms, are used. These methods are normally able to find an approximate solution at a lower time complexity than deterministic algorithms. They offer a trade-off between solution quality and solution time and generally find good enough solutions in polynomial time. They are also more general and flexible than deterministic algorithms and some of them have high probability of escaping local optima.

The basic common idea of all heuristic algorithms is that they usually start with an arbitrary solution and based on some rule, which can include randomness, choose the next solution that is supposed to improve the current one by moving towards the optimal solution. The next chosen solution can be in a neighbourhood of the current one or in some other parts of the search space, but not too far from the best known solution. This iterative process generally continuous until a maximum number of iterations or a good enough approximate solution is reached, and no more improvement can be achieved. At the end, the best of all investigated solutions is provided as the optimal one. Some of the well-known heuristic algorithms are listed below:

- *Simulated Annealing*: It is based on a physical process called annealing which is the gradual cooling down of a high temperature melted material in order to reach its minimum energy crystalline state. The essence of the algorithm is that in every iteration step a neighbourhood solution candidate that improves the current one is normally selected. However, time-to-time a perturbation occurs through which a worse solution than the current one is selected with some probability that depend on a parameter T, called the temperature, which gradually decreases with time. This permits to escape local optima especially at the beginning of the search and at the same time to converge the candidate solutions toward the optimal one.
- *Tabu Search*: As with other heuristic algorithms, it considers neighbourhood region for better candidate solution. However, its distinguished feature is that it keeps track of previously visited solutions and prohibits considering them if encountered again in order to escape repeated cycles and regions of search space already visited.
- *Genetic Algorithm*: It is an algorithm that is based on the population evolution theory to explore the search space and find the optimal solution. Since in this dissertation this algorithm is used together with a hill climbing searching method, they are both discussed more thoroughly in the following sections.

2.2.6.3 Hill-Climbing Search Algorithm

The basic idea of the hill-climbing search algorithm is simple. It starts with an initial solution, and in each iteration step it considers all candidate solutions in the neighbourhood of the current solution, picks the best one and compares it with the current solution. If the new solution improves the current one, the new solution becomes the current one and the search process begins again by considering the neighbourhood area. This iterative process continues until we reach the optimum and no improvements can be made to the current solution. This process resembles that of climbing a hill in which we go always uphill (exploring in each step better solutions) until the top of the hill is reached.

Generally, the algorithm is quite efficient even for NP-complete problems but it has the problem that it can easily get stuck to local optima because at these points no better solution can be found. One approach that deals with this problem is a variant of the basic hill-climbing algorithm called random-restart hill climbing. Basically, this variant does a number of hill-climbing searches, each starting from a random initial solution until the optimal solution is found.

2.2.6.4 Genetic Algorithm

Genetic algorithms [51] are another popular and efficient optimization method for NP-complete problems. They are based on natural evolution for exploring a search space to find the solution of an optimization problem. The basic idea of the algorithm is that it starts with a random initial population of individuals spread throughout the search space, where each individual represents a feasible solution of the problem. Each individual, called a chromosome in genetic algorithm terminology, encodes information, as a string of bits or real numbers that represents a feasible solution. Each bit or real number of the chromosome is called a gene. A genetic algorithm requires that each individual be evaluated how good its representative solution is in relation with other population individuals. This goodness value of an individual is called its fitness and is evaluated through a predefined fitness function that depends on the optimization problem. From an initial population based on selection, crossover and mutation, a new population is generated that is supposed to be better than the old one in terms of the fitness function.

The selection process selects from the population two parent individuals with high fitness values in order to insure survivable individuals with better solution quality. There are several selection methods, but a well known one is the roulette wheel method. In this method, each individual is assigned a partition of the roulette proportional to its fitness. This means that during the rotation of the roulette the individual with higher fitness has higher chance of getting selected but not excluding the probability that other lower fitness individuals being selected as well.

Then, with some probability called the crossover probability, a crossover operation is applied where different parts of the parents are combined to create one or two children that are supposed to improve their parents in terms of solution quality. The crossover probability determines if a crossover operation will happen or the children will be just exact copies of their parents. There are several crossover methods that differ on how they combine different parts of the parents to create the children. One of the most used and simplest one is one point crossover. In this method a crossover point selected randomly divides the

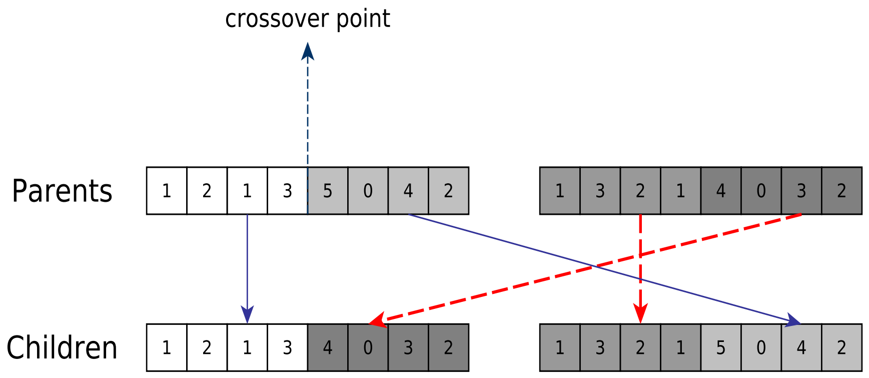


Figure 2.18: One point crossover operation

parent chromosomes in two parts as shown in Figure 2.18. Then, the children chromosome is formed by taking one part form the first parent and the other part from the second parent.

At the end, a mutation operation is applied where with a small probability each gene of the child is altered to a random value. For a binary string chromosome this means a change from bit zero to one or vice-versa. This mutation process increases the variability of population individuals to consider thus increasing the chance of avoiding local optima and reaching global optimum.

This process of selection-crossover-mutation is repeated in order to create the number of children needed for creating the new population to replace the old one. This is the most used population replacement technique but other techniques that replace not the whole population but a certain number of individuals, do exist as well. The process of creating new populations is iteratively repeated in a loop for several generations until a stopping criteria is met and the individual with the best fitness is the final solution of the problem. One of the distinguishing characteristics of the genetic algorithm compared to other heuristic methods is that it explores and improves several candidate solutions at the same time. This offers more opportunities for finding the optimum solution as it explores more than one solution at a time. This also has the benefit for a parallel implementation increasing the scalability and efficiency of the method.

2.3 Related Work

In this section, an overview of research work related to dynamic resource allocation to VMs in data centre and cloud computing environments is provided. This work is grouped into four classes based on VM resource allocation mechanism applied and if it includes VM resource demand prediction or not. One group mostly deals with global level resource allocation by dynamically mapping VMs to physical machines at runtime through live migration, to consolidate or balance the load. Another group deals with applying horizontal scaling, by changing the number of VM replicas allocated to an application. In the third group are those that as their main resource allocation mechanism use vertical scaling where the resource amount given to a VM is changed dynamically at the physical machine level. In the last group, the works are considered that take a proactive resource allocation approach by predicting VM resource demand and applying any of the allocation mechanisms mentioned above. There are also works that combine all the mentioned resource allocation mechanisms and they are considered in one of the above groups. The focus is only on the works that allocate resources to VMs to keep application performance according to SLAs while reducing resource and operating cost.

2.3.1 Live Migration based VM Resource Allocation

One of the most useful and cost effective ways to allocate resources to VMs at the global level is through VM live migration. This makes it possible to consolidate the VMs in as few physical machines as possible by turning off the unused ones when the workload reduces, therefore saving power and management costs. The problem is finding and changing at run time according to workload the optimal mapping between VMs and physical machines. Normally, this problem requires answering four main questions such as: 1) when to start VM live migrations, 2) which physical machines to consider as sources of VM live migration, 3) which VMs are migrated from the selected physical machines and, 4) which physical machines serve as destinations of VM live migrations. This is an NP-hard optimization problem that belongs to the knapsack bin packing problems family and different approaches are developed to address it.

There are a group of works [122, 63, 11, 70, 35] that use live migration for dynamic load balancing and consolidation of VMs. They avoid physical machine overloading or provide consolidation of VMs to fewer physical machines. They decide to migrate the required number of VMs based on heuristic algorithms such as First-Fit Decreasing or Best-Fit Decreasing. Most basic approaches use resource utilizations to estimate VM resource demand and utilization thresholds to signal machine overloading or under-loading for starting live migrations.

Wood *et al.* [122] propose an approach called *Sandpiper* for mitigating overloaded machines or hotspots through remapping VMs to physical machines. They use two approaches for this purpose. A black-box approach that monitors the resource data from outside the VMs that is OS and application agnostic and a gray-box approach that uses monitoring data inside the OS to make a more informed decision for hotspot detection and VM live migrations. Their hotspot detection mechanism is based on monitoring and profiling resource data for predicting when the utilization will overpass the threshold. To mitigate hotspots, they use a heuristic greedy algorithm that migrates most loaded VMs to least loaded physical machines in an iterative process. To achieve this, they measure the load of virtual and physical machines with what they call a volume metric that captures the combined load of CPU, memory and network resources.

Similarly, Khanna *et al.* [63] propose an approach for dynamic VM consolidation that increases resource utilization and keeps performance SLA violations to minimum. It depends on resource utilization thresholds to determine when a VM live migration should be initiated. The basic idea of their consolidation algorithm is to pack VMs to as few physical machines by rearranging the VMs in such a way that the variance of residual resource capacity vector of all physical machines is maximized. This is done by ordering VMs and physical machines in non-decreasing order of their utilization and residual capacities, respectively and migrating the least utilization VM (in order to reduce migration cost) to the least residual capacity physical machine.

Beloglazov *et al.* [11] propose energy-aware heuristics for dynamic consolidation of VMs, while taking into account the power consumption of the data centre infrastructure. They divide the dynamic VM resource allocation process into two steps. In the first step, they decide when and which VMs should be migrated. Basically, they achieve this by defining lower and upper utilization thresholds and monitor when VMs exceed them in order to be selected for consolidation or load-balancing. For this purpose, they developed two VM selection policies, one that minimizes the number of migrations and the other that minimizes the potential growth of physical machine utilization and therefore performance SLA violations. In the second step, a modified Best-Fit Decreasing heuristic is used. Here, the selected VMs are sorted on decreasing order of utilization and each of them is migrated to the host that provides the least increase in power consumption as the result of the migration.

Li *et al.* [70] also developed an approach for dynamic VM consolidation by packing VMs to as few physical machines as possible. Their approach takes into account the events of arriving and departing of VMs to decide when and how to rearrange them through live migration. This is done through a recursive algorithm based on first-fit and best-fit heuristics. It replaces a small workload VM with a bigger one arrived and reinserts the small workload VM to the pool of resources by considering it as a new arrived and then proceeding in the same manner. In the end, the output is a list of VMs to be migrated to other physical machines. The motivation behind this method is that small workload VMs have a higher chance to fill the residual resource gaps therefore providing a more compact packing of VMs.

Ferreto *et al.* [35] developed an approach called *dynamic consolidation with migration control*. In contrast to other works, they focus more on reducing the number of migrations by preventing migration of VMs with steady load thus reducing their performance violation as the result of VM live migration. They formulated the VM consolidation as linear programming problem to reduce the number of physical servers and extended the FFD, BFD and WFD heuristics to take into account the added constraint of not remapping VMs that do not change their workloads.

There are also works [39, 13, 5, 12] that use more sophisticated techniques than just static utilization thresholds to decide when and how to migrate for dynamic VM consolidation. For example, Gong and Gu [39] developed Pattern-driven Application Consolidation (PAC) that applies signal-processing techniques to extract VM workload resource usage patterns called signatures. This approach is based on workload signatures to achieve better VM consolidation. Similar to other works, it also uses a greedy heuristic algorithm for consolidation. However, after sorting VMs in decreasing order of their resource demand, it matches the VM workload resource usage signatures with host residual resource capacity signatures to find the best matching hosts to accommodate VMs.

Andreolini *et al.* [5] consider new techniques to identify critical hosts to migrate VMs from and select VMs to migrate in a way suitable for infrastructures with thousands of hosts. Their goal is to develop robust host overload detection mechanisms providing stability in dynamic VM migration management. First, to achieve the required stability, they identify a host as overloaded only when there is substantial change in its load state as resulted from the CUSUM algorithm. Secondly, as done in other works, they sort potential migrating VMs, based not just on their instantaneous or average resource utilization value but also based on a metric that captures load behavioral trends, and select for migration those VMs that contribute the most to the total load of the host.

Beloglazov *et al.* [13] also consider dynamic VM consolidation throughout live migration and propose an approach for detection of overloaded hosts to migrate VMs from. The goal of their overloaded host detection algorithm is to maximize the mean time between migrations, which as they showed analytically leads to better consolidation quality, and reduced performance violations. In maximizing the mean time between migrations they take into account also the QoS constrain by expressing it in a VM workload independent metric. To solve this optimization problem under a known steady workload, state configuration and QoS constrain, they developed a Markov chain model for deriving a randomized control policy. To take into account changing workloads they applied workload estimation based on a Multi-size Sliding Window providing an adaptive algorithm.

Beloglazov *et al.* [12] extended their previous work [11] on using heuristic algorithms for dynamic VM consolidation by including dynamic utilization thresholds for finding better trade-off of power and performance. The goal is to adapt the utilization thresholds according to workload variations. The thresholds are estimated based on resource utilization interval reached with 5% probability, which is derived by analysing probability distribution of historic resource utilization of a host.

The problem with all above approaches is that they base live migration decisions on low level utilization thresholds, which do not adequately map to high level metrics such as application performance or cloud provider profit. Another drawback is that these approaches are based on fixed heuristic policies such as First-Fit Decreasing that are not flexible and do not deal well with optimizing conflicting objectives such as resource costs and application performance. If the need for a policy change arises for example to make the system more aggressive in balancing the load than consolidating it, the algorithm must be changed completely. In contrast, the approaches presented in this dissertation attack the above drawbacks by basing live migration decision directly on high level metrics such as utility function value that represent the profit of the cloud infrastructure, which providers are more interested to optimize. It also offers the required flexibility by capturing in a natural way multiple conflicting objectives in the utility function and easily changing the importance weights of one objective over the other.

There is a group of works [109, 42, 92, 73, 68, 111, 59] that similarly to this dissertation's work represent the dynamic VM placement through live migration as a utility function optimization problem. For example, Goudarzi *et al.* [42] formulated a mixed integer non-linear programming function optimization that would solve the problem of dynamic VM consolidation for reducing power, migration and performance SLA violation costs. They proposed a heuristic greedy algorithm based on dynamic programming and convex optimization for finding an initial solution. Then, they apply two local search methods to improve the initial solution.

Similarly to the above work, Petrucci *et al.* [92] treat the dynamic VM consolidation, for minimizing power costs and keeping performance at SLA levels, as a function optimization problem. In the function optimization, they also include the selection of different combinations of CPU frequency and voltage. Mathematically, they formulate the problem as a mixed integer programming. To solve the optimization problem, they apply a branch-and-cut deterministic algorithm. Through simulation experiments, they have found out that the approach scales well by managing up to 350 physical hosts.

Liu *et al.* [73] also address the problem of optimizing the mapping of VMs to physical machines for reducing power costs and keeping performance levels. They represent the problem as optimization of a cost function, which includes physical machine costs, migration costs and performance costs and try to optimize it through a heuristic search algorithm. The algorithm searches over a graph where each node represents a placement of VMs to physical machines and an arrow connecting two nodes represents a transition achieved through VM live migrations. In the searching process, it keeps track of the best placement found so far and two tables, an open one holding unexplored placements and a closed one holding all already explored placements. The algorithm finishes when some criteria are meet such as the open table becomes empty or the search time limit is reached.

Van *et al.* [109] propose an autonomic resource controller that finds an optimal number and sizes of VMs, by maximizing a utility function using a Constraint Programming (CP) approach. However, the authors do not base migration decisions on the utility value, but use the CP approach to minimize the number of physical machines. They also do not support dynamic fine-grained CPU resource allocation.

Kusic *et al.* [68] propose a dynamic resource-provisioning framework that determines the number of VMs and their CPU allocation, maximizing a utility function by using limited look a head control. This takes into account node switching cost and risk of provisioning. Although the authors apply VM resource allocation based on utility function optimization, they do not consider VM live migration as an allocation mechanism for utility optimization.

The pMapper [111] system is similar to this dissertation's work because it includes a power-aware application placement controller that tries to optimize a cost-performance objective function. First, it orders the machines according to optimal power cost, but after that it views the problem as a bin-packing problem and uses some heuristic algorithm such as a modified version of First Fit Decreasing (FFD) to make migration decisions. In contrast, the heuristic algorithm presented in this dissertation directly depends on the value of the utility function, which is more flexible and supports differentiated service.

Although the above works base VM resource allocation and live migration decisions on objective function optimization, the approach presented in this dissertation has a set of unique features that others do not. The main difference is the unique combination of local fine-grained VM resource allocation based on control-theory with VM live migrations based on global utility function optimization. Some approaches solve the function optimization problem based on constrained or non-linear integer programming that can require a large running time. This makes them not suitable for practical application. In contrast, this dissertation presents a simple utility function optimization heuristic with negligible running time that is more suitable to be applied in practice. Some approaches are evaluated only based on simulated experiments making it difficult to predict how they will behave in realistic cloud environments. In this dissertation, the evaluation is done on realistic environments showing promising results in practice. Lastly, most approaches do not take into account instability resulting from VM live migration going back and forth between physical machines as the result of workload variation. In contrast, this work takes preventive measures based on workload standard deviation and filtering techniques that take into account and mitigate the instability.

There is also a group of works [126, 128, 104] that use decentralized VM allocation algorithms or take into account affinity of VMs such as communication patterns, in VM live migration decisions. Sonnek *et al.* [104] present *Starling*, a system that applies a distributed algorithm for affinity-aware live migration of VMs that take into account communication patterns between VMs. By closely placing two or more communicating VMs in the network hierarchy, the network transfer bottleneck is avoided and performance is improved. To achieve the above goal they developed a distributed bartering algorithm in which physical machines negotiate with their neighbors in order to decide which VMs are going to be migrated to improve their affinity. Their approach does not offer a complete solution in the sense that it only optimizes dynamic VM placement according to network traffic affinity, without considering other resources such as CPU or memory.

Zhang *et al.* [128] propose a similar approach to the *Starling* system, but offer a more comprehensive approach by trying to reduce the number of physical machines for power cost reduction. They apply a heuristic algorithm based on K-means clustering to consolidate VMs in clusters with low inter-communication traffic, while lowering the number of physical machines. The drawback of this approach is that the dynamic placement of VMs to physical machines through live migrations is not taken into consideration. It only considers initial consolidation or dynamic consolidation at the moment that new VMs enter or old VMs leave the data centre.

Yazir *et al.* [126] propose a distributed approach for dynamic remapping of VMs among physical machines through live migrations. Their distributed architecture is composed of a set of node agents, attached to physical machines. The decision making process for VM resource allocation is distributed among the node agents which decide in parallel based on local information when and which VMs to migrated. They use multiple criteria decision analysis based on the PROMETHEE method to select which physical machine is more suitable to migrate the selected VMs. Although this approach does not find a global near-optimal VMs to physical machines mapping, it is scalable and feasible by requiring lower number of VM migrations then would be required by a centralized global near-optimal approach. Their approach differs from this dissertation's work since it takes migration decisions based only on resource utilizations without considering optimizing high-level metrics such as cloud profit utility value. They also base their evaluations only on simulation experiments without concluding how the approach would behave in realistic environments.

2.3.2 Vertical Scaling VM Resource Allocation

Another mechanism usually applied for VM resource allocation is vertical scaling. This is concerned with dynamically changing resource shares to VMs on each physical machine in order to reduce resource under-utilization and costs, while keeping performance to required SLA levels. In this section, the literature's most important approaches to achieve VM resource allocation on physical machine level are reviewed. It is divided into three groups. One group of related works apply control-theory based feed-back controllers for VM resource allocation. Another group applies queuing theory model based optimization and the last one applies machine learning for VM performance modelling and resource allocation.

The first group [118, 130, 49, 90, 89, 61] deals with VM resource allocation on physical machine level using control-theory feed-back controllers. Their goal is to keep certain performance metrics to given reference values even in the presence of disturbances. This is achieved by using a control law derived usually from a system model identification process.

Wang *et al.* [118] developed an approach based on feed-back control for web server CPU partition sizing to lower resource waste and keep response time to acceptable levels. They proposed an integral controller with adaptive gain to keep CPU utilization to a reference value, and an adaptive PI controller to keep the response time to a reference value. They identified a non-linear system model with bimodal behavior of the relationship between CPU entitlement and response time in the under-loaded region making it difficult to control response time through CPU allocation in this region. In order to provide a more robust controller by obtaining more accurate system model parameters they included the CPU utilization measurements in the control loop. Later, they improved the above work by proposing a better approach [130] that provides a more robust and controllable system for keeping response time to reference value. They achieved this by proposing a design with two nested control loops. The inner control loop is an adaptive integral controller that keeps CPU utilization to the reference value. The outer loop is an integral controller that continuously tunes the CPU utilization reference value to keep the response time metric to the desired value. Later on, they extended the approach to include also memory allocation [49] in virtualized environments by creating joint CPU and memory controllers in order to keep CPU and memory utilizations to reference values.

Similar to the above works, Padala *et al.* [90] use control-theory feed-back controller to allocate CPU resource to VMs. However, in contrast to the above, they consider the allocation of several VMs of a multi-tier application running on different physical machines. For this purpose they developed a two-layer controller that takes into account the interdependencies between different tiers of the same application. The first layer is composed of adaptive integral utilization controllers that keep utilization to required level and decide on CPU allocation of every VM. The second layer is an arbiter controller that gets CPU allocation requests from utilization controllers. Later, based on different bottleneck scenarios, decides on the final CPU allocations for all VMs to provide a certain ratio between performance metrics of different applications.

In a later work [89], Padala *et al.* proposed *AutoControl*, an adaptive feedback controller for resource and performance management of multi-tier VMs in virtualized environments. They applied an Auto-Regressive-Moving-Average (ARMA) model estimated online using the Recursive Least Squares (RLS) method [7] to adapt the relationship between performance and resource allocation according to the workload dynamics. They developed a two-layer controller for allocation of CPU and disk bandwidth resources to the VMs of multi-tier applications. In the first layer, the Multiple-Input Multiple-Output (MIMO) application controllers are located that based on the ARMA model decide on the resource allocation to be given to multiple tiers of each application. Application controllers apply an advance control-theory technique, more specifically an optimal control approach, which uses a control law derived from a cost function optimization. In the second layer, the node controllers are located that take resource allocation requests from different application controllers and decide on the final VM resource allocations. In the case of resource contention, where the total resource requests are greater than the resource capacity, the node controller redistributes the resource allocations in such a way as to reduce the difference between the normalized application performance and its reference value.

Kalyvianaki *et al.* [61] proposed a feed-back control approach based on the Kalman filter [60]. A Kalman filter is a recursive data processing algorithm for estimating the state of a linear process over the time even in the presence of noisy measurement. The authors apply a Kalman filter to track CPU utilization and to accordingly decide on the CPU allocation to be given to VMs. They developed three controllers. The first is a basic Single Input Single Output (SISO) controller to allocate CPU resources to each VM individually. The second is a MIMO controller that allocates CPU resources simultaneously to all VMs of a multi-tier application, by taking into account the interdependences between VMs of the same application. This is achieved by estimating the pairwise covariances between VM resource utilizations. The third one is an adaptive MIMO controller, similar to the second one, except that in order to adapt to workload dynamics it dynamically changes the controller parameters at runtime.

Approaches that use control-theory have several drawbacks that are tackled in this work. Primarily, they focus on allocation of one, or maximum two, resources such as CPU and memory or CPU and disk I/O bandwidth. The approaches presented in this dissertation consider multiple resources such as CPU, memory, disk and network I/O bandwidth. Furthermore, control-theory based approaches use linear models of relationship between resource allocation and application performance, which cannot capture complex non-linear relationships and VM interferences that exist in realistic shared virtualized infrastructures. This can result in suboptimal control performance of VMs in realistic environments. Instead, this work copes with this drawback and improves over the above approaches by modelling resource performance relationships using machine-learning techniques such as Artificial Neural Network (ANN), which can capture non-linear relationships thanks to its universal approximation capability.

In recent years, a wide spread use for dynamic VM resource allocation has found the application of fuzzy control and fuzzy logic [125, 98]. This has the advantage of dealing flexibly with non-linearities and uncertainties in virtualized workload changing environments.

Xu *et al.* [125] present a two-level resource manager for VM resource allocation based on fuzzy logic modeling. In the first level, local controllers are found that determine the resource demand of each VM application and decide on resource allocation required to keep resource costs to minimum and satisfy SLA performance. Local controllers base their decisions using fuzzy logic to model in the form of fuzzy-rules the relationship between application workload (request rate) and resource demand utilization. The fuzzy logic model and rules are learned and adapted online by periodic measurements, after applying filtering and clustering on data. The global controller gets resource allocation requests from local controllers and decides on the final allocation for all VMs based on maximizing the total profit of the data center provider.

Rao *et al.* [98] present DynaQoS, a VM resource management approach based on fuzzy control that supports multi-objective optimization and service differentiation. DynaQoS is composed of two layers. In the first layer, there is a set of Self-Tuning Fuzzy Controllers (STFC) that control certain application objectives such as QoS metric or power consumption. The STFC produce a VM resource allocation request for the next interval based on the error of the objective metric with respect to reference value and the change of the error. The heart of STFC are a set of fuzzy-rules expressed as high level linguistic variable relationships that decide on the degree of changing the current resource allocation for different situations of objective metric error and error change. The STFC is complemented with a self-tuning scaling controller and an adaptive output amplifier that dynamically change the input, output scaling factors and amplifier output for better adaptability to changing workload. In the second layer, there is a gain scheduler that gets resource allocation requests from different STFC-s with different objectives and produces a final resource allocation as the weighted sum of individual requests. In this way, DynaQoS achieves optimization of multiple conflicting objectives such as application performance and power costs.

In this dissertation, an approach based on fuzzy control is presented, but in contrast to the above works it expresses conflicting objectives in a utility function and apply fuzzy control to optimize it. Furthermore, a multi-agent version of the fuzzy controller is developed where several agents apply fuzzy control for local utility function optimization in parallel with other agents, resulting in a suitable approach for large-scale multiple VM environments.

There is a group of works [28, 14, 75, 24] that apply application performance modeling based on queuing theory and utility function optimization to allocate resource and manage performance in non-virtualized and virtualized environments.

For instance, Doyle *et al.* [28] propose an approach for CPU, memory and disk I/O allocation to web service applications based on analytical performance modeling. They combined three queueing theory based models to estimate the web service response time for a certain request rate and resource allocation: a) one that predicts response time in relation to CPU allocation, b) one that predicts memory hit ration needed for memory allocation and, c) one that predicts response of disk storage in relation with bandwidth allocation. Based on these performance models, a resource allocator finds the minimum resource allocations to web services that satisfy the response time targets.

Bennani and Menasce [14] present an approach that applies autonomic computing principles in order to allocate compute resources to Application Environments (AE) for satisfying their SLA performance metrics by optimizing a global utility function. The global utility function is given as a function of AE local utility functions. AE local utility functions depend on application performance metric. They have developed analytic queuing network models for AE performance metrics. Based on these AE models, they applied a combinatorial search algorithm such as the beam search algorithm [99] to search the physical nodes configuration space, aiming to find the configuration that optimizes the global utility function. Later, they extended the work [75] in order to apply it in virtualized environments by using analytical queuing models and global utility optimization through beam-search algorithm, but in this case using CPU shares as allocation mechanism for VMs.

Cunha *et al.* [24] consider allocation of resource to VMs in multi-tier applications for cloud provider revenue optimization. For this purpose, they developed an approach that combines a pricing model with an analytical queuing model and an optimization model. The pricing model is based on two SLA layers that give the rewards and penalties for achieving or violating the performance threshold for two operating modes, normal and overload. The performance model of each application is represented as a tandem queue network, where each VM assigned to a tier is modeled as a M/M/1 queue. The optimization model is represented by an objective function, given as the sum of rewards and penalties over all applications, and represents the revenue of the cloud provider. In comparison to the author's previous approach, which used a single-resource queuing model, this one is more cost-effective.

The problem with the above approaches is the difficulty to build accurate analytical queuing models for predicting the complex behavior of applications in virtualized environments especially in the presence of VM performance interference. They are also specific to certain multi-tier transactional applications. In contrast, this dissertation's approach builds a general model during runtime without any prior application knowledge using ANNs that can capture the complex behavior of applications and VM interferences in shared virtualized environments.

The last group of works are those that apply machine learning for VM performance modelling and resource allocation. The works are divided in two groups, one that apply supervised machine learning and the other that apply reinforcement learning. In the supervised machine learning case, a model is build off-line or online after provided with training input-output samples. Based on this model, resource allocation to virtual machines can be achieved to reach required performance levels. In the reinforcement learning case, a resource allocation policy is learned without any training set supervision but only with trial-and-error actions in order to maximize a long-term reward function.

Some works apply supervised machine learning [120, 67, 119] for resource modeling and allocation in virtualized and no-virtualized environments. For example, Kundu *et al.* [67] model application performance in virtualized environments for helping with VM allocation. They developed multiple control knobs model of relationships between application performance and resource allocation and contention. To build the model they applied Artificial Neural Network (ANN) and Support Vector Machine (SVM) learning approaches and for increased accuracy they refined the basic models through techniques such as sub-modeling and clustering. Based on built models, they proposed a simple VM allocation approach.

Wildstrom *et al.* [120] present an approach for hardware resources such as CPU and memory reconfiguration according to workload changes. They applied a propositional rule learner to build an off-line model for predicting application performance given as input resource configuration and low-level system metrics. To adapt to workload changes, the model is learned for different combinations of workloads and resource configurations. The control agent adapts the configuration online using the model based on the current workload as identified by low-level system metrics.

In a later work [119], the same authors introduced CARVE, a system that predicts what would be the utility profit value of increasing or decreasing the memory resource of an application for the current workload and configuration. In this work, the utility value depends on the profit value of getting certain application performance and the costs of provisioning the required memory resource. They learned models to predict the utility value for increasing and decreasing memory by one resource unit based only on low-level system metrics. Based on these predictions, an online agent continuously makes decisions on the worth of changing memory configuration. They trained the model for different workload combinations and memory configurations based on M5' trees [121] that are decision trees with linear regression functions at the leafs. As the authors argue, the M5' trees showed better prediction accuracy than simple ANNs.

Also, this dissertation's work applies supervised machine learning techniques such as ANNs for VM resource allocation, but most of the other proposals train a single model offline for all workload and resource allocation combinations, which is not flexible in a dynamic workload varying environment. In contrast, this dissertation's approach trains a model of resource-performance relationships online using an ANN, which can be adapted according to workload changes. Furthermore, it finds a suitable performance-power trade-off through utility function optimization and deals with increased optimization time, as the result of large number of VMs, by using a distributed ANN-based VM resource allocation manager.

Another group of related work [97, 106, 124, 96] developed techniques based on reinforcement learning for dynamic resource allocation. Reinforcement learning has the potential to achieve resource allocation without learning a system model. Rao *et al.* [97] present an approach for dynamic VM resource reconfiguration based on reinforcement learning, where an agent interacts with the environment by trying different resource allocation actions to learn an optimal policy. The optimal policy gives the resource allocation action that should be taken in each state to maximize the long-term reward function value. The state is determined by the resource allocation configuration while the reward function value is determined by the average application performance over all VMs. For solving the reinforcement learning problem, they applied the well-known temporal-difference method. To cope with the problem of adaptability and scalability of a large state space, resulting in long optimal policy convergence time, they employed two supervised machine learning models. First, they learned an ANN based model to simulate several interactions with the environment and generate reward values for updating accumulative reward function. This gradually eliminates the long interaction time with the real environment. Second, they used an ANN based approximation of accumulative reward function which replaces a lookup table implementation and eliminates the long updating time of every entry in the table.

Later, the authors extended the work by developing a unified reinforcement learning approach [124] that combines two reinforcement learning agents. The VM-agent applies reinforcement learning for resource reconfiguration of all VMs using the geometric mean of performance metrics over all VMs as reward function. The App-agent applies reinforcement learning for reconfiguration of application parameters, such as *MaxClients*, for all VMs of the multi-tier application using the application performance metric as reward function.

The above-proposed solutions are applicable on a single physical machine, but become problematic from a scalability point of view for cluster wide level optimization with increased number of VMs running on several physical machines. To address this issue, the same authors proposed [96] a distributed learning approach where each reinforcement learning agent learns an optimal policy for resource allocation of its own VM in parallel with other agents. Each agent uses multiple resource utilizations of the VM as the environment state and implements a Q-function table based on Cerebellar Model Articulation Controller. In order to optimize in the same time resource efficiency and application performance, as reward function, they used a metric that captures both resource utilization and application performance metric. To coordinate resource allocation and take into account the correlation between different VMs of the same multi-tier application, the same feed-back reward function value is provided to all VM agents of the same application.

Tesauro *et al.* [106] developed a hybrid reinforcement learning approach to dynamically allocate servers to different web applications in a data center. Initially the system is driven by a build-in queuing model based control policy during which the reinforcement learning module is trained offline. This eliminates the scalability problem resulting in poor performance by the long online learning time of the optimal policy. Later, the reinforcement learning agent takes over by driving the system, using the new learned policy and collecting samples for learning a better policy through batch training. They used the Sarsa(0) learning algorithm and an ANN-based approximate utility function instead of a look-up table implementation. They applied a de-compositional method by using separate reinforcement learning for each application using a function of application performance as utility function. In this method, each application agent sends its learned utility function to an arbiter, which decides on all server allocations in order to optimize a global utility function. This utility function is given as the sum of application local utility functions.

The drawback with reinforcement learning approaches is the scalability problem from large state space and long convergence time to optimal policy, making them impractical in real cloud computing environments. In contrast, this dissertation's approach adopts VM performance modelling based on supervised machine learning and distributed allocation to cope with scalability and convergence time issues.

2.3.3 Horizontal Scaling VM Resource Allocation

In this section, the work related to horizontal scaling of VMs is reviewed, where the dynamic resources allocation is achieved by changing the number of VM replicas allocated to an application. This technique is more suitable for multi-tier applications where each tier can be replicated in several VMs in order to balance the load. The number of replica VMs can be automatically changed according to the workload to keep the performance levels and reduce resource costs. The related work can be divided in four groups: 1) rule-based approaches 2) queue modelling approaches 3) machine learning approaches and 4) control-theory approaches. There are works that use a combination of techniques but for the purpose of classification they are included in one of the groups that mostly represents the main technique used. Since the horizontal scaling mechanism is not directly related to this work, but can be used as a complementary mechanism, in the following just a few typical examples in each of the above categories will be explained.

Several works [30, 45, 46, 56] achieve horizontal VM scaling using rule-based techniques. The basic idea of all of them is to set-up rules in the form of: *when some performance metrics pass certain threshold values than increase or decrease the number of VMs.* The disadvantage of rule-based approaches is the difficulty to set the right threshold values for all applications, resources and workload dynamics. A typical example of the rule-based approach is presented by Han *et al.* [45]. The authors propose a lightweight auto-scaling approach for multi-tier applications that combines fine-grained VM resource allocation with horizontal scaling VM-level resource allocation, to achieve cost-effective performance optimization. Their system starts scaling up or down resources when application response time passes above or below a certain threshold. When this happens, it firstly applies a fine-grained resource level VM scaling algorithm to give as many needed resources to VMs. If this does not bring the response time to the required threshold interval, it then applies a VM level scaling algorithm to increase or decrease the number of VMs. This algorithm does a scale up by detecting the bottleneck tier, based on the criteria that selects the tier with least resources available and least resource costs increase from adding a VM. On the other hand, it does a scale down by detecting the bottleneck tier, based on the criteria that selects the tier with biggest resources available and biggest resource costs decrease from removing a VM.

The other group of works [108, 129, 20, 33, 15, 58] adopt queuing theory approach to model certain performance metric and decide on the number of VMs to scale. An application in virtualized environments can be modelled using a single queue model where a load-balancing component distributes the requests over all replica VMs. It can also be represented with a model of multiple queues forming a queue network that is more appropriate for multi-tier applications. The main problem with queuing theory-based approaches is the difficulty to adopt accurate analytical queuing models to represent complex real world scenarios. A typical work that represents queuing theory based VM horizontal scaling is presented by Urgaonkar *et al.* [108]. They developed an approach to determine when to allocate additional servers and how many of them to allocate on each tier using a combination of reactive and predictive techniques. Their queuing network model of the multi-tier application is composed of a set of G/G/1 queues representing servers of each tier. Using this model, and having the peak session rate and request service time of each server, is possible to estimate how many servers to allocate to each tier in order to achieve the required response time. Although the number of servers is estimated by separately modelling each tier, since this estimation is done at once and in a coordinated way for all tiers, it achieves fast allocation time and eliminates potential bottleneck shifts. For long-term allocation they apply a workload prediction technique, which based on past observations and seasonal property of the request rate, predicts the peak request rate demand for the next hours and accordingly allocates servers a head of time. To cope with long-term prediction errors and unexpected workload spikes they apply a reactive allocation technique in which the monitored request rate over the past minutes is periodically compared with the predicted request rate. If their difference is more than a certain threshold then allocation actions are applied to provision the multi-tier application with the right number of servers.

Several proposals [9, 29, 17, 18] apply machine learning for VM horizontal scaling. One of the recently applied approaches for VM horizontal scaling is reinforcement machine learning technique. Its advantage is the potential to learn a VM allocation policy without need for an environment model but just by trial-and-error interaction with the environment. One of the drawbacks of this machine leaning technique, as applied in real cloud environments, is the long convergence time to optimal policy as stated early. Recently there have been some proposals to cope with this problem such as the one presented by Barrett *et al.* [9]. They propose a parallel reinforcement learning method where different learning agents apply Q-learning in parallel with other agents on a common task to learn their optimal policy Q function while exchanging Q function values with other agents for states they have not visited. This parallel reinforcement learning approach speeds up the optimal policy convergence time. Each agent maintains a local Q_l function learned from its own experience and a global Q_g function that is the aggregated combination of Q_l functions of all other agents. Agents make decisions for adding or removing VMs based on a final Q function estimated by aggregation of local and global Q functions. During the run, if the difference between local and global Q function values for the current state is less than certain threshold, then the local Q_l function is not transmitted to other agents since the local and global functions have converged to the same value. The Q function is based on resource costs and application performance while the state of each agent is defined by three variables: a) the user requests per time interval, b) number of VMs allocated to the application and c) UTC time. This state definition with limited number of variables is another contribution to lower the state space and the optimal policy convergence time.

Lastly, there are works [2, 72, 71, 69] that apply control theory for VM horizontal scaling. Control theory approaches offer the possibility to mathematically verify the stability and guaranty of the controlled system, but require a lot of effort for manual tuning of controller parameters suitable for a certain situation and system. One typical example of applying control theory for VM horizontal scaling of multi-tier applications in cloud infrastructures is presented by Lim *et al.* [72]. The authors propose a proportional threshold policy to apply in a classical integral feed-back controller. This policy uses a dynamic target threshold range to keep a reference metric, such as CPU utilization, instead of a single target threshold that is used in classical integral controllers. If the CPU utilization goes above the range, the controller adds a VM and if it goes below it removes a VM. As experimentally shown, having a target range eliminates the oscillatory behavior that results from coarse-grained actuators typically used in existing cloud provider infrastructures such as Amazon EC2 [4]. The target threshold range is dynamic in the sense that it decreases with increasing the accumulated actuator values. This provides a more fine-grained and resource efficient solution.

Horizontal VM scaling approaches based on changing the number of VMs according to the workload are complementary to this dissertation's work. However, they have the drawback of being specific to only certain multi-tier web applications that permit tier replication, while in the dissertation are provide more general solutions by using VM live migration and fine-grained VM resource allocation mechanisms.

2.3.4 Resource Demand Prediction based VM Resource Allocation

In this section, works related to proactive VM resource allocation based on resource demand prediction techniques are reviewed. Some works [54, 76, 16] apply VM resource demand prediction for mapping VMs to physical machines. Huang *et al.* [54] propose an approach called Migration-based Elastic Consolidation Scheduling (MECS) that dynamically maps VMs to physical machines to lower the number of physical machines and satisfy application SLA performance. It makes VM allocation decisions based on VM resource demands estimated by a time series prediction technique. They used Auto-Regressive Integrated Moving Average (ARIMA) technique to predict CPU and memory usage time series. MECS algorithm is divided into resource conflict prediction phase and VM consolidation phase. The first phase, based on predicted VM resource demand time series, predicts the physical machines that will be overload and decides on VMs to migrate. The second phase decides on which physical machines the selected VMs should be live migrated in order to minimize the number of physical machines and satisfy performance SLA-s, while taking into account live migration overhead.

Meng *et al.* [76] propose an approach for consolidation of VMs on few physical machines by joint-VM capacity estimation and by multiplexing VMs with complementary resource demands therefore, offering the maximum resource saving and efficiency. They developed an SLA performance model of relationship between performance and VM capacity. They also developed a joint-VM size estimation algorithm, which based on the SLA performance model and a VM workload forecasting technique, estimates the minimum aggregated capacity of a set of VMs that can satisfy the performance SLAs of individual VMs. For more accurate forecasting of the aggregated VM workload, each VM workload is divided into regular and irregular components. The regular component represents trend or seasonal periodic workloads, while irregular component is what remains from the extraction of the regular component. Then, the aggregated workloads are forecasted separately, using time series forecasting techniques such as ARMA and Artificial Neural Networks. Based on the above methods, they developed a selection algorithm to identify the combination of VMs to be consolidated jointly that will result in the maximum resource capacity savings. A VM workload time series correlation matrix was build and those VMs that have lower correlation coefficients were selected for multiplexing. This means to select for multiplexing those VMs that have complementary workloads resulting in lower resource capacity requirements.

Bobroff *et al.* [16] propose a dynamic VM consolidation algorithm called Measure-Forecast-Remap (MFR) based on VM live migration. Its goal is to lower the number of physical machines and application SLA performance violations. The algorithm executes iteratively in discrete time intervals of length τ where in each iteration repeating three steps: a) measuring the history data, b) forecasting the future demand and c) rearranging VMs to physical machines. The authors presented an analytical formula to estimate the gain from dynamic consolidation of a VM based on resource history demand and predicted resource demand time series. This formula determines the VMs that have highly variable, auto-correlated and periodic resource demand as the ones that can gain the most from dynamic VM consolidation. They apply time series analysis [19] on the resource demand history by expressing resource demand as the sum of periodic components and residuals. After removing the periodic components, the residuals are modelled using an Auto Regressive process of second order

AR(2). Based on this model, they predicted the future demand and estimated the prediction error distribution to be used in the gain formula mentioned above. Lastly, they developed a VMs to physical machines placement algorithm that, based on predicted VM resource demands and a first-fit bin packing heuristic, places VMs to minimum number of physical machines by satisfying the constrain that the rate of VM resource demand over-passing capacity is less than a certain threshold.

There also is a group of works [40, 32, 87, 57, 53] that apply vertical or horizontal scaling VM resource allocation based on resource demand prediction. Gong *et al.* [40] developed PRedictive Elastic reSource Scaling (PRESS) system for dynamic fine-grained resource allocation to VMs, in order to reduce resource costs and application SLA performance violations. It tracks resource utilization and predicts VM resource demand in the near future, by using signal processing and machine learning techniques. It uses the Fast Fourier Transform (FFT) method to extract a repeating pattern of the resource demand called signature, if one exists, and uses it for prediction in the next time intervals. If no signature exists, then it applies a discrete-time Markov chain model of resource utilization states to predict resource demand for the next interval. After getting the prediction, it sets VM resource allocation for next interval equal to predicted resource demand, plus a small headroom to avoid any underestimation of demand because of prediction errors.

Nguyen *et al.* [87] propose AGILE, an approach for dynamic horizontal scaling of VMs according to the workload in IaaS clouds, in order to reduce resource costs and performance SLA violations. It is based on a medium-term resource demand prediction and a model of relationship between resource allocation and SLA performance violation rate. They apply a wavelet transformation technique for resource demand prediction. This technique decomposes the resource demand time series signal into several base wavelet signals that are predicted separately and summed up in the end to create the complete predicted demand. As the authors argue, this technique makes it possible to accurately predict non-periodic signals and for longer future intervals (up to 2 minutes) than other techniques. Based on online profiling and regression curve fitting, they build a black-box model that predicts what would be the amount of resource allocation to get a certain performance SLA violation rate. Based on resource demand prediction and performance model, they can predict whether the application will be overload in the future and before this happens, trigger a scaling operation of adding VMs. In order to reduce the VM provisioning time, they apply pre-copy live cloning for VM replication.

Islam *et al.* [57] also consider the problem of proactive VM resource provisioning in cloud computing based on resource demand prediction. They propose and experiment with some resource prediction techniques based on machine learning such as linear regression and artificial neural networks, combined with a sliding window method. Although, they argue that their techniques can help with VM scaling they did not integrate their prediction models with any scaling method, but only focused on analysing the prediction accuracy. Two machine learning methods used for their prediction model development are Error Correction Neural Network (ECNN) and linear regression. For estimating prediction accuracy, they used the cross-validation technique and several metrics such as Mean Absolute Percentage Error (MAPE), Root Mean Squared Error (RMSE) etc. They concluded that neural network approach is more accurate compared to linear regression as resulted from prediction evaluation metrics.

Hu *et al.* [53] present KSwSVR, a system for fine-grained VM resource allocation based on resource demand prediction. It makes prediction of the VM resource demand multiple-steps ahead in the future, based on the combination of an improved version of Support Vector Machine (SVM) and a Kalman filter. Based on resource demand prediction they allocate the right amount of CPU resource to keep costs and SLA violations to minimum. To increase the prediction accuracy, they applied an improvement to the basic SVM algorithm by giving more weight to most recent time series data. To further increase the prediction accuracy, they applied a Kalman filter for smoothing the noise of resource usage data but still estimating the main trend. They experimentally compared KSwSVR with other prediction methods such as AutoRegressive (AR) model, Back-Propagation Neural Network (BPNN) and standard SVM, showing that their approach achieves higher prediction accuracy.

The proposed proactive VM resource allocation solutions have the drawback that they predict demand time series of only one resource or of multiple resources separately, without taking into account cross-correlation between resources. This cross-correlation is especially true for multi-tier application, where there are interdependencies between tiers of the same application. The machine learning proactive resource allocation approach proposed in this dissertation makes prediction of multiple resource demand time series simultaneously, taking into account cross-correlation between resources of multiple VMs of same multi-tier application. This results in improved prediction accuracy and resource allocation decisions.

2.4 Summary

This chapter has been divided into two large parts. The focus of the first part is to present theoretical background on which this dissertation's work is based upon. Definitions and technical descriptions have been given for concepts such as cloud computing, virtualization, supervised machine learning, artificial neural networks, support vector machines, feed-back control systems and utility function optimization. For each concept introduced, basic algorithms and supporting techniques have been described in the required detail for understanding the rest of the dissertation. The focus of the second part is to present works related to VM resource allocation solutions in cloud computing. The related work has been divided into different groups depending on the VM resource allocation mechanism used. Three VM resource allocation mechanisms have been considered: VM live migration, vertical scaling VM resource allocation, and horizontal scaling VM resource allocation. Work related to proactive VM resource allocation using resource demand prediction has been presented. For each related work, a short description has been given, and at the end of a group of related works, a comparison with this dissertation's work has been given.

3 Utility-based Virtual Machine Load Balancing and Consolidation

3.1 Introduction

Live migration of VMs offers the possibility to provide resources without interrupting running services during migration, which is important for interactive services with particular quality of service (QoS) requirements. However, this flexibility comes with increased complexity in resource management. Consequently, the main challenge is finding an adequate approach to dynamically allocate resources to VMs while taking QoS requirements and operating costs into account.

A promising approach to solve this cost-performance trade off is a utility function. In the field of economics, a utility is a scalar value to express the satisfaction gained from a set of goods or services. In this chapter, a two-tier resource manager that maximizes an adequate utility function to dynamically allocate resources to VMs such that QoS constraints are satisfied and operating costs are minimized is presented. This work explicitly focuses on the CPU as the resource to be allocated, but the approach can straightforwardly be extended to consider other resources such as memory and network bandwidth. The proposed two-tier resource manager consists of local node controllers and a global controller. The local node controllers dynamically allocate CPU shares to VMs with the aim of maximizing a local node utility function by giving higher CPU shares to VMs with higher priorities and thus with higher revenues for a cloud provider. The global controller periodically gets the CPU requirements of every VM from the local node controllers, and decides to migrate particular VMs to

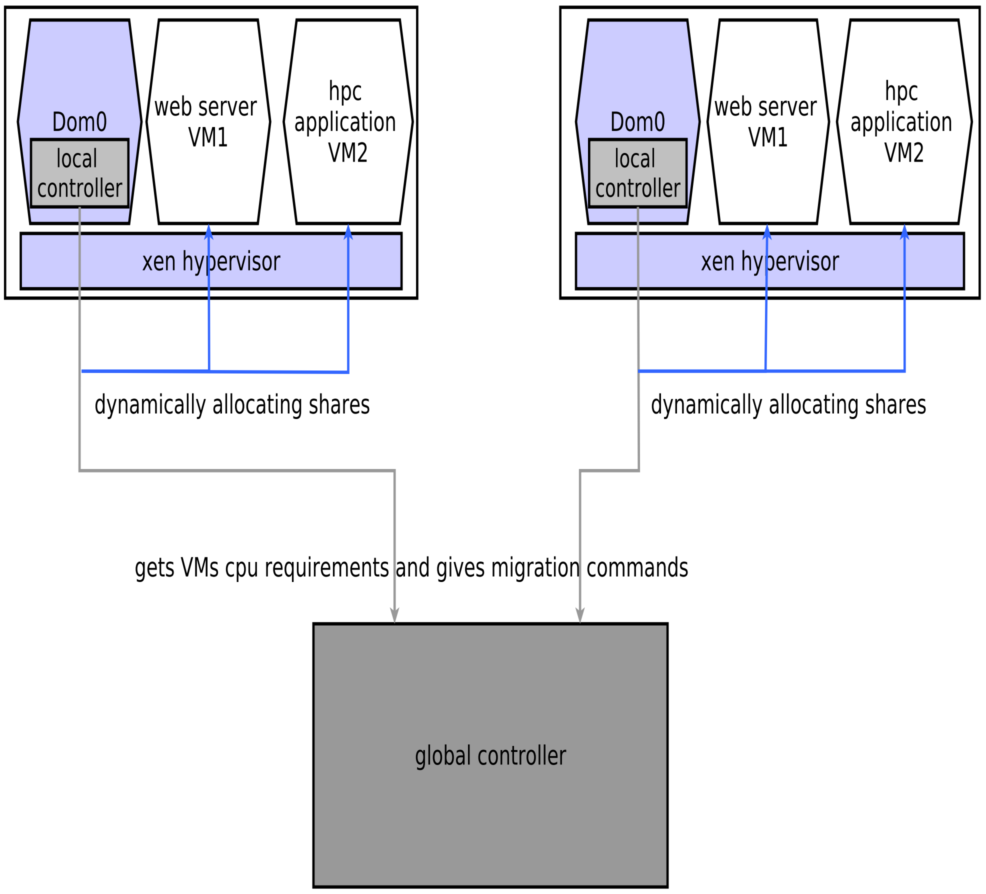


Figure 3.1: Resource manager architecture

other physical machines based on a heuristic algorithm that is aimed at maximizing a global system utility function by considering machine load criteria for VM consolidation. Maximizing the global system utility function through optimized local CPU share settings and global live migrations means finding a proper trade off between cost and performance. The novelty of the proposed approach is to consider VM live migration as an important resource allocation mechanism and to present a new heuristic algorithm for increasing the utility of a cloud provider. Experimental results show that the approach is beneficial for controlling the cost-performance trade off in a virtualized environment.

First the architecture, the utility function, the algorithms of the resource manager and stability issues are described. Then, implementation issues and experimental results are discussed. The chapter is concluded with a summary of the approach. The work described in this chapter has been published in [80].

3.2 Utility-based Resource Allocation

This section presents the architecture of the utility-based resource manager, the utility function model, the local controller, the global controller and stability issues.

3.2.1 Resource Manager Architecture

The resource manager manages an IaaS cloud environment consisting of several VMs running on top of physical machines. It is assumed that the physical machines are homogeneous and the environment offers the possibility to live migrate VMs from one physical machine to another one. The architecture of the resource manager is shown in Figure 3.1. It is composed of local node controllers responsible for maximizing a local node utility through dynamic CPU share allocation and a global controller responsible for maximizing a global system utility through live migrations.

3.2.2 Utility Function Model

In this work, an environment is considered where cloud consumers run their applications on VMs offered by a cloud provider and pay for the virtual resources (in this case CPU resources) they get. The cloud provider tries to maximize the profit, i.e., the difference between the amount of money it gets from the consumers and the operating costs of the infrastructure. A VM utility function is defined to represent the amount of money paid by the consumer for using the VM for a given control interval. A local node utility function is defined to represent the profit of the cloud provider generated by a single node for a given control interval and the global system utility function to represent the profit of the cloud provider produced by the entire system for a given control interval.

The VM utility function is defined as a linear function of the CPU resources the VM of a consumer gets from the cloud provider. A VM utility function can also be a function of some performance metrics stated in Service Level Agreements (SLAs) - a written contract signed between the consumer and the cloud provider that formally defines service levels. Since other works [117, 21] have shown the suitability of using linear utility functions to allocate resources to VMs, also in this work are used linear utility functions combined with classic control theory. More concretely, the utility function U_i for VM_i is given as:

$$U_i = \alpha_i * S_i \tag{3.1}$$

where α_i is the amount of money paid per unit of CPU resource (i.e., 1% of CPU utilization) allocated (e.g., $\alpha_i = 4\$/1\%$ of CPU), S_i is the CPU resource allocated (i.e., CPU share) to VM_i for a given control interval measured in % of CPU utilization. The rationale behind this utility function is that the more

CPU resources a VM gets, the more money the consumer has to pay to the cloud provider. QoS objectives are taken care of by a local feed-back controller and a live migration mechanism that automatically determine the required resources to be given to VMs to meet them. This model of cloud computing is different from that of Amazon EC2 where consumers pay per-instance per hour, where each VM instance has a fixed size chosen from the set of small, large, and extra large sizes; in the model of Amazon EC2, there is no support for fine-grained CPU allocation according to the load of the machines.

The local node utility function N_j for node j is given as the sum of the VM utilities hosted on the node minus node cost:

$$N_j = U_{j1} + U_{j2} + ... + U_{jm} - C_j \tag{3.2}$$

where m is number of VMs hosted on the node, U_{ji} is the VM_i utility on node j and C_j is the cost of the node.

The global system utility function U_g is given as the sum of all local node utilities N_j:

$$U_g = N_1 + N_2 + ... + N_n \tag{3.3}$$

where n is the total number of physical nodes. Inserting equation (2) into (3), we get another form of global system utility function

$$U_g = (U_{11} + U_{12} + ..) - (C_1 + C_2 + .. + C_n) = U_v - C \tag{3.4}$$

In Equation (3.4), the global system utility function expresses the cost-performance trade-off, where the sum of the VM utilities represent performance (i.e., more VM utility means more resources and improved performance) and the sum of node costs represents the total cost of the cloud provider.

The aim of the resource manager is to maximize the global system utility function through live migrations and adequate local resource allocation. In the following sections, the algorithms implemented by the local and global controllers for this purpose will be discussed.

3.2.3 Local Node Controller

The local node controller is responsible for dynamically allocating CPU resources to different VMs to maximize the local node utility function. Its structure is shown in Figure 3.2. It is composed of a monitoring component, a feed-back controller and an arbitrator. The monitoring component measures the average CPU utilization of VMs in every control interval. The control interval can be set by the administrator depending on how quickly the local node controller should react to the load. Its value also depends on the time taken by the arbitrator (in this work it is around 1 second). The control interval is set the to 8 seconds, which in practice provides an adequate adaptation to the load. The feed-back controller [118] uses control theory to determine the

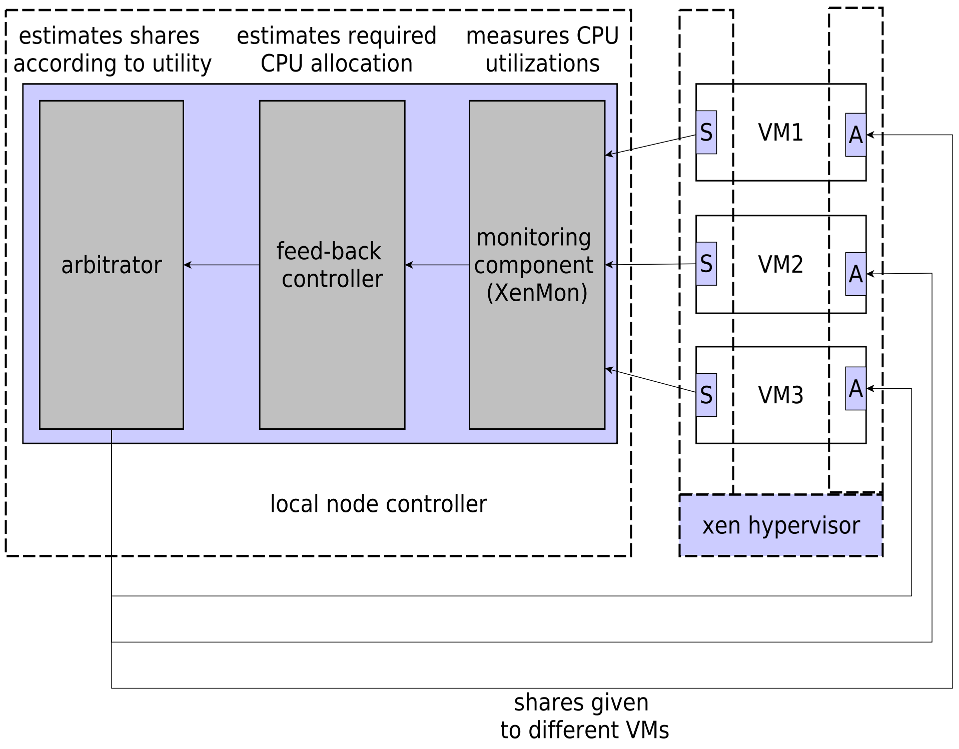

estimates shares according to utility
estimates required CPU allocation
measures CPU utilizations
arbitrator
feed-back controller
monitoring component (XenMon)
local node controller
S
VM1
A
S
VM2
A
S
VM3
A
xen hypervisor
shares given to different VMs

Figure 3.2: Local node controller

required CPU allocations to be given to each VM in order to keep the CPU utilization at a desired level to maintain QoS objectives. The arbitrator takes CPU allocation requests for all VMs from the feed-back controller, and according to the utility of each VM, sets the CPU shares using an algorithm described below in order to maximize the local node utility function.

The arbitrator algorithm works as follows. If the sum of required CPU allocations of all VMs does not exceed the total CPU capacity of 100% (for a single CPU), then the arbitrator sets the CPU shares for every VM equal to its required CPU allocation. In this case, the local node utility function is maximized because each VM gets its required allocation. If the sum of the required CPU allocations of all VMs exceeds the total CPU capacity, then the arbitrator maximizes the local node utility function by first satisfying requests of VMs that offer a higher utility per unit of CPU resource and setting their CPU shares equal to their required CPU allocation. Then, in decreasing order of utility, other requests are satisfied. If there is not enough CPU capacity to satisfy all requests, lower utility VMs will never get CPU resources. To solve this problem, every VM is always given a minimum of CPU share which can be stated in SLA contracts, in this work 5% of the CPU, despite their low utility values. Then, the remaining CPU capacity is given to first satisfy the requests of higher utility VMs, then lower utility VMs. Of course, some VMs will get a CPU share that is lower than their required allocation, but here live migration is used to increase the global system utility by moving VMs to some underloaded CPU. This algorithm supports differentiated-service because higher priority VMs have a higher utility and therefore get a higher CPU share.

3.2.4 Global Controller

The global controller runs on its own node, and in each control interval queries the local controllers for the CPU requirements and CPU shares of all VMs. The control interval can be set by the administrator depending on how agile the virtual infrastructure should be. Small values mean that the manager reacts quickly to load, and large values mean that it reacts slowly. This value also depends on the time taken by global controller (in the implementation and experimental setup: 2 seconds). Based on this value, the global controller interval is set to 20 seconds, which provides a sufficiently agile environment. This interval does not include live migrations or times for turning on new nodes, so this is the minimum time the global controller waits. If there are live migrations or new nodes turned on, the global controller waits until all live migrations are finished and new nodes are turned on. In its general form, the problem of finding the mapping of VMs to physical machines that maximizes a utility function is a multi-knapsack problem in which items packed in knapsacks are elastic in size. Since this problem is NP-hard, it is devised a heuristic algorithm to solve it. This algorithm determines if the global system utility can be increased by live migrating VMs, and if this is the case, it suggests a list of live migrations that achieve a maximum increase of the global system utility.

There are two cases when the global system utility is increased: a) if a VM is migrated from an overloaded node where its CPU requirements are not satisfied to an underloaded or a newly turned on node; b) if all VMs are migrated from an underloaded node to some other node to eliminate node cost by turning off the node. Based on these two observations, the algorithm used by the global controller works as follows.

First, based on CPU shares and required allocations of all VMs, the nodes are divided into three groups: overloaded, underloaded and unstable. A node is considered as overloaded if the total required CPU allocations of VMs exceed the total CPU capacity for a sustained number of consecutive local controller intervals (which is set to 3 as a default value), and there is more than one VM running on the node. An underloaded node is one where the total required CPU allocations of VMs does not exceed the total CPU capacity for a sustained number of consecutive local controller intervals (which again is set to 3 as a default value). A node is unstable if it does not satisfy any of the above conditions, which means that the node oscillates between the overloaded and underloaded states. To maintain the stability of the algorithm, unstable nodes are not included in migration decisions.

Then, the algorithm checks if there are any overloaded nodes in order to improve the global system utility through balancing the load. If there are overloaded nodes, for each of them the algorithm picks the lowest utility VM and simulates a migration of it to every underloaded node and also to a newly turned on node. The logic behind picking the lowest utility VM is to permit degraded performance for low priority VMs during live migration. For each overloaded node, the algorithm selects as the destination node of a migration the node that gives the highest increase in global system utility (if there is any increase) and marks the VM as moved. If there is no utility increase, the migration will not be included in the list. At the end of this procedure, the algorithm will produce a list of migrations of a VM from each overloaded node to some underloaded one.

In case there are no overloaded nodes, the algorithm selects the under-loaded node with the least local node utility, which intuitively will lead to the highest increase in global system utility by migrating all its VMs and turning off the node. A VM is selected on this node and its migration to every other under-loaded node is simulated. As the destination of migration, the node that gives the highest increase in global system utility is selected, and the VM is marked as moved. This is repeated for every VM on this node. Finally, a check is made if the global system utility is increased by all these simulated migrations and turning off the node. If the global system utility is increased, a list of migrations for all VMs of the under-loaded node in question to other different under-loaded nodes will be produced. After all migrations are done for load balancing or consolidation purposes, improving the global system utility for the current interval, the whole algorithm is repeated in the next control interval. If the algorithm does not produce any list of migrations, because there is no increase in global system utility, then the global controller has nothing to do for the current control interval and re-initiates the algorithm in the next interval.

The procedure of finding an increase of the global system utility by migrating a VM from one node to another one is as follows. First, the utility of the source node denoted as S_b (S : source b : before removal) is evaluated, then the removal of the VM from the source node is simulated and the CPU shares of this new state are recalculated using the same algorithm used by the local node controller, and the utility of the source node after the VM removal denoted as S_a (a : after removal) is evaluated. Then, the algorithm evaluates the utility of the destination node denoted as D_b (D : destination b :- before addition), simulates the addition of the VM on the destination node, recalculates the CPU shares of the new state and evaluates the utility of the destination node after the addition of VM, denoted as D_a. After these calculations, the increase I of the global system utility obtained by this migration is given as:

$$I = (D_a - D_b) - (S_b - S_a) - [NodeCost] \tag{3.5}$$

Thus, the increase of the utility of the destination node due to the addition of a VM, denoted as $(D_a - D_b)$, should be greater than the decrease of the utility of the source node due to a VM removal, denoted as $(S_b - S_a)$, in order to get an increase of the global system utility. The optional $NodeCost$ term is added if the destination node is a newly turned on node, in which case it should be taken into account the node cost in the utility increase, while in the calculations above the utility of the destination node before VM addition is just the node cost. In the consolidation case, the sum of the increases of global system utility for all VM migrations from the node to be turned off to other nodes is evaluated. To this sum, the cost of a node to be turned off is added, and it is tested whether there is any increase. Thus, the value to be tested is:

$$I = I_1 + I_2 + ..I_i... + I_m + NodeCost \tag{3.6}$$

where I_i is the global system utility increase for migrating VM_i, and $NodeCost$ is the cost of the node to be turned off.

3.2.5 Stability Issues

Since the load can change frequently, VMs might unnecessarily migrate from one node to another node frequently. To address these problems, two approaches are followed to balance stability with agility, as described below.

First, since the global system utility depends on the VM utilities (see Equation (3.4)), and a VM utility depends on CPU shares that change dynamically according to the load, the global system utility also changes according to the load. This should be taken into account when checking for an increase of global system utility, because it is possible to see a utility increase when simulating migrations, but this can be a "fake increase" in the sense that is caused by load decrease/increase in the current control interval. In the next control interval, the load could change in reverse order, making the migrations useless and the system unstable, forcing again to perform migrations to put VMs back in their initial locations. To eliminate this instability, in each control is estimated the standard deviation of the last 5 values of the global system utility and migration decisions are made only if the increase of the global system utility is greater than the standard deviation. This means that migrations are performed only if the increase of the global system utility is really caused by migrations and not caused by load changes. Thus, in the global controller algorithm, all statements about an increase of global system utility in fact mean an increase of global system utility greater than the standard deviation.

Second, there could be spikes of increased or decreased load in the form of noise just for a few seconds. In this case, it is not preferable to perform migrations in order to eliminate the overhead of unnecessary migrations. Since the VMs' CPU requirements can also have spikes for short periods, their values are smoothed by passing them through a stability filter [64] before giving them to the global controller. In this case, any spikes of load changes are filtered out, and the global controller makes migration decisions based on a smoothed view of the load. This makes the system resistant to short spikes, and it reacts only if the changed load persists for a longer period. Thus, unnecessary migrations due to frequent changes of the utility value are eliminated. A thorough formal analysis of the stability of the method is an interesting area of future research.

3.3 Implementation

The local controller is implemented as a two-threaded application in Python running in Dom0 of the Xen system. The first thread listens for requests from the global controller and sends CPU resource consumption and requirement values. The second thread implements the monitoring component, the feed-back controller and the arbitrator. Since it is used Xen, the monitoring component is based on the XenMon tool [43]. The arbitrator uses the Pyxen binding library to interface the Xen scheduler for changing CPU shares. In total, the local controller implementation consists of 800 lines of Python code. The overhead of the local controller is comparable to that of XenMon, adding only 1% of CPU utilization for Dom0.

The global controller is also implemented as a Python application consisting of 530 lines of code, and it initiates live migrations using the libvirt interface to the Xen platform. To simply evaluate the functionality of the global controller, new physical machines are not really turned off/on . Instead, the global controller waits some time to simulate the startup or shutdown of nodes, and the local controllers of the supposed turned off nodes do not respond to the request of the global controller. Since it is used Xen, there is a privileged management VM called Dom0 that requires some CPU share. Since it serves other VMs, its required allocation should be satisfied first; therefore, it is given the highest utility of all VMs. Furthermore, since Dom0 does I/O work on behalf of other DomU VMs, this is taken into account by including a part of the Dom0 share that corresponds to a VM in the calculation of its utility function. The part of the Dom0 share corresponding to a VM is determined using the rate of the number of I/O pages exchanged with Dom0 and the total I/O pages exchanged. This information is obtained from XenMon. Thus, the modified VM utility function becomes

$$U_i = \alpha_i * (S_i + S0_i) \tag{3.7}$$

$S0_i$ is part of the CPU share of Dom0 corresponding to VM_i. It is also used the Xen credit scheduler in work-conserving mode to set shares given by the arbitrator, since it has been shown that is more efficient than capped mode.

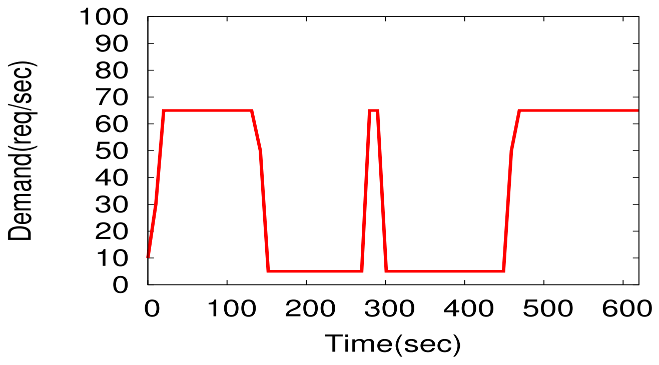
3.4 Experimental Set-up and Evaluation

The experimental setup is composed of 5 physical machines connected over a Gigabit/s Ethernet: three physical machines run consumer VMs, one physical machine runs the global controller and the last physical machine runs a load generator. The physical machines have one 1.80GHz Intel Pentium 4 CPU with 512MB of RAM and 55GB of hard disk. They run Xen 3.0.3-1 and Debian/GNU

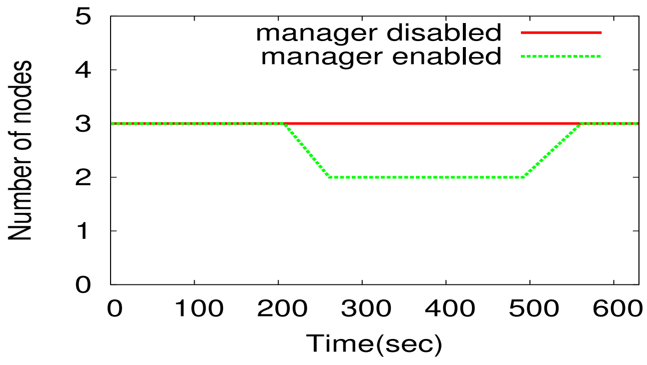
Linux 5.0 OS with the 2.6.18-5-xen-686 kernel. The machine with the global controller also runs an NFS server to support live migrations based on shared storage. Each VM has 128MB or 64MB of RAM, depending on the experiment. Two different applications are run on the VMs to test the resource manager: one application is an Apache 2.2.9 web server serving CPU intensive PHP scripts, the other application is the HPL benchmark that solves dense linear systems. Requests to the web server are generated with httperf 0.9.0 crafted in Python scripts to generate desired request patterns.

3.4.1 Experiment 1: Improving Global System Utility

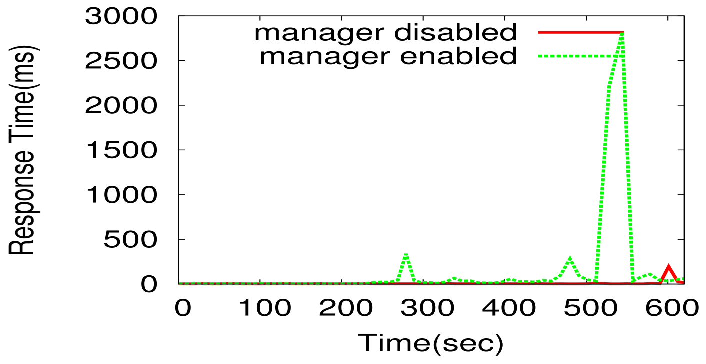
The goal of this experiment is to evaluate the approach in two cases: a) when the global controller is disabled, and b) when it is enabled to show the resource manager's ability to improve the global system utility. In this experiment, five physical machines are used, PM1, PM2, PM3, PM4, PM5. PM1, PM2, PM3 each hosts a virtual machine VM1, VM2, VM3, respectively. PM4 runs the global controller and PM5 runs the web request generator. VM1 and VM2 each run MPI processes of the HPL benchmark. VM3 runs the Apache web server. VM1 and VM2 have utility 4 per unit of CPU, while VM3 has utility 8 per unit of CPU and and node cost is set to a constant value of 60. At the start of the experiment, each VM is running on its own physical machine.



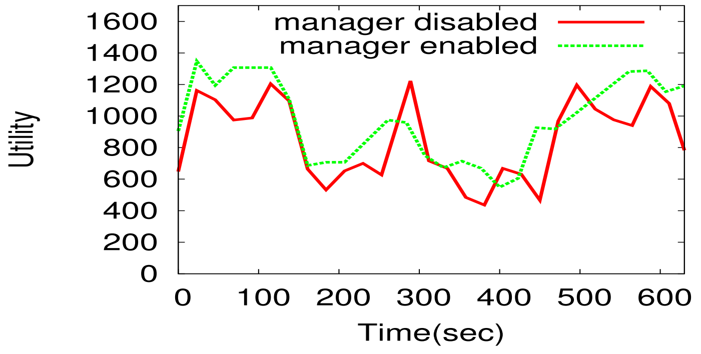
(a) demand for web server in VM3



(b) number of active nodes



(c) VM3 web server response time



(d) global system utility

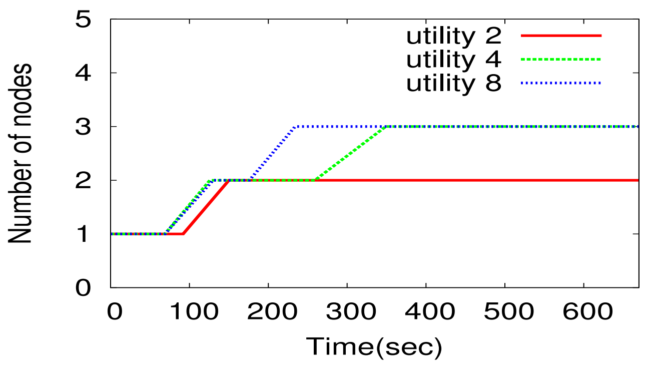
Figure 3.3: Experiment 1: Results

Figure 3.3(a) shows generated request patterns for the web server. In the case when the global controller is disabled, each VM runs in its own physical machine for the whole duration of the experiment, providing maximum achievable performance but higher operating cost because all three machines are active. In the case when the global controller is enabled, after 55 seconds from the moment that the web server load decreased at second 145, the controller evaluates that the global system utility can be increased by consolidating the web server and shutting down the node, so it initiates at second 200 the live migration of VM3 to the physical node PM2 and shuts down PM3. This is also shown in Figure 3.3(b) where at second 200 the number of physical machines is reduced to 2. In Figure 3.3(a), it is seen a spike of increased load for VM3 at second 280 for a short period of time in which case the controller does not initiate any migration actions, showing the ability of the controller to resist to short load spikes, eliminating unnecessary migration overhead. At second 450, there is an increased load for VM3 persisting for a long period, and the controller sees an improved global system utility by distributing the load, so it turns on PM3 and live migrates VM2 to it. For the rest of the experiment, the load on VM3 remains high, so all three physical machines remain active. It is evident that there are cost savings for the global controller, because over the whole experimental time, it uses less physical nodes, as shown in Figure 3.3(b).

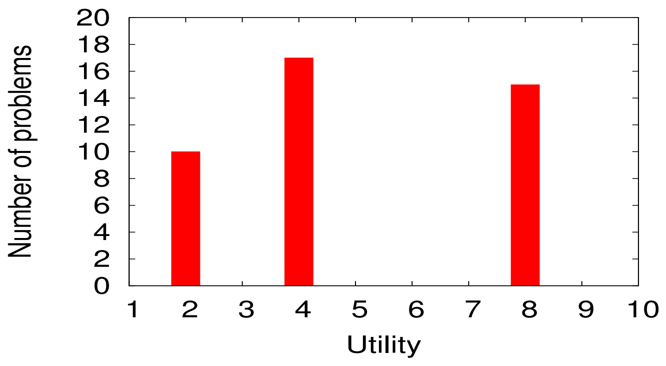
As Figure 3.3(c) shows, the response time of the web server for both cases is the same for most of the experimental time. There is only a high response time for a duration of 40 seconds, caused by the overhead of live migration. This result shows that in general the resource manager is able to reduce cost while maintaining performance at an acceptable level. Figure 3.3(d) shows the global system utility for both cases where in the case of a disabled global controller, local controllers are enabled. It can be seen that the case when the global controller is enabled achieves a better global system utility especially at time interval [150:470], due to the reduced node cost resulting from shutting down one physical machine. This shows the benefits of including live migration to improve the global system utility.

3.4.2 Experiment 2: Controlling Cost-Performance Tradeoff

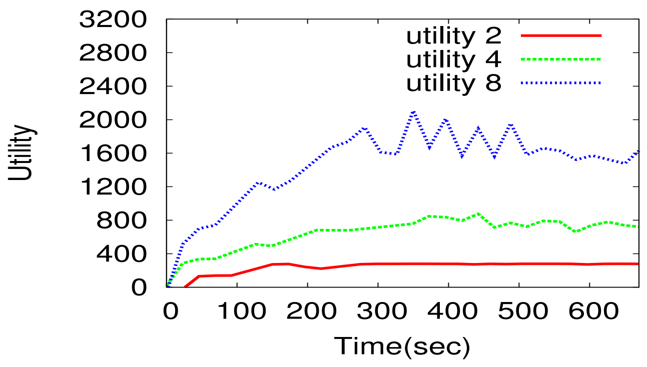
The goal of this experiment is to show the ability of the resource manager to provide a controllable cost-performance trade-off. In this experiment, four VMs are used to run the HPL benchmark composed of four MPI processes running each in its own virtual machine. Three cases are considered. In the first case, an utility of 2 per unit of CPU is given to the VMs, in the second case an utility of 4, and in the third case an utility of 8, while again node cost is 60. The experiment is started for all three cases by running four VMs on one physical node and turning on new nodes as decided by the global controller. Figure 3.4(a) shows the number of active nodes over time. In all three cases, the global controller sees at some point in time that the global system utility can be improved by turning on a new node and migrating a VM to it. The times when



(a) active of nodes for different utilities



(b) problems solved for different utilities



(c) global system utility for different VM utilities

Figure 3.4: Experiment 2: Results

new nodes are needed and their maximum number are different for the different cases. The utility 2 case uses less nodes over experimental time, a maximum of 2, followed by utility 4 and utility 8. This is because when the VM utility is low, the cost of turning on a new node cannot justify a global system utility increase achieved by migrating the VM to it. This shows that consolidation and load balancing aggressiveness can be controlled by changing VM utility or node cost. Figure 3.4(b) shows the number of problem instances solved by the HPL benchmark for the three cases above. The utility 2 case solves about half of the problem instances solved by the utility 4 and utility 8 cases. This is because the utility 2 case uses less nodes to solve the problem. While the utility 4 and utility 8 cases in general solve the same number of problems because both of them use a maximum of 3 nodes, it is seen a small difference in this number but it is believed it is just a random event possibly happening because in the utility 8 case, the live migration of a VM could have taken longer. These results show the cost-performance trade-off. If the consumer of a VM has a low utility per unit of CPU, it gets poor performance, leading to reduced resource costs. An increased VM utility means increased performance and increased resource costs. More importantly, this trade-off can be controlled by changing the VM utility or cost coefficients. Figure 3.4(c) shows global system utility over time for the three cases where in each case it is seen a global system utility improvement, showing the resource manager's ability to increase global system utility through live migration and turning on an adequate number of nodes.

3.5 Summary

In this chapter, an approach for dynamically allocating CPU resources to VMs in IaaS clouds has been presented, taking into account QoS objectives and operating costs. To solve this problem in a flexible manner, a two-tier resource manager has been developed based on local node utility functions and a global system utility function that considers VM live migration as an important resource allocation mechanism. Experimental results have shown the ability of the resource manager to adapt to changing load, to improve the global system utility using VM live migration, and to maintain the performance at acceptable levels while reducing costs. The experiments also have demonstrated the ability of the resource manager to control the cost-performance tradeoff by changing VM utilities or node costs. The method can preferably be used to manage VMs in IaaS clouds running interactive applications (e.g., a web server), but it is also suitable for other applications based on VMs, as indicated by the good experimental results.

There are several areas of future work. For example, the resource manager could be extended by considering other utility function models or by including other factors, such as a reliability cost measured at runtime that represents the reliability of a node. Other future work could be the inclusion of memory, disk and network as dynamically managed resources to increase the global system utility. Finally, it would be interesting to consider a distributed resource manager for large cloud computing environments composed of hundred of nodes and thousand of VMs, where the properties of eliminating a performance bottleneck and a single point of failure are important factors to consider.

4 Allocation of Multiple Types of Resources to Virtual Machines Using Control Theory

4.1 Introduction

There has been a lot of research on how to dynamically allocate particular types of resources to VMs, but approaches on how to manage multiple types of resources are lacking. The motivation for a combined management approach is that different types of applications or the same application in different execution phases require different types of resources, e.g., some applications are disk I/O bound, while other applications are CPU bound, such that the overall resource utilization can be improved by finding a proper match between application characteristics and resource types.

In this chapter, an approach for dynamically managing different types of resources, i.e., CPU, memory, disk, and network, by maximizing a utility function, is presented. In the field of economics, utility is a scalar value to express the satisfaction gained from a set of goods or services. In this work's context, a utility function is used to drive the allocation of resources to VMs such that quality of service (QoS) requirements are satisfied and the financial profit of the cloud infrastructure provider is maximized. This approach consists of resource managers situated in each physical machine of the cloud infrastructure that dynamically manage resources of the VMs running on a physical machine. For each type of resource to be managed, the manager contains a) a monitoring component to measure resource usage, b) a feed-back controller to allocate resource shares to each VM such that QoS requirements are met, and c) an arbitrator that attempts to maximize a node utility function by considering the utility values of the VMs that run on it. The novelty of the proposed approach is to consider multiple types of resources and a new arbitration algorithm trying to maximize a utility function for the dynamic allocation of multiple types of resources. Experimental results indicate that the approach is beneficial for efficiently allocating different types of resources to VMs in Infrastructure-as-a-Service clouds.

In the following sections, the utility function model, the architecture and the algorithm of the resource manager are described. Then, implementation issues, the experimental set-up and evaluations are presented. The chapter is concluded with a summary of the approach. The work described in this chapter has been published in [81].

4.2 Utility-driven Resource Manager

This section presents the utility function model, the architecture and the algorithm used by resource manager for the allocation of resources.

4.2.1 Utility Function Model

In this work, an environment is considered where cloud consumers run their applications on VMs offered by a cloud provider and pay for the performance characteristics of the VMs they use. The cloud provider tries to maximize the profit, i.e., the difference between the amount of money obtained from the consumers and the operating costs of the infrastructure. A VM utility function is defined to represent the amount of money paid by a consumer for using a VM for a given control interval, and a node utility function to represent the profit of the cloud provider generated by that node for a given control interval. The profit of the cloud provider produced by the entire cloud infrastructure for a given control interval is given by the sum of all node utilities. The VM utility function is defined as a function of some performance metric (e.g., response time) and the VM resources the consumer gets from the cloud provider. The performance metric and the utility function can be stated in a Service Level Agreement (SLA) – a signed written contract between the consumer and the cloud provider that formally defines service levels. More concretely, the utility function U_i for VM_i studied in this work is given by:

$$\begin{aligned} U_i\left(R, resp\_time\right) = &\left(\alpha_i * R_{cpu} + \beta_i * R_{net} + \gamma_i * R_{disk}\right. \\ &\left. + \omega_i * R_{mem}\right) * f_i\left(resp\_time\right) \end{aligned} \tag{4.1}$$

where α_i is the amount of money paid per unit of CPU resource consumed (e.g., $\alpha_i = 4\$/1\%$ of CPU), R_{cpu} is the mean CPU resource consumed by VM_i for a given control interval measured in % of CPU consumption. β_i is the amount of money paid per unit of network bandwidth consumed and R_{net} is the mean network bandwidth consumed by VM_i for a given control interval measured

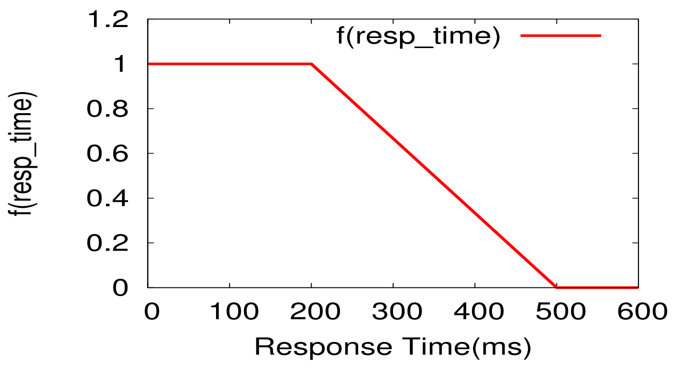


Figure 4.1: Utility performance function

in Mbits/sec. γ_i is the amount of money paid per unit of disk bandwidth consumed and R_{disk} is the mean disk bandwidth consumed by VM_i for a given control interval measured in MBytes/sec. ω_i is the amount of money paid per unit of memory consumed and R_{mem} is the memory consumed by VM_i for a given control interval measured in MBytes. $f_i\,(resp\_time)$ is a function of the response time stated in the SLA. The particular function used in this work is shown in Figure 4.1: when the response time is between zero and 200 ms, the utility has a maximum value (i.e., a value of 1), when the response time is between 200 ms and 500 ms, the utility is reduced in a linear fashion, and when the response time is above 500 ms, the utility becomes zero. The rationale behind this VM utility function is that the amount of money the consumer has to pay depends on the QoS and the resources a consumer gets from the cloud provider. Thus, when he gets QoS outside the boundaries specified in the SLA, he has to pay nothing, but when there are two VM consumers that get the same QoS inside specified boundaries, the consumer that requires more resources to achieve this performance should pay more money. This model of the utility function offers flexibility in giving different resources different priorities for different applications.

A node utility function N_j for node j is given by the sum of the VM utilities hosted on node j minus node cost:

$$N_j = U_{j1} + U_{j2} + ... + U_{jm} - C_j \tag{4.2}$$

where m is number of VMs hosted on the node, U_{ji} is the VM_i utility on node j and C_j is the cost of the node. The aim of the resource manager is to maximize the node utility function. This means allocating resources such that the values of the utility functions of all VM on that node are maximized, assuming that the node cost is constant. Maximizing the node utility functions of all nodes means maximizing the total profit of the cloud provider.

In the following sections, a simple algorithm implemented by the resource manager to achieve this is presented. This model of cloud pricing is different from that of Amazon EC2 where consumers pay per-instance per hour, where each

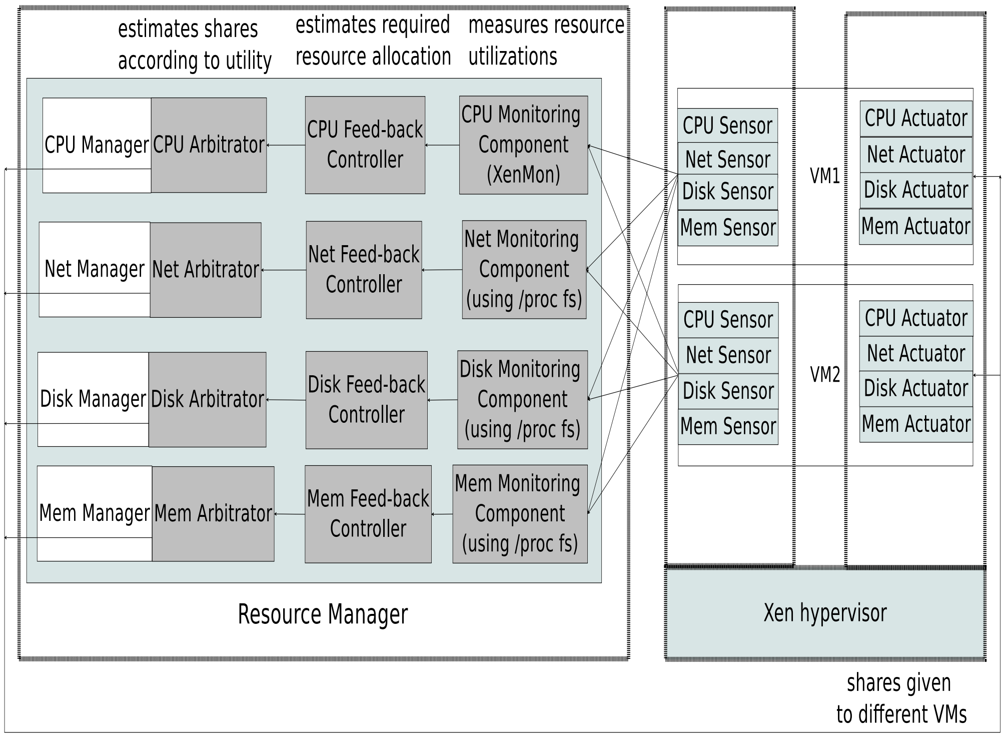

estimates shares according to utility
estimates required resource allocation
measures resource utilizations
CPU Manager
CPU Arbitrator
CPU Feed-back Controller
CPU Monitoring Component (XenMon)
Net Manager
Net Arbitrator
Net Feed-back Controller
Net Monitoring Component (using /proc fs)
Disk Manager
Disk Arbitrator
Disk Feed-back Controller
Disk Monitoring Component (using /proc fs)
Mem Manager
Mem Arbitrator
Mem Feed-back Controller
Mem Monitoring Component (using /proc fs)
Resource Manager
CPU Sensor
Net Sensor
Disk Sensor
Mem Sensor
VM1
CPU Actuator
Net Actuator
Disk Actuator
Mem Actuator
VM2
Xen hypervisor
shares given to different VMs

Figure 4.2: Resource manager architecture

VM instance has a fixed size chosen from the set of small, large, and extra large sizes; in the model of Amazon EC2, there is no support for fine-grained resource allocation according to the load of the machines.

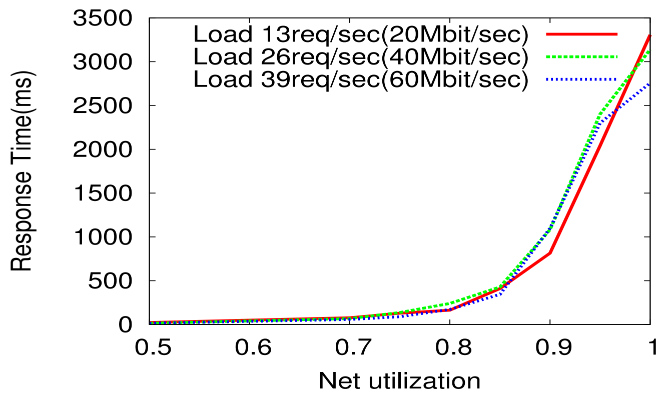
4.2.2 Resource Manager Architecture

The resource manager is local to each node, and its aim is to dynamically allocate resources to different VMs in order to maximize the node utility function. Its architecture is shown in Figure 4.2. It consists of four main components: CPU manager, Network manager, Disk manager, Memory manager; each of them is independent of the others and is responsible for managing a particular type of resource. Each manager consists of a monitoring component, a feed-back controller and an arbitrator. The monitoring component measures the average resource consumption of VMs in every control interval. The control interval can be set by the administrator depending on how quickly the resource manager should react to the load. Its value also depends on the time taken by the arbitrator (in this case it is less than 1 second). The control interval is set to 5 seconds, which in practice provides an adequate adaptation to the load. The feed-back controller [118] uses control theory to determine the required resource allocations to be given to each VM in order to keep the utilization at a desired level to maintain QoS objectives. By utilization it is meant the fraction of the amount of resource consumed and the amount of resource allocated to a VM. The arbitrator receives resource allocation requests for all VMs from the feed-back controller, and according to the utility function of each VM for that resource sets resource shares using an algorithm described below in order to maximize the node utility function. This architecture of independent managers and corresponding arbitrators for each type of resource offers flexibility in resource management by giving different priorities to different resources for different applications.

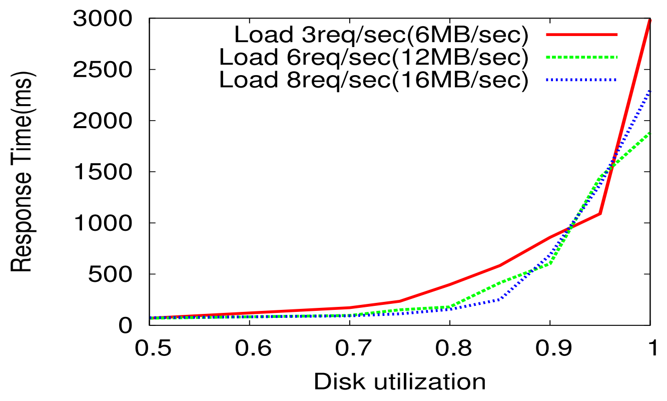
As shown in other proposals [49], QoS of interactive applications such as web servers can be satisfied by keeping CPU or memory utilization to a desired level using feed-back control theory. To motivate the use of such a feed-back controller for satisfying QoS of an interactive application with respect to two other types of resources such as disk and network, are performed the following experiments to find the relationship between application performance and its resource utilization level. The experiments are based on an Apache web server in a Xen virtual machine running Linux in one physical machine and load generated with httperf installed in another physical machine connected to the first one via a 100 Mbit/sec Ethernet.

In the first experiment, the relation between performance and network bandwidth utilization is tested. To limit ingoing and outgoing network bandwidth allocation of the web server VM, it is used the Traffic Control (tc) utility of the iproute2 [48] collection of utilities. To create network load, a workload with exponentially distributed inter-arrival times lasting for 60 seconds and requesting a 200 KB static html file is generated. Three mean request rates are used to generate three different workload intensities (low, normal and high). For each of them, the network bandwidth allocation is changed to provide different utilizations and measured the response time for each utilization. The results of this experiment are shown in Figure 4.3(a). When the utilization is below 0.8, the response time is at acceptable levels for all three workloads, and when the utilization is increased above 0.8, the response time increases. This shows that keeping network bandwidth utilization at a certain level through the use of a feed-back controller provides acceptable QoS for an interactive application such as a web server.

In the second experiment, the relation between performance and disk bandwidth utilization is tested. To limit disk bandwidth allocation to the VM, the dm-ioband I/O bandwidth controller [105] in Linux is used. To create disk bandwidth load, requests are ran against an Apache module created for this purpose, which for each request reads a 2 MB data block (the file pointer moves sequentially from one request to the other) from a file larger than the memory allocated to the VM in order to avoid file system caching. Again, workload with exponentially distributed inter-arrival times is generated lasting for 60 seconds using three mean request rates to generate three different workload intensities (low, normal and high). For each of them, the disk bandwidth allocation is changed and the response time for each utilization is measured. As shown in Figure 4.3(b), when the utilization is below 0.75, the response time is at accept-



(a) Response time vs network bandwidth utilization



(b) Response time vs disk bandwidth utilization

Figure 4.3: Performance vs. utilization for network and disk resources

able levels, and when the utilization is above 0.75, the response time increases. This shows again that also for the disk resource keeping utilization at certain levels provides acceptable QoS, justifying the use of a feed-back controller that regulates disk bandwidth utilization.

4.2.3 Resource Manager Algorithm

In this section, the algorithm used by the resource manager to allocate shares to different types of resources in order to maximize the node utility function is presented. The algorithm is executed by each resource arbitrator to allocate shares for its particular type of resource. It works as follows. If the sum of the resource requirements of all VMs as decided by the feed-back controller does not exceed the total resource capacity, then the arbitrator sets the resource shares for every VM equal to the resource requirements. If the sum of the resource requirements of all VMs exceeds the total resource capacity, then the arbitrator maximizes the local node utility function by first satisfying requests of VMs that offer a higher utility per unit of resource and setting their resource shares equal to their resource requirements. Then, in decreasing order of utility, other requests are satisfied. If there is no sufficient resource capacity to satisfy all requests, lower utility VMs will never get resources. To solve this problem, every VM is always given a minimum of resource share that can be stated in SLA contracts (e.g., 5% in the case of the CPU resource), despite their low utility values. Then, the remaining resource capacity is used to first satisfy the requests of higher utility VMs, then lower utility VMs. This algorithm supports differentiated-service because higher priority VMs have a higher utility per unit of resource and therefore get a higher resource share.

The reason why this algorithm increases the node utility function is as follows. If a resource is not overloaded, i.e., all requirements for a resource can be satisfied, the resource consumption R_i of all VMs is at its maximum, and also the function $f_i\,(resp\_time)$ has a maximum value of 1 because utilization is at the required level. Therefore, according to equations (4.1) and (4.2), this means that the node utility is at its maximum. If a resource is overloaded, i.e., all requirements of VMs for that resource cannot be satisfied, the resource consumption R_i of the VM that offers a higher utility per unit of resource will be higher and the corresponding $f_i\,(resp\_time)$ function will a have higher value because the resource is overloaded, its required allocation will be satisfied first and its utilization will be at the required level. Therefore, according to Equations (4.1) and (4.2), the node utility has a maximum value. This structure of the utility function and the algorithm above offer flexibility in prioritizing allocations not only at the application level, but at a more fine-grained resource level by giving different resources different priorities for the same application. This is not unusual in cloud computing where different applications use different resources and therefore have different priorities on different resources.

4.3 Implementation

The resource manager has been implemented in the Python programming language running in Dom0 of the Xen virtualization platform. To keep the structure of the main manager modular, the resource managers for CPU, network, disk, and memory are programmed as separate procedures that are called one after the other in a loop in the main procedure of the program for every control interval. Since each manager is independent of the other mangers, they could run as separate threads, each managing its own type of resource with its own control interval, but to keep the implementation simple and to use the same control interval, it is decided to implement a single thread approach where the only requirement is that the control interval must be larger than the sum of the execution times of all arbitrators. The main components of each resource manager (monitoring component, feed-back controller and arbitrator) are programmed as separate procedures and called in that order by the main resource manager procedure.

Since Xen is used, the CPU monitoring component is based on the XenMon tool [43]. The network, disk and memory monitoring components are based on /proc filesystem to get resource consumption values. More concretely, the network monitoring component gets from /proc filesystem the bandwidth consumption of the virtual interface corresponding to the VMs to be monitored. The disk monitoring component gets from /proc filesystem the bandwidth consumption of the block tap (blktap) process corresponding to the VMs to be monitored. To measure the memory consumption, a monitoring module inside a VM is installed to get the free and used memory from /proc filesystem. It passes this information through a socket connection to the main memory monitoring component when requested in each control interval.

To set the allocation shares of the arbitrators to different types of resources, the following actuators are used. For CPU allocation, the Pyxen binding library of the Xen hypervisor is used to change shares dynamically and the Xen credit scheduler is used in work-conserving mode (which is more efficient than in capped mode). For network allocation, the tc utility in Linux is used using a Hierarchical Token Bucket (HTB) queueing discipline that allows to shape network bandwidth to and from virtual network interfaces corresponding to VMs. For disk allocation, the dm-ioband I/O bandwidth controller is used that allows limiting disk bandwidth allocation per process, such that it can limit the bandwidth to the blktap process responsible for disk I/O of a VM. The weight-iosize policy of dm-ioband is used. It distributes bandwidth proportional to the weights given to different VMs. The weights given to VMs, which are important when disk bandwidth is overloaded, are calculated as follows. For example, if there is a total maximum disk throughput equal to 24 MB/sec and the arbitrator decides to give one VM an allocation of 16 MB/sec and another VM an allocation of 8 MB/sec, then the weight given to the first VM is calculated as 16/24*100=66.66 and the weight for the second VM as 8/24=33.33, thus the weights are proportional to their required allocation. For memory allocation, the Pyxen binding library of the Xen hypervisor is used by first allocating a maximum of 460 MB to each VM and using the Xen balloon driver capability to change the memory allocation dynamically. In total, the resource manager implementation consists of 1163 lines of Python code.

To generate memory and disk load, an Apache module in the C programming language is created that can receive four different kinds of requests, three related to memory load and one to disk load. The first request related to memory can increase the memory of the Apache module for each request by 5 MB using malloc, the other one can decrease memory for each request by 5 MB using free and malloc, and the third one can access allocated memory randomly for each request. In this way, the memory usage of every VM can change dynamically and memory workload can be generated. To generate disk load, the fourth request to the module is used which reads a 2 MB data block for each request (the file pointer moves sequentially from one request to the other) from a 1 GB file which is bigger than the memory allocated to the VM in order to avoid file system caching. To generate network load, a 200 KB html file that consumes some network bandwidth during transfer is requested from the web server. Finally, to generate CPU load, requests are sent to the web server via a PHP script that executes a loop for increasing some variable and printing it to the web page to consume some CPU cycles.

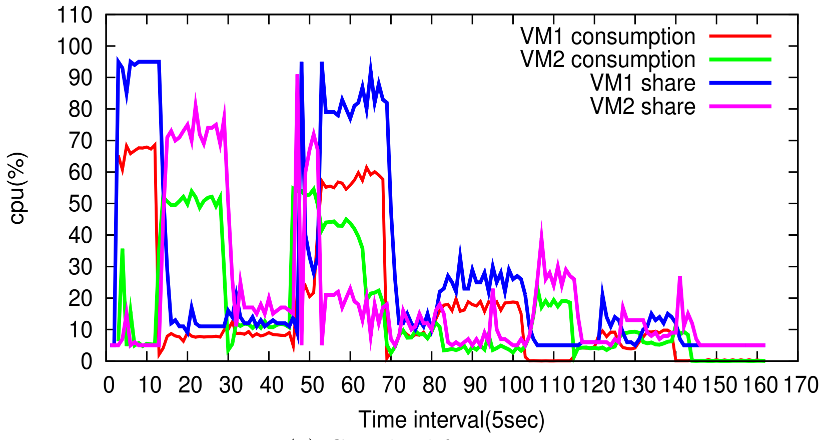
4.4 Experimental Setup and Evaluation

To evaluate the resource manager, an experimental testbed is set up consisting of two physical machines. The first physical machine is used to host two VMs and has a dual core 2.4 GHz Intel(R) x86_64 CPU with 3 MB L2 cache and 2 GB of RAM. It runs the Xen 4.0.1 hypervisor as the virtualization platform using Ubuntu 10.10 OS with 2.6.32.27 pv_ops Linux kernel in Dom0 and Ubuntu 10.10 OS with 2.6.35-25-server Linux kernel in DomU VM. The two DomU VMs to be managed by the resource manager get two VCPU each and are pinned to a physical core, while the Dom0 gets one VCPU and is pinned to the other core to avoid interfering with the DomU VMs. The other physical machine is used to generate load for the VMs and has a dual core 1.5 GHz Intel(R) x86_64 CPU with 2 MB L2 cache and 2 GB of RAM. It runs Ubuntu 10.10 OS with a 2.6.35-22-generic Linux kernel. Both machines are connected over a 100 Mbit/sec Ethernet in the same Local Area Network. Each DomU VM gets a maximum of 460 MB of RAM, but the allocation varies according to the load. On each VM, an Apache 2.2.16 web server is run as a representative of an interactive application to be managed. The requests to the two web server VMs are generated by two instances of httperf-0.9.0 crafted in Python scripts to generate the desired request patterns, each running in its own CPU core for not interfering with each other. The dm-ioband-1.14.0.patch is used as a disk I/O bandwidth controlling utility and the iproute2-ss100519 utility suite that includes tc is used as a network bandwidth controlling utility.

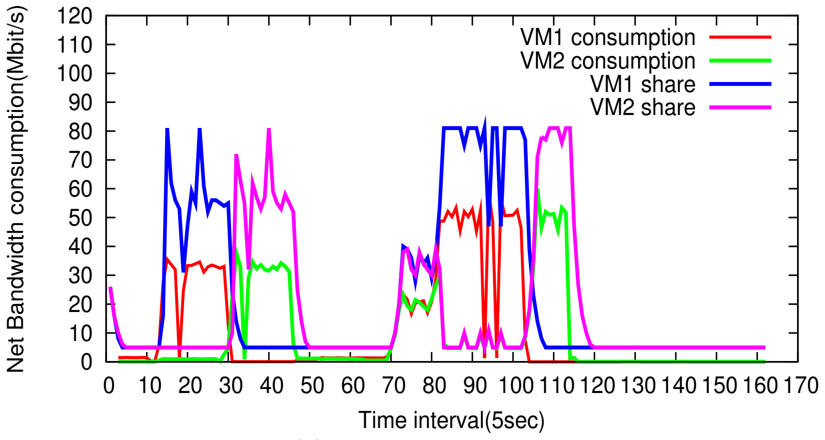
To test the ability of the resource manager to dynamically allocate resources according to the load, to keep QoS at acceptable levels and to increase the node utility function value, an experiment is performed where load is generated for the two VMs for all resources in different combinations and the results are compared when the resource manager was enabled and when it was disabled. For the experiment, the configuration described above is used. To synchronize the workloads of two VMs, the experimental time is divided into two phases: in the first phase, the resource manager is tested with load generated for CPU, disk and network, and in the second phase, with load generated for memory. A utility value of 4 per unit of resource is assigned for VM1 for all resources, and a utility value of 2 per unit of resource for VM2 for all resources.

4.4.1 Experiment Phase 1: CPU, Disk and Network Load

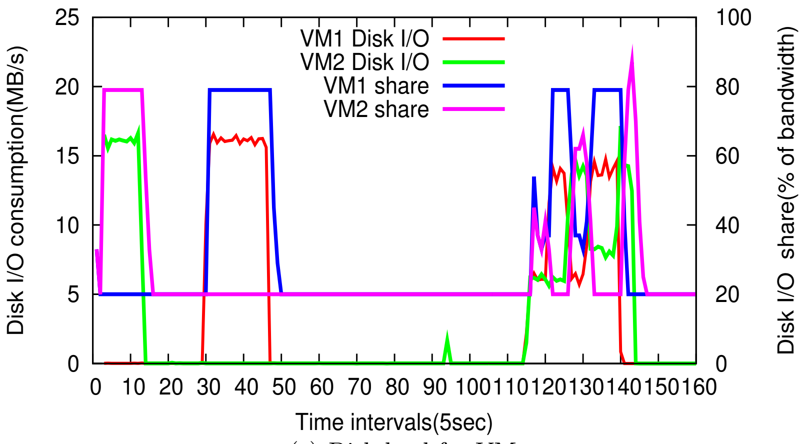
In Figures 4.4(a), 4.4(b), 4.4(c) the load patterns as represented by the resource consumption of the two VMs for CPU, network and disk bandwidth resources



(a) CPU load for VMs



(b) Net load for VMs



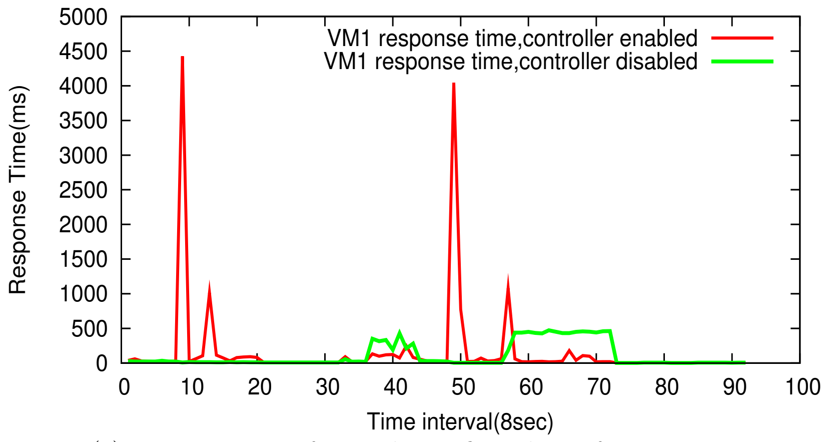
(c) Disk load for VMs

Figure 4.4: Load generated to VMs for different resources for resource manager enabled case

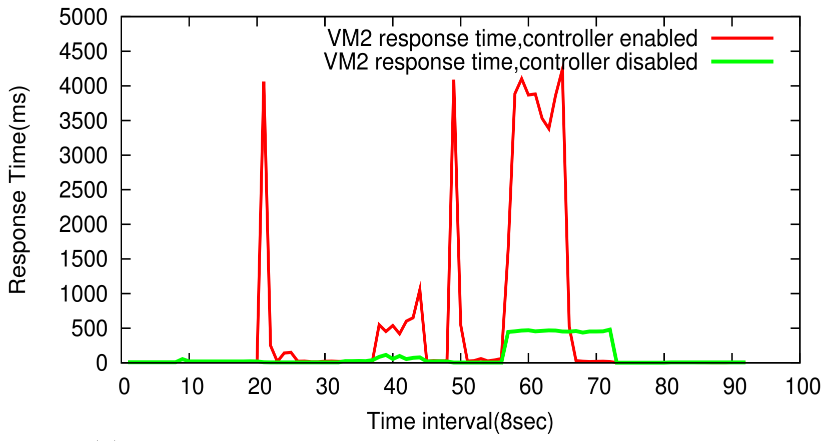
and the shares given by the resource manager are shown. What can be quickly noticed from these figures is that the resource manager is able to dynamically set shares according to the load for all types of resources, providing only resource allocations that are needed to satisfy QoS. From time interval 0 to 50, there are combined loads for different resources, but no resource is overloaded, meaning that when VM1 uses a resource, VM2 uses another resource (e.g., from interval 0 to 12, VM1 has CPU load, while VM2 has disk load), while each VM gets it required resource allocation. From interval 50 to the end of this phase, load is generated such that every resource becomes overloaded for some period of time. For example, from interval 50 to 70, both VMs consume CPU resources. If the resource manager is disabled, each VM gets the same CPU allocation independent of the utility per unit of CPU, giving VM1 less than the required allocation, which has an impact on the node utility and the response time of VM1. If the resource manger is enabled, VM1 gets the required CPU allocation, since it has more utility per unit of CPU than VM2, showing the ability of manager to solve overload situations by trying to increase the node utility. The same can be observed for the disk resource in time interval [130:140] and for the network resource in time interval [80:105].

Figure 4.5 shows the response time for both VMs. Figure 4.5(a) indicates that the response time of VM1 for ”manager enabled case” is better than for the ”manager disabled” case, especially in resource overload intervals, such as [35:45] or [55:75]. There are high spikes in response time for short periods of time, such as in time intervals around 9, 13 or 49, but these are caused by the load in the network resource, the reactive nature of feed-back controller and the interference of the network controller with the CPU scheduler. A thorough investigation of this problem is an interesting area of future research. Figure 4.5(b) demonstrates that the response time of VM2 for the ”manager enabled” case for most of the time (besides the short spikes discussed above) is the same as for ”manager disabled” case, and it is worst in overload intervals such as [37:45] or [55:66]. This behavior is expected since the manager in overload intervals gives more priority to allocating resources to VM1 in order to increase the node utility and thus does not satisfy the requirements of VM2. These results show that the manager is able to solve resource overload situations by keeping QoS of high utility VMs at acceptable levels.

In Figure 4.6, the node utility values for the ”manager enabled” and ”manager disabled” cases are shown. Most of the time, the utility values are equal in both cases, but the utility value for the ”manager enabled” case is better in overload intervals ([53:60] or [92:110]). There are short time intervals around 14, 21 or 78, where the utility value for the ”manager enabled” case is lower than for the ”manager disabled” case, but this is caused by the spikes of high response times for VM1 discussed above. These results show that the manager is able to improve the node utility especially in resource overload time intervals.



(a) Response time of VM1 during first phase of experiment



(b) Response time of VM2 during first phase of experiment

Figure 4.5: Response time of VMs for resource manager enabled and disabled cases

4.4.2 Experiment Phase 2: Memory Load

In this phase of the experiment, the ability of the manager to manage the memory resource is tested. Figure 4.7 shows the memory usage of the VMs for the ”manager enabled” case and their corresponding allocations as given by the resource manager. It is evident that the manager allocates memory to the VMs according to their memory usage. In the interval [0:15], the total memory load is low, and each VM gets its required allocation. At around time interval 15, the load for VM1 is increased and the load for VM2 is decreased, but the total required allocation does not exceed the total capacity, and again the resource manager gives each VM its required allocation. In time interval 50, it is seen an increase of the load of VM2 which creates a memory overload situation where the required allocations of both VMs exceed the total memory capacity. Since VM1 has a higher utility, it gets its required allocation first,

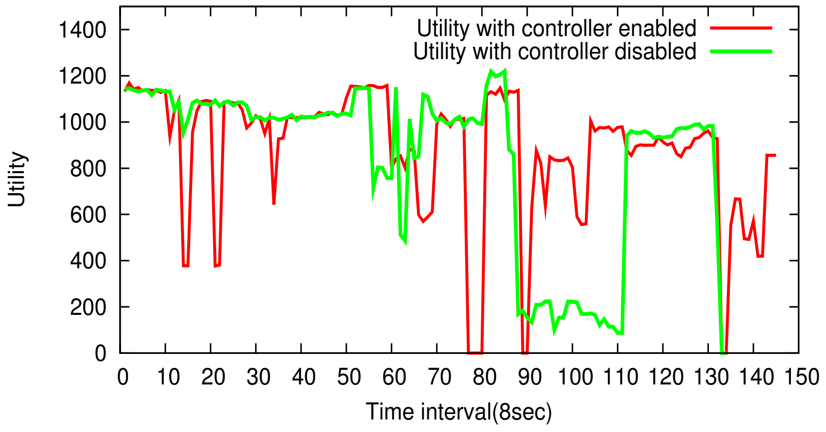


Figure 4.6: Utility during the first phase of experiment

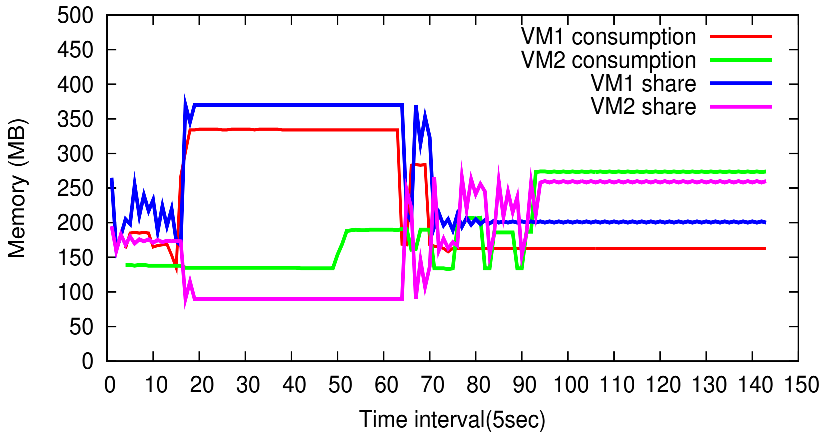
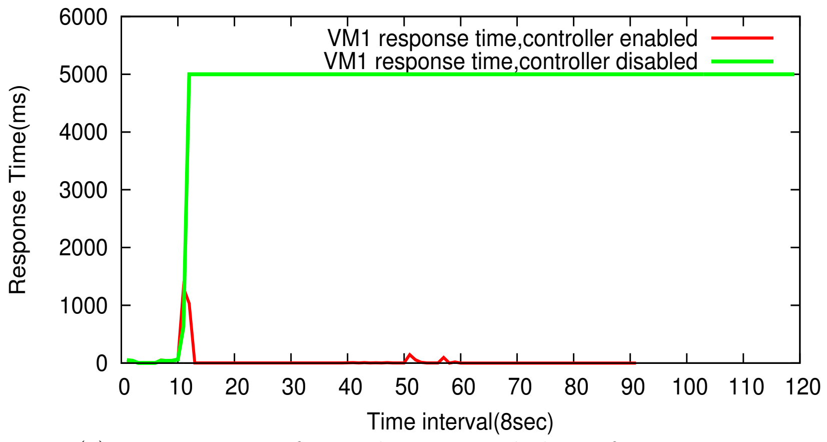


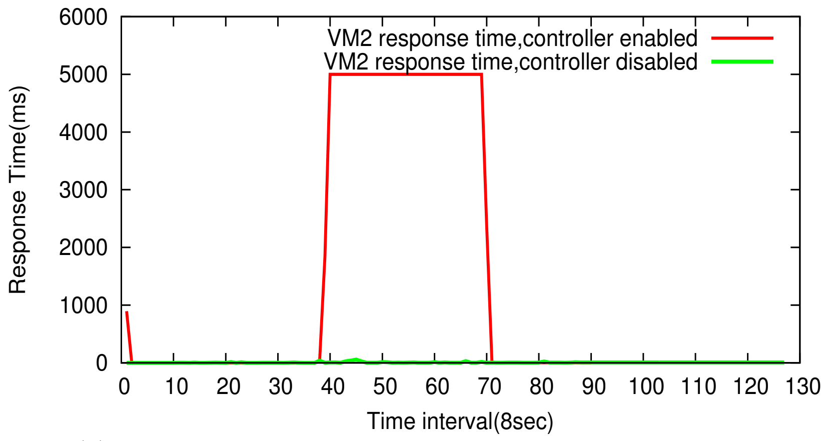
Figure 4.7: Memory load generated to VMs for resource manager enabled case

and the rest of the memory is given to VM2. At first glance it seems strange that in interval [15:65] VM2 consumes more memory than the given allocation, but this can be explained by the fact that for calculating the memory used and the memory shares, the algorithm leaves a 100 MB memory region free for the file system cache. In a near total capacity situation, the consumption of this reserved memory is in fact the negative difference between the VM2 memory share and its usage. At around time interval 65, the memory load for VM1 is decreased, which makes it possible to have more free memory, and after some time at around time interval 92, the required allocation to VM2 to be satisfied. These results show that the manager is able to allocate memory according to the load and to solve overload situations by giving the required allocation to VMs that have a higher VM utility in order to increase node utility.

Figure 4.8(a) shows the VM1 response time in this phase of the experiment for the "manager enabled" and the "manger disabled" cases. The response time of VM1 for the "manager enabled" case is at acceptable levels, because



(a) Response time of VM1 during second phase of experiment



(b) Response time of VM2 during second phase of experiment

Figure 4.8: Response time of VMs for resource manager enabled and disabled cases

its memory requirements are satisfied even in overload situations, while for the "manager disabled" case it is very high for most of the time, because in this case VM1 requires more memory than was statically allocated and forces it to use swapping, thus lowering QoS. In Figure 4.8(b) the response time of VM2 for "manager enabled" and "manager disabled" cases is shown. Most of the time, the response time remains at acceptable levels for both cases. It is worst for the "manager enabled" case for the memory overload period [38:71], which is expected, since VM2 has a lower utility per memory unit and thus its required allocation is not satisfied increasing swapping activity and lowering QoS.

Figure 4.9 shows node utility value for this phase of the experiment for the "manager enabled" and "manager disabled" cases. The utility value is much better for the "manager enabled" case, demonstrating the ability of the resource manager to improve the node utility by allocating memory first to VMs that offer a higher VM utility.

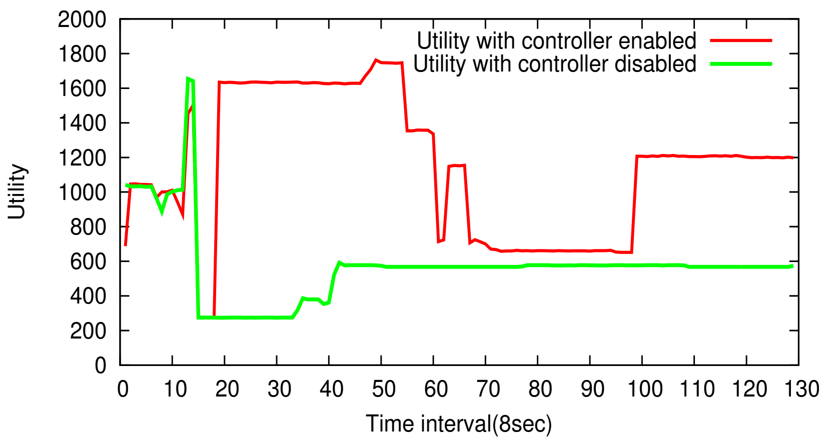


Figure 4.9: Utility during the second phase of experiment

4.5 Summary

In this chapter, an approach is presented for dynamically allocating different types of resources such as CPU, memory, disk, and network to virtual machines in Infrastructure-as-a-Service clouds by maximizing a utility function, such that QoS requirements are satisfied and the profit of the cloud provider is optimized. The proposed approach is based on resource managers situated in each physical machine of the cloud infrastructure that dynamically manage resources of VMs running on that physical machine. For each type of resource to be managed, the manager contains a) a monitoring component to measure resource usage, b) a feed-back controller to allocate resource shares to each VM, and c) an arbitrator that attempts to maximize the node utility function by considering the utility values of the VMs that run on it. Experimental results have shown that the approach allocates different types of resources to VMs in Infrastructure-as-a-Service clouds in a satisfactory manner.

There are several areas of future work. For example, the resource manager could be extended by considering other types of utility functions. Furthermore, live migration of virtual machines should be considered as a resource allocation mechanism to handle overload situations and to further increase the utility value of a cloud provider. Finally, further investigations of the interdependencies of multiple resource allocation would provide more insights in the best allocations mechanisms to be used.

5 Virtual Machine Resource Allocation via Multi-Agent Fuzzy Control

5.1 Introduction

In this chapter, an approach is presented to support the Virtual Machine Monitor (VMM) in allocating resources to VMs running on a physical machine of the cloud provider. The two conflicting goals of maintaining application performance and reducing operating costs are expressed in a utility function that represents the profit of a cloud provider. The problem to be solved is optimizing the utility function, and the solution proposed in this work is based on fuzzy control to optimize it. To provide scalability for a potentially large number of VMs, a multi-agent version of the fuzzy controller is presented where each agent is responsible for resource allocation of a single VM in parallel with other agents, while an agent coordinator redistributes the maximum amount of resources that can be used by all agent VMs.

The focus of the work is on fine-grained dynamic allocation of VM resources locally on each physical machine of a cloud provider, considering CPU and memory as resources to be managed. Resources are allocated through the mechanisms offered by the used virtual machine technology for changing the CPU share and the memory allocation at runtime. Since the used virtualization technology does not adequately support disk I/O bandwidth allocation, it is not considered in this work. However, the proposed approach is principally designed to also include disk I/O bandwidth allocation. Fuzzy control is used to maximize a global utility function (a utility function over all VMs) using a hill-climbing heuristic implemented as fuzzy rules. In the multi-agent version, the global utility function is divided into local utility functions that are optimized separately by each agent. The advantage of using fuzzy control is that it does not require any mathematical model to be built beforehand to control VM resources. The novelty of this work consists in developing a fuzzy control approach for utility function optimization and supporting a potentially large number of VMs by developing a multi-agent fuzzy controller.

The performance of the proposed approach is compared to an adaptive optimal control approach developed by Magnus *et al.* [74] that is adapted for VM resource allocation. It represents an advanced control theory approach and is one of the state-of-the-art techniques used recently for virtualized resources [89]. Experimental results show the effectiveness of the multi-agent fuzzy control approach achieving better utility values than the approach of Magnus *et al.* [74] and the centralized fuzzy controller.

First the VM resource allocation framework is described. Next, the adaptive optimal control approach is discussed. Then, the centralized fuzzy control approach and its multi-agent version is introduced. In the end, the experimental evaluation is described and the chapter is concluded with a summary. The work described in this chapter has been published in [82].

5.2 Virtual Machine Resource Allocation Framework

The problem considered is how resources of a physical machine of a cloud provider should be (re)-allocated to VMs dynamically in response to workload changes to keep the performance according to the SLAs while reducing the operating costs. An SLA is a contract between a consumer and a cloud provider, and in this work it is represented by the utility function that describes the monetary value that a consumer pays for obtaining the desired performance level. The operating cost includes any cost from management costs to power costs of the physical machine. In this work, the average amount of resources allocated to all VMs as an approximation of the operating costs is used. The problem of finding the right trade-off of performance and operating costs is solved by optimizing the following global utility function:

$$U = \delta * (\alpha \frac{\sum_{i=1}^{n} V_i}{n} - \beta C) \tag{5.1}$$

U represents the average profit the cloud provider gets from one physical machine during a control interval, n is number of VMs, V_i is the performance utility that represents the monetary value paid by the consumer to the cloud provider for getting a certain performance level from VM_i during a control interval, and C represents the operating cost of the physical machine in a control interval. The coefficients α and β are used to give more priority to one objective over the other. δ is a constant to make the utility value larger than 1 for display convenience. To measure application performance, the application heartbeat framework [50] is used. Although, this technique is used to measure applica-

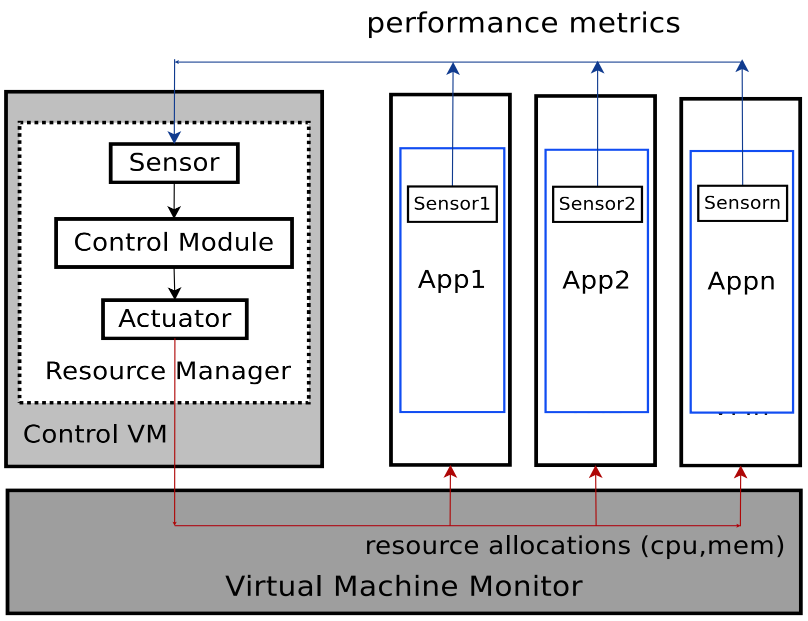


Figure 5.1: Resource manager architecture

tion performance, the approach is independent of any performance measuring method, since it uses the normalized performance calculated by dividing the actual performance by the desired performance level in an SLA contract. V_i is a simple linear function of normalized performance n_perf:

$$V_i = \begin{cases} n\_perf & \text{if } n\_perf < 1, \\ 1 & \text{if } n\_perf >= 1. \end{cases} \tag{5.2}$$

The operating cost C is the average of the sum of resource allocation amounts of two resource types, CPU and memory:

$$C = \frac{\sum_{i=1}^{n} cpu_i + \sum_{i=1}^{n} mem_i}{2} \tag{5.3}$$

where cpu_i and mem_i are portions of CPU and memory capacity given to VM_i expressed as numbers in the interval [0, 1], obeying the constraints that the total sum should be less than 1, the resource capacity. The approach is implemented as a resource manager responsible for resource allocation of VMs running on a physical machine. Its architecture shown in Fig. 5.1 consists of three components: 1) sensor, 2) actuator, 3) control module. Sensors gather performance metrics in each interval, normalize them and pass them to the main sensor in the resource manager. The actuator gets resource allocations from the control module and applies them via the VMM. The control module implements the resource allocation approach. In each interval, it decides which resource allocation should be applied in the next interval.

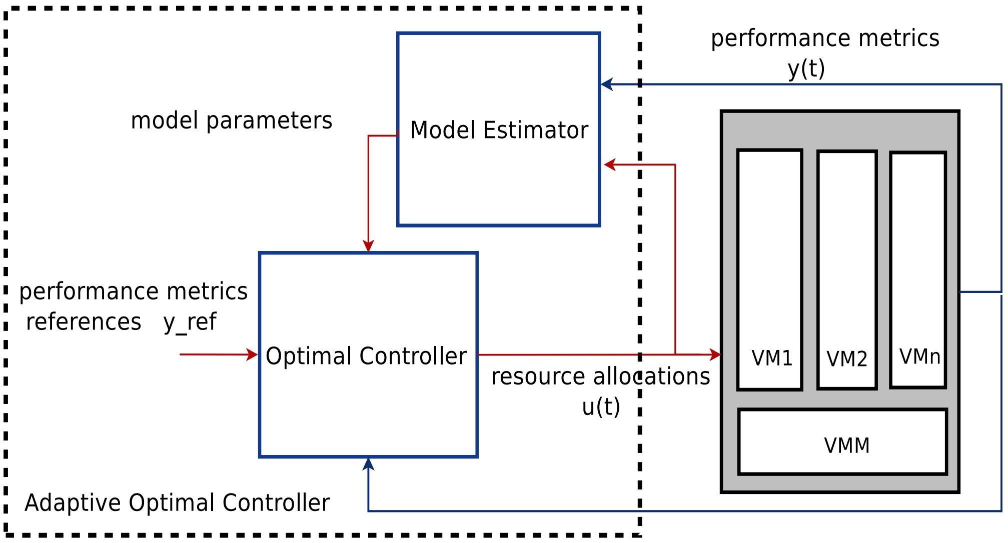


Figure 5.2: Adaptive optimal controller

5.3 Adaptive Optimal Control

Adaptive optimal control is an advanced design approach to build feed-back controllers. The method of adaptive optimal control used in this work is based on the work of Magnus et al. [74], but it is adapted to a virtualized environment where VM CPU and memory shares are used as controller outputs instead of shares for scheduling requests. The architecture of the adaptive optimal controller is shown in Fig. 5.2. It consists of two components: 1) *model estimator* and 2) *optimal controller*.

The model estimator computes a linear autoregressive-moving-average Multi-Input Multi-Output (MIMO) model of the relationship between multiple resource allocations and multiple application performance metrics.

Since workload dynamics can change overtime, this linear model may become invalid and thus should be recomputed. For this purpose, a recursive least squares (RLS) method is used to recompute the time-varying model parameters online. The optimal controller aims to find resource allocations that minimize a quadratic cost function from which a control law can be derived to calculate resource allocations [74].

Algorithm 1 shows the main steps of the adaptive optimal controller. In the algorithm, the control interval between resource allocation decisions is set to 30 seconds through sleep() which puts the control module to sleep. This value is chosen as a trade-off between a low value with noisy measures and a high value of slow reaction.

Algorithm 1: Adaptive optimal controller algorithm

```
1  apply few predefined samples to build an initial model
2  while true do
3      sleep(30sec)
4      obtain performance measurements y(t)
5      estimate model parameters using RLS
6      calculate resource alloc u(t) through the control law
7      if sum of resource allocations greater than capacity then
8          redistribute resources in the same proportions to be lower than capacity
9      end
10     apply resource alloc u(t)
11 end
```

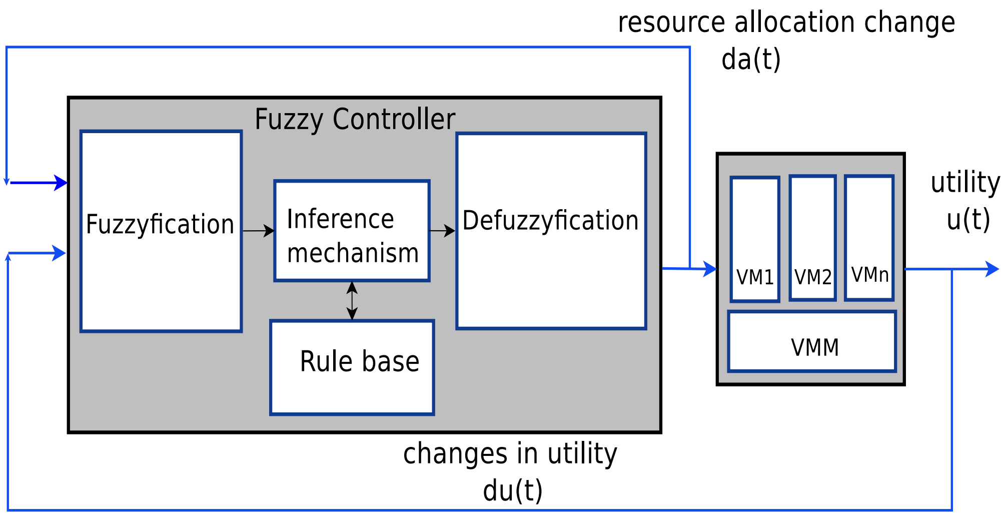


Figure 5.3: Fuzzy controller

5.4 Centralized Fuzzy Control

Fuzzy control [91] provides a way to design a controller based on heuristic knowledge needed to control a dynamic process. The fuzzy controller is shown in Fig. 5.3. The output of the VM environment is the utility value $u(t)$ to be maximized. The change of the utility value $du(t)$ between two intervals is the input of the fuzzy controller. The output of the fuzzy controller is the allocation change $da(t+1)$ determining whether a resource should be increased or decreased in the next interval. This is also applied as the input to the fuzzy controller itself as the current resource allocation change $da(t)$ to be used for the next decision.

Based on the change of the utility and the change of allocation in the previous interval, the controller determines the change of allocation for the next interval. Since the fuzzy controller internally works with linguistic variables and

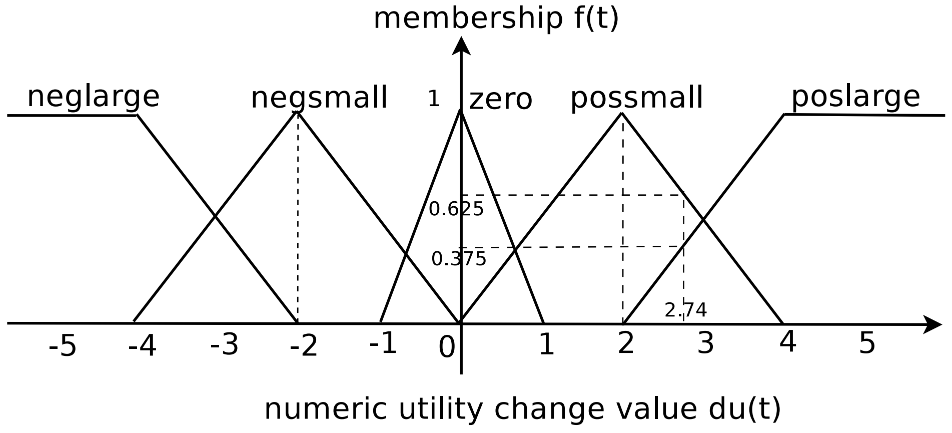


Figure 5.4: Utility change membership function

linguistic values, the fuzzyfication module performs the conversion from numeric values measured from outside the fuzzy controller to the corresponding linguistic values (e.g., *poslarge*, a value that is positive and large), while the defuzzyfication module performs the reverse conversion. In the rule base, a set of condition-action rules is stored that implements the heuristic to optimize the utility function. The inference mechanism based on the linguistic input values selects the rules that apply and produces linguistic output values. The conversion in the fuzzyfication and defuzzyfication modules from numeric values to linguistic values is achieved by a membership function. This function maps a numerical value to certainty levels, a number in the interval [0,1] (0 completely uncertain, 1 completely certain), for different linguistic values it corresponds to.

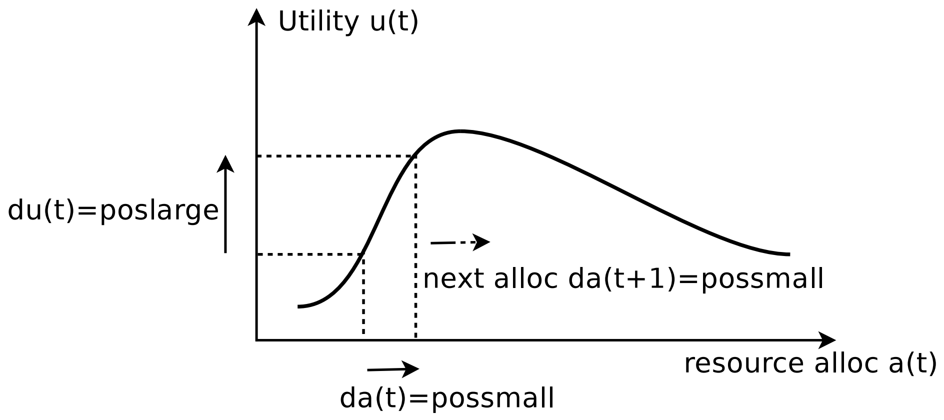


Figure 5.5: Hill-climbing heuristic for one resource

In Fig. 5.4, the membership function that converts a numeric utility change to linguistic values with different certainties is shown, where the x-axis represents numeric values and the y-axis certainty values. Five linguistic values are used to represent utility changes. The membership function for resource allocation changes is not shown for space reasons, but it has almost the same form, and for the input it is used to convert numeric resource allocation changes to linguistic values, and for the output linguistic values to numeric resource allocation changes. The fuzzy controller maximizes the utility value by a set of

Table 5.1: Hill-climbing heuristic rule table

da(t+1)		**du(t)**				
		0	1	2	3	4
da(t)	0	1	0	2	3	3
	1	0	1	2	4	4
	2	2	2	2	2	2
	3	4	3	2	0	0
	4	3	4	2	1	1

rules based on a hill-climbing heuristic. Fig. 5.5 shows how the utility value changes by changing the allocation of a resource for a VM. If we increase the resource allocation in the current interval by a small amount and get a positive large increase of the utility value, we are on the left side of the hill and the slope of the curve is high. Thus, to maximize the utility, we should continue to increase the allocation in the next interval by a small amount in order not to overpass the top of hill. The idea of using hill-climbing to maximize the utility is based on work of Diao *et al.* [26]. Similar rules are created for all situations, as shown in Table 5.1. This rule set has been created by experimental testing, and the results indicate that it performs quite well. The following mapping between numbers and linguistic values is used: *0-possmall*, *1-poslarge*, *2-zero*, *3-negsmall*, *4-neglarge*. These rules tell how to maximize utility for one resource of a VM, and since utility is a function of multiple resources and multiple VMs, the utility function is maximized by applying these rules for each resource and each VM one by one as shown in Algorithm 2 without lines 27-29, which are used only for multi-agent fuzzy control. The algorithm starts with minimum resource allocations and takes a first round, in lines 3-14, of only increasing allocations by applying fuzzy rules until it reaches the optimal utility. Then, it takes a second round, in lines 15-26, of only decreasing allocations until a possibly new optimal utility is reached. This repeated switching between the two rounds allows the controller to adapt to changes in optimal utility due to workload changes.

5.5 Multi-Agent Fuzzy Control

Since an increased number of VMs leads to an increased number of iterations of the for loops in lines 3 and 15, the runtime of Algorithm 2 for global utility optimization increases correspondingly. Thus, a multi-agent fuzzy controller is developed as shown in Fig. 5.6. Each agent is responsible for the resource

Algorithm 2: Fuzzy control algorithm

```
set resource allocations of all VMs to minimum levels
while true do
    for $i \leftarrow 1$ to $numberOfVMs$ do
        for $j \leftarrow 1$ to $numberOfResources$ do
            nextalloc = possmall // increase resource
            repeat
                apply nextalloc to resource j of $VMi$
                sleep(30sec)
                based on utility change apply fuzzy rules and get nextalloc
            until utility change zero or negative ;
            apply nextalloc to resource j of VMi
            sleep(30sec)
        end
    end
    for $i \leftarrow 1$ to $numberOfVMs$ do
        for $j \leftarrow 1$ to $numberOfResources$ do
            nextalloc = negsmall //  decrease resource
            repeat
                apply nextalloc to resource j of $VMi$
                sleep(30sec)
                based on utility change apply fuzzy rules and get nextalloc
            until utility change zero or negative ;
            apply nextalloc to resource j of VMi
            sleep(30sec)
        end
    end
    synchronization barrier() // only in multi-agent
    send() to coordinator resource consumption and request
    receive() from coordinator maximum resources allowed
end
```

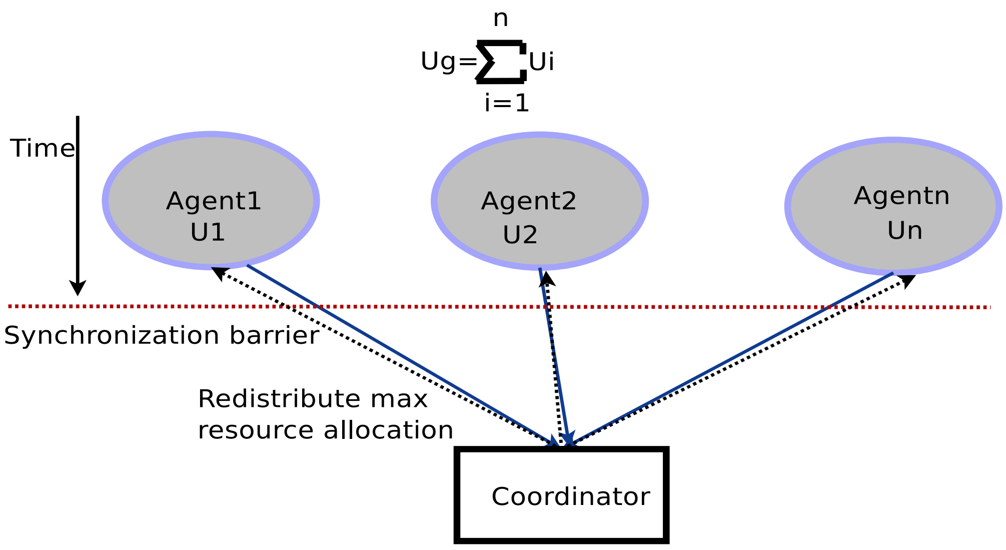


Figure 5.6: Multi-agent fuzzy controller architecture

allocation of a single VM by optimizing its own local utility function in parallel with other agents. Since the goal is to optimize the global utility function given by (5.1), this function should be divided into local utility functions in such a way that optimizing each of them separately would lead to global utility optimization. The global utility function in (5.1) permits us to express it as the sum of local utility functions:

$$
\begin{aligned}
U &= \delta(\alpha\frac{V_1 + V_2 + ..}{n} - \beta\frac{cpu_1 + cpu_2.. + mem_1 + mem_2..}{2}) \\
&= \delta((\alpha\frac{V_1}{n} - \beta\frac{cpu_1 + mem_1}{2}) + (\alpha\frac{V_2}{n} - \beta\frac{cpu_2 + mem_2}{2}) \\
&... + (\alpha\frac{V_n}{n} - \beta\frac{cpu_n + mem_n}{2})) \\
&= \delta(U_1 + U_2 + ... + U_n)
\end{aligned}
\tag{5.4}
$$

where U_1,U_2,..U_n represent local utility functions that are assigned to each agent. A local utility function U_i depends only on resource costs ($\frac{cpu_i+mem_i}{2}$) and application performance V_i of the corresponding VM_i, meaning that it can be optimized separately by each agent. Since the global utility is the sum of local utilities, optimizing each of them separately optimizes the global utility.

Resources allocated to VMs in the system should not exceed the total capacity of the resources, but since the agents operate independently, they can allocate resources in such a way that this condition is violated. To solve this problem, a coordinator is introduced that periodically calculates and distributes to all agents the maximum amount of resources that can be allocated by them, obeying to the total capacity condition.

In the beginning, the coordinator assigns the same amount of maximum resources to all agents. At the time of calculation, it gets the resource consumption and the maximum resource allowed for each agent. From the difference between the two, it calculates the amount of resources that is not used by each agent. Summing up all free resources of all agents it obtains the total amount of free resources that can be redistributed to them. To do this, the total amount of free resources is divided by the number of agents, and the result is added to the resource consumption of each agent to get the maximum amount of resources allowed for each of them. This method is applied for each resource separately.

The maximum amount of resources is redistributed after each agent has passed two rounds of increasing and decreasing the resource allocations (see Algorithm 2). This scheme is fair by giving each agent the same number of rounds for optimizing its local utility functions before getting its allowed maximum amount of resources. Since agents can finish their two rounds with different speeds, they are synchronized with each other through a synchronization barrier primitive at the moment of requesting the maximum amount of resources.

The algorithm followed by each agent is shown in Algorithm 2. In this case, the fuzzy rules are applied for allocating resources to a single VM, so $numberOfVMs = 1$. After two rounds of increasing and decreasing resource allocations, the agent is synchronized with other agents through the barrier primitive. After synchronization, it sends the actual resource consumption and the maximum resource allocation request to the coordinator. After receiving from the coordinator the maximum amount of allowed resources, it continues with the next rounds of resource allocations.

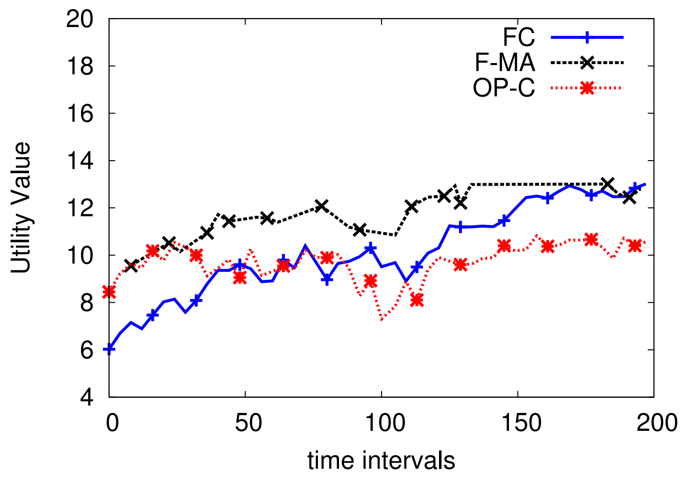
5.6 Implementation

The resource allocation approach is implemented in the C++ programming language. The Xen hypervisor is used as the virtualization technology. The performance sensor runs as a daemon inside each VM and is attached to an application through a shared memory mechanism of Linux to gather performance metrics using the heartbeat API framework [50]. The main sensor that is part of the resource manager requests performance metrics through the IP socket interface from all application sensors in each control interval. The actuator sets the CPU and memory allocation using the xm sched-credit and xm mem-set commands of the Xen hypervisor. The multi-agent version of the fuzzy controller is implemented as a multi-threaded program. The value of α is set to 2 and the value of β to 1 in (5.1) to give more priority to performance than to resource costs.

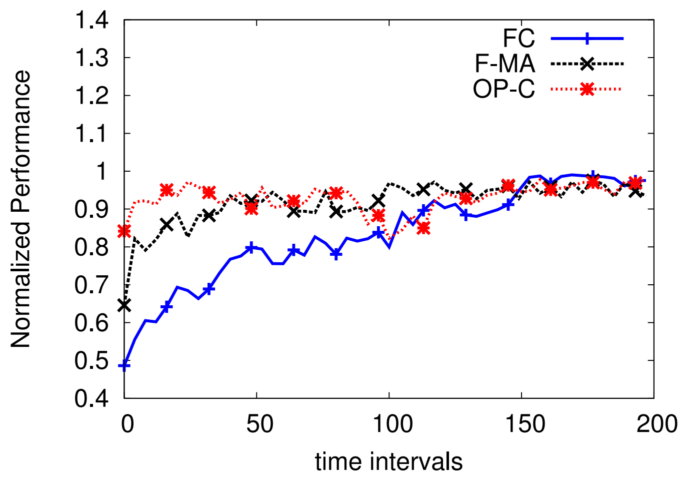
5.7 Experimental Set-up and Evaluation

A physical machine is set up as a testbed to run VMs to be managed. It has two dual-core AMD Opteron 2.4 GHz processors and 8 GB of RAM. It runs Xen 4.1 and the Ubuntu 12.04 operating system in the Dom0 and DomU VMs. For evaluation, the following benchmarks are used: a) *cpu_bench* is a CPU intensive benchmark, created for this work, that performs a set of mathematical calculations, such as estimating the factorial of a number; b) *filebench* [36] is a file system benchmark that allows us to generate a variety of workloads emulating a number of applications such as web, file, and database servers.

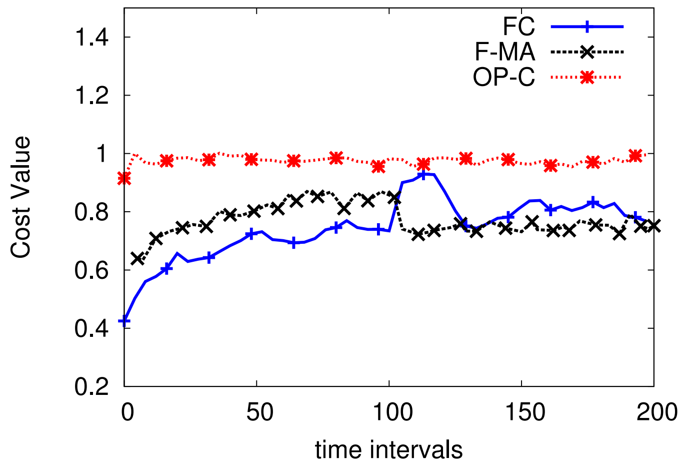
Experiments are performed with four VMs and eight VMs, where in each case half of them run filebench with the web server profile and the other half run cpu_bench. Four VMs are pinned in two CPU cores and eight VMs to three CPU cores and is given each of them one VCPU while Dom0 is pinned to another core. 1600 MB of memory is given to four VMs, while 3200 MB to eight VMs. A minimum of resource allocation of 20% is set to each VM for not letting them starve for resources. The following resource granularities are used: *(neg/-pos)small*=10%, *(neg/pos)large*=20% for the CPU and *(neg/pos)small*=80 MB, *(neg/pos)large*=160 MB for the memory. The experiment is run for 200 intervals, and in interval 100 it is initiated a workload change. The performance



(a) Utility



(b) Average performance



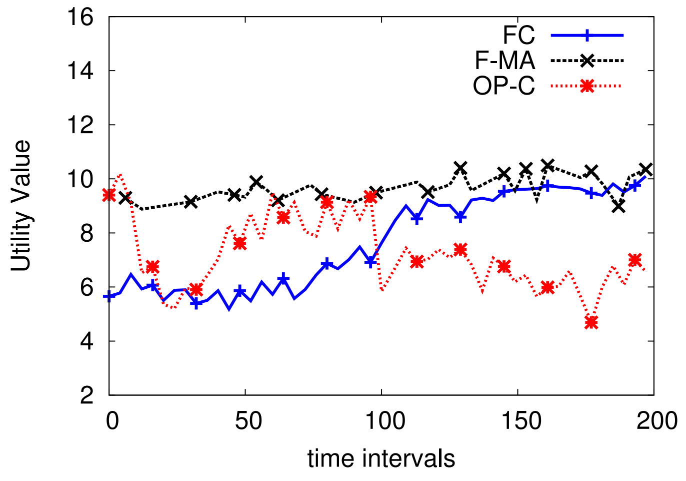
(c) Average resource cost

Figure 5.7: Utility, average performance and average resource cost for 4 VMs

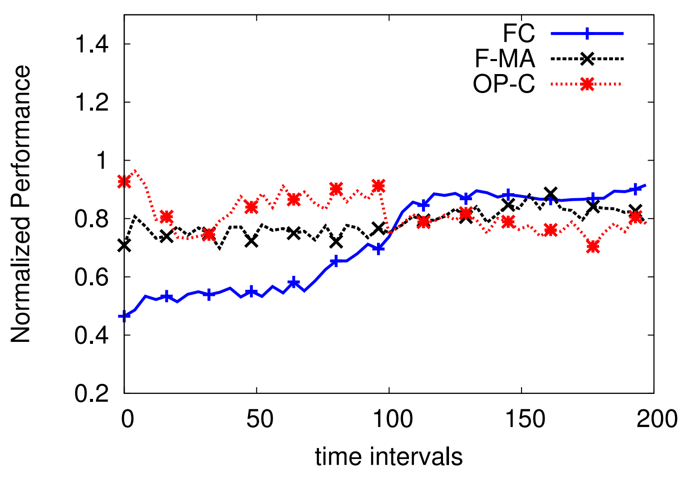
levels of two filebench VMs and two cpu_bench VMs are changed to emulate a workload change. The experiment was run for each approach five times and the average results are shown. The following abbreviations are used: Fuzzy Control - FC, Fuzzy Control Multi-Agent - F-MA, and Adaptive Optimal Control - OP-C.

In Fig. 5.7(a) and 5.8(a), the average utility over time for the three approaches is shown for 4 and 8 VMs. It is evident that F-MA has better average utility value during the whole experimental interval compared to the other approaches for both cases. FC has more difficulty to reach the optimal utility value with increasing the number of VMs from 4 to 8, as indicated by a larger difference of the utility value to F-MA for 8 VMs. This is because the number of resource allocations steps to reach the optimal utility increases with the number of VMs, which is not the case for F-MA. It is also evident that FC reaches the same final maximal utility as F-MA after the workload change in the interval from 100 to 200. This is because the resource allocation in time interval 100 changes in a few steps to obtain the resource allocation with the optimal utility in time interval 200, and thus this can be achieved faster by FC. To understand whether the utility difference of F-MA to the other approaches is statistically significant, it is performed paired t-tests over all intervals between F-MA and each of the other approaches with a 95% confidence interval. The utility value difference between F-MA and FC is statistically significant over only 37% of the intervals for the 4 VM case and 50% for the 8 VM case. This is because the advantage of F-MA can appear only with an increased number of VMs. The utility value difference between F-MA and OP-C is statistically significant over 72% of the intervals for 4 VMs and 67% for 8 VMs. The average performance over all VMs was calculated for each interval, and the average of this value over five runs for each approach is shown in Fig. 5.7(b) and 5.8(b). It is evident that FC has difficulties to keep the performance to desired levels especially until interval 100 with a statistically significant difference with other approaches, while F-MA is able to keep the performance near the desired level over all intervals together with OP-C.

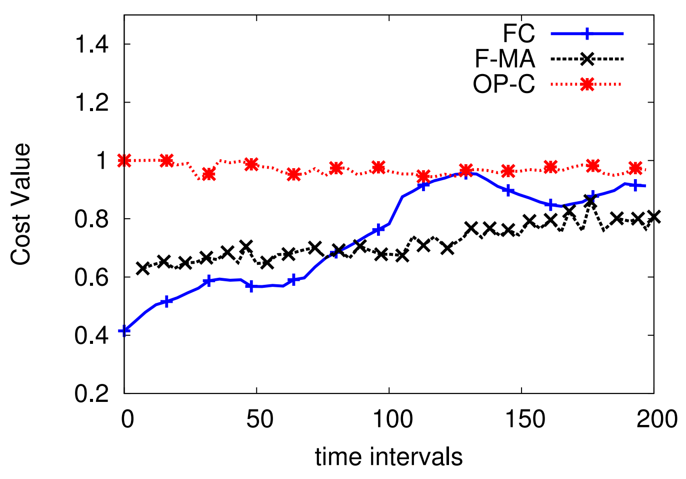
Fig. 5.7(c) and 5.8(c) show the average resource cost over all intervals as calculated by (5.3). It can be observed that FC and F-MA achieve comparable resource costs, sometimes F-MA achieves lower cost as shown in the second phase for 8 VMs. OP-C achieves higher resource costs than the other approaches, since OP-C does not take into account resource costs through optimization of the utility value, but focuses only on performance. F-MA achieves almost the same resource costs as FC but with higher average performance resulting in higher utility values, as shown in Fig. 5.7(a) and 5.8(a). F-MA achieves the right amount of resource costs needed to keep the performance to high levels by finding the best performance-cost trade-off.



(a) Utility



(b) Average performance



(c) Average resource cost

Figure 5.8: Utility, average performance and average resource cost for 8 VMs

5.8 Summary

In this chapter, an approach to address the performance-cost trade-off for VM resource allocation in cloud computing has been presented. It is based on expressing the two conflicting goals in a utility function and using fuzzy control for optimizing it. To support an increased number of VMs, a multi-agent fuzzy control approach has been developed.

Experiments comparing the multi-agent approach with centralized fuzzy control and a state-of-the-art adaptive optimal control approach show the effectiveness of the proposed multi-agent fuzzy controller. It performs better than the other approaches, especially with an increased number of VMs.

In the future, it is planned to include power consumption and disk I/O bandwidth in the utility function. Furthermore, the approach will be evaluated in a complex environment with multiple physical machines and multi-tier applications.

6 Distributed Resource Allocation to Virtual Machines via Artificial Neural Networks

6.1 Introduction

This chapter is focused on vertical resource scaling and presents an approach to assist the Virtual Machine Monitor (VMM) of an IaaS cloud to optimize resource allocation to VMs running on the physical machines. The proposed approach addresses the mentioned performance-power trade-off by expressing the two conflicting objectives of application performance and power consumption in a utility function and optimizes the utility function at runtime. A resource manager for utility function optimization is presented that is based on an artificial neural network (ANN) to model the relationship between the resources allocated to VMs, the performance of applications and the power consumption of physical machine. ANNs are well known for being universal approximators [52] to model any complex non-linear function. Another feature of this approach is that the model is learned and adapted on-line to workload changes, avoiding the need to build the model beforehand.

There are several versions of the proposed resource manager. In the centralized resource manager, it is used a single Multiple-Input Multiple-Output (MIMO) ANN based model for capturing the relationship between resource allocation to all VMs running on a physical machine and their corresponding performance metrics. Another similar MIMO model is used to capture the relationship between VM resource allocation and power consumption of the physical machine. To provide support for a potentially large number of VMs, a distributed resource manager is presented where the single MIMO model is divided into several mod-

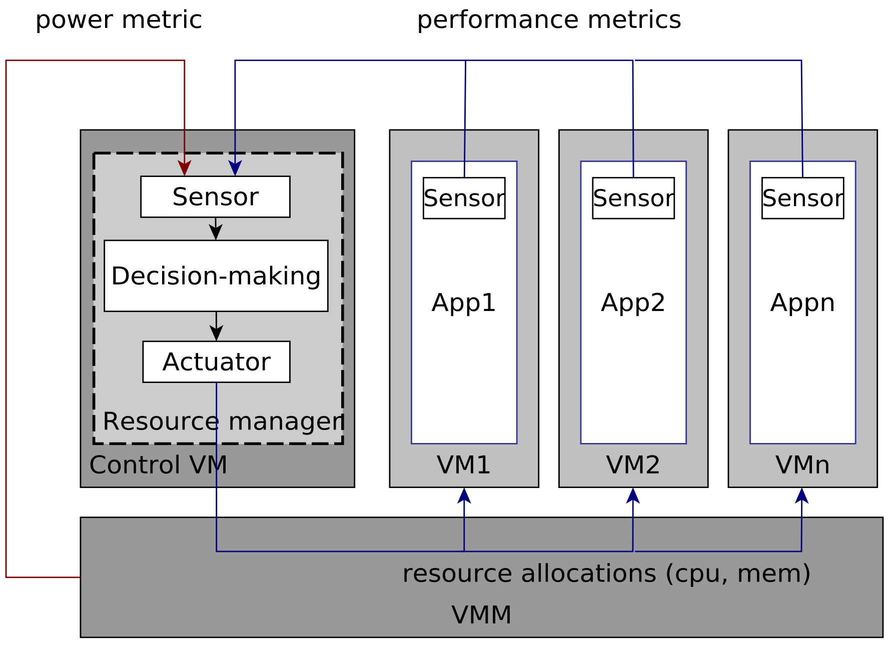


Figure 6.1: Resource manager architecture

els, each of them being responsible for performance and power modeling of a single VM. Each of the distributed resource managers optimizes its own utility function through its ANN model, but exchanges information with other resource managers to coordinate resource allocations. The approach is evaluated via simulation and real experiments. The results show the effectiveness of the decentralized resource manager over static allocation, a centralized version and a distributed non-coordinated version.

First, the resource manager architecture, the utility function and the power model are described. Then, the centralized resource manager approach and its distributed version are introduced. Next, implementation issues and experimental results are discussed. The chapter is concluded with a summary of the approach. The work described in this chapter has been published in [84].

6.2 Resource Manager Architecture

The proposed resource manager architecture is shown in Fig. 6.1. It consists of three components: sensor, actuator and decision-making module. It works in discrete time intervals where at the end of each time interval it gathers application performance and power metrics through the sensor module, performs resource allocation to all VMs for the next interval through the decision-making module and allocates the resources through the actuator.

The problem addressed is how to allocate resources to VMs running on each physical machine of the cloud provider in order to maintain application performance according to SLA levels while reducing power consumption. To solve this performance-power trade-off, both objectives are expressed in a utility function and based on ANN modeling, the resource allocations that optimizes the utility function are found. The utility function used is shown in Equation (6.1):

$$U = \delta * (\alpha \frac{\sum_{i=1}^{n} V_i(perf_i)}{n} - \beta U_p(power)) \tag{6.1}$$

U is the average global utility over all VMs representing the average profit the cloud provider gets from one physical machine during a time interval, n is number of VMs, V_i is the performance utility that represents the profit the provider gets from the consumer for guaranteeing a certain performance level to VM_i during a time interval, and U_p is a power utility function that represents the power consumption costs of the physical machine in one time interval. The coefficients α and β are used to control the priority given to both objectives. δ is a constant to make the utility value larger than 1 for display convenience. The performance utility function V_i used in this study is given as a function of normalized application performance $perf_i$, as shown in Equation (6.2). Normalized performance is used to make the resource manager independent of specific application performance metrics. Since the throughput is used as the performance metric (the number of operations served per second), the normalized performance is the ratio of the actual throughput to the desired throughput stated in an SLA.

$$V_i(perf_i) = \begin{cases} perf_i & \text{if } perf_i < 1, \\ 1 & \text{if } perf_i >= 1. \end{cases} \tag{6.2}$$

The power utility function $U_p(power)$ is a function of the *power* consumed by a physical machine in a time interval. Since power consumption of a physical machine generally depends on resource utilization, it is adopted a linear model for CPU and disk I/O utilization [47] to approximate power consumption as shown below:

$$power = P_{idle} + P_{cpu}\frac{U_{cpu}}{C_{cpu}} + P_{disk}\frac{U_{disk}}{C_{disk}} \tag{6.3}$$

P_{cpu} is the maximum dynamic power consumption of the CPU at full utilization, U_{cpu} is the CPU utilization as a percentage of the total CPU capacity, C_{cpu} is the total CPU capacity, P_{disk} is the maximum dynamic power consumption of the disk at full bandwidth utilization, U_{disk} is the disk utilization as a percentage of the total disk bandwidth capacity, and C_{disk} is the total disk bandwidth capacity. P_{idle} is the power consumed by a physical machine when it is idle. These metrics are average values measured in a single time interval.

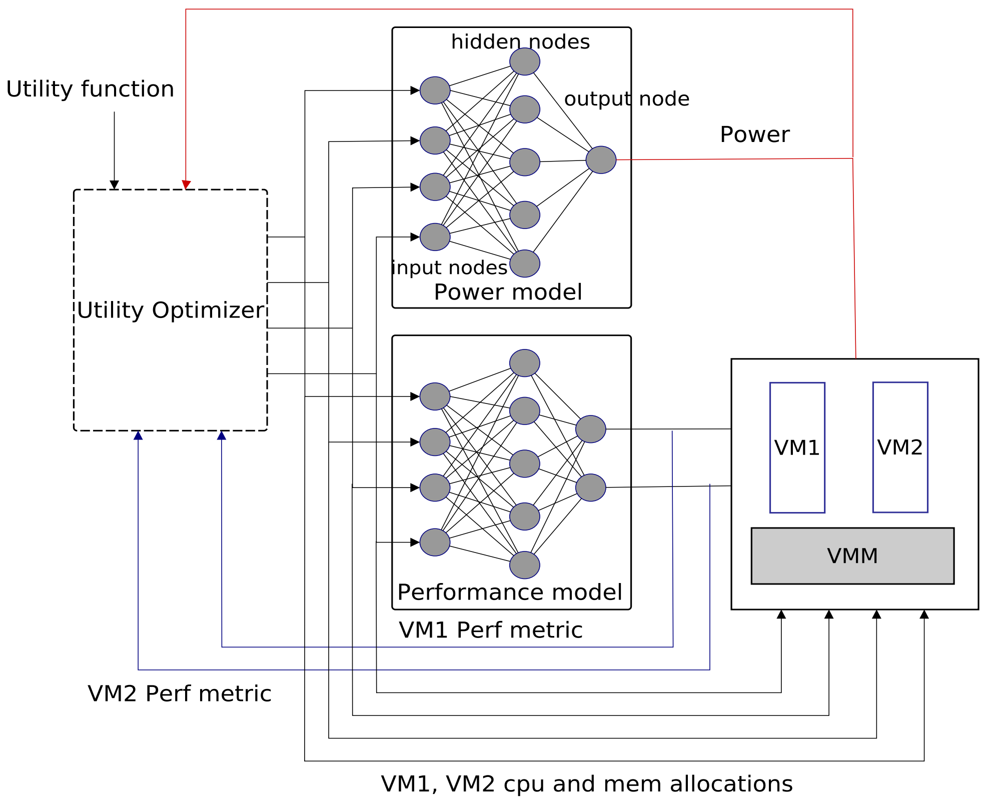


Figure 6.2: Centralized ANN resource manager

A power utility function is used that is given as the ratio of actual dynamic power consumption to the maximal dynamic power at full utilization:

$$U_p(power) = \frac{power - P_{idle}}{P_{cpu} + P_{disk}} \tag{6.4}$$

6.3 Centralized Resource Manager

The architecture of the centralized resource manager approach is shown in Fig. 6.2. It consists of three components: 1) a *performance model* representing the relationship between VM resource allocation and application performance metrics, 2) a *power model* representing the relationship between VM resource allocation and physical machine power consumption 3) a *utility optimizer* that uses the above models to search for resource allocations that reach maximum utility.

6.3.1 Performance Model

In the centralized resource manager, a MIMO model of the relationship between VM resource allocation and application performance is used. For two

VMs, its structure is shown in Fig. 6.2. It has four inputs representing resource allocations, two inputs (CPU and memory) for each VM and two outputs representing performance metrics, one output for each VM. The model predicts the performance of each VM if certain resource allocations are given to all VMs. To build the model an ANN is used to approximate the possibly non-linear function of the mentioned multi-input multi-output parameters.

An ANN is a structure of multiple layers of nodes connected through links, as shown in Fig. 6.2. It has three types of nodes: input, hidden and output nodes. Each link passes the output value of a node to the input of another node. The output of a node is calculated as the weighted sum of all inputs to the node which is then fed into a (non-linear) activation function. The ANN learns a model in a training phase where it is exposed to a number of (input, output) samples from a training set. Since it is difficult to decide which training algorithm performs best for which task, one of the well known training algorithms for ANNs is used, namely the resilient backpropagation (RPROP) algorithm [100]. To build the ANN, the cascade correlation algorithm [31] is also used that starts with a minimal network, then automatically adds new hidden nodes one by one during training, creating a structure with the number of hidden nodes necessary for the current training set size.

6.3.2 Power Model

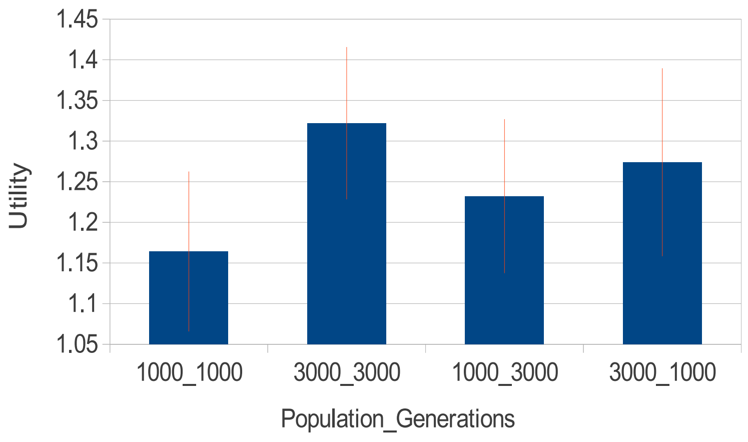
A MIMO model of the relationship between VM resource allocation and physical machine power consumption is built. For two VMs, its structure is shown in Fig. 6.2. It has four inputs representing resource allocations, two inputs (CPU and memory) for each VM and one output representing the power. The model predicts the power consumption of the physical machine if certain resource allocations are given to all VMs. To model power consumption, an ANN is also used.

6.3.3 Utility Optimizer

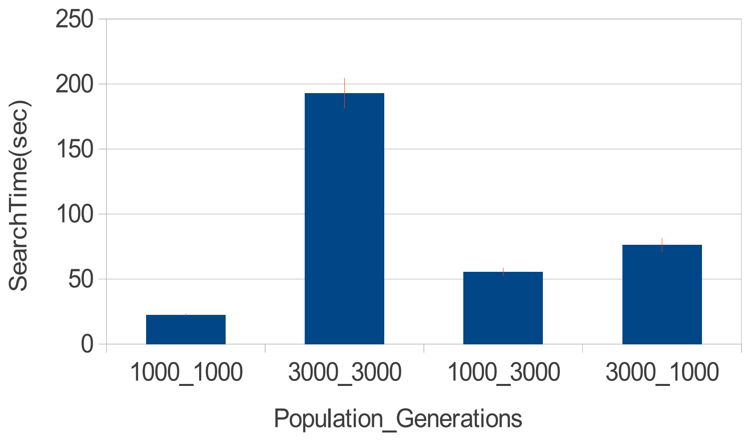
The goal of the utility optimizer is to find the resource allocation configuration that maximizes the utility function given in Equation (6.1). It does this by trying all resource configurations, predicting the performance using the performance model and power consumption using the power model, calculating the resulting utility and selecting the configuration that gives the maximum utility value. The total number of configuration combinations to try depends on the granularity of the resource allocations (minimum allocation change from one configuration to another) and the number of VMs. For a larger number of VMs, the total number of combinations to try can require considerable time, making it impractical for an exhaustive search, and in this case a stochastic search technique, namely a genetic algorithm [51] is used to optimize the utility function.

A genetic algorithm is based on natural evolution for exploring a search space to find the solution of a problem. It starts with a random initial population of individuals where each individual represents a feasible solution to the problem. There is a function that evaluates the fitness of an individual; the utility function is used as the fitness function. Then, from an initial population based on selection, crossover and mutation, a new population is generated that is supposed to be better than the old one in terms of the fitness function. This process is repeated for several generations until a maximum generation number is reached and individual with the best fitness found so far is the final solution of the problem. Each individual encodes information that represents a feasible solution. In this case, this is a resource allocation configuration in the form of a string of real numbers that represents resource shares given to VMs. The selection process selects from the population two parent individuals based on their fitness values according to a roulette wheel scheme. Then, with some probability, a crossover operator is applied where different parts of parents are combined to create two children such that each child takes a CPU allocation configuration from one parent and a memory allocation configuration from the other parent. At the end, a mutation operation is applied where with a small probability each gene of the child is altered to a random value. This process of selection-crossover-mutation is repeated to create the number of children needed for the new population to replace the old one for the next generation. Since population size and number of generations are two important parameters that can influence the genetic algorithm performance, experiments are performed to determine their influence on the utility optimization problem. In these experiments, 16 VMs are run, managed by a centralized ANN manager, for 100 control intervals, and the average utility value over all intervals has been measured. Each parameter is given a low and high value of 1000 and 3000, respectively, creating four combinations. Values lower than 1000 were not tested, since they already have shown acceptable search times. In Fig. 6.3, the average utility and search time in seconds of five runs are shown for the four combinations X_Y where X represents population size and Y the number of generations. Increasing any of the parameters to higher than 1000 does not have any effect on the average utility value, since their differences are statistically insignificant, as shown from an ANOVA statistical test. Since increasing each of the above parameters affects search time, the combination 1000_1000 is used, which has the lowest search time for the utility optimization problem.

The steps followed by the centralized ANN resource manager are given in Algorithm 3 without lines 4 to 8 and 11 to 13 which are used only in the distributed version. It starts by applying a fixed number of samples (configurations) to build initial models of performance and power. The initial training set is constructed by generating random samples to cover the whole range of resource allocations that can be given to VMs. Then, based on the built models, it finds an optimal configuration that is applied to the system. After a time interval of S seconds (S=30 in this case) set by sleep, it gets performance and power metrics that constitute a new sample that is added to the training set. Then, the models are trained with the new training set, and this process is repeated. This



(a) Utility for combination of population_generations



(b) Search time for combination of population_generations

Figure 6.3: Utility and search time for different combinations of population size and number of generations

repeated training makes it possible to adapt the models to workload changes. Since the training set/time grows by adding new samples, the training set is fixed to 50 samples, as a trade-off between prediction accuracy and training time, and adding a new sample means removing the oldest one.

6.4 Distributed Resource Manager

In the distributed version, the resource manager is divided into several resource managers, each responsible for resource allocation of a single VM. Its architec-

Algorithm 3: Resource manager algorithm

```
1  apply few predefined samples to build an initial model
2  while true do
3      based on models find the optimal allocation
4      for $j \leftarrow 1$ to $N$ do
5          broadcast(resource allocation of the VM)
6          receive(resource allocation of other VMs)
7          based on models find the optimal allocation
8      end
9      apply optimal allocation and sleep(S seconds)
10     get performance and power metrics
11     broadcast(resource allocation of VM)
12     receive(resource allocation of other VMs)
13     calculate_max_resource_capacity_every_M_intervals
14     add the new sample to the training set
15     train perf and power model with new training set
16 end
```

ture for three VMs is shown in Fig. 6.4. Each manager builds an ANN based performance model of its own VM that has only two inputs (CPU and memory allocation of one VM). It also builds an ANN based power model for predicting the power consumption of the physical machine. Since the power consumption of the physical machine depends on resource allocation to all VMs, the number of inputs to the power model is two times the number of VMs. In the distributed resource manager, the global utility given in Equation (6.1) is divided into local utility functions, given in Equation (6.5), one for each resource manager. This function depends on the performance of the corresponding VM and power consumption of the physical machine. Based on the performance and power models, the utility optimizer finds resource allocations given to the VM to optimize the local utility function.

$$U_i = \alpha V_i(perf_i) - \beta U_p(power) \tag{6.5}$$

Each resource manager controls the resource allocation of only one VM, therefore the number of resource allocation combinations to try for local utility optimization can be performed in negligible time. On the other hand, since the power consumption depends on the resource allocation to all VMs, each resource manager exchanges resource allocation decisions with other resource managers through a coordination procedure organized in a number of iterations where in each iteration a search for different resource allocation combinations to find the optimal utility is made. For each resource allocation combination, the resource manager uses ANN models to predict the performance of the VM and the power consumption of the physical machine. To predict power consumption, the resource manager fixes the resource allocations to other VMs to the values decided by other managers in the previous iteration (in the beginning, it sets them to some random values), while changing the resource allocations

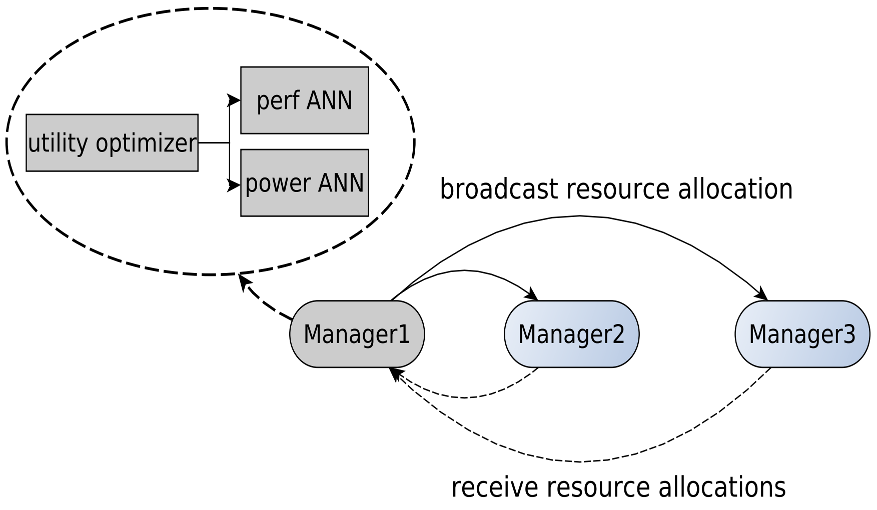


Figure 6.4: Distributed resource manager

of its VM. After this search, it selects the resource combination that has the maximum local utility value. At the end of the iteration, the resource manager exchanges this allocation decision with other managers to be used for the next iteration. This process of search and exchange is repeated for a number of iterations N as shown in lines 4 to 8 of Algorithm 3. The intuition behind this iterative process is that after predicting the power consumption for the current iteration, the resource allocation decision of other VMs may have changed, and in the next iteration after getting the new resource allocations, an improved prediction can be made. At the end of N iterations, the last predicted resource allocation that has the maximal utility is applied to the system.

The steps followed by each resource manager are similar to the centralized resource manager shown in Algorithm 3, but they are used for allocating resources to only one VM and there is the addition of lines 4 to 8 and 11 to 13. In order for the resource manager to train the power model, it gets the final resource allocation decisions of other resource managers corresponding to the measured power metric by exchanging allocation information with other managers through lines 11 and 12 in Algorithm 3.

Since resource managers act independently, they can allocate resources to VMs that overpass the maximum physical machine resource capacity. To overcome this problem, a maximum resource capacity is assigned to each resource manager. The method, which is shown in line 13 of Algorithm 3, to calculate the maximum resource capacity is executed by each resource manager every M time intervals ($M = 5$ in our case). Through the broadcast and receive operations in lines 11 and 12 of Algorithm 3, apart from resource allocations in the current time interval, resource managers exchange resource consumption information. By resource consumption, an exponentially weighted moving average of resource allocations of several time intervals in the past is meant. After having obtained resource consumptions of all VMs, the method proceeds as follows. In the beginning, each resource manager has the same maximum resource capacity. Then, at the time of calculation, the free resource amount (i.e., the difference between the maximum capacity and the resource consumption) is computed for each VM and also the total free resource amount. This total free resource amount is divided by the number of VMs to obtain the new free resource amount for each VM. By adding this new free resource amount to the VM resource consumption, the new maximum resource capacity is obtained for each VM. This method is executed by each resource manager to compute its own maximum resource capacity and also the capacity of others needed for the next calculation.

6.5 Implementation

A simulated environment in the C++ programming language is implemented and two type of applications are simulated. One is a High Performance Computing (HPC) application running CPU intensive calculations. Its performance, measured as the number of calculations per second, depends linearly on the CPU share allocated to the virtual machine. The other is a web application that has file I/O intensive workload serving web data from the disk. Its performance metric, the throughput, depends on both CPU and memory allocation to the VM. The relationship between web performance and resource allocation of CPU and memory used in these simulations is shown in Fig. 6.5. By increasing the resource allocation, the performance is increased until some point is reached where it is saturated, showing bimodal behavior that is typical for this kind of application [118]. For the web application, two types of workloads called workload1 and workload2 are simulated, that have the general form of Fig. 6.5, but differ with respect to the slope of the performance surface and the load intensity. For both applications, CPU and disk I/O utilizations the power model depends on are also simulated. CPU utilization of the HCP application is linearly depending on the CPU allocation, while for the web application it shows the same bimodal behavior as in the case of performance. Disk I/O utilization is high with low memory allocation and is low with high memory allocation. To the performance metric and the resource utilization generated from the models, some small random number generated from a normal distribution with zero mean is added to simulate measurement noise and uncertainty present in real environments.

Experiments are also performed in a realistic IaaS cloud environment based on the Xen virtualization technology. To measure application performance, the application heartbeat framework [50] is used that allows application code to emit heartbeats. The number of heartbeats per second represents application performance. The application performance sensors shown in Fig. 6.1 run as daemons inside each VM and are attached to an application through a shared memory mechanisms to gather performance metrics using the heartbeat framework. The main sensor that is part of the resource manager requests per-

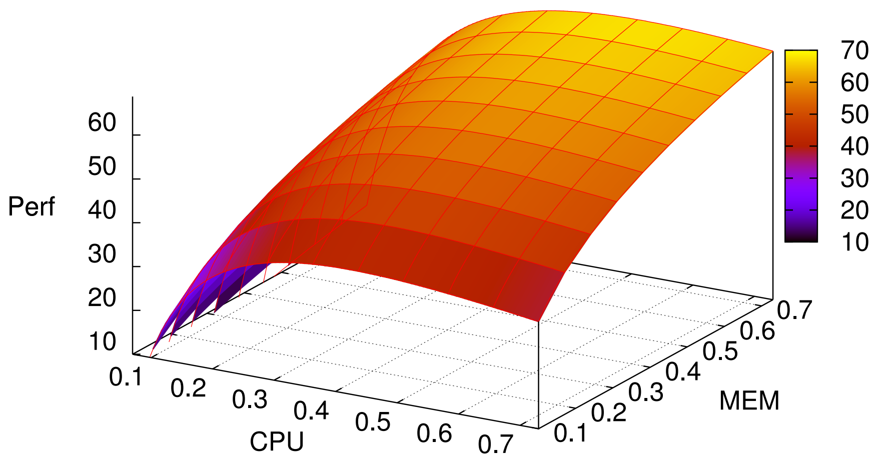


Figure 6.5: Web performance vs. resource allocation

formance metrics through the IP socket interface from all application sensors in each control interval. CPU and disk utilizations are gathered using xentop and iostat commands in Xen environment. The actuator sets the CPU and memory allocations using the xm sched-credit and xm mem-set commands of the Xen hypervisor.

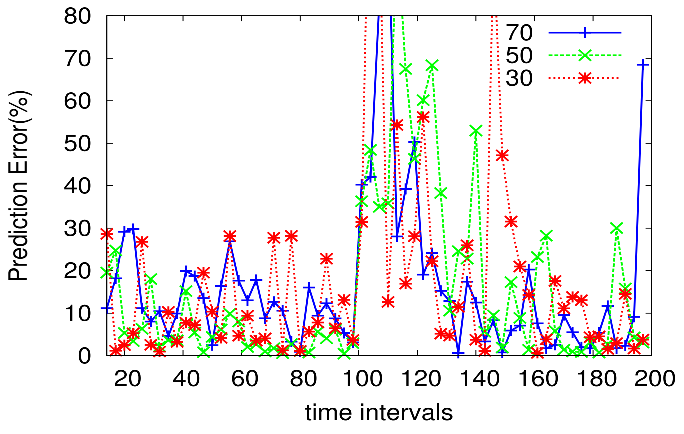
The approach is implemented using the Fast Artificial Neural Network library (FANN) [88] that offers multilayer artificial neural networks in the C programming language. The genetic algorithm has been implemented using the GAlib [116] C++ library. The coefficients α and β in Equation (6.1) are set to value 1. The following genetic algorithm parameters are set : crossover probability to 0.85 and mutation probability to 0.01. The distributed resource manager is implemented as a multi-threaded application where each manager is run by one thread in parallel with other managers. After testing several values, the parameter N of Algorithm 3 has been set to the value of 10, since increasing it further did not lead to further improvements.

6.6 Experimental Setup and Evaluation

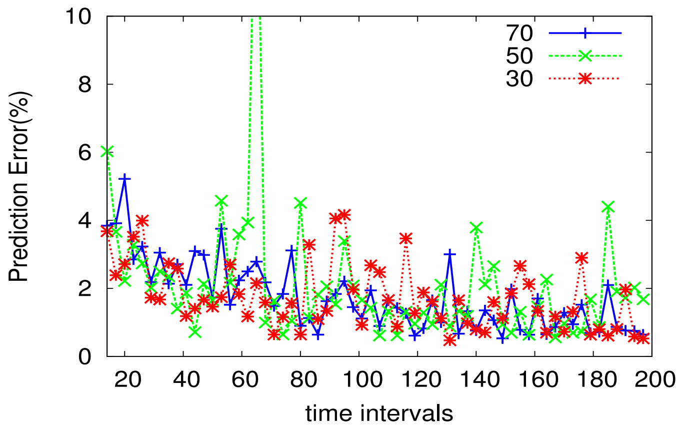
The simulation experiments are based on a simulated two quad-core CPU machine with a total of 800% CPU capacity and 16 GB of memory. The maximum power of the physical machine at full utilization has been set to 288 Watt, the power at idle state to 156 Watt and the maximum dynamic power at full utilization to 132 Watt. Since the CPU is the biggest power consumer, its dynamic power at full utilization is set to 106 Watt and the dynamic power of the disk at full utilization to 26 Watt. These are typical values found in real servers [107].

For the experiments in a realistic IaaS environment, a physical machine is set up that has two dual-core AMD Opteron 2.4 GHz processors and 8 GB of RAM. It runs Xen 4.1 and the Ubuntu 12.04 operating system in the Dom0 and DomU VMs. The following benchmarks are used: a) cpu_bench is a CPU intensive benchmark created for this work that performs mathematical calculations such as computing the factorial of numbers, b) filebench [36] is a file system benchmark that emulates a number of applications such as web and file servers. The maximum power of the physical machine at full utilization is set to 212 Watt, the power at idle state to 132 Watt, the maximal dynamic power at full utilization to 80 Watt, the CPU dynamic power at full utilization to 56 Watt and the dynamic power of the disk at full utilization to 24 Watt. These values have been derived from the power values of a simulated machine, taking into account the fact that the physical machine has half of the CPU cores of the simulated machine.

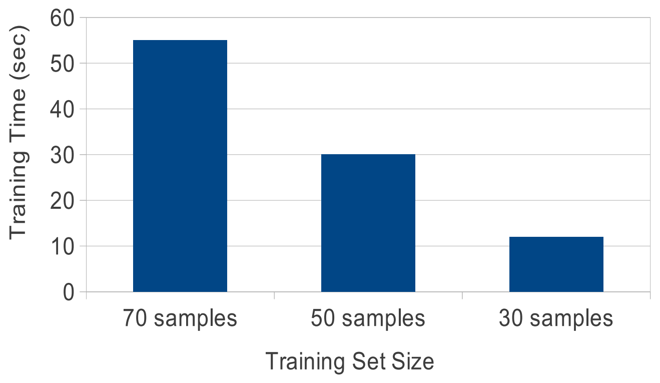
Simulations are performed with 4, 8 and 16 VMs where in each case half of them run the HPC application and the other half runs the web application. The simulations are restricted to 16 VMs, since the focus is on a single physical machine, and running 16 VMs is reasonable and sufficient to study the behavior of the different resource allocation approaches when the number of VMs is increased. For 4 VMs, a maximum of 2 CPU cores to all VMs is assigned, for 8 VMs 4 CPU cores and for 16 VMs 8 CPU cores. For simplicity, the control VM and its power consumption is not simulated, but this has been considered in the experiments in a realistic IaaS environment. For each of the cases above, two workload mixes are considered : a) a steady mix and b) a periodic mix. In the steady workload mix, the experiment is run for 200 time intervals where from interval 0 to 100 all web applications run workload1 and from 100 to 200 they run workload2. In the periodic workload mix, the web application workload changes between workload1 and workload2 every 24 intervals. It is experimented with 4 resource allocation techniques abbreviated as follows: ANN is the centralized ANN resource manager, DANN is the distributed ANN resource manager, STATIC is an allocation where the total resource capacity is distributed equally to all VMs, and DANNNOC is a distributed ANN resource allocation technique without coordination between the ANNs where the power model is based only on resource allocation to the corresponding VM. For each combination of the allocation techniques, the number of VMs and the workload mixes, the experiment is repeated five times and the collected results were exposed to ANOVA and TukeyHSD statistical tests.



(a) Prediction error of performance model



(b) Prediction error of power model



(c) Training time of performance and power models

Figure 6.6: Prediction error and training time for different training set sizes

6.6.1 Simulation Experiment: Results

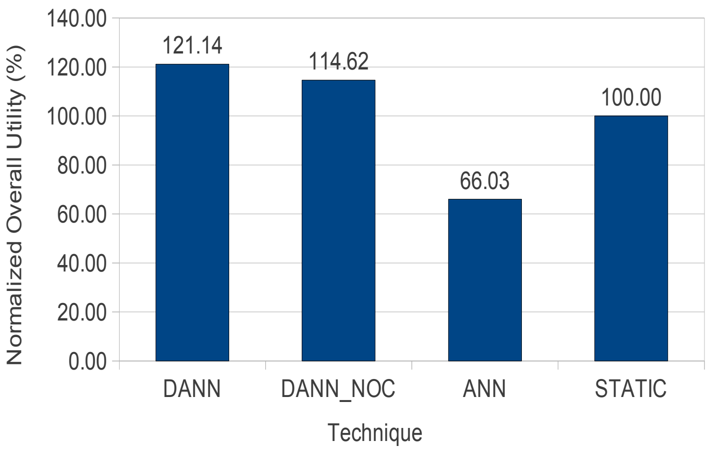
6.6.1.1 ANN Model Prediction Accuracy and Training Time

First, simulation experiments are performed to determine the size of the training and the prediction accuracy of the model. In this experiment, 16 VMs are executed with a steady workload mix in a single simulated machine. In Fig. 6.6(a), the average performance prediction error over all VMs performance models for DANN is shown. Three different training set sizes of 70, 50 and 30 samples are tested. For all training set sizes, the average prediction error, excluding the time of adaptation to new workload from 100 to 130, is less than 12%. On the average, a slight decrease of the prediction error is evident when the training set size is increased from 30 to 50 and to 70. Since the training time increases when the training set size is increased, as shown in Fig. 6.6(c), a training set size of 50 samples has been chosen as a trade-off for the rest of the experiments. In Fig. 6.6(a), the time needed to achieve an acceptable prediction error of the performance model after a workload change in time interval 100 can be observed: it is around 30 time intervals. The power prediction error is not affected by the training set size, and all sizes achieve an error of less than 3%. This is because the dynamic power change is small compared to the actual power, making it possible to build an accurate ANN power model.

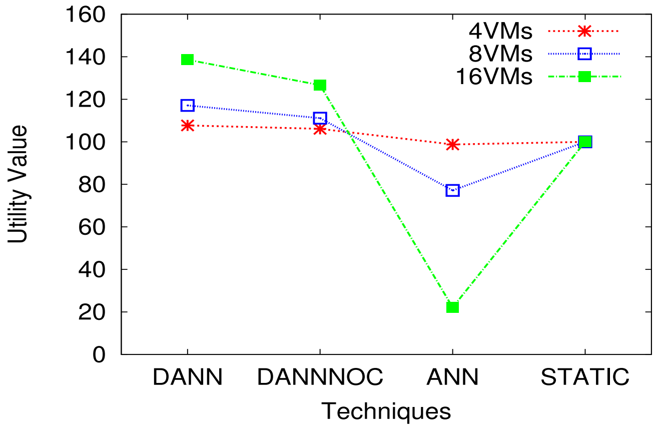
6.6.1.2 Utility, Performance and Power Results

In Fig. 6.7(a), the cumulative utility is shown, which is the sum of the global utility values for all time intervals, for each allocation technique averaged over all numbers of VMs and workload mixes. The cumulative utility is shown as a percentage of the STATIC allocation utility. DANN achieves a higher utility than all other approaches. Although DANNNOC has a better utility than the STATIC and ANN techniques, it is worse than DANN with statistical significance, indicating that coordination between ANNs is important to perform better. In Fig. 6.7(b), for each technique it is shown how the cumulative utility changes with the number of VMs. With an increased number of VMs, DANN becomes more effective than the other techniques.

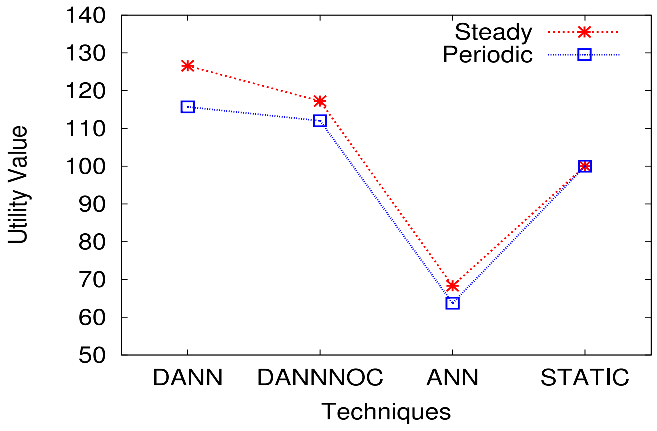
This is because by increasing the number of VMs, the uncertainty about the power modeling without considering all VMs increases, making the coordination between the ANNs a necessity for more accurate power modeling. In Fig. 6.7(c), it is shown how the cumulative utility changes with the workload mix. The utility achieved for each technique with a periodic workload mix is lower than with a steady workload mix. This is because with a periodic workload, the accuracy of the performance and power models decreases, since they need to adapt frequently to workload changes. In Fig. 6.8(a), the overall performance for each technique averaged over all numbers of VMs and workload mixes is shown. DANN has a better performance than the other techniques (with sta-



(a) Overall utility

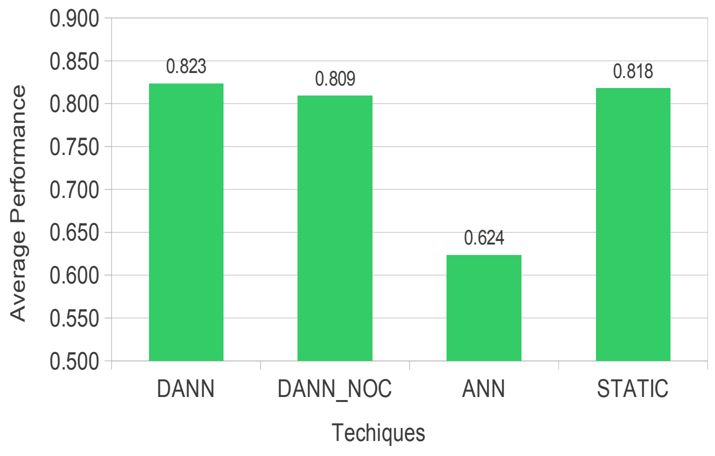


(b) Utility for different number of VMs

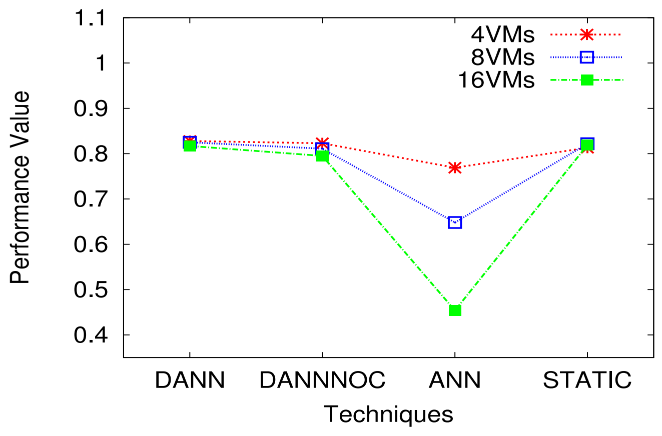


(c) Utility for different workload mixes

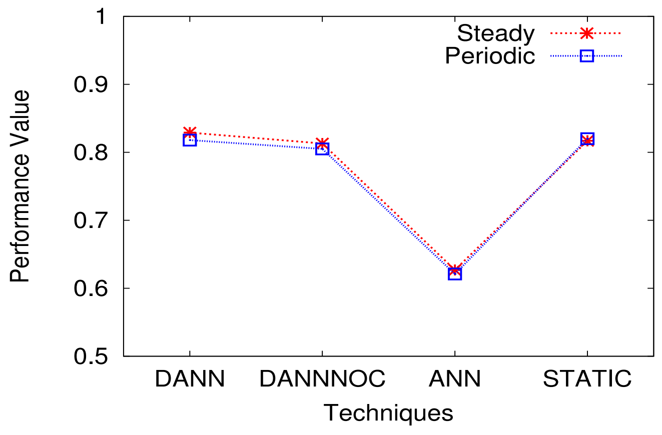
Figure 6.7: Overall utility and utility for different number of VMs and workload mixes.



(a) Overall average performance

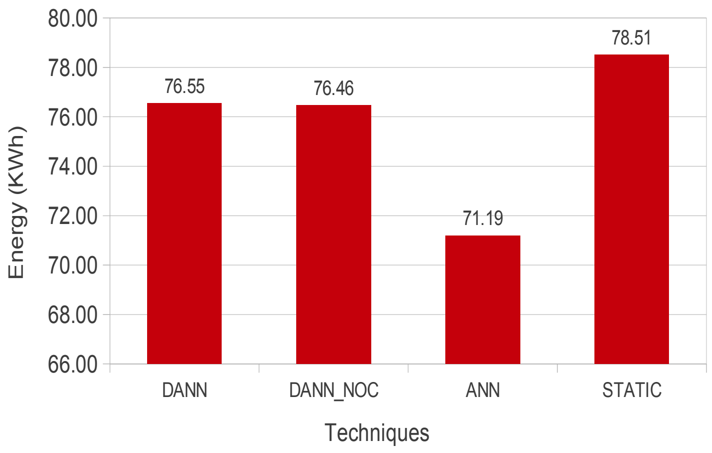


(b) Performance for different number of VMs

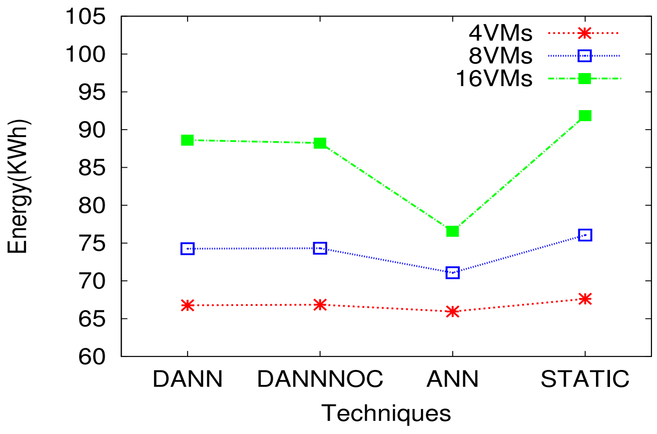


(c) Performance for different workload mixes

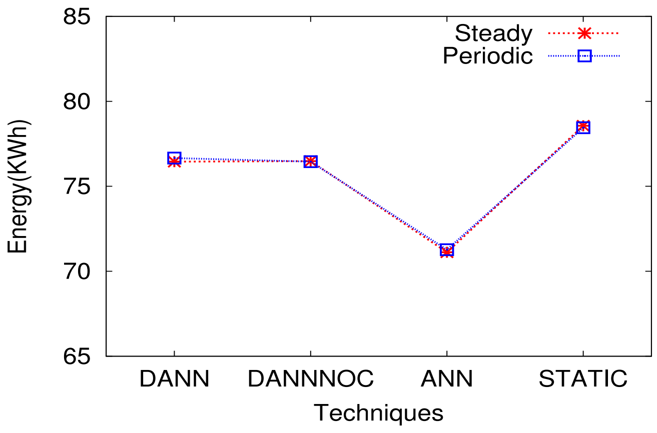
Figure 6.8: Overall average performance and average performance for different number of VMs and workload mixes.



(a) Overall Energy



(b) Energy for different number of VMs



(c) Energy for different workload mixes

Figure 6.9: Overall energy and energy for different number of VMs and workload mixes.

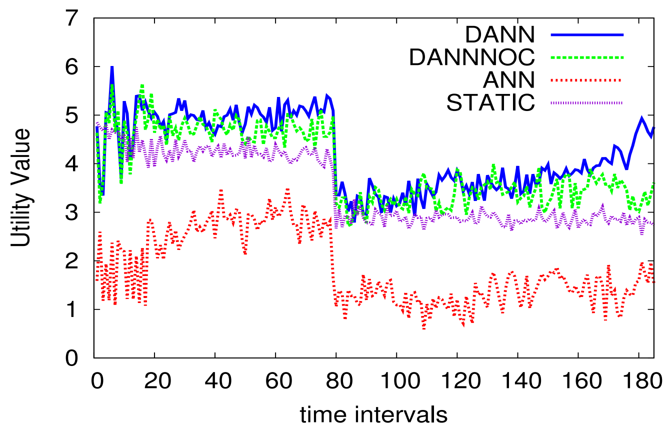
tistical significance). Although DANN has a slightly better performance than STATIC, it consumes less power, resulting in a higher utility. In Fig. 6.8(b), it is shown how the performance changes with the number of VMs. The performance of DANN is not influenced by increasing the number of VMs, which is not the case for ANN. In Fig. 6.8(c), it is shown how the performance changes with the workload mix. With statistical significance, the performance achieved by DANN is only slightly reduced by changing the workload type.

In Fig. 6.9(a), for each technique the energy consumption during the experiment time for 200 machines averaged over all numbers of VMs and workload mixes is shown. DANN achieves energy savings compared to STATIC with statistical significance. Although DANN consumes the same energy as DANNNOC, it achieves a higher utility due to the higher performance than DANNNOC. The least energy consumption is achieved by ANN, but with the price of a lower performance. In Fig. 6.9(b), it is shown for each technique how the energy changes with the number of VMs. The energy savings of DANN compared to STATIC become larger with an increased number of VMs. This is because the range of dynamic power consumption becomes larger with an increased number of VMs, increasing the potential of the technique to influence the energy consumption. In Fig. 6.9(c), it is shown how the energy changes with the workload mix. The energy consumption of each technique is not influenced by the workload mix, indicating the robustness of the techniques to keep the same level of energy savings.

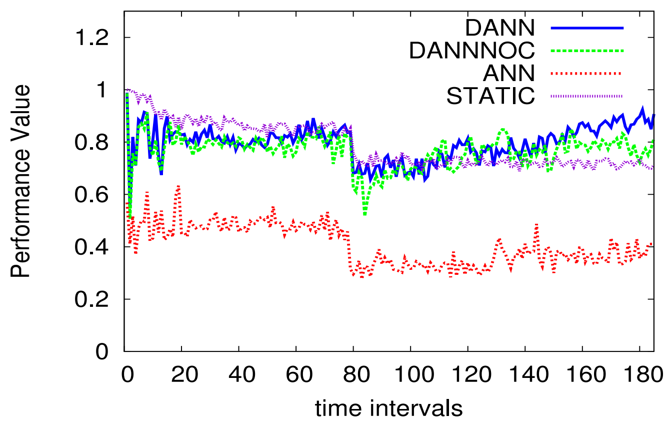
6.6.2 Realistic IaaS Experiment: Results

Experiments are performed in a realistic Xen environment with 8 VMs, half of them running cpu_bench and the other half filebench. 8 VMs are pinned to 3 CPU cores and are given 3200 MB of memory, while Dom0 is pinned to the remaining CPU core. To each VM is assigned one VCPU. The experiment has been run for 185 time intervals, and in interval 80 it is triggered a workload change emulated by increasing the SLA performance level of 4 VMs. The experiment has been run 5 times and the average results are shown.

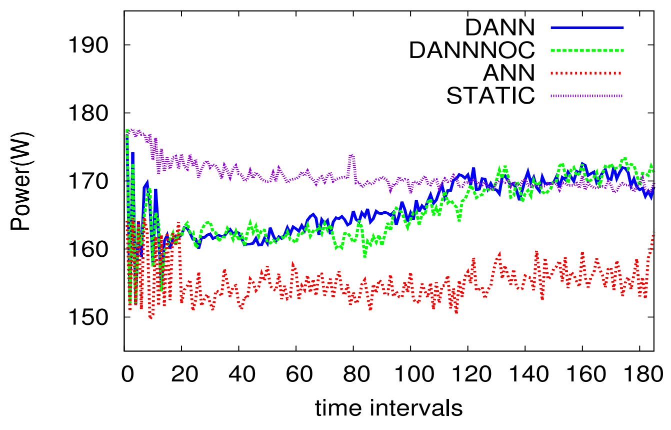
Fig. 6.10(a), 6.10(b) and 6.10(c) show utility, average performance and power consumption over time for each technique. DANN has a better utility value than the other approaches. Although the difference with DANNNOC is not statistically significant for most of the experimental time, there are intervals where it has a better utility, such as in the intervals (60:80) and (170:185). Although most of the time DANN achieves the same levels of average performance as DANNNOC and STATIC, in the interval (150:185) DANN achieves better values. This is because in the second half of the experiment, the performance SLA levels are increased, making it more difficult for DANNNOC and STATIC to satisfy them. With respect to power, although DANN achieves the same power consumption as DANNNOC, it achieves much lower power levels than STATIC in the first half of the experimental time and increases it during the



(a) Utility



(b) Average performance



(c) Power consumption

Figure 6.10: Utility, average performance and power consumption for 8 VMs

second half. During this interval (0:80), the SLA performance levels can be achieved with much less power consumption than using STATIC, while in the second half the power consumption should be increased to meet the new SLA performance level.

6.6.3 Overhead Results

The overhead of DANN comes from two factors: a) the coordination time in lines 4-8 of Algorithm 1 and b) the models' training times in line 15 of Algorithm 1. Running the resource manager on the physical machine with 4 CPU cores and managing 16 VMs, it gives, on the average, 2.5 sec for the first factor and 30 sec for the second factor, showing the feasibility of the resource manager in practice.

6.7 Summary

In this chapter, a resource management approach for VM resource allocation in IaaS clouds has been presented. It finds an adequate performance-power trade-off by expressing the two conflicting objectives of performance and power in a utility function and optimizing it using ANN based performance and power models. To cope with a potentially increased number of VMs, a distributed resource management approach has been developed where each ANN is responsible for modelling performance and power of a single VM, while exchanging information with other ANNs to coordinate resource allocation. Simulated and real experiments show that the distributed ANN resource manager achieves better utility values and performance-power trade-offs than a centralized ANN approach, a distributed non-coordinated version and a static allocation approach.

In the future, the disk and the network can be included as resources to be allocated. It is also planned to apply the approach in more complex environments with multi-tier applications and multiple physical machines where the coordination procedure for power modelling could be applied for coordinating multi-tier VMs running on different physical machines.

7 Cross-Correlation Prediction for Virtual Machine Resource Allocation Using Support Vector Machines

7.1 Introduction

Many cloud providers assign resources to VMs statically according to the peak demands. This static allocation leads to a waste of resources, higher costs and inefficient use of computational resources, probably leaving some applications starving for resources and others having more resources than needed. A better way is to allocate resources in an elastic manner dynamically according to the current demand, by just assigning the minimum amount of resources required to satisfy application performance. However, this is a challenging task, since resource demand is dynamic and changes quickly over time. Ideally, what is required is a proactive resource allocation approach that predicts resource usage in advance and allocates resources according to the current demand in an automatic way without human intervention.

In this chapter, an approach called Automatic Proactive Resource Allocation (APRA) is presented that proactively allocates resources to VMs by predicting resource usage demand using Support Vector Machine (SVM) for time series forecasting. Since there are interdependencies between VMs of a multi-tier application, and the resource usages of resources of multiple VMs of the same application are correlated with each other, cross-correlation prediction is applied by making predictions for multiple resources of a multi-tier application at the same time. The focus of the work is on two resources to be assigned to VMs, namely CPU and memory, since current virtualization technologies do not offer adequate support for the allocation of other resources, such as network or disk bandwidth. Based on predictions of resource usage for the next interval, the proposed approach allocates the most adequate amount of resources to VMs to satisfy the demand. The approach is general and non-intrusive, since its resource allocation decisions are based on resource usage metrics monitored outside of the VMs, and are independent of the type of application. Experimental results of the evaluation of a multi-tier application cloud environment testbed running web serving benchmarks of CloudSuite [34] show better application performance and prediction accuracy than a non-cross-correlation approach for resource demand prediction.

First, the APRA design is described. Then, implementation issues and experimental results are presented. In the end, the chapter is concluded with a summary of the approach. The work described in this chapter has been published in [83].

7.2 Automatic Proactive Resource Allocation

In this section, the design of the APRA approach is presented by starting with an overview of its architecture and its main components. Then, the application of the SVM approach for cross-correlation prediction of resource demand is described. Finally, it is presented how the resource allocation decision to multiple resources is made based on resource demand prediction.

7.2.1 APRA Architectural Overview

The architecture of APRA is shown in Fig. 7.1. It is applied in an IaaS cloud environment composed of several physical nodes on top of which several VMs are running. Each VM is running one tier of a multi-tier application. Fig. 7.1 shows the case of three physical nodes and three two-tier applications running in total six VMs spread randomly over the physical nodes. The components of APRA are as follows.

In each physical node, a monitoring daemon called *mond* is running that is responsible for gathering, in each time interval, resource usage metrics of all VMs running on the corresponding physical node. Each multi-tier application has a management module called *Agent* that is responsible for the resource allocation of its own VMs. The *Agent* does not necessarily have to run on the same physical machine. In each time interval, the *Agent* gets resource usage metrics from the *mond* daemons of each physical node that its VMs are running on. Then, it predicts the resource usage demand for the next interval for all its VMs. Based on this prediction, it determines the resource allocations to be assigned to them for the next interval. In each physical node, a module called *Arbiter* is running that gathers resource allocation assignments from all

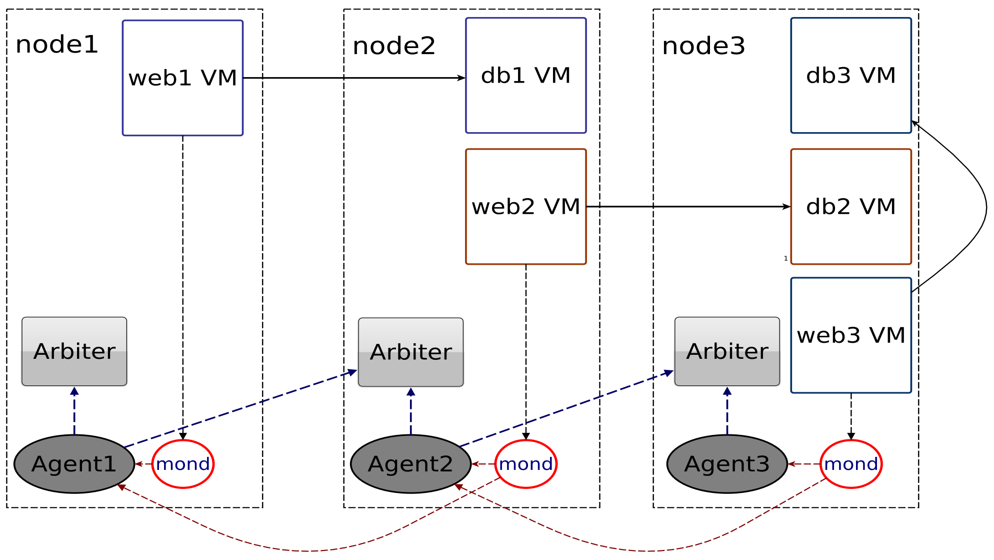


Figure 7.1: APRA architecture.

Agents that have VMs on the physical node, decides the final resource allocation assignment for all VMs, and actually applies them through the Virtual Machine Monitor (VMM). Its main responsibility is to resolve conflicts that result from assignments that overpass the resource capacity of the physical node.

This architectural arrangement of distributing the resource allocation decisions on several *Agents* running on different physical nodes has the benefit of providing scalability of the resource management in a large scale cloud infrastructure.

7.2.2 SVM for Cross-correlation Prediction

One of the responsibilities of the *Agent* module is to take resource usage data from the *mond* daemons, build a time series of resources of all its VMs, and based on that predict the resource usage for the next time interval. This prediction is a function of a number of past time series usage values called a time window. This is given mathematically by:

$$x(t+1) = f(x(t), x(t-1), x(t-2), .., x(t-16)) \tag{7.1}$$

where $x(t)$ is resource usage at time interval t, $x(t+1)$ is the predicted resource usage for the next interval and $f()$ is the prediction function.

The prediction function is learned using a machine learning approach, i.e., in this proposal an SVM. The motivation for using this technique are its advantages over alternative techniques, as shown by other research work [102] and summarized below:

- Non-parametric method avoiding the need define a model structure beforehand.
- Applicable to non-linear models and non-stationary processes.
- Guaranteed to find the global optimum, unlike artificial neural networks that can get stuck to local minima.
- A small number of parameters are required to tune the algorithm.
- Better generalization by avoiding over-fitting.

SVMs, as other supervised machine learning techniques, undergo a training phase by applying past time series samples to generate a model to be used for prediction. Before the time series data is applied to the learning algorithm, a transformation phase is needed to make it more suitable for learning by removing the temporal ordering of input samples by creating additional inputs called "lagged" variables. The training is performed repeatedly for each time interval in order to adapt to changing characteristics of time series. Since there are dependencies between tiers of a multi-tier application, the resource usage time series of VMs belonging to the same application are cross-correlated, meaning that the usage prediction for one resource depends on past resource usages of other VMs of the same application. To take this cross-correlation into account, The SVM algorithm is applied to model multiple time series simultaneously.

7.2.3 Proactive Resource Allocation

The other responsibility of the *Agent* is to determine the resource allocation to be given to all resources of all VMs of the application it is responsible for. This decision is made in discrete time intervals where in each interval, the resource allocation to be given to each resource for the next interval is determined. This decision is based on resource usage prediction of the next interval as a result of the SVM algorithm. More specifically, the amount of resources allocated is estimated as the resource usage prediction plus a small margin of 5% of the resource capacity. This makes it possible to allocate just the needed resources to keep the cost to a minimum and to leave room for any possible prediction error in order to keep the performance to acceptable levels.

Due to prediction errors, it can happen that the resource amount allocated is less than the usage demand, leading to poor performance and influencing the future predictions. To mitigate this problem, in each control interval, if resource starvation is encountered, meaning that the difference between the resource allocated and resource usage is less than 5% of the resource capacity, the resource allocation margin for the next time interval is doubled to 10% of the resource capacity. This leaves room to satisfy the resource demand and get realistic usage data for future predictions. If resource starvation is not encountered in a time interval, the normal resource allocation margin of 5% is set for the next interval.

After determining the resource allocations for all its VMs according to the algorithm above, the *Agent* submits them to the corresponding *Arbiters*. The *Arbiter* collects, in each time interval, resource allocations from all *Agents* that have VMs running on its physical machine and actually applies them through the VMM. In case the sum of the resource allocations of all VMs running on the physical machine is greater than the physical resource capacity, the *Arbiter* redistributes resource allocations to VMs proportionally to their requirements. Each VM gets the final resource allocation after redistribution according to the formula:

$$A = \frac{Alloc}{SumAlloc} * Capacity \tag{7.2}$$

where A is the final allocation, $Alloc$ is the preliminary allocation, $SumAlloc$ is the sum of all VMs' preliminary allocations and $Capacity$ is the total resource capacity.

7.3 Implementation

All APRA components run in Dom0 of the physical nodes and are written in Java, except for the *mond* monitoring daemons that are written in the C language. The *Mond* daemons use the libvirt API to get CPU usage data in each time interval. Since there is no simple way in current VM technology to measure memory consumption of a VM outside of it, it is implemented a small daemon called *mmond* running inside each VM to get memory usage data from the /proc/meminfo file system in Linux. *Arbiter* sets CPU and memory allocations through the xm sched-credit and xm mem-set commands of the Xen hypervisor. The *agent* is implemented as a two-threaded application. One thread communicates with *mond*, maintains usage time series and predicts new usage values for the next interval. The other thread makes allocation decisions and communicates with the *Arbiters*. The *Agents* use the SVM implementation of the WEKA [44] machine learning framework for time series cross-correlation forecasting of resource demand. All communication between the components of APRA is based on the IP socket interface. The time interval for sampling resource usage data, training the forecasting model and applying resource allocation decisions, is set to 10 seconds. The time window interval of the past time series usage data is set to 16 samples according to the results of some experiments and to keep the training time to a minimum.

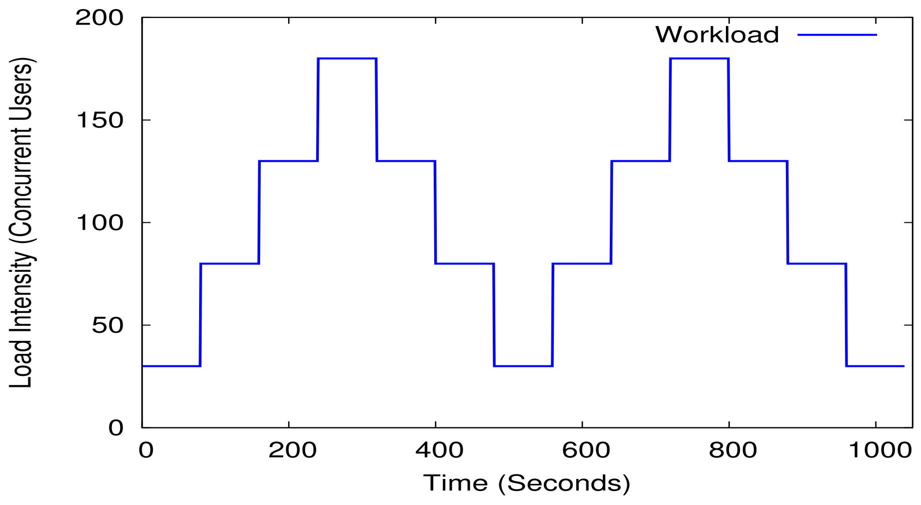
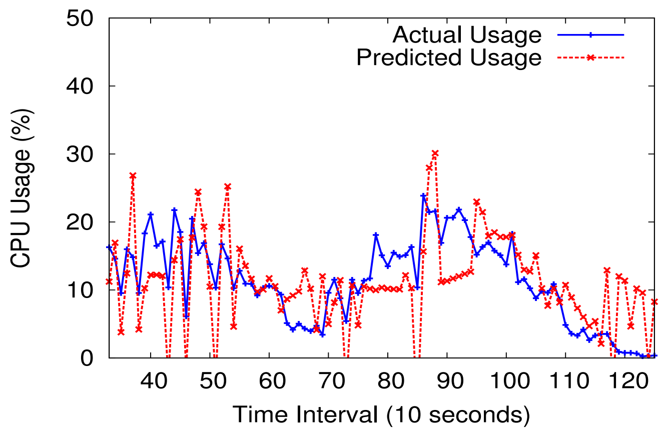


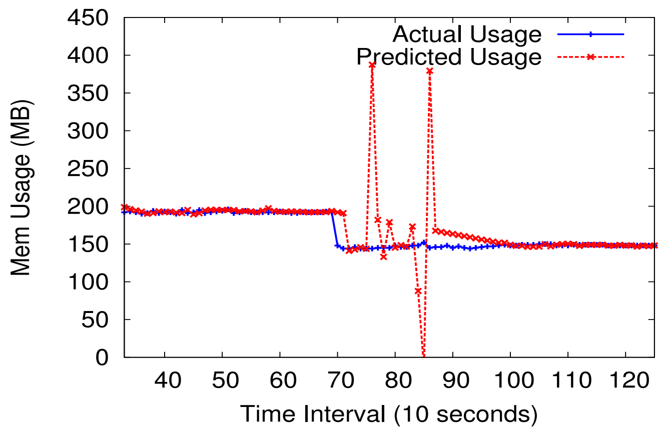
Figure 7.2: Workload.

7.4 Experimental Results

A testbed composed of 3 physical nodes and 3 two-tier applications (App1, App2, App3) is set-up as shown in Fig. 7.1. Each tier runs inside one VM and the VMs of the same application can run on different physical machines. CloudSuite [34] web-serving benchmark is used as a representative multi-tier application. APRA is tested with the two-tier configurations where the first tier is the web front-end tier that implements the *olio* web application and the second is the *mysql* database tier that stores information about social events. Three clients are running on the first node that generate load for the three applications through the Faban workload generator. The physical machines are blade servers with two Intel quad-core Xeon 2.00 GHz CPUs with a total of 8 CPU cores and 4 GB of RAM. Xen 4.1.2 Hypervisor is used as the virtualization platform for managing VMs and used the Ubuntu 12.04 OS with 3.8.0-29-generic Linux kernel in Dom0 and DomU VMs. The machines are connected over a 1 Gbit/sec Ethernet. In the experiments, Dom0 is pinned for all physical nodes to 2 CPU cores and is given 2 VCPUs. Web1 VM is pinned to one CPU core and is assigned 1024 MB of RAM, db1 and web2 VMs are pinned to 1 CPU core and are assigned 1024 MB of RAM and finally db2, web3 and db3 VMs are pinned to 2 CPU cores and assigned 1536 MB of RAM. It is assigned to all application VMs 1 VCPU. The workload generated by the clients for the three applications with varying intensity lasting for about 17 minutes is shown in Fig. 7.2. On the y-axis, the workload intensity set by the number of concurrent users that send requests to the web application is shown. The workload starts with 30 concurrent users at time 0 seconds and changes every 80 seconds by $\pm$ 50 users.



(a) Actual and predicted CPU usage for web front-end VM without cross-correlation

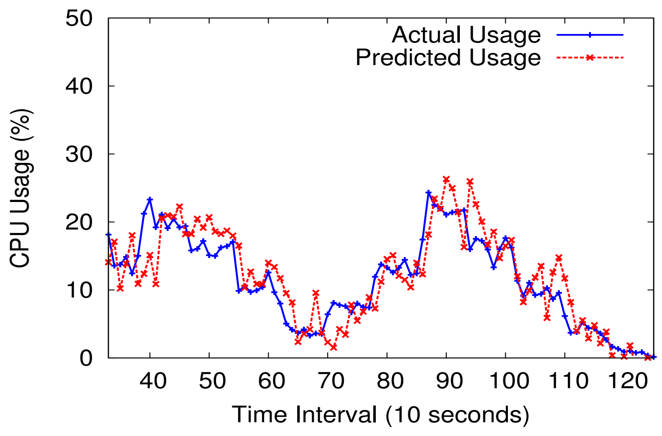


(b) Actual and predicted memory usage for web front-end VM without cross-correlation

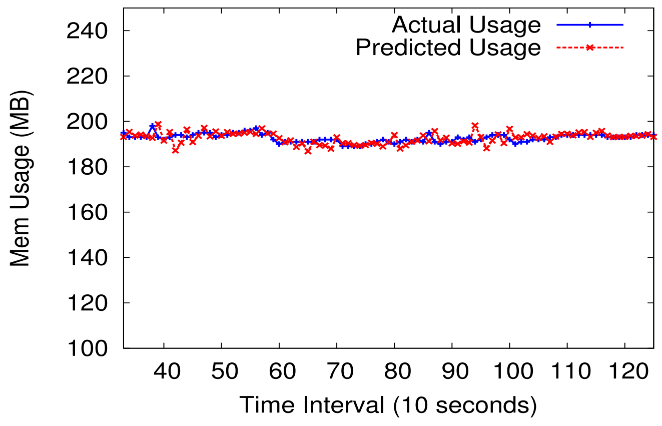
Figure 7.3: Actual and predicted resource usage of the web front-end VM without cross-correlation.

Table 7.1: Prediction errors for 2 approaches

Resource	**Approach**	**MAE**	**RMSE**	**MAPE**
WEB CPU	*Cross-Correlation*	2.78	3.7	0.26
	No-Cross-Correlation	4.52	5.81	0.53
WEB MEM	*Cross-Correlation*	2.29	3.26	0.011
	No-Cross-Correlation	4.37	10.95	0.026
DB CPU	*Cross-Correlation*	0.48	0.66	0.25
	No-Cross-Correlation	0.79	1.17	0.48
DB MEM	*Cross-Correlation*	0.5	0.71	0.001
	No-Cross-Correlation	9.44	52.54	0.038



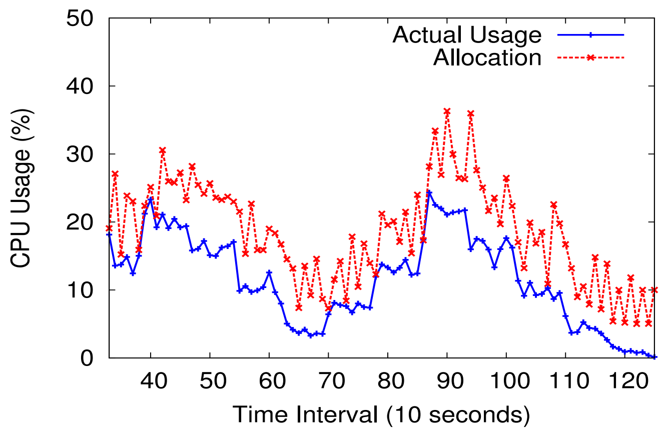
(a) Actual and predicted CPU usage for web front-end VM with cross-correlation



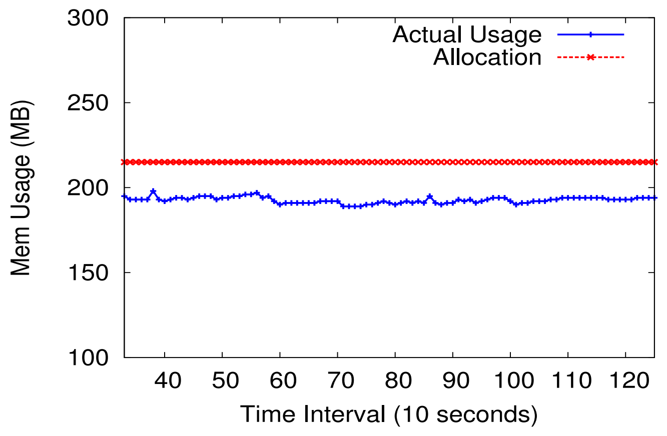
(b) Actual and predicted memory usage for web front-end VM with cross-correlation

Figure 7.4: Actual and predicted resource usage of the web front-end VM with cross-correlation.

Different aspects of two approaches are compared: a) one that applies SVM to make predictions of usage time series of each resource separately and b) the other one that makes predictions of multiple usage time series of the same multi-tier application simultaneously to take cross-correlation into account. Fig. 7.3 and Fig. 7.4 show the actual and predicted CPU and memory usage of the web front-end VM of App2 with using cross-correlation prediction and without using it. Similar results are taken also for the CPU and memory resources of the database VM, but these results are not shown. It is evident especially for the CPU resource that applying cross-correlation leads to better resource usage predictions than without it. This is because there are correlations between resource usage time series of the same multi-tier application as the result of the interdependencies between its VMs, thus making cross-correlation time series predictions a necessity for getting better results.



(a) Actual and allocated CPU usage for web front-end VM

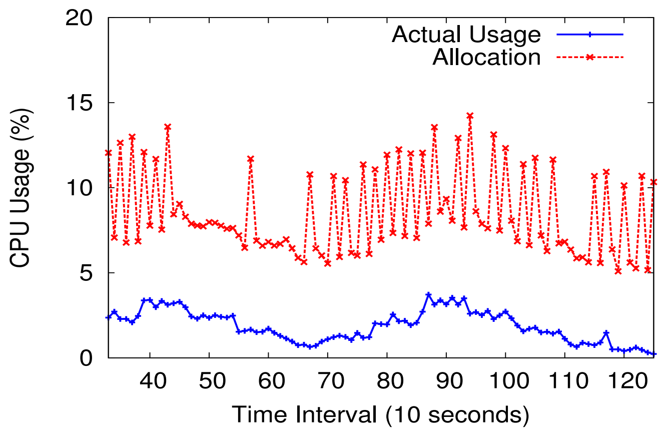


(b) Actual and allocated memory usage for web front-end VM

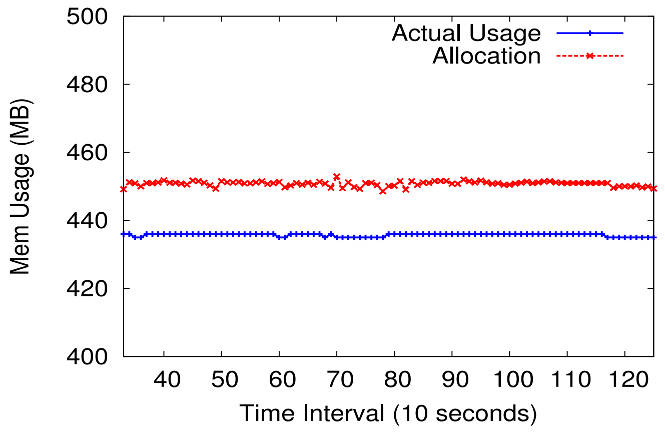
Figure 7.5: Actual and allocated CPU and memory resource usage of web front-end VM with cross-correlation prediction.

To verify the prediction accuracy of the two approaches quantitatively, Table 7.1 shows the results for three widely used metrics of prediction errors for App2: MAE, RMSE and MAPE. For each resource, smaller prediction errors are seen with cross-correlation than without it.

To understand how proactive resource allocation based on cross-correlation prediction works, Fig. 7.5 and Fig. 7.6 show the used and allocated resources determined by APRA for the two VMs of App2. APRA allocates the minimum amount of resources needed and changes it dynamically according to the demand. This keeps resource costs to a minimum while satisfying application performance. Fig. 7.5(a) shows that only in a few cases the allocation is equal to the usage demand, while most of the time it is within some margin above the demand, resulting in good application performance.



(a) Actual and allocated CPU usage for database VM



(b) Actual and allocated memory usage for database VM

Figure 7.6: Actual and allocated CPU and memory resource usage of database VM with cross-correlation prediction.

Finally, it is presented how the two proactive resource allocation approaches, one based on cross-correlation prediction and the other one based on separate time series predictions, affect application performance. In Table 7.2, the average and the 90th percentile of the response time in seconds for the three applications with cross-correlation prediction and without it is shown. Resource allocation with cross-correlation prediction achieves lower application response times on all applications except for the App1 application. The almost equal performance of the two approaches for the App1 application is explained by the fact that its performance is degraded by the interference of workload generation client VMs running on the same physical node with its web front-end VM. The better application performance with cross-correlation approach is explained by the fact that it achieves better prediction accuracy and therefore better allocations.

Table 7.2: Response times of three applications for the two approaches

Appl.	**Approach**	**Avg RespTime**	**90th% RespTime**
APP1	*Cross-Correlation*	2.05	5.82
	No-Cross-Correlation	2.43	6.25
APP2	*Cross-Correlation*	0.25	0.45
	No-Cross-Correlation	0.4	0.87
APP3	*Cross-Correlation*	0.29	0.63
	No-Cross-Correlation	0.4	0.95

7.5 Summary

In this chapter, Automatic Proactive Resource Allocation (APRA) based on multiple resource demand predictions using SVM has been presented. APRA is motivated by the interdependencies that exist between resources of multiple VMs of the same multi-tier application. This necessitates a cross-correlation resource demand prediction approach for better accuracy. Based on predicting CPU and memory resource demand, APRA makes allocation decision to lower costs and keep application performance to acceptable levels. Experimental results with the CloudSuite web serving multi-tier application benchmark have shown that the cross-correlation prediction approach achieves better prediction accuracy and resource allocation decisions compared to the non-correlation prediction approach.

As future work, it is planned to include VM live migration as another resource allocation mechanism to handle the case when the predicted physical machine resource demand overpasses the total capacity. Furthermore, the inclusion of VM replication as another resource allocation mechanism is an interesting approach for future work.

8 Conclusions

This chapter summarizes the research challenges encountered in cloud computing resource management and the different VM resource allocation approaches developed. Finally, a number of directions for future work are outlined.

8.1 Summary

Cloud computing, with its supporting technology of virtualization, is changing the computation model and the way IT infrastructures are accessed. This new computation model is securing many benefits for the cloud consumer and the cloud provider. For the consumer, it offers as many needed resources on demand, in a pay-as-you-go fashion, resulting in cost-efficient solutions and optimal performance. For the provider it offers efficient resource utilization, resulting in reduced operational and power costs. The key element to provide these promising features is a dynamic and autonomic resource allocation, for consumer applications and physical machines, through the use of VM technology.

Although progress has been made and many VM resource allocation techniques have been developed, effective and optimal resource management still remains an open research challenge. In the course of this work several VM resource allocation approaches are presented to overcome some of the limitations and drawbacks of existing solutions, operating both, at a global data center level and a local physical machine level of a cloud computing infrastructure.

In order to reduce power and management costs of the cloud infrastructure, dynamic VM consolidation through live migration according to workload is a promising technique of allocating VM resources at the global data center level. Most of the approaches to date base their VM live migration decisions on low-level utilization metrics and thresholds that do not capture and guarantee high level metrics, such as application performance or cloud provider profit. They also rearrange mapping of VMs to physical machines using fixed heuristic based policies without offering the required flexibility of later changing the policy from consolidation to load balancing and vice versa. In this dissertation, a VM dynamic consolidation and load balancing approach based on utility function optimization has been presented. The resource allocation approach is composed of two tiers: (1) local utility function optimization, through VM share allocation and, (2) global utility function optimization through VM live migration. In this case, the global utility function value represents the profit received by the provider from the cloud computing infrastructure. The utility function offers a natural way to capture the trade-off between application performance and infrastructural costs, by expressing two conflicting objectives in the same function. At the heart of the approach lays a simple heuristic method for global level utility function optimization through VM live migrations. The novel characteristic of the approach, and what cloud providers are mostly interested in, is the VM resource allocation based directly on high-level metric, such as cloud provider profit that is represented by utility function value. Furthermore, resource allocation based on utility optimization offers more flexibility in changing the resource allocation policy by changing the weights of utility function components. As experiments show, the approach can find an adequate trade-off between application performance and cloud infrastructure costs.

Another important resource allocation mechanism for cloud computing resource management is vertical scaling of VM resources. This mechanism holds when shares of different resources of each VM, running on physical machines, can be changed dynamically at run time. Most of the existing approaches apply vertical scaling resource allocation to VMs to only one resource such as the CPU. In this respect, a feed-back control theory approach has been developed to allocate multiple resources to VMs such as CPU, memory, disk and network I/O bandwidth. Its goal is to keep resource utilizations to certain percentage and in the case of resource contention it applies an algorithm to maximize a utility function, by giving more resource shares to higher utility VMs. The utility function, given as the difference between VM utilities and resource costs, represents cloud provider's profit for one physical machine. On the other hand, the VM utility represents the charged monetary value of the cloud consumer and depends on application performance and resource consumption. This is in contrast to commercial models, such as Amazon EC2 [4], where the consumer is charged only for the resources consumed.

Allocating VM resources by keeping utilization to certain levels does not guarantee application performance since the correlation between resource utilization and performance in shared virtualized infrastructures is not clear and can change over time. Also, feed-back control theory approaches usually apply a linear system model to control allocation (for design simplicity and stability guaranties). This is not adequate and can result in suboptimal control, since the performance and VM resource allocation is generally a non-linear relationship. To overcome these difficulties, a VM vertical scaling resource allocation approach has been developed using ANN-based modelling of application performance and VM resource allocation relationship. By applying a supervised machine learning based modelling approach (ANN), which is well known for its universal function approximation capabilities, it is possible to capture non-linear relationships between application performance and VM resource allocation and provide better resource allocation decisions. The approach is general and can be applied to any application, since it builds an ANN model online without any prior application knowledge. This is not the case for other approaches such as queuing theory based modeling approaches. The approach finds an adequate trade-off between conflicting goals of performance and power costs by optimizing an utility function. To cope with a large utility optimization time, as the result of increased number of VMs, a distributed resource manager has been developed. Each manager, based on the ANN model, allocates resources to one VM through local utility optimization and exchanges resource allocation decisions with other managers, coordinating global resource allocations. Simulated and realistic experiments, have demonstrated the superiority of distributed resource allocation approach over a centralized and a distributed non-coordinated version.

Modelling based VM resource allocation offers a general approach for VM performance management in cloud computing, but has its own drawbacks. An important one is the long online model adaptation time to fast changes of VM workloads. An improvement has been made by proposing a model-free approach based on utility optimization using fuzzy control. Also, in this approach the utility function provides the profit for the cloud provider for one physical machine. Utility function optimization is done using a hill-climbing local search algorithm, implemented as fuzzy rules. The main advantages of fuzzy-control, besides not requiring to build a model beforehand, is the fuzziness of utility and resource allocation values, resulting in an adaptive resource allocation amount of increasing or decreasing the current allocation. The fuzziness also allows to make appropriate resource allocation decisions by taking into account noisy utility value measurements. To cope with long utility optimization time, as the result of increased number of VMs, a multi-agent approach, dividing the global utility function into local utility functions, has been developed. Each agent, in parallel with other agents, allocates resources of one VM by optimizing through hill-climbing fuzzy control its own local utility function. Experimental evaluations show the superiority with respect to utility value of the multi-agent approach, over the centralized version and a state-of-the-art optimal control approach.

Most of the solutions to date do take a reactive approach for VM resource allocation. They make allocation decisions based on previous time interval resource utilizations, which can often result in SLA performance violations. These violations occur because too often the allocation decisions are taken in reactive manner, when the performance SLA violation has already happened. To overcome these drawbacks, the Automatic Proactive Resource Allocation (APRA) approach has been developed, which takes VM resource allocation decisions proactively based on resource demand prediction for the next time interval. For VM resource demand prediction, a machine learning technique is used, such as Support Vector Machine (SVM), showing good accuracy for time series forecasting. The APRA system is composed of several agents, spread over all physical machines of the cloud infrastructure, each responsible for resource allocation of all VMs of one multi-tier application. An arbiter, running on each physical machine, is responsible for taking resource allocation requests from all agents running VMs on that physical machine, resolving resource contentions and deciding on the final VM resource allocations. Existing approaches to proactive VM resource allocation base their allocation decisions on predicting utilization time series of one resource or of multiple resources separately. Since there are interdependencies between VMs of the same multi-tier application, their resource usage time series are correlated. Taking this into account, the approach presented in this dissertation applies SVM-based cross-correlation prediction on multiple resource usage time series of VMs of the same application. As experimental results show, the cross-correlation prediction achieves better prediction accuracy compared to prediction of each time series separately, resulting in better VM resource allocation decisions and application performance.

8.2 Future Work

There are several directions of future work for augmenting and improving the proposed solutions in this dissertation. Regarding global data center resource allocation through VM live migration, an interesting direction to follow is to develop a distributed resource allocation approach where on each physical machine a resource allocation agent responsible for taking VM live migration actions for all VMs can run. The challenge here is how agents, based only on local information, autonomously take VM live migration actions and simultaneously coordinate with each other in order to find near optimal VM to physical machine mapping. Ideas from distributed load balancing algorithms, distributed function optimization and multi-agent systems can be borrowed and investigated further. Distributed VM live migration resource management offers a scalable approach in large-scale IaaS cloud infrastructures. It mitigates the drawbacks of centralized management such as single point of failure, performance bottlenecks and long optimization times. Also, an interesting direction to follow would be experimenting with utility functions that include other resources, such as hard-disk and network bandwidth I/O, and other objectives such as physical machine reliability metrics.

There are several challenges and improvements that can be investigated further regarding fine-grained VM resource allocation at the physical machine level. One of the main challenges is coordinating the resource allocations of different VMs of the same multi-tier application, running on different physical machines.

In this respect, each agent should allocate resources to its own VM, optimizing its local utility function, and at the same time coordinate actions with other agents in order to optimize a global utility function defined over all VMs. The coordination of actions in the same multi-tier application is needed, since the optimization is done over the same performance metric and the allocation of a single VM influences the common application metric of them all. Moreover, for coordinating resource allocation actions, techniques and methods from multi-agent coordination and distributed utility function optimization could be investigated. Interesting could also prove the investigation of techniques from game theory. This can offer a rich variety of methods for agents to allocate resources based on incentives for optimizing their goals.

Further investigation is also needed with respect to VM resource allocation based on application performance modelling through machine learning techniques. Since model training is done online, the applied training samples should be as representative of the dynamic range of allocation values as possible. On one hand, the samples should aim to build an accurate model, on the other hand, they should make sure to not violate application performance SLAs. Solving this trade-off requires investigating intelligent sample selection and model training techniques. Also, online model adaptation time to workload changes needs improvements, since a long adaptation time, especially for very fast workload changes, can influence in reverse the application performance. Aiming at eliminating the long model adaptation time, the investigation of techniques for training and caching different models, detecting workload changes and applying directly the model for the corresponding workload could be interesting. Another challenge is training accurate models, in the presence of allocation actions with delayed effects several time intervals after they are applied. Here, the integration of Markov Decision Processes (MDP) into the model training process could help.

A promising VM resource management approach is the proactive VM resource allocation where allocation decisions are taken on the basis of resource demand predicted over several time intervals. It is of interest to apply resource demand prediction for taking proactive allocation actions at different time scales and levels such as global resource allocation and local physical machine allocation. Here, improvements are needed to increase prediction accuracy especially for long-term future predictions. This is useful for planning decisions far ahead in time for allocations actions, such as VM live migration, that take long time to complete. Also, more intelligent and robust techniques to correct allocation actions in the presence of prediction errors are needed. Interesting is also the investigation of combining reactive and proactive allocation approaches, to provide better resource management solutions that incorporate the advantages of both approaches.

Finally, an important challenge is the combination and integration of multiple resource allocation mechanisms (actions) in one complete resource management solution that operates at different levels and time scales. More specifically, the combination of resource allocation actions such as VM live migration, horizontal scaling VM replication and vertical scaling VM resource allocation is desired. The challenge is to decide, (1) which allocation actions to make for which VM and each workload change event and, (2) how to coordinate different actions in different physical machines without having conflicts and negative interferences on each other.

List of Figures

List of Tables

Bibliography

[1] Advanced Micro Devices, Inc. Amd virtualization technology. http://www.amd.com/en-us/solutions/servers/virtualization, 2014.

[2] Ahmed Ali-Eldin, Johan Tordsson, and Erik Elmroth. An adaptive hybrid elasticity controller for cloud infrastructures. In *Proc. IEEE/IFIP Network Operations and Management Symposium*, NOMS '12, pages 204–212. IEEE Press, 2012.

[3] Amazon. http://www.amazon.com/, 2014.

[4] Amazon elastic compute cloud (amazon EC2). http://aws.amazon.com/ec2/.

[5] Mauro Andreolini, Sara Casolari, Michele Colajanni, and Michele Messori. Dynamic load management of virtual machines in cloud architectures. In *CloudComp*, volume 34 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 201–214. Springer, 2009.

[6] Earl Bryson Arthur and Yu-Chi Ho. *Applied Optimal Control: Optimization, Estimation and Control.* Blaisdell Pub. Co., 1st edition, 1969.

[7] Karl Johan Astrom and Bjorn Wittenmark. *Adaptive Control.* Addison-Wesley Longman Publishing Co., Inc., 2nd edition, 1994.

[8] 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. In *Proc. 19th ACM Symposium on Operating Systems Principles*, pages 164–177. ACM Press, 2003.

[9] Enda Barrett, Enda Howley, and Jim Duggan. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience*, 25(12):1656–1674, 2013.

[10] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

[11] Anton Beloglazov, Jemal H. Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Comp. Syst.*, 28(5):755–768, 2012.

[12] Anton Beloglazov and Rajkumar Buyya. Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers. In *Proc. 8th International Workshop on Middleware for Grids, Clouds and e-Science*, MGC '10, pages 4:1–4:6. ACM, 2010.

[13] Anton Beloglazov and Rajkumar Buyya. Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. *IEEE Transactions on Parallel and Distributed Systems*, 24(7):1366–1379, 2013.

[14] Mohamed N. Bennani and Daniel A. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *Proc. 2nd International Conference on Automatic Computing*, pages 229–240. IEEE Press, 2005.

[15] Jing Bi, Zhiliang Zhu, Ruixiong Tian, and Qingbo Wang. Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center. In *Proc. IEEE 3rd International Conference on Cloud Computing*, CLOUD '10, pages 370–377. IEEE Press, 2010.

[16] N Bobroff, A Kochut, and K Beaty. Dynamic placement of virtual machines for managing sla violations. In *Proc. 10th IFIP/IEEE International Symposium on Integrated Network Management (IM'07)*, pages 119–128. IEEE Press, 2007.

[17] Peter Bodík, Rean Griffith, Charles Sutton, Armando Fox, Michael Jordan, and David Patterson. Statistical machine learning makes automatic control practical for internet datacenters. In *Proc. Conference on Hot Topics in Cloud Computing*. USENIX Association, 2009.

[18] Peter Bodik, Rean Griffith, Charles Sutton, Armando Fox, Michael I. Jordan, and David A. Patterson. Automatic exploration of datacenter performance regimes. In *Proc. 1st Workshop on Automated Control for Datacenters and Clouds*, ACDC '09, pages 1–6. ACM Press, 2009.

[19] George E. P. Box, Gwilym M. Jenkins, and Gregory C. Reinsel. *Time Series Analysis: Forecasting and Control.* Wiley, 2008.

[20] Rodrigo N. Calheiros, Rajiv Ranjan, and Rajkumar Buyya. Virtual machine provisioning based on analytical performance and qos in cloud computing environments. In *Proc. International Conference on Parallel Processing*, ICPP '11, pages 295–304. IEEE Press, 2011.

[21] Michael Cardosa, Madhukar R. Korupolu, and Aameek Singh. Shares and utilities based power consolidation in virtualized server environments. In *Proc. 11th IFIP/IEEE International Symposium on Integrated Network Management (IM'09)*, pages 327–334. IEEE Press, 2009.

[22] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proc. 2nd Conference on Symposium on Networked Systems Design and Implementation*, NSDI'05, pages 273–286. USENIX Association, 2005.

[23] Community project, supported by Parallels, Inc. OpenVZ, Linux containers. http://openvz.org/Main_Page, 2014.

[24] Italo S. Cunha, Jussara M. Almeida, Virgilio Almeida, and Marcos Santos. Self-adaptive capacity management for multi-tier virtualized environments. In *Integrated Network Management*, pages 129–138. IEEE Press, 2007.

[25] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51:107–113, 2008.

[26] Y. Diao, J. L. Hellerstein, and S. Parekh. Using fuzzy control to maximize profits in service level management. *IBM Syst. J.*, 41(3):403–420, 2002.

[27] Jeff Dike. A user-mode port of the linux kernel. In *Proc. 4th Annual Linux Showcase & Conference - Volume 4*, ALS'00, pages 7–7. USENIX Association, 2000.

[28] Ronald P. Doyle. Model-based resource provisioning in a web service utility. In *Proc. 4th USENIX Symposium on Internet Technologies and Systems*, pages 5–5. USENIX Association, 2003.

[29] Xavier Dutreilh, Sergey Kirgizov, Olga Melekhova, Jacques Malenfant, Nicolas Rivierre, and Isis Truck. Using Reinforcement Learning for Autonomic Resource Allocation in Clouds: towards a fully automated workflow. In *Proc. Seventh International Conference on Autonomic and Autonomous Systems, ICAS 2011*, pages 67–74. IEEE Press, May 2011. MoVe INT LIP6.

[30] Xavier Dutreilh, Nicolas Rivierre, Aurlien Moreau, Jacques Malenfant, and Isis Truck. From data center resource allocation to control theory and back. In *Proc. 3rd IEEE International Conference on Cloud Computing*, CLOUD'10, pages 410–417. IEEE Press, 2010.

[31] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In *Proc. Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann, 1990.

[32] Wei Fang, ZhiHui Lu, Jie Wu, and ZhenYin Cao. Rpps: A novel resource prediction and provisioning scheme in cloud data center. In *Proc. 9th IEEE International Conference on Services Computing*, SCC '12, pages 609–616. IEEE Press, 2012.

[33] Guofu Feng, Saurabh Garg, Rajkumar Buyya, and Wenzhong Li. Revenue maximization using adaptive resource provisioning in cloud computing environments. In *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*, GRID '12, pages 192–200. IEEE Press, 2012.

[34] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. In *Proc.*

17th International Conference on Architectural Support for Programming Languages and Operating Systems, pages 37–48. ACM Press, 2012.

[35] Tiago C. Ferreto, Marco Aurlio Stelmar Netto, Rodrigo N. Calheiros, and Csar A. F. De Rose. Server consolidation with migration control for virtualized data centers. *Future Generation Comp. Syst.*, 27(8):1027–1034, 2011.

[36] Filebench: file system and storage benchmark. http://sourceforge.net/apps/mediawiki/filebench/index.php.

[37] Forrester Research, Inc. http://www.forrester.com/home/, 2014.

[38] Gartner, Inc. https://www.gartner.com/, 2014.

[39] Zhenhuan Gong and Xiaohui Gu. Pac: Pattern-driven application consolidation for efficient cloud computing. In *Proc. IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS'10, pages 24–33. IEEE Press, 2010.

[40] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *Proc. International Conference on Network and Service Management (CNSM'10)*, pages 9–16. IEEE Press, 2010.

[41] Google App Engine. https://cloud.google.com/products/app-engine/, 2014.

[42] Hadi Goudarzi, Mohammad Ghasemazar, and Massoud Pedram. Sla-based optimization of power and migration cost in cloud computing. In *Proc. 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, CCGRID '12, pages 172–179. IEEE Press, 2012.

[43] Diwaker Gupta, Ludmila Cherkasova, Rob Gardner, and Amin Vahdat. Enforcing performance isolation across virtual machines in xen. In *Proc. ACM/IFIP/USENIX International Conference on Middleware*, pages 342–362. Springer-Verlag Inc., 2006.

[44] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[45] Rui Han, Li Guo, Moustafa M. Ghanem, and Yike Guo. Lightweight resource scaling for cloud applications. In *Proc. 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGRID '12, pages 644–651. IEEE Press, 2012.

[46] Masum Z. Hasan, Edgar Magana, Alexander Clemm, Lew Tucker, and Sree Lakshmi D. Gudreddi. Integrated and autonomic cloud resource scaling. In *Proc. IEEE/IFIP Network Operations and Management Symposium*, NOMS'12, pages 1327–1334. IEEE Press, 2012.

[47] Taliver Heath, Bruno Diniz, Enrique V. Carrera, Wagner Meira, Jr., and Ricardo Bianchini. Energy conservation in heterogeneous server clusters. In *Proc. 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 186–195. ACM Press, 2005.

[48] Stephen Hemminger. Iproute2 Homepage at Linux Foundation. http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2, 2011.

[49] Jin Heo, Xiaoyun Zhu, Pradeep Padala, and Zhikui Wang. Memory overbooking and dynamic control of xen virtual machines in consolidated environments. In *Proc. 11th IFIP/IEEE international conference on Symposium on Integrated Network Management (IM'09)*, pages 630–637. IEEE Press, 2009.

[50] Henry Hoffmann, Jonathan Eastep, Marco D. Santambrogio, Jason E. Miller, and Anant Agarwal. Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments. In *Proc. International Conference on Autonomic Computing*, pages 79–88. ACM Press, 2010.

[51] John H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, 1992.

[52] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[53] Rongdong Hu, Jingfei Jiang, Guangming Liu, and Lixin Wang. Kswsvr: A new load forecasting method for efficient resources provisioning in cloud. In *Proc. IEEE International Conference on Services Computing*, SCC '13, pages 120–127. IEEE Press, 2013.

[54] Qingjia Huang, Sen Su, Siyuan Xu, Jian Li, Peng Xu, and Kai Shuang. Migration-based elastic consolidation scheduling in cloud data center. In *Proc. IEEE 33rd International Conference on Distributed Computing Systems Workshops*, pages 93–97. IEEE Press, 2013.

[55] Intel Corporation. Hardware-based intel virtualization technology. http://www.intel.com/content/www/us/en/virtualization/virtualization-technology/hardware-assist-virtualization-technology.html, 2014.

[56] Waheed Iqbal, Matthew N. Dailey, David Carrera, and Paul Janecek. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Gener. Comput. Syst.*, 27(6):871–879, June 2011.

[57] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Gener. Comput. Syst.*, 28(1):155–162, January 2012.

[58] Jing Jiang, Jie Lu, Guangquan Zhang, and Guodong Long. Optimal cloud resource auto-scaling for web applications. In *Proc. 13th IEEE/ACM*

International Symposium on Cluster, Cloud and Grid Computing, CC-Grid'13, pages 58–65. IEEE Press, 2013.

[59] Gueyoung Jung, Matti A. Hiltunen, Kaustubh R. Joshi, Richard D. Schlichting, and Calton Pu. Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In *Proc. IEEE 30th International Conference on Distributed Computing Systems*, ICDCS '10, pages 62–73. IEEE Computer Society, 2010.

[60] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME-Journal of Basic Engineering*, 82:35–45, 1960.

[61] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *Proceedings of the 6th International Conference on Autonomic Computing*, ICAC '09, pages 117–126. ACM Press, 2009.

[62] Poul-Henning Kamp and Robert N. M. Watson. Jails: Confining the omnipotent root. In *Proc. 2nd International System Administration and Networking Conference*, SANE 2000.

[63] G Khanna, K Beaty, G Kar, and A Kochut. Application performance management in virtualized server environments. In *Proc. 10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006)*, pages 373–381. IEEE Press, 2006.

[64] Minkyong Kim and Brian Noble. Mobile network estimation. In *Proc. 7th Annual International Conference on Mobile Computing and Networking (MobiCom '01)*, pages 298–309. ACM, 2001.

[65] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the Linux virtual machine monitor. In *Proc. Linux Symposium*, pages 225–230, 2007.

[66] Younggyun Koh, Rob C. Knauerhase, Paul Brett, Mic Bowman, Zhihua Wen, and Calton Pu. An analysis of performance interference effects in virtual environments. In *Proc. IEEE International Symposium on Performance Analysis of Systems and Software.*, pages 200–209. IEEE Press, 2007.

[67] Sajib Kundu, Raju Rangaswami, Ajay Gulati, Ming Zhao, and Kaushik Dutta. Modeling virtualized applications using machine learning techniques. In *Proc. 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments*, pages 3–14. ACM Press, 2012.

[68] Dara Kusic, Jeffrey O. Kephart, James E. Hanson, Nagarajan Kandasamy, and Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. In *Proc. 2008 International Conference on Autonomic Computing (ICAC'08)*, pages 3–12. IEEE Press, 2008.

[69] Palden Lama and Xiaobo Zhou. Autonomic provisioning with self-adaptive neural fuzzy control for percentile-based delay guarantee. *ACM Trans. Auton. Adapt. Syst.*, 8(2):9:1–9:31, July 2013.

[70] Bo Li, Jianxin Li, Jinpeng Huai, Tianyu Wo, Qin Li, and Liang Zhong. Enacloud: An energy-saving application live placement approach for cloud computing environments. In *Proc. IEEE International Conference on Cloud Computing (CLOUD)*, pages 17–24. IEEE Press, 2009.

[71] Harold C. Lim, Shivnath Babu, and Jeffrey S. Chase. Automated control for elastic storage. In *Proc. 7th International Conference on Autonomic Computing*, ICAC '10, pages 1–10. ACM Press, 2010.

[72] Harold C. Lim, Shivnath Babu, Jeffrey S. Chase, and Sujay S. Parekh. Automated control in cloud computing: Challenges and opportunities. In *Proc. 1st Workshop on Automated Control for Datacenters and Clouds*, ACDC '09, pages 13–18. ACM Press, 2009.

[73] Liang Liu, Hao Wang, Xue Liu, Xing Jin, Wen Bo He, Qing Bo Wang, and Ying Chen. Greencloud: A new architecture for green data center. In *Proceedings of the 6th International Conference Industry Session on Autonomic Computing and Communications Industry Session*, ICAC-INDST '09, pages 29–38. ACM Press, 2009.

[74] Karlsson Magnus, Zhu Xiaoyun, and Karamanolis Christos. An adaptive optimal controller for non-intrusive performance differentiation in computing services. In *Proc. IEEE Conference on Control and Automation*, pages 709–714. IEEE press, 2005.

[75] Daniel A. Menasce and Mohamed N. Bennani. Autonomic virtualized environments. In *Proc. International Conference on Autonomic and Autonomous Systems*, pages 28–38. IEEE Press, 2006.

[76] Xiaoqiao Meng, Canturk Isci, Jeffrey Kephart, Li Zhang, Eric Bouillet, and Dimitrios Pendarakis. Efficient resource provisioning in compute clouds via vm multiplexing. In *Proc. 7th International Conference on Autonomic Computing*, ICAC '10, pages 11–20. ACM Press, 2010.

[77] Microsoft. Microsoft virtual server. http://www.microsoft.com/virtualserver/, 2014.

[78] Microsoft. Windows virtual pc. http://www.microsoft.com/windows/virtual-pc/, 2014.

[79] Microsoft Windows Azure. https://www.windowsazure.com/en-us/, 2014.

[80] D. Minarolli and B. Freisleben. Utility-based resource allocation for virtual machines in cloud computing. In *Proc. IEEE Symposium on Computers and Communications*, ISCC '11, pages 410–417. IEEE Press, 2011.

[81] Dorian Minarolli and Bernd Freisleben. Utility-driven allocation of multiple types of resources to virtual machines in clouds. In *Proc. IEEE 13th Conference on Commerce and Enterprise Computing*, CEC '11, pages 137–144. IEEE Press, 2011.

[82] Dorian Minarolli and Bernd Freisleben. Virtual machine resource allocation in cloud computing via multi-agent fuzzy control. In *Proc. 2013 International Conference on Cloud and Green Computing*, CGC '13, pages 188–194. IEEE Press, 2013.

[83] Dorian Minarolli and Bernd Freisleben. Cross-correlation prediction of resource demand for virtual machine resource allocation in clouds. In *Proc. 6th International Conference on Computational Intelligence, Communication Systems and Networks*, CICSYN'14, pages 119–124. IEEE Press, 2014.

[84] Dorian Minarolli and Bernd Freisleben. Distributed resource allocation to virtual machines via artificial neural networks. In *Proc. 22Nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, PDP '14, pages 490–499. IEEE Press, 2014.

[85] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. Q-clouds: managing performance interference effects for qos-aware clouds. In *Proc. 5th European conference on Computer systems (EuroSys '10)*, pages 237–250. ACM Press, 2010.

[86] National Institute of Standards and Technology (NIST). http://www.nist.gov/, 2014.

[87] Hiep Nguyen, Zhiming Shen, Xiaohui Gu, Sethuraman Subbiah, and John Wilkes. Agile: Elastic distributed resource scaling for infrastructure-as-a-service. In *Proc. 10th International Conference on Autonomic Computing*, pages 69–82. USENIX Association, 2013.

[88] Steffen Nissen. Fast Artificial Neural Network Library. http://leenissen.dk/fann/wp/.

[89] Pradeep Padala, Kai-Yuan Hou, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, and Arif Merchant. Automated control of multiple virtualized resources. In *Proc. 4th ACM European Conference on Computer Systems*, pages 13–26. ACM Press, 2009.

[90] 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. In *Proc. 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 289–302. ACM Press, 2007.

[91] Kevin M. Passino and Stephen Yurkovich. *Fuzzy Control*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1997.

[92] Vinicius Petrucci, Orlando Loques, and Daniel Mossé. A dynamic optimization model for power and performance management of virtualized clusters. In *Proc. 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 225–233. ACM Press, 2010.

[93] John C. Platt. Advances in kernel methods. chapter Fast Training of Support Vector Machines Using Sequential Minimal Optimization, pages 185–208. MIT Press, 1999.

[94] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17:412–421, 1974.

[95] Herbert Potzl. Linux-vserver technology. http://linux-vserver.org/Paper, 2014.

[96] Jia Rao, Xiangping Bu, Kun Wang, and Cheng-Zhong Xu. Self-adaptive provisioning of virtualized resources in cloud computing. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, SIGMETRICS '11, pages 129–130. ACM, 2011.

[97] Jia Rao, Xiangping Bu, Cheng-Zhong Xu, Leyi Wang, and George Yin. Vconf: A reinforcement learning approach to virtual machines auto-configuration. In *Proc. 6th International Conference on Autonomic Computing*, pages 137–146. ACM Press, 2009.

[98] Jia Rao, Yudi Wei, Jiayu Gong, and Cheng-Zhong Xu. Dynaqos: model-free self-tuning fuzzy control of virtualized resources for qos provisioning. In *Proc. 19th International Workshop on Quality of Service*, pages 1–9. IEEE Press, 2011.

[99] V.J. Rayward-Smith. *Modern heuristic search methods.* Wiley, 1996.

[100] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Proc. IEEE International Conference on Neural Networks*, pages 586–591. IEEE Press, 1993.

[101] Salesforce.com. http://www.salesforce.com/platform/overview/, 2014.

[102] Nicholas I. Sapankevych and Ravi Sankar. Time series prediction using support vector machines: A survey. *IEEE Computational Intelligence Magazine*, 4(2):24–38, 2009.

[103] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14:199–222, 2004.

[104] Jason Sonnek, James Greensky, Robert Reutiman, and Abhishek Chandra. Starling: Minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration. In *Proc.*

39th International Conference on Parallel Processing, ICPP '10, pages 228–237. IEEE Press, 2010.

[105] Sourceforge, Open Source Software. Dm-ioband Project Page. http://sourceforge.net/apps/trac/ioband/, 2011.

[106] Gerald Tesauro, Nicholas K. Jong, Rajarshi Das, and Mohamed N. Bennani. On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing*, 10(3):287–299, 2007.

[107] Dimitris Tsirogiannis, Stavros Harizopoulos, and Mehul A. Shah. Analyzing the energy efficiency of a database server. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 231–242. ACM Press, 2010.

[108] Bhuvan Urgaonkar, Prashant Shenoy, Abhishek Chandra, Pawan Goyal, and Timothy Wood. Agile dynamic provisioning of multi-tier internet applications. *ACM Trans. Auton. Adapt. Syst.*, 3(1):1:1–1:39, March 2008.

[109] Hien Nguyen Van, Frederic Dang Tran, and Jean-Marc Menaud. Sla-aware virtual resource management for cloud infrastructures. In *Proc. Ninth IEEE International Conference on Computer and Information Technology (CIT '09)*, pages 357–362. IEEE Press, 2009.

[110] 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.*, 39:50–55, 2008.

[111] Akshat Verma, Puneet Ahuja, and Anindya Neogi. pmapper: Power and migration cost aware application placement in virtualized systems. In *Proc. ACM/IFIP/USENIX 9th International Middleware Conference (Middleware '08)*, pages 243–264. Springer-Verlag, 2008.

[112] VMWare Inc. VMWare Homepage. http://www.vmware.com/, 2011.

[113] VMware, Inc. Vmware esx server. http://www.vmware.com/products/esxi-and-esx/overview, 2014.

[114] VMware, Inc. Vmware workstation. http://www.vmware.com/products/workstation/, 2014.

[115] Werner Vogels. Beyond server consolidation. *Queue*, 6:20–26, 2008.

[116] Matthew Wall. A C++ Library of Genetic Algorithm Components. http://lancet.mit.edu/ga/.

[117] William E. Walsh, Gerald Tesauro, Jeffrey O. Kephart, and Rajarshi Das. Utility functions in autonomic systems. In *Proc. First International Conference on Autonomic Computing (ICAC'04)*, pages 70–77. IEEE Press, 2004.

[118] Zhikui Wang, Xiaoyun Zhu, Sharad Singhal, and Hewlett Packard. Utilization and slo-based control for dynamic sizing of resource partitions. In *Proc. 16th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, pages 24–26. Springer-Verlag, 2005.

[119] Jonathan Wildstrom, Peter Stone, and Emmett Witchel. Carve: A cognitive agent for resource value estimation. In *Proc. 5th IEEE International Conference on Autonomic Computing*, pages 182–191. IEEE Press, 2008.

[120] Jonathan Wildstrom, Peter Stone, Emmett Witchel, and Mike Dahlin. Machine learning for on-line hardware reconfiguration. In *Proc. 20th International Joint Conference on Artifical Intelligence*, pages 1113–1118. Morgan Kaufmann Publishers Inc., 2007.

[121] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.

[122] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Sandpiper: Black-box and gray-box resource management for virtual machines. *Comput. Netw.*, 53(17):2923–2938, 2009.

[123] Xen Project. www.xenproject.org, 2014.

[124] Cheng-Zhong Xu, Jia Rao, and Xiangping Bu. Url: A unified reinforcement learning approach for autonomic cloud management. *J. Parallel Distrib. Comput.*, 72(2):95–105, 2012.

[125] Jing Xu, Ming Zhao, José Fortes, Robert Carpenter, and Mazin Yousif. Autonomic resource management in virtualized data centers using fuzzy logic-based approaches. *Cluster Computing*, 11(3):213–227, September 2008.

[126] Yagiz Onat Yazir, Chris Matthews, Roozbeh Farahbod, Stephen Neville, Adel Guitouni, Sudhakar Ganti, and Yvonne Coady. Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, CLOUD '10, pages 91–98. IEEE Press, 2010.

[127] L.A. Zadeh. Fuzzy sets. *Information Control*, 8:338–353, 1965.

[128] Bolei Zhang, Zhuzhong Qian, Wei Huang, Xin Li, and Sanglu Lu. Minimizing communication traffic in data centers with power-aware vm placement. In *Proc. Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, IMIS '12, pages 280–285. IEEE Press, 2012.

[129] Qi Zhang, Ludmila Cherkasova, and Evgenia Smirni. A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In *Proc. Fourth International Conference on Autonomic Computing*, ICAC '07, page 27. IEEE Press, 2007.

[130] Xiaoyun Zhu, Zhikui Wang, and Sharad Singhal. Utility-driven workload management using nested control design. In *Proc. American Control Conference*, ACC '06, page 6. IEEE Press, 2006.

Curriculum Vitae

Personal Data

Name	Dorian Minarolli
Birthday	09.07.1979, Shkoder
Contact	`minarolli@Mathematik.Uni-Marburg.de`

Education

1997-2002	*Polytechnic University of Tirana;* Diploma: Electronic Engineer, Computer Direction

Academic and Working Experience

07/2002-03/2004	*Albanian Mobile Communications, Tirana, Albania*, Specialist
04/2004-07/2004	*Procredit Bank. Tirana, Albania*, IT specialist.
10/2004-12/2008	*Polytechnic University of Tirana, Albania*, Teaching and Research Assistant.
01/2009-present	*Department of Mathematics and Computer Science, University of Marburg,Germany*, PhD student.

Other Qualifications

2004	*National Technical University of Athens*, work on diploma thesis.
07/2006	*CE EGEE and SEEGRID-2 Summer School on Grid Applications, held in MTA SZTAKI, Budapest, Hungary.*
09/2009	*First DAAD Summer School on Current Trends in Distributed Systems (CTDS09), Gammarth, Tunisia*