Autonomous Resource Allocation in Clouds: A Comprehensive Analysis of Single Synthesizing Criterion and Outranking Based Multiple Criteria Decision Analysis Methods

by

Yağmur Akbulut
B.Sc., University of Victoria, 2011

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Autonomous Resource Allocation in Clouds: A Comprehensive Analysis of Single Synthesizing Criterion and Outranking Based Multiple Criteria Decision Analysis Methods

by

Yağmur Akbulut
B.Sc., University of Victoria, 2011

ABSTRACT

Cloud computing is an emerging trend where clients are billed for services on a pay-per-use basis. Service level agreements define the formal negotiations between the clients and the service providers on common metrics such as processing power, memory and bandwidth. In the case of service level agreement violations, the service provider is penalised. From service provider's point of view, providing cloud services efficiently within the negotiated metrics is an important problem. Particularly, in large-scale data center settings, manual administration for resource allocation is not a feasible option. Service providers aim to maximize resource utilization in the data center, as well as, avoiding service level agreement violations. On the other hand, from the client's point of view, the cloud must continuously ensure enough resources to the changing workloads of hosted application environments and services. Therefore, an autonomous cloud manager that is capable of dynamically allocating resources in order to satisfy both the client and the service provider's requirements emerges as a necessity.

In this thesis, we focus on the autonomous resource allocation in cloud computing environments. A distributed resource consolidation manager for clouds, called IMPROMPTU, was introduced in our previous studies. IMPROMPTU adopts a threshold based reactive design where each unique physical machine is coupled with an autonomous node agent that manages resource consolidation independently from the rest of the autonomous node agents. In our previous studies, IMPROMPTU demonstrated the viability of Multiple Criteria Decision Analysis (MCDA) to provide resource consolidation management that simultaneously achieves lower numbers of reconfiguration events and service level agreement violations under the management of three well-known outranking-based methods called PROMETHEE II, ELECTRE III and PAMSSEM II. The interesting question of whether more efficient single synthesizing criterion and outranking based MCDA methods exist was left open for research. This thesis addresses these limitations by analysing the capabilities of IMPROMPTU using a comprehensive set of single synthesizing criterion and outranking based MCDA methods in the context of dynamic resource allocation. The performances of PROMETHEE II, ELECTRE III, PAMSSEM II, REGIME, ORESTE, QUALIFEX, AHP and SMART are investigated by in-depth analysis of simulation results. Most importantly, the question of what denotes the properties of good MCDA methods for this problem domain is answered.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank my supervisors Dr. Sudhakar Ganti and Dr. Yvonne Coady for their support and mentoring throughout this degree. I would also like to thank Dr. Onat Yazır for the academic work that lead to this thesis, for his endless support and trust.

DEDICATION

I dedicate this thesis to my wonderful family; my mother Ergülen Akbulut, my aunt Aksel Erenberk, my uncle Pertev Erenberk, and my grandparents -may they rest in light - Hatice and Sami Ulusoy. All my love to you, for finding me the light, whenever it was far away.

Chapter 1

Introduction

In 1961, John McCarthy declared publicly that computer time-sharing technology may possibly result in a future where computational resources and applications could be billed as utility, like water and electricity, in a public speech to celebrate MIT's centennial [37]. The idea was clearly ahead of its time due to the insufficient software, hardware and network technologies but it surely provided a glimpse into the future. Before skipping ahead to investigate the current status of cloud computing, in order to put matters in perspective, it is beneficial to provide a definition and working examples of distributed systems, as well as, the evolution of distributed computing systems by investigating milestone achievements in hardware, software and network technologies.

In a general sense, distributed computing systems refer to the multiple computational units that communicate over a network to achieve a common goal [32,66]. The concept is especially useful and becomes more meaningful when the problem at hand is large and computational complex. Naturally, a task is broken into manageable mutually exclusive sub-tasks to be processed concurrently by a number of computational entities in the system. The system appears as a single entity to the end user that provides a level of abstraction. Although, a former description is not present [31,66], the two defining properties of distributed systems are: 1) Each computational unit in the distributed system has its own local memory [4,21,31,47,54], and 2) the units are communicated and coordinated by the means of message parsing [4,31,54].

The architecture of distributed system consists of a server and multiple clients. The server typically employs a job dispatcher which keeps track of the progress of the task. In addition, the server is responsible for generating work packages and communicating them to the clients. On the other hand, the clients carry out the required tasks and return the result to the server. Let us investigate a distributed system by examining distributed.net's RC5-72 project [20]. The challenge is to decrypt an encrypted English message with a key of length 2^{72}. The servers starts from the key at zero and waits for a client to connect. Once a client connects and communicates the work package request to the server, it receives a work package that consists of the pair : (key + range of current key). The client attempts to decrypt the message with the given keys in the range and returns the result of completed work package to the server. The server checks whether the decrypted message is English and reports the key as the answer. In the case of failure, the key range is incremented and sent out to clients until all the keys are exhausted.

The concept of executing mutually exclusive tasks by communicating via message parsing in order to achieve a common goal dates back to 1960s where a large number of operating systems were designed based on the idea of time-sharing [4]. During this era, voice and data communication was established by allocated, end to end circuit switched networks. ARPANET, the predecessor of internet, was the first system to implement packet switching functionality in order to connect geographically separated computer systems. The lack of sufficient network technologies and protocols led to the development of current communication technologies such as Ethernet, TCP/IP and FTP in 1970s [1]. In addition, the most successful application of ARPANET was the introduction of e-mail in the early 1970s which can be considered the as one of the first large scale distributed applications . In the following decade, the connectivity of local area networks (LANs) and wide area networks (WANs) made possible by the developments in high speed communication technologies. This lead to global distributed systems such as FidoNet and Usenet.

The popularity of Internet rapidly increased in the 1990s by connecting millions of users worldwide. Internet is the largest distributed system where despite the geographical differences millions of private, public, academic, business and government networks are connected in a global scope. In his 1965 paper, Intel co-founder Gordon E. Moore stated that the computer chip performance would double approximately every 18 months [41]. Although, this forecast remained valid for a number of decades, currently, the amount of computation that can be carried out by a single computational unit is not likely to enhance dramatically over the course of next few years [45]. This is mainly due to the physical limits being pushed in the semiconductor technology and the ability to cool the over heating chips to operational levels [38]. In attempts to overcome this limitation, the CPU manufacturers turned into developing multi core technologies. However, the number of cores within a computational unit is not linearly associated with the processing power of the CPU as similar bottlenecks mentioned above remain in place. In all likelihood of these limits being lifted by new paradigms and technologies, other bottlenecks such as the bus speed would still be present [45]. Thanks to the communication technologies provided by the Internet, the usage of distributed systems are essential in order to carry out computational tasks faster.

An average computer user only uses a fraction of the available computational power at hand when performing tasks such as web browsing and text editing. This, in turn, leaves a vast proportion of computational power unused considering the machines being connected via the Internet. A few projects attempt to harvest this computational power in order to execute immense jobs such as Folding@home and SETI@home. Folding@home aims to find a cure for diseases such as Parkinson's, Alzheimer's, Huntington's and many cancers by trying to reveal the mystery surrounding protein folding [28]. Whereas, SETI@home aims to process radio signals gathered from outer space in order to search for extra terrestrial life [64]. The volunteer everyday computer users can install a client on their machine which asks for jobs to be processed when the screen saver comes on. These two and many other similar projects are solid examples where distributed computing is useful and can relate to other scientific fields.

The usage of distributed system apply to a various problem domains with the concept of cloud computing being the most popularly emerging trend. Cloud computing applies the idea of utility computing where the computational power is billed on a pay-per-use basis. This new paradigm uses an already existing concept of virtualization by converting physical resources such as processors and storage into virtually scalable and shareable resources by using the Internet [39]. In the next section, we are going to provide a detailed description of cloud computing technologies, highlight the underlying architecture, explore the resource allocation problem that forms the basis of this thesis.

1.1 Motivation and Problem Description

Cloud computing provides an abstraction between the end users and the infrastructure by provisioning the underlying computational resources. This, in turn, makes it possible to bill customers on a pay per use basis in a transparent way. The services provided by this technology can be summarized in three major categories: Infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS) [39].

Infrastructure as a service (IaaS), provides the lowest level of abstraction and the most customizable services by leasing the computing resources and storage. Typically, users run a hypervisor such as Xen or KVM in order to manage the virtual machines. The hypervisor can increase or decrease the reserved resources to match the requirements of the applications hosted by the virtual machine. It is important to note that, the users are responsible for installing an operating system of choice and their entire application software on the virtual machines. This, in turn, enables the cloud environment to be fully customizable to fit the needs of the user. Generally, IaaS is billed as a utility where the cost reflects the amount of resources used [39].

The second level of the cloud computing service stack is platform as a service (PaaS). In this model, the cloud service providers offer a bundle of hardware, operating system, programming tools, database and web server. The users can run or develop their solutions without the added cost and complexity of the lower level architecture. As in IaaS, the hypervisors can manage the assigned resources to fit the requirements of the active platform [39].

Software as a service (SaaS) lies at the top level of the cloud computing services. In this model, the cloud service providers offer and manage the infrastructure to run a specific application. This, in turn, saves the user from maintaining the software as well as the hardware necessary to run the computationally heavy applications. In the case of computationally intensive applications, load balancer are used to distribute the work load between numerous virtual machines. Software as a service is usually billed as a monthly, or a yearly flat rate service [39].

In contrast to the advantages provided by the cloud computing, many additional problems and complexities are also introduced. At first glance, the most obvious challenges can be seen as communication and coordination in the data center. All of these being valid challenges, an equally interesting and complex problem lies in the resource management for the hosted virtual machines. In an IaaS setting, the data center is constantly faced with dynamic, open and accessible environments, in which, the conditions change rapidly, unpredictably and in a continuous manner [40, 43].

Although not being a new paradigm, virtiualization is an essential ingredient in cloud computing [29]. Virtualization provides the ability to stop, start and migrate virtual machines within the data center which, in turn, facilitates flexible resource consolidation management in clouds. The ability to migrate virtual machines with little to no interruption to their on going services provides means to dynamically configure the distribution of computational resources. As a direct result, in the case where the user requires an increase to the resources assigned to their software environments, the service provider can relocate the virtual machines in the data center to satisfy the computational needs. In addition, the service provider can downgrade users resource allocation by hosting the software environment on a lesser performance hardware in the data center. Finally, the virtual machines can be migrated in order to perform a scheduled maintenance with no interruption.

Service level agreements (SLAs) , in a cloud setting, define the formal negotiations between the user and the service providers on common metrics such as available memory, processing power and bandwidth. In the case of violation of SLAs, the service providers are generally faced with a fine. In this manner, providing cloud computing services within the negotiated metrics is crucial for the service providers. In a setting where the resource requirements and conditions change rapidly, unpredictably and continuously, dynamic resource allocation emerges as a critical necessity. In the current state of cloud computing, resource allocation process is done statically where a system administrator assigns resource to a software environment. Due to the low-level and rigid definitions of the SLAs, generally static resource allocation is deemed acceptable [72]. However, SLAs are likely to evolve towards higher-level definitions that will abstract lower-level computational resources [72, 82]. In particular, response time is a valid example to some of the soft metrics that can be used in high-level SLAs. Response time of a software application can change through out its life time in the cloud, as the amount of computational resources needed to deliver a predefined response time may not be available. In addition, static resource allocation suffers from under-provisioning and over-provisioning. Under-provisioning results in wasted computational resources, whereas, over-provisioning results in frequent SLA violations.

Particularly, in large-scale data center settings like clouds, manual administration for virtual machine provisioning is not a feasible option. The problem at hand requires finding solutions to the following conflicting goals. The service provider aims to maximize the resource utilization in the data center in order to make sure that the available computational resources are used to the maximum possible capacity. From the client's point of view, an IaaS layer must continuously ensure enough resources to the changing workloads so that SLA violations are minimized. Therefore, an autonomous IaaS level manager that is capable of dynamically allocating resources in order to satisfy both the user and the service provider's requirements emerges a necessity.

1.2 Scope

In this thesis, we focus on the problem of dynamic resource allocation in cloud computing environments. In this context, we extend the proposed methodology for reactive self-management based on multiple criteria decision analysis (MCDA) with eight additional methods. It is important to note here that the framework for the dynamic resource allocation manager was already implemented [80]. However, only a single MCDA methods was investigated under this framework. This, in turn, left an interesting research area where additional MCDA methods needed to be implemented on this framework, in order to gain an in depth understanding of the problem.

In various studies, the generally adopted approach is to view the self-management process as a continuous procedure that runs regardless rather than a function of perceived changes [35, 67, 72]. In the proposed framework, the adaptive behaviour is defined through the means of reacting to changes in conditions, which, in turn, establishes a system that is capable of being self-aware. Although, the idea of reactive behaviour not being a new approach [26, 76], the usage of MCDA during the reaction process is an innovative idea. In particular, the former approaches need to search for a near-optimal solution in a vast solution space. However, MCDA based reactions do not suffer from this limitation. The next course of action during a reaction to a change in the system is carried out through mathematically explicit multiple criteria aggregation procedures (MCAP) from a pre-defined and finite set of alternative actions. To best of our knowledge, the variety of MCDA methods investigated in this thesis have not been previously used in the domain of resource allocation in cloud computing systems.

The thesis provides a simulation case study regarding distributed dynamic resource allocation in clouds using eight MCDA methods on the existing IMPROMPTU model. IMPROMPTU is a reactive resource consolidation manager where the responsibility of distributing resources are handled by autonomous node agents through the calculations of virtual machine to physical machine mappings.

In summary, the contributions of this work can be listed as follows:

- We extend IMPROMPTU with the following eight MCDA methods: SMART, AHP, ELECTRE III, PROMETHEE II, PAMSSEM II, ORESTE, REGIME and QUALIFLEX.
- We provide an in-depth analysis of simulation results regarding the performance measures of newly introduced MCDA methods.
- We answer the question of what denotes the properties of good MCDA methods for this problem domain.

We provide a comprehensive assessment of the introduced MCDA methods under IMPROMPTU management by comparing each method under a identical scenarios in a simulated data cloud data center.

1.3 Organization

The rest of this thesis is organized as follows. Chapter 2 provides detailed information on the related work done in the domain of dynamic resource allocation. In Chapter 3, we formally explain the Multi Criteria Decision Making Analysis methods implemented in this thesis. Chapter 4 reviews the already existing resource consolidation manager called IMPROMTU and the application of the MCDA methods used. In Chapter 5, the simulation case study is revealed and the collected results are analysed. Finally, Chapter 6 summarizes this thesis, and outlines future directions.

Chapter 2

Related Work

The problem of dynamic resource allocation management in data centers have been investigated mainly under two categories where the new configurations in the data center are calculated in a centralized or a distributed manner that employ methods that employ utility functions, queueing models, reinforcement learning, hybrid solutions, decompositional learning, analytic and statistical performance models, genetic algorithms and rules based approaches. In order to review the evolution of the proposed solutions and research, we are going to divide and analyse the related work under two sections.

2.1 Centralized Resource Allocation Management

A two-tier generalized design is first proposed approach in the context of non-virtualized data centers through the usage of utility functions [72]. In this study, the outlined data center consists of a number of logically separated Application Environments (AEs). Each AE has a pool of resources that directs the workload to servers via a router. In addition, AE's are assigned a service level utility function which reflects the business value of the service level agreements at the granularity of penalties and goal terms under specific situations. The utility function consists of two vectors that represent service level and demand space in the form of service metrics such as response time and throughput. Local application agents or local decision makers are assigned to each AE in the lower layer of the two-tier structure adopted by this method. The local agents are responsible for calculating the utilities of the associated AE and then communicating these values to a global arbiter. In the second layer of the design, the global arbiter tries to find an optimal solution in order to optimize a system goal defined by another utility function. The authors claim that the utility functions provide a natural way to represent a value, however, expressing a complex systems resource needs through utility functions is difficult. The need for a carefully designed interface that extracts utility functions from human input is also mentioned. In our view, the system suffers from scalability and performance issues. Although, the proposed system is feasible for small-scale systems, the configuration times present a bottleneck in large-scale settings as the global arbiter attempts to optimize a system wide utility function.

A similar two-tier approach is further adopted with changes regarding the storage of the collected data and the way the solution space is searched [7]. In [72], a table driven approach is used to store the resource metrics. This approach has several limitations that can be listed as follows: (1) the solution is not scalable with respect to number of application environments, and (2) the solution does not scale well with respect to the resource dimensions and resource types. Furthermore, building a table from experimental data is time consuming and computationally expensive. In [7], the table driven approach is replaced with predictive multi-class queuing network models. In this setting, each application manager is assigned a work load manager that is responsible for collecting and storing resource usage data into a database. In addition, a workload forecaster analyses the collected information in order make predictions about the future workload intensity levels. These values are transferred to a global controller algorithm to evaluate the general utility function. The algorithm starts the redeployment process when a better configuration is found. In contrast to the previous work done in this area, the local decision modules can compute the utility values depending on either of the information available before passing them to the global arbiter. Also, the new configurations calculated and implemented in fixed time intervals or in a reactive manner. This method differs in the way that the solution space is computed. Beam search algorithm is used by the global arbiter when searching for new configurations in attempt to reduce the size of the solution space.

In [51], a similar two-tier approach is used in the context of autonomic virtualized environments. Each virtual machine has an associated utility function which reflects the local resource usages. As in the previously adopted architectures, each utility function of virtual machines is attempted to be maximized by a global utility function for the virtualized environment. In this context, the authors explore two approaches: (1) dynamic CPU priority allocation and (2) the allocation of CPU shares to the various virtual machines. Priority allocation refers to each virtual machine's dispatching priority at the CPU. On the other hand, CPU share allocation refers to clock cycle ratios or simply percentage of the total processing power allocated for each virtual machine. Similar to the previous studies, the authors adopt a beam search method in order to efficiently find an optimal configuration. Finally, in this research, the only research dimension in consideration is the CPU. In our view, a single criterion is not sufficient in the context of dynamic resource allocation, as other important criteria such as memory and bandwidth is not considered.

Autonomic Virtual Resource Management for Service Hosting Platforms adopts the generally used two-tier approach with the following two assumptions: (1) the number of physical machines within the data center is fixed and that they belong to the same cluster and (2) the system has the ability to perform live migration [70]. In the first level, an application specific local decision maker is coupled with an application environment. The local decision maker computes a goal specific utility function which gives a degree of satisfaction in terms of resource usage. In the top layer, a global decision module combines the individual utility functions that would satisfy the business level needs of the service level agreements. Each local decision maker is treated as a black box by only knowing the utility function of the module and performance metrics. The global decision module carries out the two following tasks: (1) determining the virtual machine allocation vectors for each application layer and (2) placing the selected virtual machines on to the physical machines in order to minimize the number of machine usage. It is important to note that both of these problems are expressed through constraint satisfaction problems which are handled by a constraint solver. The simulation environment consists of a cluster of four physical machines that can each host two application environments. Therefore, the proposed solution is not validated in large scale settings.

The work outlined in [35], proposes a dynamic resource manager for homogeneous clusters, called Entropy. Entropy aims to provide virtual machine to physical machine mappings that (1) gives enough resources to each virtual machine, and (2) the configuration uses the minimal number of physical machines. The proposed solution is different from the previous approaches where utility functions were used. The authors attempt to overcome the scalability issues introduced with the former method using constraint programming. Constraint programming defines a problem in terms of a set of logical constraints that needed to be satisfied in order to solve the problem.

Instead of implementing a custom constraint solver, Choco library is used, which can solve constraint satisfaction problems where the goal is to minimize or maximize the value of a single variable [18]. In this context, the problem at hand needs a slight modification as it has multiple criteria. The task is carried out in two phases in order fit Choco model. In the first phase, the minimum number of physical machines that are necessary to host all virtual machines are found. In the second step, an equivalent configuration that minimizes the reconfiguration time is computed. It is important to note here that, the time needed to carry out these phases are extremely large with regards to the problem domain. The authors mentioned that the system cancels the search if the computation time is longer than one minute. In our view, this is likely to result in stale virtual machine to physical machine mappings which is not useful. Although, the proposed solution is elegant and is an improvement over the previous research, the computational complexity limits the viability of the method. A strong point of this research is the consideration of the overhead of virtual machine migrations and placement conflicts.

Online resource allocation using decompositional reinforcement learning was first suggested in the work of [69]. The proposed solution uses the two-tier architecture used in the previous studies with the global arbiter trying to maximize the utility functions of each application environment through reinforcement learning. The main contribution of this idea is that the proposed model needs no previous knowledge about the system or performance models. However, reinforcement learning needs training data set. In this manner, the author employs two strategies of quick learning and overnight training. It is important to note here that the real time learning is considerably slower than the change in workloads, which, in turn, causes problems in the real system. The author claims that the experimental results suggest validity of the solution with the need of comparison using queue models and a possible hybrid method.

The authors in [67], investigate two methodologies in order to reduce the immense computational burden of the utility functions. The proposed architecture is the same one in [72], where each application environment is coupled with a utility function at the lower level. A global resource arbiter allocates resources to these application environments at the higher level. When computing new configurations, the authors investigate the following two methodologies: (1) a queuing-theoretic performance model and (2) a model-free reinforcement learning. In the first approach, the system is modelled using M/M/1 queues where a periodic garbage collection is used. In the second approach, the application manager uses an algorithm known as Sarsa(0) to learn about the long range expected value functions of virtual machines. The performance of the two proposed methods are virtually identical, and better than static and random allocations according to the experiments. However, both of these methods need training times. Furthermore, the authors express concerns regarding scalability issues as the methods are tested in a small-scale environment.

The work outlined in [72] is continued with focusing on a hybrid reinforcement learning approach in [68]. In their previous work, the authors inspected both reinforcement learning approach and queueing models. In theory, reinforcement learning approach needs no explicit performance model or management policies but suffers from training times. On the other hand, the queueing model doesn't suffer from the same problems but the workload model is only an estimation of the actual system. In order to overcome these limitations, the proposed hybrid approach combines the positive sides of the two methods where reinforcement learning is used to train offline on the collected data and the queueing model controls the system. Furthermore, reinforcement learning is used to approximate non-linear functions instead of look-up tables for forecasting changes in workload in the data center.

Research outlined in [3], focuses on the problem of resource allocation in virtualized web server environments with respect to quality of service. The authors underline the following two critical requirements, (1) Short term planning, defined as how to minimize the service level agreement violations while maximizing resource utilization and, (2) long term capacity planning, defined as how to plan the size of the data center in order to maximize the net revenue from service level agreement contracts while minimizing the cost of ownership. Accordingly, the authors proposed a two phased solution. In both phases, the system is optimized using performance estimations extracted from analytical queueing models. The ad hoc solver employed in the optimizer computes configurations for up to 400 virtual machines under 15 seconds. Although, the authors claim that the configuration time is very efficient and practical for real online applications, we strongly believe that the workload in real time systems change more rapidly.

Dynamic placement of virtual machines for managing service level agreement violations are addressed in [8]. The solution proposes an analytical formula that is used to classify virtual machines which benefit the most from dynamic migrations. The formula predicts gain based on read metrics using p-percentile of appropriate distributions. In addition, forecasting predicts the probability distribution of a demand in the future intervals. The management algorithm combines series forecasting and bin packing heuristic to choose virtual machines for migration. This study is further extended by [46], where a similar analytical model of gain from dynamic reallocation of virtual machines is examined.

A commercialized computing system called Unity was built using the same generalized architecture [15, 19]. Unity was designed at IBM's J. Watson Research Center with respect to the principal that the behaviour of a system results from the behaviours and relations among its components. The authors claim that for a computing system to be self managing, it has to fulfil the following properties: (1) to be self-configuring, (2) self-optimizing, (3) self-protecting and (4) self-healing. The architecture of Unity is examined under these four principals. Unity is composed of several structured autonomic elements similar to the two-tier approach mentioned above where application environments are closely monitored by local decision modules and a global global arbiter computing new possible configurations. The proposed system uses utility functions during the decision-making process. In addition, the architecture uses goal-driven self-assembly to congure itself and the design patterns that enable self-healing within the system. The system employs a policy repository element that allows human administrators to input high-level policies for the guidance of the system through a web interface which consists of servlets, portlets and applets. The interface allows the direction of system wide goals, which, in turn, is passed down to the underlying components by the manager.

In the paper called *Application Performance Management in Virtualized Server Environments*, an algorithm is proposed to address dynamic resource allocation problem [44]. The system monitors key performance metrics such as CPU utilization and the collected data triggers migration of virtual machines based on certain thresholds. The decision making progress is carried out by the proposed algorithm as follows: (1) A heap is constructed by selecting the virtual machines with the lowest utilization, (2) The virtual machine with the lowest utilization is moved to the physical machine with the minimum residual capacity. In case of failure, the heap is iterated using the next minimum. It is important to not here that the authors do not make a distinction between the types of virtual machines and assume homogeneity within the data center. In addition, the proposed solution is only tested within a small scale testing environment where no forecasting techniques are used.

A virtual-appliance based autonomic provisioning framework for large outsourcing data centers is presented in [74]. The system consists of heterogeneous physical machines that are capable of hosting a number of virtual machines. In this manner, the proposed solution consolidates small server virtual machines on high capacity physical machines. This, in turn, provides a sense of isolation for the resource demanding virtual machines. The proposed solution employs a constrained non-linear optimization technique to dynamically allocate resource within the date center. In addition, important factors such as virtualization overhead and reconfiguration delay is also considered in order to establish realistic model.

Black-box and Gray-box Strategies for Virtual Machine Migration proposes a system called Sandpiper in order to monitor and configure virtual machine to physical machine mappings in data centers [75] using statistic models. In this context, two main approaches are considered: (1) black-box approach where the proposed solution is independent of the operating system and the application environment, and (2) gray-box approach where the statistics regarding the operation system and the application environment is considered. In the cases where it is not possible to 'peek inside' a virtual machine to collect resource usage statistics, the black-box approach is used where CPU, network and memory is monitored for each virtual machines. In addition, the collected usages are aggregated to calculate the resource usages of physical machines. On the other hand, the gray-box approach employs a light-weight monitoring daemon installed in each virtual machine that tracks statistics as well as application logs. In this manner, service level agreements are detected in an explicit manner, as opposed to the proxy metric in black-box approach. Finally, the profiling engine produces profiles for each virtual machine based on the collected statistics which is used for dynamic resource allocation in Sandpiper.

In addition to the usage of utility functions, constraint problems, queueing models and reinforcement learning, genetic algorithms are widely used in the context of dynamic resource allocation in clouds [2,5,30,78,84,85]. One such solution is outlined in [2] where genetic algorithms are grouped for solving server consolidation problem with item incomparability, bin-item incomparability and capacity constraints. The authors claim that the method performs particularly well when the conflict graph is sparse and disjoint.

The study outlined in [30], provides a solution similar to the usage of genetic algorithm proposed in [2]. The system architecture is based on applications that run in Web 2.0 environment. The authors claim that these type of applications provide good examples for the resource consolidation and high application availability due to their long running nature and range of variability. The latter is caused by visitor loyalty which forms a constant load on the servers and random events that capture the high interest of users. The variability in loads are expected in the case of predictable events such as sports games. On the other hand, media hype or disasters are unpredictable. Genetic algorithm described in this work is carried out in an iterative manner by mutations until the solution converges to a near optimal solution. Therefore, starting with a large initial population is significantly increases the chances of obtaining an improved configuration. In this manner, each perturbation corresponds to a virtual machine migration. The process starts with determining the set of overloaded nodes. For every such node, a randomly selected virtual machine is migrated to another physical machine which is under-loaded. If no such node is available, a new node is created. Every mutated element replaces the existing ones if high application availability and utilization is achieved. The performance of this method is compared against a Round-Robin algorithm. Although the proposed system increases application availability and establishes better resource utilization, the average time required for convergence is 4 seconds within a small-scale environment. Therefore, it is safe to conclude that the method is likely to suffer from feasibility and scalability issues.

Furthermore, the study outlined in [49], proposes a self-adaptive knowledge management and resource efficient service level agreement enactment for cloud computing infrastructures. In order to achieve goals of minimizing service level agreement violations, the authors propose a escalation level approach which divides all possible actions into five levels. The proposed method starts by dividing resource allocation problem into smaller sub-problems using a hierarchical approach. These levels are called escalation levels and refer to the idea that a problem should be handled with the lowest possible level at all times to increase configuration speed. Every problem is tried to be solved at the lowest level, in case that this is not possible, a next level is exploited. The first escalation level tries to change a virtual machine configuration using local resources such as assigning more processing power to the virtual machine from the corresponding physical machine. Second level consists of migrating light weight applications between virtual machines on the same physical machine. Virtual machine migration is considered in level 3 which is formulated into a binary integer problem known to be NP-complete. In level 4, physical machines are switched on and off in order to avoid service level agreement violations. Level 5 is the last resort where the problematic virtual machines are outsourced to other cloud providers.

2.2 Distributed Resource Allocation Management

In contrast to using a global arbiter, some solutions adopt dynamic approaches in which aim to compute new resource allocations in a distributed manner. In the work outlined in [73], a cooperative distributed control frame work is suggested where each physical machine can host virtualized application environments. The system assumes one-time static placement of virtual machines onto physical machines where the goal is to optimize the CPU share of each virtual machine in order to meet desired response times. The main problem is decomposed into sub-problems where applications are mapped to a physical machine via a dispatcher. The physical machines are coupled with a look ahead controller which manages CPU shares of each virtual machine under their control where statistical forecasting is used to predict workloads. In this manner, the suggested distributed framework has the following two properties: (1) look ahead controllers having a low computational complexity, and (2) the framework being able to tolerate virtual machine failures.

In [25], a holistic approach for energy management in IaaS clouds is outlined. The solution proposed in this paper is concerned with the wasted energy when hosting cloud services due to the idle physical machines. In this manner, dynamic virtual machine placement is used to conserve energy as well as detecting service level agreement violations by using limit checking. The architecture uses a semi-distributed approach where the physical machines are arranged in groups each of which has a group leader. Each physical machine monitors the resource usage and the collected information is transmitted to the associated group leader which aims to calculate new virtual machine to physical machine mappings in order to minimize the number of machine used in a group.

A similar study is conducted in [16] where dynamic resource allocation is carried out by distributed decisions in cloud environments. In this architecture, each physical machine is coupled with a capacity and utility agent which is responsible for maximizing the utility of an associated physical machine. The agent collects resource usage information from underlying virtual machines via API calls and computes an index based on the collected data that represents the degree of utility. Each computed index is transmitted to a distributed capacity agent manager which is responsible for the following tasks: (1) maintain a database with the indexes, (2) calculate new virtual machine to physical machine mappings periodically for reallocation.

Dynamic Resource Allocation Using Virtual Machines for Cloud Environment proposes an architecture where each physical machine is coupled with a Xen hypervisor [77]. In addition, each virtual machine can employ one or more applications of type Web server, remote desktop, DNS, E-mail and Map-Reduce. The resource usage of each virtual machine is monitored by Xen and the collected data is processed by an Usher local node manager which employs a scheduler. The scheduler uses a predictor to forecast virtual machine resource demands of CPU and memory. In this context, a hot spot and a cold spot solver is integrated into the module in order to cope with over and under utilization respectively. Furthermore, a concept of skewness is proposed which helps to blend virtual machines together in order to achieve a overall utilization among all resource dimensions on a physical machine.

A SLA-compliant Cloud Resource Allocation Framework for N-tier Applications outlines a distributed three layer architecture called Innkeeper [50]. Innkeeper provides three brokers, one for each physical host, one for each cluster of hosts and one for the cloud of clusters. Host Innkeeper either accepts or rejects virtual machine chosen for placement by checking the local resource usage. The cluster innkeeper is responsible for rejecting or accepting virtual machines at the cluster level with a given service level agreement. The cloud innkeeper is the central system that decides on the placement of virtual machines in a cluster. The placement problem is formulated through a multi-objective knapsack problem and the authors suggest a range of optimization techniques as well as genetic algorithm. Although the design is inherently scalable, the solution is only tested within a small-scale environment where only CPU utilization is considered.

IMPROMPTU aims to provide another solution to the problem of resource consolidation in computing networks [80, 81, 83]. The architecture defines two thresholds for over and under utilization, which is likely to cause service level agreement violations, and tries to optimize the overall system by trying to keep resource usage between these thresholds. The main contribution of this paper is two-fold. First, the authors adopt a distributed approach where resource management is carried out by autonomous node agents that are tightly coupled with each physical machine. Second, node agents carry out configurations in parallel through Multiple Criteria Decision Analysis (MCDA). Each node agent observes the resource usage on a physical machine by aggregating the resource usages of hosted virtual machines. In this manner, the proposed system reacts to workload changes based on the lower and higher thresholds. In the case of lower thresholds, also commonly referred to as cold spots in literature, the node agents try to evacuate all of the hosted virtual machines to hibernate the physical machine. This behaviour allows for energy conservation as well as better utilization in the data center. On the other hand, in the case of over utilization, virtual machines with the highest amount of resource usage is chosen for migration. Two types of decisions are made by the node agents: (1) choosing a virtual machine for migration except for the case of evacuation, and (2) choosing a target physical machine for migration. The decision making process is carried out by one of the investigated MCDA methods such as PROMETHEE II, ELECTRE III and PAMSSEM II. Furthermore, similar to the concept of skewness mentioned in [77], IMPROMPTU uses a set of influence criteria in both types of decision making process. During the VM selection process, it can be defined as the measure of the effect of a virtual machine on threshold violations. This measure is captured over time on hosting physical machines for all resource dimensions. In this context, the evaluation process favours virtual machines with higher influence criteria. On the other hand, the influence criteria of a physical machine is defined as the sum of all influence criteria of the hosted virtual machines. In this case, the evaluation process favours physical machines with lower influence criteria. The reason behind both of these choices is to blend problematic virtual machines with the lower influence ones in order to avoid future threshold violations and establish an even utilization through out the data center. It is important to note that the proposed solution is highly scalable and efficient. Finally, it leaves the question of whether more efficient MCDA methods exist in the domain of dynamic resource allocation.

2.3 Conclusion and a Brief summary

In the current state of virtualized servers, two main approaches are generally adopted during the computation of new possible virtual machine to physical machine mappings. The most common method is the usage of thresholds where a target service level agreement is considered to be violated when certain resources are under or over utilized. On the other hand, some researchers considered the problem of dynamic resource allocation through the means of autonomous control. Therefore, various methods such as reinforcement techniques and control feedback systems are widely used. Besides the challenges that surface with this problem, deciding and setting the parameters require utmost care in the case of threshold based approaches. On the other hand, applying reinforcement learning techniques require a full fledged integration, good choice of policy exploration strategies and convergence speed-ups [22].

The approaches using centralized global arbiters outlined in studies [7, 15, 19, 51, 70, 72] can be efficient in small scale data servers however they suffer from scalability and feasibility issues in realistic large-scale settings. Moreover, some solutions propose constraint programming which also suffers from performance issues [35]. This in turn, results in potentially stale solutions with respect to the speed at which the workloads change in modern data centers [80,81]. A queueing model based approach is outlined in [69] which is then used to combined with reinforcement learning to produce a hybrid method [67,68].

The analytic and statistics based approach generally compute new configurations faster [3,8,44,74]. However, the computed placement reconfigures the entire resources in the data center making the implementation require a vast number of migrations, which in turn, lead to critical feasibility issues [80, 83]. In addition, the centralized approaches share the same nature of tightly packing virtual machines onto physical machines resulting in a high number of service level agreement violations. Furthermore, the new configurations are computed on a time interval basis, which naturally has a negative impact on responsiveness.

Genetic algorithms are also widely used in the context of dynamic resource allocation in clouds. Given a starting sample, each mutation represents a new configuration for the data center. In the case where the new mutations is a more efficient configuration, the old is replaced and the procedure is continued until the solution converges. Therefore, the effectiveness of the method is associated with the initial sample size. Due to the computationally complex nature of the algorithm, the solutions outlined in [2, 5, 30, 78, 84, 85] also suffer from performance and responsiveness issues.

Distributed solutions proposed in [16, 25, 50, 73, 77] are inherently resistant to scalability and feasibility issues. However, the semi-distributed architecture outlined in [25] requires local monitors to report to group leaders which in turn suffers in the context of responsiveness. Although [73] is a fully distributed model, the usage is limited to data centers that host Map-Reduce applications.

After the investigation of the existing approaches and related research on the problem of dynamic resource allocation in data centers, it can be concluded that that a valuable solution needs to have the following properties: (1) issues related to scalability and feasibility needs to eliminated, (2) virtual machine to physical machine configurations need to be computed fast in order to avoid stale solutions, and (3) these configurations need to be computed in a reactive manner in order for the system to be adaptive and self-optimizing.

Chapter 3

Multi Criteria Decision Analysis Methods

In a perfect world, choosing between alternatives according to multiple criteria is a trivial task as the decision maker is faced with a dominant alternative. In such cases, a winner is deemed by a clear decision and no further evaluation or consideration is necessary. However, a decision making process can be extremely complex and challenging in real world situations. In personal life, a decision maker may be trying to decide on purchasing a car or a house. In this context, many criteria come into play such as, safety, comfort, style, initial costs, maintenance, insurance options and resell value. In a professional setting, a company may be trying to decide on an employee for promotion. In this manner the criteria could be education, salary, job experience, charisma and social skills.

At first glance, making a decision may look simple easy as human beings are excellent decision makers. Naturally, this ability is present to us ever since we were born and it improves with experience. However, almost in all cases, the problem has conflicting criteria. This, in turn, complicates the process. Let's assume the first example of purchasing a car which presents multiple conflicting criteria. The most stylish vehicle is not necessarily the most economical purchase. In addition, the insurance and maintenance costs are most likely to be higher. Furthermore, the most economical purchase may not have the greatest resell value and safety. Such problems quickly turn into a complex mathematical problem that does not a have a trivial solution.

Let A denote the set of alternative actions and C be the set of criteria to be considered when evaluating the performances of the alternative actions. Mathematically, for $a_1, a_2 \in A$, alternative a_1 dominates a_2, if, $c(a_1) \geq c(a_2), \forall c \in C$, and $\exists c_l \in C$ such that $c_l(a_1) > c_l(a_2)$. That is, an alternative action is "at least as good as" all the other alternative actions according to all criteria and there is at least one criterion which the alternative performs strictly better than the other alternative action. In this manner, an alternative is *Pareto Optimal*, if no other alternative outperforms the alternative at hand over any evaluation criteria. In addition to the conflicting criteria, non-measurable or intangible criteria are often present. This calls for the need of forming a comprehensive judgement measure in order to determine the better-performing alternative by evaluating the alternative actions according to the all criteria. Such problems are often referred to as an *aggregation problem* [58].

Aggregation problems form the basis of many MCDA methods which can be summarized in two categories: 1) Methods that are based on mathematically explicit multiple aggregation procedures (MCAP), and 2) Methods that let the decision maker interact with the implicit mathematical procedures.

MCAP based approaches are designed to provide a clear solution to the aggregation problem for any pair of alternatives according to a various *inter-criteria parameters* and a *logic of aggregation* [58]. The addition of the inter-criteria parameters help to define a relation between the criteria taking into consideration factors such as conflicts. These parameters are mainly referred to as veto thresholds, concordance thresholds, weights, scaling constants, aspiration and rejection levels [9, 57]. The inter-criteria parameters are assigned as mathematical values by a logic of aggregation. Logic of aggregation, usually gathered from the decision maker or a system administrator, represents the relationship where an alternative actions performance is deemed satisfactory or unsatisfactory. Hence, these parameters can only apply to a specific decision making problem and is defined for single use [9,57]. MCAP methods can be categorized as follows: 1) Multi-attribute Utility Theory Methods and 2) Outranking Methods [71] which are also called Methods Based on Synthesizing Criterion and Methods Based on Synthesizing Preference Relation System [58]. In addition to these two major categories, some other approaches include methods based on simulated annealing and evolutionary algorithms [17], rough sets [33,65], artificial intelligence [55] and fuzzy subsets [52]. Generally, a set of feasible solutions are produced on the majority of these methods. Whereas, on MCAP based methods, either a single better-performing alternative or a rank of alternative actions are produced depending on the choice problematic.

In the remaining of this chapter, we focus on the following MCAP based methods: (1) Single Synthesizing Criterion Methods and (2) Outranking Methods. These two approaches are chosen due to the mathematically explicit nature of the problem domain of this thesis. A set of MCDA methods used in this thesis are explained in detail which also provides a brief survey of existing methods belong to associated family of approaches mentioned above.

3.1 Single Synthesizing Criterion Methods

As the name suggests, single synthesizing criterion approach MCDA methods aim to reduce the different criteria used in the decision making process into a one single comprehensive index by making use of some rules, processes and mathematical formulae. In addition, imperfect knowledge and inconsistency is tolerated to some mathematical extent [58]. These methods are often considered the most traditional approach and the alternative actions form a complete pre-order [23, 58].

In the rest of this section, two well-known MCDA methods that are used in this thesis, Simple Multi-Attribute Rating Technique (SMART) and Analytic Hierarchy Process (AHP) is explained in detail.

3.1.1 Simple Multi-Attribute Rating Technique

Simple Multi-Attribute Rating Technique (SMART) is the simplest form of multi-attribute utility theory models [24]. In order to rank the alternatives, different performance measures of alternatives under various scales need to be converted into a common internal scale. Mathematically, this process is done by using weighted linear averages, which, in turn, provides a very close approximation to utility functions.

Let A be the set of alternatives, $u_i(a)$ be the performance of alternative a under criterion i. Then, the aggregated performance value u_a is given by the utility function,

$$u_a = \sum_{i}^{n} w_i a_i \tag{3.1}$$

where w_i is the weight of the criterion i.

3.1.2 Analytic Hierarchy Process

Analytic Hierarchy Process (AHP) is a single-synthesizing MCDA method where the decision maker derives priority scales from the relative judgements through pairwise comparisons among all alternatives under the supervision of a hierarchical structure [61, 62]. The method has the ability to deal with both tangible and intangible criteria, focusing mainly on the latter, by enabling the decision maker to use domain expertise or collected statistics. The main strength of AHP lies in its capability of making comparisons of intangibles in relative terms. It is due to this feature that the experience and the knowledge of the decision maker is reflected on the final answer. In this context, the decisions are made in a qualitative fashion, and expressed numerically through the fundamental scale of AHP [62]. AHP is based on four main axioms [62]: (1) reciprocal judgements, (2) homogeneous elements, (3) hierarchical or feedback dependent structures, and (4) rank order expectations. In addition, a small inconsistency in the judgement is also allowed.

The mathematical structure of AHP can be investigated by considering a case where the decision maker needs to express judgements on a set of alternatives $A = a_1, a_2, ... \geq, a_n$. The process starts by attaching a positive real number, v_i, to each alternative action, a_i, where v_i denotes the importance of alternative a_i. At this point, the following assumption is made. After when the decision maker assigns a numerical value to each alternative, the relative importance of two alternatives a_i and a_j can be expressed as ratio of v_i/v_j. In the cases where, $v_i/v_j < 1$, then a_j is more important than a_i by a factor of v_j/v_i. When an alternative is compared to itself, naturally, $v_i/v_i = 1$. Once an importance value is assigned to each alternative, then the decision maker can construct an $n \times n$ matrix $M = (m_{ij})$, where $m_{ij} = v_i/v_j$, that has the following properties [6, 61]:

1. $m_{ii} = 1, m_{ij} > 0$ and $m_{ji} = m_{ij}^{-1} \forall i, j$

2. $m_{ij} m_{jk} = m_{ik} \forall i, j, k$

3. The matrix M has rank 1, with each column vector proportional to the vector $C = (v_1, v_2, ..., v_n)^T$ and each row proportional to the vector $C = (v_1^{-1}, v_2^{-1}, ..., v_n^{-1})$.

4. 0 is an eigenvalue of M with the multiplicity $n - 1$, and the trance of M is n;

it follows from this that there is a remaining eigenvalue which is simple and equal to n.

5. C is a column eigenvector and R is a row eigenvector of M corresponding to the eigenvalue n. Thus, in this special case, the relative importance measurements of alternative a_i appears in the form of an eigenvector corresponding to the largest positive eigenvalue of a matrix with positive entries.

Commonly, due to the challenges faced by the decision maker when assessing the value of the comparison of two alternatives i, j, the exact value of m_{ij} can not be given. Instead, this value is only an estimation. In this context, the judgement matrix M is perturbed, which, in turn, makes the eigenvectors and eigenvalues perturbed. In the case where this perturbation is small, then there is an eigenvalue close to n whose column vector can be regarded as good approximation to the relative importance judgements of the alternative actions. In this manner, the ranking problem can be defined as follows. There exists an ideal, yet unknown, ranking for the n alternatives, expressed in the form of a vector. When the judgement matrix M is constructed as defined above, naturally, the first property of being a reciprocal matrix is satisfied. Also, the matrix is consistent when the entries m_{ij} satisfy the second property. Then, the k^{th} column is equal to a_{jk} multiplied by the j^{th} column, so the rank of M is 1. Consequently, M satisfies the fourth property [6, 62]. In addition, if $(c_1, c_2, ..., c_n)^T$ is any column eigenvector and $(r_1, r_2, ..., r_n)$ any row eigenvector with eigenvalue n, then it can be concluded that $r_i/r_j = c_i/c_j = m_{ij}$. In other words, the weights of the alternatives can be extracted from the eigenvectors that is consistent with the pairwise judgements [6, 62].

AHP assists the decision maker when assigning a value , m_{ij}, to a judgement. Judgements express the property that to what extent an alternative is dominant or better than the other when compared under a specific criterion. This measure comes from the fundamental 1 - 9 scale of AHP which is derived from stimulus response ratios [61–63]. The following value are assigned when:

1, two alternatives equally contribute to the goal

3, an alternative has moderate importance over the other

5, an alternative has strong importance over the other

7, an alternative has demonstrated importance over the other

9, an alternative has extreme importance or domination over the other

The relative weights for the alternatives are determined as the entries of the eigenvector associated with the eigenvalue of the matrix.In addition to this approach, there are other methods that can be used to determine the extract the weights from the impact matrix. The logarithmic least squares method operates by calculating the geometric mean of each row in the impact matrix, then taking the sum of geometric means, and normalizing each geometric mean by the computed sum.

Structurally, AHP arranges the criteria in a tree structure with the goal being at the root, followed by criteria, sub-criteria and alternatives. At each level, pairwise comparisons are carried out to express judgements among criteria or alternatives with respect to the fundamental scale of AHP. These judgements are placed in the impact matrix, which is used to derive the relative performance values or weights through calculating its eigenvector. In the final step, these values are aggregated with respect to the weight of upper hierarchies in order to produce a final ranking.

AHP provides a mathematical way to investigate the consistency of the judgements computed from the pairwise comparisons. In the cases of inconsistency, an inconsistent impact matrix M is subject to Perron's theorem [63], which states that if M is a matrix with strictly positive entries, then M has a simple positive eigenvalue, λ_{max} which is not exceeded in absolute value by any of its complex eigenvalues; every row eigenvector or column eigenvector corresponding to λ_{max} is a constant multiple of an eigenvector with strictly positive entries. Let $M = (m_{ij})$ be an $n \times n$ reciprocal matrix, if λ_{max} is the maximum absolute eigenvalue of M, $\lambda_{max} \geq n$, and M is consistent if and only if $\lambda_{max} = n$. This, in turn, means that the difference $\lambda_{max} - n$ can be used to measure the consistency of the impact matrix M [62, 63]. The sum of all the eigenvalues of M, also called the trace of M, is n, $\lambda_{max} - n$ is the negative sum of the remaining eigenvalues of M. The average of these eigenvalues is $-\mu$,

$$\mu = \frac{\lambda - n}{n - 1} \tag{3.2}$$

where μ is the consistency index of M. For a sample of 500 randomly generated reciprocal matrices, μ is satisfactory if it is less than 10%. The judgements are more likely to be inconsistent, as this measure gets larger [62].

3.2 Outranking Methods

The outranking based MCDA methods follow a successive pairwise comparison of alternative actions under all criteria. As opposed to the Single Synthesizing Criterion methods which consider each alternative action in isolation to produce a complete pre-order, outranking approaches produce a synthesizing preference relation system based on the successive pairwise comparisons [58].

It is important to note that, the pairwise comparisons may not always be transitive, which, in turn produces inconsistent preference relations. Furthermore, incomparability can also cause flaws in the design of such relations. In Single Synthesizing Criterion methods, the aggregation procedure is sufficient to rank the alternative actions since no such in-transitivity and incomparability is present. On the other hand, Outranking Methods call for another step called exploitation procedure in order to overcome this challenge [58].

In the rest of this section, the following six well-known outranking methods, ELECTRE III, PROMETHEE II, PAMSSEM II, ORESTE, REGIME and QUALIFLEX, are explained in detail.

3.2.1 ELECTRE III

ELECTRE methods model preferences by using outranking relations "*at least as good as*", denoted by S [27,57]. The following four situations occur when two alternatives a and b are compared under the preference relation S: (1) aSb and bSa, or aPb, meaning a is strictly preferred to b, (2) bSa and aSb, or bPa, meaning b is strictly preferred to a, (3) aSb and bSa, or aIb, meaning a is indifferent to b, (4) Not aSb and not bSa, or aRb, meaning a is incomparable to b. ELECTRE methods introduce the incomparability preference relation, R, when the decision maker is not able to compare two alternatives. Note that this is not present in the MAUT methods [27].

Two major concepts, *concordance* and *non-discordance* provides the basis for the outranking relations [59,60]. The former concept states that in order for aSb to be valid, a *sufficient* majority of criteria should be in favour of this assertion. Whereas, the latter states that, aSb is valid if none of the criteria in the minority oppose too strongly to the assertion. Conclusively, the assertion aSb is valid only when these two conditions are satisfied. Due to the Condorcet effect and incomparabilities, an outranking relation may not be transitive. This requires the need for another procedure, called exploitation, in order to extract results from such relations that fit a given problematic - choice, ranking or sorting problematic [27].

The importance coefficients and the veto thresholds define the relative importance attached to each criteria [59, 60]. The importance coefficients are the intrinsic weights of each criterion. In detail, for a given criterion, the weight w_j, reflects is voting power when it contributes to the majority which is in favour of an outranking. Whereas, the veto threshold is the power assigned to a given criterion to be against this assertion. That is, in order for *"a outranks b"* to be valid, the difference between the evaluation of a and b is not greater than the veto threshold assigned to this criterion [59, 60]. The thresholds can be variable or constant along a scale [58]. Other important concepts that underline ELECTRE methods are the preference and indifference thresholds. These are referred to as discrimination thresholds that lead to a pseudo-criterion model. When the difference between evaluations of two different alternatives under a specific criterion is considered: (1) preference thresholds, p_j, justifies the preference in favour of one of the two actions, (2) indifference thresholds, q_j, defines compatibility with indifference between two alternatives. In addition, these thresholds can be interpreted as an hesitation between opting for a preference or an indifference between two alternatives [58].

ELECTRE III belongs to the ranking problematic domain of ELECTRE methods. In ranking problematic, the process ranks all the candidates belonging to a set of alternatives from the best to worst, possibly with *ex aequo*. ELECTRE III addresses the consideration of inaccurate, imprecise, uncertain or ill-determined information. The main contribution of ELECTRE III when compared to the rest of the ELECTRE methods is the introduction of pseudo-criteria instead of true-criteria [27, 59]. In ELECTRE III, the outranking relation is defined as a fuzzy relation, that is constructed by a credibility index. The credibility index, $\rho(aSb)$, characterizes the credibility of the assertion *"a outranks b"* and is defined by both concordance index, $c_j(aSb)$, and the discordance index, $d_j(aSb)$, for each criterion $g_j \in F$ [27].

The concordance index is defined as follows

$$c_j(a,b) = \begin{cases} 1 & \text{if} \quad g_j(a) + q_j \geq g_j(b) \\ 0 & \text{if} \quad g_j(a) + p_j \leq g_j(b) \\ \frac{p_j + g_j(a) - g_j(b)}{p_j - q_j} & \text{otherwise} \end{cases} \tag{3.3}$$

Let w_j be the importance coefficient or weight for criterion j. The valued outranking relation is defined as follows

$$C(a,b) = \frac{1}{w} \sum_{j=1}^{r} w_j c_j(a,b) \tag{3.4}$$

where $w = \sum_{j=1}^{r} w_j$

The discordance index can be represented as follows

$$d_j(a,b) = \begin{cases} 0 & \text{if} \quad g_j(a) + p_j \geq g_j(b) \\ 1 & \text{if} \quad g_j(a) + v_j \leq g_j(b) \\ \frac{g_j - g_j(a) - p_j(b)}{v_j - p_j} & \text{otherwise} \end{cases} \tag{3.5}$$

The credibility index is defined based on these definitions of both indexes as

$$S(a,b) = \begin{cases} C(a,b) & \text{if} \quad d_j(a,b) \leq C(a,b) \quad \forall j \\ C(a,b) \prod\limits_{j \in J(a,b)} \frac{1 - d_j(a,b)}{1 - C(a,b)} & \text{otherwise} \end{cases} \tag{3.6}$$

where $J(a,b)$ is the set of criteria such that $d_j(a,b) > C(a,b)\ \forall j$.

The definition of credibility index is based on three concepts [27]: (1) In the case where there is no discordant criterion, the credibility of the assertion is equal to the comprehensive concordance index, (2) If a discordant criterion uses veto option, then the assertion is deemed not credible, (3) When the comprehensive concordance index is lower than the discordance index on the discordant criterion, the credibility index becomes lower than the comprehensive concordance index, because of the opposition effect on this criterion.

Once the credibility index is computed for all alternatives, ELECTRE III proceeds with the exploitation procedure. Two pre-orders Z_1 and Z_2 are derived from the constructed fuzzy relations. The former is computed by a descending distillation, whereas, the latter is computed by an ascending distillation. The partial pre-order Z is defined as the intersection of the two pre-orders to come up with a final ranking of alternatives.

3.2.2 PROMETHEE II

The PROMETHEE methods use outranking to rank alternatives in a multiple criteria decision making problem. It needs the following rather clear and simple information, in order to assist the decision maker: (1) information between criteria, and (2) information within each criterion [10, 13].

Information between criteria refers to the relative importance of each criterion's influence during the decision making process. The set, w_j, $j = 1, 2, ..., k$, represents the weights associated with each criteria. Furthermore, these weights are non-negative and independent from the scales of evaluation criteria [11, 12]. Consequently, higher weights result in a higher influence on the decision made. Also, these weights can be normalized, so that $\sum_{j=1}^{k} w_j = 1$.

The PROMETHEE methods gather information within each criterion by pairwise comparisons of alternatives under a specific criterion. The pairwise comparisons are carried out for all alternatives, where, little deviations represent either weak or no preference at all. On the contrary, large deviations suggest the strength of preference proportionally. The strength of preference can be represented by real numbers between 0 and 1, that are computed by the preference function $P_j(a,b) = F_j[d_j(a,b)], \forall a, b \in A$, under a specific criterion where $d_j(a,b) = g_j(a) - g_j(b)$ and $0 \leq P_j(a,b) \leq 1$ [11, 12, 14]. The function $P_j(a,b)$ represents the preference of alternative a over b for observed deviations between their evaluations under criterion $g_j(\cdot)$. In the case where the evaluation difference is negative, the preference equals 0. Mathematically, $P_j(a,b) > 0 \Rightarrow P_j(b,a) = 0$. In addition, reversing the preference function results in minimized criteria so that $P_j(a,b) = F_j[-d_j(a,b)]$.

The pair $\{g_j(\cdot), P_j(a,b)\}$ is named the *generalised criterion* associated with the criterion $g_j(\cdot)$. PROMETHEE offers six main preference functions in order to define identification for each criterion [13].

Type 1: Usual Criterion

$$P(d) = \begin{cases} 0 & \text{if} \quad d \leq 0 \\ 1 & \text{if} \quad d > 0 \end{cases}$$

Type 2: U-Shape Criterion

$$P(d) = \begin{cases} 0 & \text{if} \quad d \leq q \\ 1 & \text{if} \quad d > q \end{cases}$$

Type 3: V-Shape Criterion

$$P(d) = \begin{cases} 0 & \text{if} \quad d \leq 0 \\ \frac{d}{p} & \text{if} \quad 0 \leq d \leq p \\ 1 & \text{if} \quad d > p \end{cases}$$

Type 4: Level Criterion

$$P(d) = \begin{cases} 0 & \text{if} \quad d \leq q \\ \frac{1}{2} & \text{if} \quad q < d \leq p \\ 1 & \text{if} \quad d > p \end{cases}$$

Type 5: V-Shape with Indifference Criterion

$$P(d) = \begin{cases} 0 & \text{if} \quad d \leq q \\ \frac{d-q}{p-q} & \text{if} \quad q < d \leq p \\ 1 & \text{if} \quad d > p \end{cases}$$

Type 6: Gaussian Criterion

$$P(d) = \begin{cases} 0 & \text{if} \quad d \leq 0 \\ 1 - e^{-\frac{d^2}{2s^2}} & \text{if} \quad d > 0 \end{cases}$$

where q is a threshold of indifference, p is a threshold of preference and s is an intermediate value between s and p. The indifference threshold q represents the largest evaluation difference that can be considered negligible. Whereas, the preference threshold, p is the smallest evaluation difference sufficient for full preference. Other generalized criteria can also be used in addition to the six types of functions defined above.

After information between criteria, w_j, and within each criterion $\{g_j(\cdot), P_j(a,b)\}$ are evaluated, PROMETHEE computes *aggregated preference indices.* For an alternative pair $a, b \in A$, the aggregated preference indices are computed as follows: $\pi(a,b) = \sum_{j=1}^{k} P_j(a,b)w_j$ and $\pi(b,a) = \sum_{j=1}^{k} P_j(b,a)w_j$. The former measures to what degree alternative a is preferred over alternative b over all criteria, and the latter vice versa [11, 12].

In a set of alternatives A, each alternative is compared to remaining $n-1$ alternatives. The outranking flows are defined as follows [10–12].

Positive Outranking Flow:

$$\phi^{+} = \tfrac{1}{n-1} \textstyle\sum_{x \in A} \pi(a,x)$$

Negative Outranking Flow:

$$\phi^{-} = \tfrac{1}{n-1} \textstyle\sum_{x \in A} \pi(x,a)$$

The positive outranking flow represents to what degree an alternative a outranking all the others. Whereas, the negative outranking flow represents to what degree an alternative a is outranked by the others. In other words, these are a measure of the sum of powers and weaknesses [13].

In the last step, PROMETHEE II method produces a complete ranking on the set of alternatives by combining the positive and the negative outranking flows into a new measure called *net outranking method* defined as follows:

Net Outranking Flow:

$$\phi(a) = \phi^{+}(a) - \phi^{-}(a)$$

3.2.3 PAMSSEM II

PAMSSEM II is a hybrid outranking method that combines the key features of ELECTRE III, PROMETHEE II and NAIDE methods [34]. The outranking method is designed to deal with heterogeneous and missing data. As in ELECTRE III, PAMSSEM II is making use of pseudo-criteria such as discrimination thresholds and considers two types of inter-criterion information: (1) a set of coefficients or weights that represent the relative importance between each criteria, and (2) veto thresholds between criteria when needed.

PAMSSEM II computes a concordance, local discordance and outranking indexes for each pair of alternatives in the aggregation phase exactly the same way as in ELECTRE III. Once the concordance matrix is constructed, an outranking degree is computed for each pair of alternatives $a, b \in A$, represented by $\sigma(a, b)$. Similar to PROMETHEE II, this measure defines to what degree alternative a is preferred over alternative b over all criteria globally.

The second and the last step in PAMSEEM II is the exploitation phase, which is carried out exactly the same way as in PROMETHEE II. A total pre-order forced by computing a net outranking flow for each alternative a as $\sigma(a) = \sigma^{+}(a) - \sigma^{-}(a)$. Finally, the alternatives are sorted in decreasing order according to this measure in order to produce a final ranking.

3.2.4 ORESTE

ORESTE extends traditional MCDA approaches to include both the ranking of alternatives by criteria as well as the ranking of the criteria themselves [56]. In this context, the decision maker is not responsible for determining weights, but instead assigns a preference structure on the set of criteria F. This in turn, is used to define a complete preorder S such that $S = I \cup P$, where the preference relation S is strongly complete and transitive, the indifference I is symmetric and the preference P is asymmetric. The method is designed to form a global preference structure on the set of alternatives A that reflects the following two relations: (1) preference relations between all alternatives under each criterion and, (2) preference structure among the criteria.

ORESTE method outputs a global preference structure such that the alternatives are evaluated according to each criteria where the preference of criteria reflects on the outcome. In order to come up with such preference structure, the method operates in three phases. In the first phase a projection of the position-matrix is constructed followed by a ranking of the projections in the second phase and lastly these global ranks are aggregated.

Before starting the first phase, each criterion is given a rank related to its position in complete preorder among criteria represented by r_j where J is the set of criteria and $j \in J$. Also, each alternative is given a rank with respect to each criterion as $r_j(a)$ where $a \in A$. This procedure results in n preoders. Besson's mean rank is used to convert these preorders into numerical values. Note that, these preorders can be represented as a position table.

In the projection phase, an origin is introduced to calculate the distances of each alternative from the position table. Several definitions of this measure is possible. One common way is a linear oblique projection. This defines the distances of the projections from the origin by

$$d(0, a_j) = \alpha r_j(a) + (1 - \alpha) r_j$$

where α stands for substitution rate ($0 < \alpha < 1$). The substitution rate is used to reflect the weight of the ranks of criteria and the ranks of alternatives. In cases where, α is set to 0.5, the substitution rate defines equal importance for both criteria and alternatives.

In the ranking phase, since the relative positions of the projections are more important than the numerical value of the distances, the projections needs to be ranked. The distance matrix is converted into a global rank matrix using Besson's mean rank.

In the aggregation phase, a global weak order is constructed by summation of the global ranks for each alternative over the set of criteria;

$$R(a) = \sum_j R(a_j)$$

This order is not considered reliable as neither indifference nor incomparability are taken into account. Preference intensities are introduced to overcome this limitation; $C'(a, b) = \sum_{j:aP_jb}[R(b_j) - R(a_j)]$ and then normalized by division of upper bound $C'_{\text{upperbound}} = (a - 1)k^2$ where a is the number of alternatives and k is the number of criteria. The net preference intensity between alternatives a and b is denoted by $C(a, b)$.

Three thresholds β, C^* and γ are introduced to distinguish between indifference, incomparability and preference. The threshold β is used to make the distinction between indifference and preference where as C^* is used for indifference and incomparability. γ indicates a difference between preference and incomparability and represents the risk that decision maker selects one candidate against another without being aware of the conflicting situation between both candidates [56]. These thresholds can be mathematically represented [56] as follows: β and C^* are upper bounded by; $\beta \leq 1/[(a-1)k]$ and $C^* \leq 1/[2(a-1)]$ as γ is lower bounded by; $\gamma > (k-2p)/4p$

where p is the degree of perturbation. In order to make a clear distinction between preference and incomparability, the values of β and C^* need to be fixed to their upper bounds.

In the decision making procedure, the preference intensities between two candidates a and b are investigated. If the difference between $C(a,b)$ and $C(b,a)$ is smaller than the threshold β, then indifference occurs when both of these intensities are smaller than the threshold C^*, else there is incomparability. Preference or incomparability occurs when the difference between two intensities is larger than or equal to β. If the relation of the smallest normalized preference intensity to the difference in normalized intensities is larger than or equal to γ then incomparability occurs, else preference is assessed. This procedure results in a global preference structure, which in turn, is used to rank all alternatives.

3.2.5 REGIME

REGIME can be viewed as ordinal generalization of pairwise comparison methods where the main part of the decision making process relies on concordance analysis [36].

Let A be the set of alternatives and F be the set of criteria under consideration. The concordance set of two alternatives a and b is defined as the set of criteria for which a is as good as b. The concordance measure between candidates a and b is given by, $C_{ab} = \sum_{j \in C^*_{ab}} \pi_j$ where C^*_{ab} is the concordance set and π_j is the weight of criterion $g_j \in F$. The preference is based on the difference between C_{ab} and C_{ba}. The focus is on the sign of this measure. Naturally, if the sign is positive a is preferred to b, and if the sign is negative then it is assessed that b is preferred to a. Incomparability occurs in the cases when $C_{ab} - C_{ba} = 0$

In the first step of the REGIME method, a regime matrix is constructed. This matrix is populated by pairwise comparisons of the alternatives under all criteria. For alternatives a and b, +1 is noted when a has a better rank than b. -1 is noted when b has a better rank than a. The resulting matrix consists of the pairwise comparisons of the alternatives for each criterion that belongs to F. Mathematically, comparing a and b under criterion j can be represented as;

$$C_{ab,j} = \begin{cases} +1 & \text{if } r_{aj} < r_{bj} \\ 0 & \text{if } r_{aj} = r_{bj} \\ -1 & \text{if } r_{aj} > r_{bj} \end{cases}$$

where r_{aj} is the rank of alternative a according to criterion g_j. When two alternatives are compared under all criteria, a vector called regime $(C_{ab,j})$ is formed. These regimes are used to determine rank order of alternatives. The concordance index can be represented as;

$$C_{ab} = \sum_j \pi_j c_{ab,j}$$

. It is important to note that, Regime method can deal with the situation where the weights are not explicitly known but an ordinal information is given.

In the decision making phase, when deciding on a preference relation between a and b the concordance indexes are compared. aPb is assessed when $C_{a,b} - C_{b,a} > 0$ and bPa when $C_{b,a} - C_{a,b} > 0$. Also, incomparability ,aRb, is denoted when $C_{a,b} - Cb, a = 0$

3.2.6 QUALIFLEX

QUALIFLEX is an outranking decision making method based on the evaluation of all possible permutations of rankings in an alternative set A. This method is a generalization of Jacquet-Lagreze's permutation method and the aggregation procedure is based on Kemeny and Snell's rule [42].

The method begins by constructing an *impact matrix*, F, where the columns consist of the alternatives and the rows consist of the criteria. The impact matrix is populated with the ranks of alternatives according to each criterion, and the outcome of this evaluation is placed in the appropriate position. Naturally, each row in the matrix is ordinal and take the form of preorders. Once the impact matrix is constructed, another matrix is formed to compute the comprehensive concordance/discordance indices for every possible permutation of the alternative set. In this matrix, the columns consist of the permutations and the rows consist of the criteria. Concordance/discordance index is calculated for each pair of alternatives in the permutation for each criterion and then aggregated using the associated relative importance or weights to come up with the index of the permutation. The permutation with the biggest measure is deemed the winner. In the case of equal measures, one permutation is randomly chosen from the set of winner permutations.

More particularly, given a set of alternatives A, QUALIFLEX defines *concordance* and *discordance* indexes for each tuple of alternatives $(a, b) \in A$, at the level of preorder. The ranking of the k^{th} permutation under criterion j is then given by [48]:

$$I_{jk}(a,b) = \begin{cases} 1 & \text{if there is concordance} \\ 0 & \text{if there is ex aqeuo} \\ -1 & \text{if there is discordance} \end{cases}$$

Concordance is defined to be when a tuple (a, b) are equivalently ranked to within two preorders of each other. Discordance is defined to occur when (a, b) are not in concordance. Finally, ex aequo is defined to occur when (a, b) are exactly equivalent in rank. The concordance/discordance index, I_{jk}, for the k^{th} permutation and according to criteria j and the is then given by:

$$I_{jk} = \textstyle\sum_{a,b \in A} I_{jk}(a,b)$$

The comprehensive concordance/discordance index for the k^{th} permutation can then be given by:

$$I_k = \textstyle\sum_j \pi(f_j) I_{jk}(a,b)$$

where $\pi(j) \in [0, 1]$ is the weighting assigned to criteria j. In general, $|A|!$ possible permutations exist, and the permutation with the biggest comprehensive concordance/discordance measure is deemed the winner.

Chapter 4

IMPROMPTU Model

IMPROMPTU is a threshold based reactive autonomous resource consolidation manager that distributes the responsibility of computing new virtual machine to physical machine mappings among autonomous node agents. Autonomous node agents are tightly coupled with a unique physical machine in the data center. In this context, virtual machines hosted on that physical machine can be stopped, started, suspended or made subject to live migration by node agents. Which, in turn, gives them full control over the computational resources at hand.

The main responsibilities of the node agents can be categorized under the following three points: (1) watching the resource usages of each virtual machine hosted on a particular physical machine, (2) detecting undesirable conditions as they occur on these environments, and (3) eliminating under-utilization and over-utilization on the physical machine by carrying out a set of virtual machine migrations within the date center. In this manner, IMPROMPTU provides a completely reactive computational resource management system as new configurations are only computed when prompted by the sudden change in resource usages of a given environment. Once the undesirable conditions are captured, the node agents decide on the next course of action based on the MCDA model in management. In this thesis, we are investigating the behaviour of the IMPROMPTU model under the management of eight MCDA methods.

The rest of the section is organized as follows. In Section 4.1, we outline the reactive behaviour of the system by formally defining the physical machine states, the triggers and the reactions caused by undesirable conditions. As well as, describing the means of keeping the system self-aware. Section 4.2 describes the decision making process carried out by the node agents. In Section 4.3, we provide insight to the

Table 4.1: Table of notations

Notation	Description
P	Set of physical machines
p_i	A unique physical machine
V^i	Set of virtual machines
v_j^i	A unique virtual machine hosted on p_i
G_R	Set of resource metrics
$g_n(\cdot)$	Measured usage on resource dimension n
$g_n(v_j^i)$	Measured resource usage of v_j^i on n
$g_n(p_i)$	Measured resource usage of p_i on n
L_n	Lower threshold on n
H_n	Higher threshold on n
R_i	A unique reaction of a node agent
w_n	Dynamic weight assigned to n
y_n	Violation parameter on n
G_I	Set of influence criteria n
$u(p_i)$	Performance index of a unique physical machine
$u(v_j^i)$	Performance index of a unique virtual machine
S_i	A unique state of a physical machine

applications of eight MCDA methods on the IMPROMPTU model during the decision making process.

4.1 Reactive Resource Consolidation Management

IMPROMTU models the cloud computing environment using two main entities. Firstly, virtual machines represent software environments of the end user that requires computational resource to carry out its tasks. Secondly, physical machines provide these computational resources by hosting various number of virtual machines. In this context, the granularity of resources in the cloud are set by the physical machines, which are the most general and the abstract unit in the data center. Therefore, the computational resources in the data center is the total number of the physical machines available. Let the set of physical machines be denoted by $P = \{p_i \in P | i = 1, 2, ..., I\}$ where p_i represents a unique physical machine available in the cloud. Similarly, let the set of virtual machines be denoted by $V^i = \{v_j^i \in V^i | j = 1, 2, ..., J\}$ where each v_j represents a unique virtual machine hosted on the physical machine p_i. Therefore, the set of all virtual machines in the cloud at a given point of time can be defined as $V = \bigcup_{i=1}^{I} V_i$, given that,

$$V^i \neq V^{i'} \text{and} V^i \cap V^{i'} = \emptyset, \forall i, i' \text{where} i \neq i' \tag{4.1}$$

which denotes that each virtual machine can be hosted by only a specific physical machine at a given time [80].

The first responsibility of the autonomous node agents is to observe the resource usage of the associated physical machine. In order to perform this task, a set of parameters need to assigned to each node agent. The available resource on a physical machine can be viewed under different categories of resource dimensions such as computation, storage, and network. Then, the set of resource metrics can be defined as $G_R = \{g_n(\cdot) \in G_R | n = 1, 2, ..., N$, where each $g_n(\cdot)$ denotes the measured usage on a specific resource dimension n. The amount of resources assigned to a particular virtual machine v_j^i on a resource dimension n can be denoted by $g_n(v_j^i)$. In this context, the node agents can calculate the total resource usage on a particular physical machine p_i by summing the computational resource assigned to each hosted virtual machine represented by $g_n(p_i) = \sum_{j=1}^{J} g_n(v_j^i)$, where $0 \leq g_n(v_j^i) \leq 1$. Furthermore, it is assumed that the hypervisors are assigned a static predetermined amount of resources to carry out the decision making [80]. In this manner, the total amount of resources used on a physical machine is separate from the hypervisor's own resource usage.

The second responsibility of the autonomous node agents is to detect the state of the associated physical machines using the computed values of $g_n \forall N$. This task is carried out by checking whether the total resource usage on a physical machine stays within a set of predetermined desirable limits. These limits are referred to as lower and higher resource thresholds represented by L_n and H_n respectively, where $0 < L_n \leq H_n \leq 1$. Lower resource usage thresholds are used in order to determine under-utilization, whereas, higher resource usage thresholds are used to over-utilization. In particular, L_n is associated with wasting the available resources and H_n is associated with possible SLA violations.

The last task of the node agents is to eliminate under-utilization and over-utilization on the physical machine that it is coupled with. This requires the node agents to be self-aware by continuously updating their perception of the physical machines using the lower and higher thresholds. In IMPROMPTU, a physical machine can be found in four distinct states represented as follows [80],

$$S_i = \begin{cases} 1 & \text{if } \exists n, g_n(p_i) > H_n, \\ 2 & \text{if } (\exists n, g_n(p_i) \neq 0) \ \wedge (\forall n, g_n(p_i < L_n), \\ 3 & \text{if } \forall n, g_n(p_i) = 0, \\ 4 & \text{if } (otherwise). \end{cases} \tag{4.2}$$

In this context, the four distinct states can be categorized into desired and undesired ones defined by $S_i = 4$, $S_i = 4$ and $S_i = 1$, $S_i = 2$ respectively. The desirable state, $S_i = 3$, happens when no virtual machines are hosted on a physical machine. In other words, the physical machine is suspended and can be shut off in order to save resources such as energy in the data center. The second desirable state, $S_i = 4$, is encountered when usages on each resource dimension is between the corresponding lower and higher thresholds.

The undesirable state, $S_i = 1$, represents that at least one of the resource usages in the set of resource dimensions exceeds the higher thresholds. In particular, this behaviour is associated with a possible SLA violation. On the other hand, the second undesirable state, $S_i = 2$, is encountered when the physical machine is under-utilized.

In order to eliminate undesirable conditions, the node agents determine the next course of action based on a reaction to the captured state of the physical machine. In IMPROMPTU, the node agents can react in the three following ways [80]

$$R_i = \begin{cases} \text{Tune} & \text{if } S_i = 1, \\ \text{Evacuate} & \text{if } S_i = 2, \\ \text{No Reaction} & \text{if } S_i = 3 \wedge S_i = 4. \\ & \end{cases} \tag{4.3}$$

In particular, *No Reaction* refers the physical machine being in a desirable state, whereas, *Tune* and *Evacuate* reactions call for distinctive procedures of their own which are explained in detailed in the next section.

4.2 Multiple Criteria Decision Analysis Model

Autonomous node agents carry out the *Tune* procedure in order to bring the total usage on each resource dimension in desirable levels, that is below the higher threshold values. As over-utilization of a physical machine is associated with possible SLA violations, Tune can be viewed as an attempt to eliminate SLA violations. During this procedure, the node agents need to move one or more hosted virtual machines to another physical machine in the cloud. From a local point of view, each migration must have a solid effect on the decrease in the total resource usage, whereas, the physical machine chosen for migration needs to have enough computational resource to host this virtual machine without having to exceed the higher thresholds values. It is important note that the reaction is carried out in a iterative manner in order to avoid searching combinations in a large solution space which can rise feasibility issues and result in stale solutions. In each iteration of the Tune reaction two following sequential decisions take place: (1) deciding on a candidate virtual machine for migration, and (2) choosing a physical machine to host the migrated virtual machine without encountering immediate problems [81].

In the case of under-utilization in the physical machine, the *Evacuate* procedure takes place. In this procedure, the node agent attempts to migrate all virtual machines hosted on that particular physical machine in order to bring the physical machine to a desirable hibernation state. As the Tune procedure, Evacuate reaction is also carried out in an iterative manner for the reasons stated above. In contrast to Tune procedure, only one decision needs to be made since choosing a candidate virtual machine is not necessary. In the event of undesirable conditions, these two reactions decide on the next course of action in order to eliminate wasting resources and cause SLA violations.

The process of selecting the most suitable virtual machine to migrate needs an assessment of the problematic alternatives. Mathematically, let the set of alternatives on a such physical machine p_i be denoted by $A_1 = V^i$. In this set, the evaluation favours the candidates with the higher resource usage on a given resource dimension since the removal of such virtual machines are more likely to bring the system back to a desirable state. The process of choosing a target physical machines to be the new host for the migrated virtual machine needs a similar, yet more complex analysis. Let A_2 denote the set of all physical machines that can host the virtual machine to be migrated with no immediate threshold violations. This set is generated by a node agent in the following mathematical manner [80],

$$A_2 = \{p_{i'} \in A_2 | (g_n(p_{i'}) + g_n(v_j^i) < H_n, \forall n) \wedge (S_{i'} = 4), \forall i^{'}\} \tag{4.4}$$

which defines the set of physical machines that operate between the thresholds of L_n and H_n on all resource dimensions, and also can host the chosen virtual machine without immediate higher threshold violations. In the case where $A_2 \neq \emptyset$, each alternative in the set is evaluated by the node agent.

However, in the case where $A_2 = \emptyset$, the alternative set needs be redefined in order to account for further possible candidates. Mathematically, this is done in the following way [80],

$$A_2 = \{p_{i'} \in A_2 | (g_n(p_{i'}) + g_n(v_j^i) < H_n, \forall n) \wedge (S_{i'} = 2), \forall i^{'}\} \tag{4.5}$$

It is important to note that in this second attempt, the set of candidate physical machines are chosen from the undesirable state 2, in which, the physical machines are under-utilized in at least one resource dimensions. This benefits the stability of the system in two ways. Firstly, the migrated virtual machine is likely to bring the under-utilized physical machine in a desirable state where the resource usage is within the threshold limits. Which, in turn, improves utilization and avoids wasting computational resources. Secondly, the virtual machine can be migrated without having to use a physical machine in the hibernation phase. However, in the case where the redefined set of $A_2 = \emptyset$, then a physical machine needs to be chosen from the set of available ones in the hibernation phase. In the worst possible case where no idle physical machines are left in the system, the node agent is forced to skip the decision making step, in which, a SLA violation is very likely to surface. The decision making process of this type, favours the physical machines in the set A_2 with the least amount of resource usage, or in other words, the most amount of available resources with the intention of avoiding the SLA violations once the migration is carried out.

IMPROMPTU is designed with the principal of being self-aware, where the relative importance weight for each resource dimension is assigned in a dynamic manner. Let w_n denote the relative weight of the criterion $g_n(\cdot) \in G_R$. In addition the dynamic weights are normalized so that $\sum_{n=1}^{N} w_n = 1$ [82]. It is important to note that, these weights can also be set statically by the system administrators in order to reflect on the purpose of the data center. In the case of dynamic weight allocation, the higher threshold violations result in the increase of the weight of a given criterion. Modulation of these weights are done by defining a separate violation parameter y_n for each resource dimension as [79],

$$y_n = \begin{cases} \frac{g_n(p_i) - H_N}{1 - H_N} & \text{if} \quad g_n(p_i) > H_N, \\ 0 & \quad \text{otherwise.} \end{cases} \tag{4.6}$$

In addition, the violation parameters are normalized with respect to the higher threshold violation on each resource dimension as follows,

$$w_n = \frac{y_n}{\sum_{n=1}^{N} y_n} \tag{4.7}$$

where $\sum_{n=1}^{N} y_n = 1$. In the cases where the threshold violation occurs only on one resource dimension, the relative weight of that particular criterion is assigned to 1 in order to neglect the importance of non-violating resource usages on the decision making process.

In IMPROMPTU model, the node agents also use another set of parameters called influence criteria in the decision making phase of choosing both virtual and physical machines. Let $G_I = \{g_m(\cdot) \in G_I | m = 1, 2, ..., M\}$ denote the set of influence criteria. In the case of choosing virtual machines for migration, the influence criteria reflects on the history of both higher and lower threshold violations on the physical machine. Since the resource usage on every dimension varies, the influence measures for each resource dimension needs to be captured separately. Therefore, the relation between G_R and G_I is bijective. In this context, the influence measure of a given virtual machine v_j^i on the threshold violations on a specific resource dimension n can be formally defined as [80, 82],

$$g_m(v_j^i) = \frac{\sum_{t=0}^{T}(e_n^t \Delta t)}{T} \tag{4.8}$$

where T is the total amount of time that a given virtual machine is hosted in the cloud, and e_n^t denotes the portion of impact that v_j^i had on the change of resource usage on the hosted physical machines. e_n^t between the time $t^{'}$ and time t can be formally defined as follows [80, 82],

$$e_n^t = \begin{cases} \frac{g_n^t(v_j^i) - g_n^{t^{'}}(v_j^i)}{g_n^t(v_j^i) - H_n} & \text{if} \quad S_i = 1 \wedge g_n^t(v_j^i) - g_n^{t^{'}}(v_j^i) > 0 \\ \frac{g_n^{t^{'}}(v_j^i) - g_n^t(v_j^i)}{L_n - g_n^t(v_j^i)} & \text{if} \quad S_i = 2 \wedge g_n^t(v_j^i) - g_n^{t^{'}}(v_j^i) < 0 \\ 0 & \text{if} \quad \text{otherwise} \end{cases} \tag{4.9}$$

On the other hand, the influence measure of a physical machine is defined as the total of the influence measures of the hosted virtual machines at a given time. Such measure is represented as $g_m(p_i) = \sum_{j=1}^{J} g_m(v_j^i)$. In addition, when choosing the virtual machine to be migrated, the evaluation process favours the virtual machines with higher influence measures. On the contrary, during the selection of physical machines, the evaluation process favours the ones with the lower measures. The reason behind this process is to blend problematic virtual machines with more stable ones in order to reduce further possible threshold violations [81].

4.3 Applying Multiple Criteria Decision Analysis Methods

As outlined in the previous section, there are two main decision making processes carried out in IMPROMPTU under undesirable conditions. Firstly, a virtual machine is chosen for migration from an under-utilized or an over-utilized physical machine at hand. Secondly, a target physical machine is chosen from the set of all available physical machines in desirable states. These two types of decision making processes are executed by the node agents sequentially. In both decision types, the node agents encounter conflicting multiple criteria problems, which, in turn, makes the problem a great application domain for Multiple criteria decision analysis methods.

In Chapter 3, MCDA methods are formally explained in detail. In this section, we are going to revisit SMART, AHP, PROMETHEE II, ELECTRE III, PAMMSEM II, ORESTE, REGIME and QUALIFLEX, in the context of decision making process carried out by the node agents in the IMPROMPTU model. Furthermore, we are going to provide details about the implementation of these methods.

4.3.1 Simple Multi-Attribute Rating Technique

Simple Multi-Attribute Rating Technique, SMART, is a computationally simple and a light weight single synthesizing MCDA method. In this context, computing virtual machine to physical machine mappings are extremely efficient, which in turn, contributes to the system being reactive by producing never stale solutions.

The application of SMART on the IMPROMPTU model can be defined formally as follows. In the first decision making process, the goal of the node agents is to choose a virtual machine for migration. The evaluation process favours the virtual machines with both higher resource usages and influence criteria. The performance index of the alternative virtual machine v_j^i is denoted by:

$$u(v_j^i) = \sum_{n=1}^{|G_R|} g_n(v_j^i) w_n + \sum_{m=1}^{|G_I|} g_m(v_j^i) w_m \tag{4.10}$$

The node agents compute the function outlined above for each alternative in the set of possible virtual machines and sort them in a descending order. The top ranking virtual machine in the sorted list is picked for the migration.

A similar procedure is performed in the second decision type. The goal is to pick the physical machine with the most amount of available resources and the lowest influence criteria. The performance index of a given physical machine p_i is denoted as follows:

$$u(p_i) = \sum_{n=1}^{|G_R|} g_n(p_i) w_n + \sum_{m=1}^{|G_I|} g_m(p_i) w_m \tag{4.11}$$

When defining the performance measure for a physical machine, the sum of the resource usages and influence criteria of each hosted virtual machine on all dimensions are added together. In contrast to the decision process for virtual machine selection, the ranking is done in an ascending order. Therefore, the physical machine with the most available resources and the least amount of influence measure is deemed the winner.

It is important to note that, influence criteria and resource usages are weighted equally. However, it is possible to steer the judgement towards either set of measure by assigning weights and normalizing the end value so that $u(p_i) = 1$.

4.3.2 Analytic Hierarchy Process

Analytic Hierarchy Process process is a single-synthesizing criteria method that derives performance indexes of alternatives through pair-wise comparisons according to a hierarchical organization. This organization can be described by a tree structure, where the goal is placed at the root, and followed by criteria, sub-criteria and alternatives. At each level, pairwise comparisons are carried out to express judgements among criteria or alternatives with respect to the fundamental scale of AHP. These judgements are placed in the impact matrix, which is used to derive the relative performance values or weights through calculating its eigenvector. In the final step, these values are aggregated with respect to the weight of upper hierarchies in order to produce a final ranking.

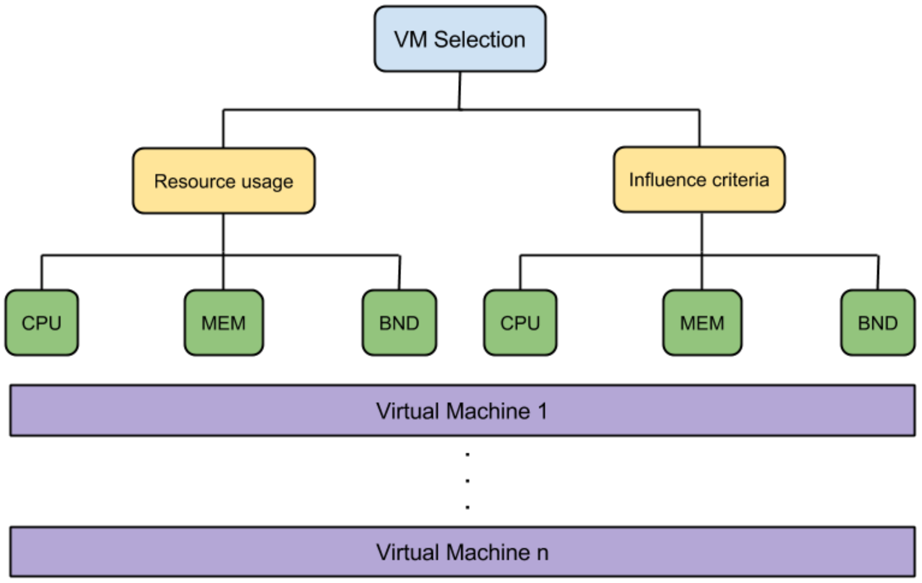

Figure 4.1: Virtual machine selection hierarchy

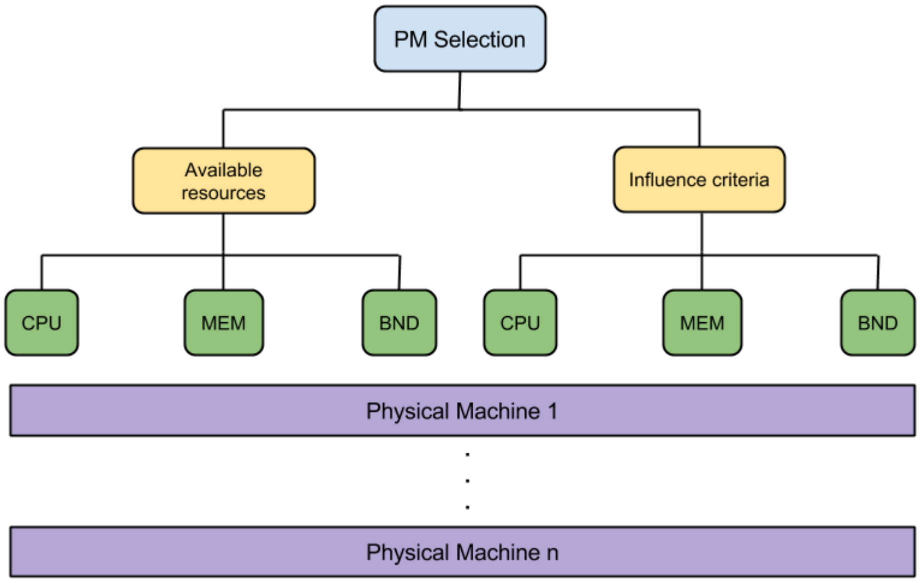

Figure 4.2: Physical machine selection hierarchy

In the decision making type of selecting a virtual machine, the tree structure consists of the goal of selecting a virtual machine for migration placed at the root, followed by the two sets of measure, resource usage and influence criteria. Furthermore, at the lower level, these criteria are followed by CPU, memory and bandwidth. A similar structure is used for the physical machine selection. The only difference is that instead of resource usage for the virtual machine, the process considers the available resources on a given physical machine. In both structures, the relative weights are derived by the means of first comparing the numerical values through a linear transformation in order to fit into the fundamental scale of 1 to 9. Performance values at each level is computed through pairwise comparisons, and entered into an impact matrix.

The relative weights for the alternatives are extracted from the impact matrix using the logarithmic least squares method. The method operates by calculating the geometric mean of each row in the impact matrix, then taking the sum of geometric means, and normalizing each geometric mean by the computed sum. Once the relative weights for each criteria, sub-criteria and alternatives are computed, a performance index for the alternative is computed by multiplying the relative weights of the parents in the hierarchical structure. In the final step, the performance indexes are sorted in order to choose a virtual machine or a physical machine.

The hierarchical structure used by the node agents are shown if in Figures 4.1 and 4.2.

4.3.3 PROMETHEE II

PROMETHEE II is an outranking MCDA method that can rank alternatives based on the evaluation differences when carrying out pairwise comparisons. For each evaluation preference, PROMETHEE II uses a generalized criterion for the pair a, b such that $\{g_k(\cdot), P_k(a, b)\}$ where the $P_k(a, b)$ is the preference function associated with $g_k(\cdot)$. As outlined formally in Chapter 3, PROMETHEE II uses a set of six predefined preference functions. In IMPROMPTU, three of these are selected to reflect on the nature of the resource usage. The load dependent nature of resources such as CPU and bandwidth are represented by a Gaussian function, whereas a V-Shaped function is chosen for the memory. Furthermore, the influence criteria is modelled through Usual preference function, in which, the strength of preference is represented by a Boolean value.

In both decision types, the aggregate preference indices are given by [81],

$$\pi(a,b) = \sum_{j=1}^{G} P_k(a,b) w_k \tag{4.12}$$

where $G = G_R \cup G_I$ and w_k is the dynamic weight associated with the criterion k.

Once the aggregation phase is completed, the exploitation process needs to carried out by the node agents. In this step, a positive outranking and a negative outranking flow is computed, where the positive outranking flow reflects on to what degree an alternative outperforms the rest and the negative outranking flow reflects on the degree of weakness against each alternative. These two measures are defined as follows [81],

$$\phi^+ = \frac{1}{n-1} \sum_{x \in A} \pi(a,x) \tag{4.13}$$

$$\phi^- = \frac{1}{n-1} \sum_{x \in A} \pi(x,a) \tag{4.14}$$

In the final step, the node agents combine these two measures in order to rank the alternatives denoted by the net outranking flow as:

$$\phi(a) = \phi^+(a) - \phi^-(a) \tag{4.15}$$

4.3.4 ELECTRE III

ELECTRE III is an outranking based method where the evaluation of alternatives are based on pairwise comparisons by using concordance, discordance and veto measures. In both decision types carried out by the node agents, the decision process starts by calculating a concordance index for each alternative as follows [81],

$$c_g(a,b) = \begin{cases} 1 & \text{if} \quad g_g(a) + q_g \geq g_g(b) \\ 0 & \text{if} \quad g_g(a) + p_g \leq g_g(b) \\ \frac{p_g + g_g(a) - g_g(b)}{p_g - q_g} & \quad \text{otherwise} \end{cases} \tag{4.16}$$

where q_g and p_g are the thresholds of indifference and preference defined for criterion g and $g \in G$ where $G = G_R \cup G_I$.

Let k_g be the importance coefficient or weight for criterion g, which is the dynamic weights assigned by the IMPROMPTU model. The valued outranking relation is defined as follows [81],

$$C(a,b) = \frac{1}{k}\sum_{g=1}^{r} k_g c_g(a,b) \tag{4.17}$$

where $k = \sum_{g=1}^{r} k_g$

In order to calculate the credibility index, a final measure of discordance index needs to be calculated as follows [81],

$$d_g(a,b) = \begin{cases} 0 & \text{if } \; g_g(a) + p_g \geq g_g(b) \\ 1 & \text{if } \; g_g(a) + v_g \leq g_g(b) \\ \frac{g_g(b) - g_g(a) - p_g}{v_g - p_g} & \text{otherwise} \end{cases} \tag{4.18}$$

In the last step of the aggregation phase, concordance and discordance indexes are combined in order to come up with the credibility index as follows [81],

$$S(a,b) = \begin{cases} C(a,b) & \text{if } \; d_g(a,b) \leq C(a,b) \;\; \forall g \\ C(a,b) \prod\limits_{g \in J(a,b)} \frac{1 - d_g(a,b)}{1 - C(a,b)} & \text{otherwise} \end{cases} \tag{4.19}$$

where $J(a,b)$ is the set of criteria such that $d_g(a,b) > C(a,b)$.

After the aggregation phase, the node agents proceed with the exploitation procedure, in which, two pre-orders are derived from the credibility matrix as Z_1 and Z_2. These two pre-orders reflect on the descending and ascending distillation procedures respectively. Finally, the intersection of two pre-order are taken by the node agents, in order to compute a final ranking of the alternatives. It is important note that the distillation procedure is a computationally complex procedure which presents a downside in terms of configuration times.

4.3.5 PAMMSEM II

PAMMSEM II is a hybrid outranking method that combines the aggregation phase of ELECTRE III and the exploitation phase of PROMETHEE II. Both of these procedures are explained in detail in the previous two subsections.

However, since PAMMSEM II combines the key and the most beneficial phases of two well known MCDA methods, this brings very interesting features together. In this context, PAMMSEM II does not suffer from the computationally complex distillation procedure introduced by ELECTRE III method as it uses a net outranking flow in this step of the decision making procedure. In addition, PAMMSEM II involves the detailed aggregation phase introduced by ELECTRE III. This, in turn, lifts the negative effects of a computationally complex decision making procedure in terms of configuration times.

4.3.6 ORESTE

ORESTE method assigns ranks to criteria as well as the alternatives themselves in the first step of the decision making process. Each criterion $g \in G$ where $G = G_R \cup G_I$ and each alternative $a \in A$ is under criterion g is given ranks denoted by r_g and $r_g(a)$ respectively. In the context of IMPROMPTU, r_g is associated with the dynamic weights and the influence criteria, whereas, $r_g(a)$ is associated with the resource usages and the influence criteria under dimension n when $g \in G_R$ and $g \in G_I$ respectively. The procedure of assigning ranks results in $|G| + 1$ pre-orders.

A position table is formed by converting these pre-orders into numerical values by using Besson's mean rank. In the next step, these values are projected into a distance matrix by using a linear oblique projection. The distance of a projection from the origin is to the alternative is defined as $d(0, a_g) = \alpha r_g(a) + (1 - \alpha) r_g$ where α denoted the substitution rate $(0 < \alpha < 1)$. The substitution rate is used to weigh the importance of the ranks of criteria and alternatives. The distance matrix is converted into a global rank matrix using Besson's mean rank in order to construct a global weak order of the alternatives. The global rank of an alternative $a \in A$ under the criterion $g \in G$ is denoted by $R(a) = \sum_g R(a_g)$.

In the aggregation phase, preference indices are introduced in order to consider indifference and incomparability. After the aggregation phase, the node agents are left with preference relations between all alternatives. In order to extract a total pre-order from these preference relations, a distillation procedure is carried out. During the distillation, a set consisting of non-dominated alternatives under all criteria is constructed. Finally, an alternative is chosen at random from this set in order to pick virtual and physical machines.

4.3.7 REGIME

REGIME is an outranking method where the outcome of the pairwise comparisons under a given criteria depend on the mathematical sign obtained as follows:

$$C_n(a,b) = \begin{cases} +1 & \text{if } g_n(a) < g_n(b) \\ 0 & \text{if } g_n(a) = g_n(b) \\ -1 & \text{if } g_n(a) > g_n(b) \end{cases} \tag{4.20}$$

where $C_n(a,b)$ represents the concordance measure of alternative a compared to b under the criterion n.

In both types of decision making processes, the node agents compute the comprehensive performance measure when comparing alternatives a, b as follows:

$$C(a,b) = \sum_{n=1}^{|G_R|} C_n(a,b)w_n + \sum_{m=1}^{|G_I|} C_m(a,b)w_m \tag{4.21}$$

Consequently, the decision between two alternatives are done in following manner: (1) aPb is assessed when $C(a,b) - C(b,a) > 0$, (2) bPa is assessed when $C(b,a) - C(a,b) > 0$, and (3) aRb is assessed when $C(a,b) - C(b,a) = 0$.

After the aggregation phase, the node agents are left with preference relations between all alternatives. In order to extract a total pre-order from these preference relations, a distillation procedure is carried out. During the distillation, a set consisting of non-dominated alternatives under all criteria is constructed. Finally, an alternative is chosen at random from this set in order to pick virtual and physical machines.

4.3.8 QUALIFLEX

In the both decision types carried out by the node agents, QUALIFLEX method evaluates all possible permutations of rankings in the alternative set by default. The procedure starts by constructing an impact matrix, in which, each vector represents a permutation in the alternative set also referred to as pre-order. From each pre-order, QUALIFLEX defines *concordance* and *discordance* indexes for each tuple of alternatives. The ranking of the tuple a, b from the k^{th} permutation under criterion n is defined as follows:

$$I_{nk}(a,b) = \begin{cases} 1 & \text{if} \quad g_n(a) > g_n(b) \\ 0 & \text{if} \quad g_n(a) = g_n(b) \\ -1 & \text{if} \quad g_n(a) < g_n(b) \end{cases} \tag{4.22}$$

When all tuples in a given permutation of alternatives is ranked, these values are combined in a comprehensive concordance/discordance index. The node agents use the dynamic weights and the influence criteria at this step as follows:

$$I_k(a,b) = \sum_{n=1}^{|G_R|} I_{nk} w_n + \sum_{m=1}^{|G_I|} I_{mk} w_m \tag{4.23}$$

In the exploitation step, comprehensive concordance/discordance indices for each permutation is sorted. The virtual machine with the highest value is deemed the winner, whereas the physical machine with the small values is chosen for the reasons explained in previous sections. It is important to note that, QUALIFLEX brings a heavy computational burden on the decision making process as in order to rank n alternatives, $n!$ permutations are needed to be ranked. In QUALIFLEX implementation used by IMPROMPTU model, the alternative list is limited to a size of 5 by applying a lexicographical ordering of virtual machine resource usages on each dimension. On the other hand, the first 5 available physical machines in a steady state is chosen in order to reduce and limit the solution space to $5! = 120$ permutations.

Chapter 5

Simulation Case Study: IMPROMPTU Under The Management Of MCDA Methods

In this section, we are going evaluate the performance of eight different resource allocation managers within the context of IMPROMPTU that employ the MCDA methods given in Chapter 3.

The ideal scenario for testing the performance of the resource allocation managers is to use an actual physical system that enables virtualization. In this context, we have investigated several existing cloud computing technologies. However, to the best of our knowledge, the existing commercial clouds and public research data centers do not provide enough flexibility to allow a custom resource allocation management. In most of the cases, when a virtual machine is requested from a service provider, the end user has no information about the physical whereabouts of the hardware let alone allowing access to underlying physical layer. Resource allocation research requires explicit and unrestricted control over how the physical resources are distributed among application environments. Therefore, the ideal scenario cannot be utilized. This, in turn, leaves the option of using a closed local distributed setting where the researchers have full control over the layout and the design of the data center. As mentioned in Chapter 2, some research was done in a small scale physical settings which may arguably be acceptable for assessing the performance of a single resource allocation manager. However, in this thesis, we are equally interested in the scalability as well as the performance of the solution. In addition, the difference of performance when multiple managers are cross compared could diminish in such small settings. Therefore, a research of this nature requires a realistic large scale physical setting. There are two main restrictions of using a such platform. First of all, the financial burden introduced with hardware, power, cooling and co-location is extremely excessive and not feasible. Secondly, working with an actual large scale data center introduces additional complexities that are beyond the scope of this research such as virtual machine placement conflict, the cost of migrations and the mapping of physical resources to virtual machine demands. As a result of all the limitations, a custom built simulation platform was used during the course of this work, in order to lift such limitations, provide explicit control over the data center and prevent loss of information introduced by a small scale setting [80–82].

The rest of this chapter is organized as follows. Section 5.1 explains the details of the custom built simulation platform that is used to carry out the tests as well as the performance metrics used in this study. In Section 5.2, the empirical results are analysed and compared using graphs and tables. In addition, the reason behind the observed performances associated with the mathematical workings of MCDA methods is explained. Finally, Section 5.3 provides a summary.

5.1 Simulation Platform

In this section, we are going to explain the details of the custom built simulation platform in terms of the architecture and the simulation settings as well as the performance metrics. In this context, it is beneficial to divide this section into two subsections.

5.1.1 Simulation Platform Architecture

The simulation platform is custom built for testing the performance of various resource allocation managers in the context of clouds using the Python programming language [80,81]. The design of the platform consists of three main layers: (1) resource layer, (2) simulation engine, (3) consolidation management. Figure 5.1 represents the architecture of the simulation platform.

At the lowest level of the architecture, resource layer provides the abstraction of the cloud in terms of virtual and physical machines. In this context, it is important to note that the simulation platform is designed to accommodate scenarios for clouds

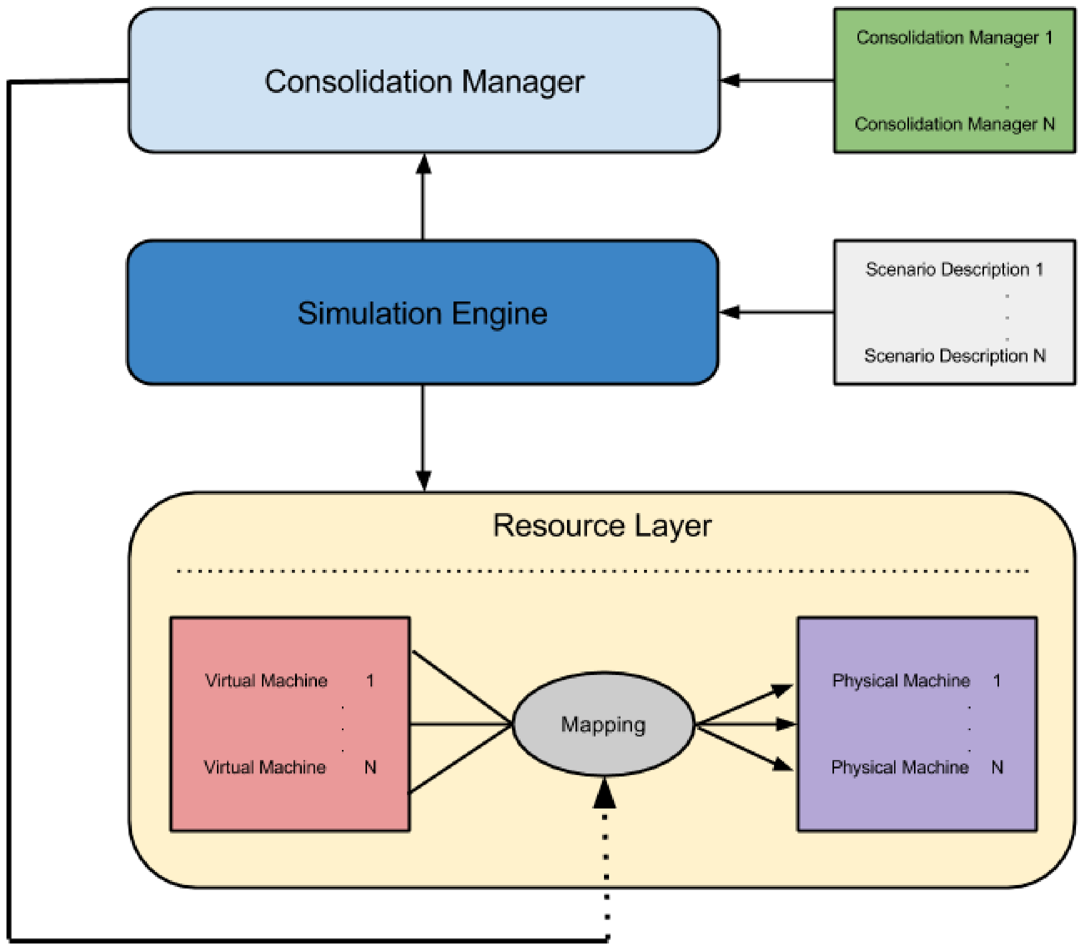

Figure 5.1: Simulation Platform Architecture

at the IaaS level. This, in turn, provides the right amount of granularity needed to compare dynamic resource allocation managers without having to worry about the details of other various components.

In a general sense, the available resources under all dimensions in a cloud are represented by a number of physical machines. Hence, a physical machine is the most basic unit of resources. The main responsibility of the physical machines is to host any number of virtual machines given that the total resource consumption is kept between a set of predetermined limits. As described in Chapter 4, a service level agreement violation is assumed to be encountered in the case of over utilization. On the other hand, virtual machines represent a total application environment that demands resources from the physical machines. The resource consumption is modelled based on statistical distributions such as Gaussian, Exponential, Pareto, etc [81]. In addition, resource consumption on each dimension is independent of each other that change based on the assigned statistical models depending on the type of the virtual machine. The simulation platform is capable of hosting two types of virtual machines: (1) the ones that carry out batch processes, (2) the ones that carry out online processes. The batch type is modelled to simulate a scientific calculator, or a number cruncher that continuously demand certain amount of resources on each dimension. The online type virtual machines follow two different resource consumption schemes. During the on-line period, they behave much like the batch processes demanding a continuous amount of resources under all dimensions. On the other hand, during the off-line periods, they only require a small amount of resource to keep them active, such as memory. It is important to note that the lengths of on-load and off-load periods follow statistical distribution pairs of either {pareto,exponential} or {exponential,pareto} in order to mimic a realistic cloud setting [80]. Figure 5.2 represents the abstract view of the simulated data center in terms of virtual and physical machines.

The consolidation manager lies at the top level of the architecture. The main responsibility of this unit is to compute virtual machine to physical machine mappings based on the reactions of the node agents [80]. Chapter 4 explains the nature and the cause of these reactions in detail. The main novelty of the consolidation manager layer lies in its flexibility of computing virtual machine to physical machine mappings based on the employed decision making method.

The middle layer consists of the simulation engine which is responsible for coordinating the simulation runs [80]. First of all, a user populated scenario script is loaded in the initialization procedure. This file describes various components of the data center such as the statistical distributions used when simulating virtual machine resource demands, the number of virtual machines, the type of consolidation manager, the number of simulation steps, the simulation seed and the resource limits on each physical machine. Based on the loaded parameters, the simulation engine runs the defined scenario in a step wise manner. Each step consists of the following two turns: (1) signalling the resource layer with the necessary resource changes depending on the statistical models, (2) starting the process of consolidation management by providing a current image of the cloud to the employed manager. In this context, a new virtual machine to physical machine mapping is produced at the end of each step. When run under identical scenarios with different consolidation managers, the step wise

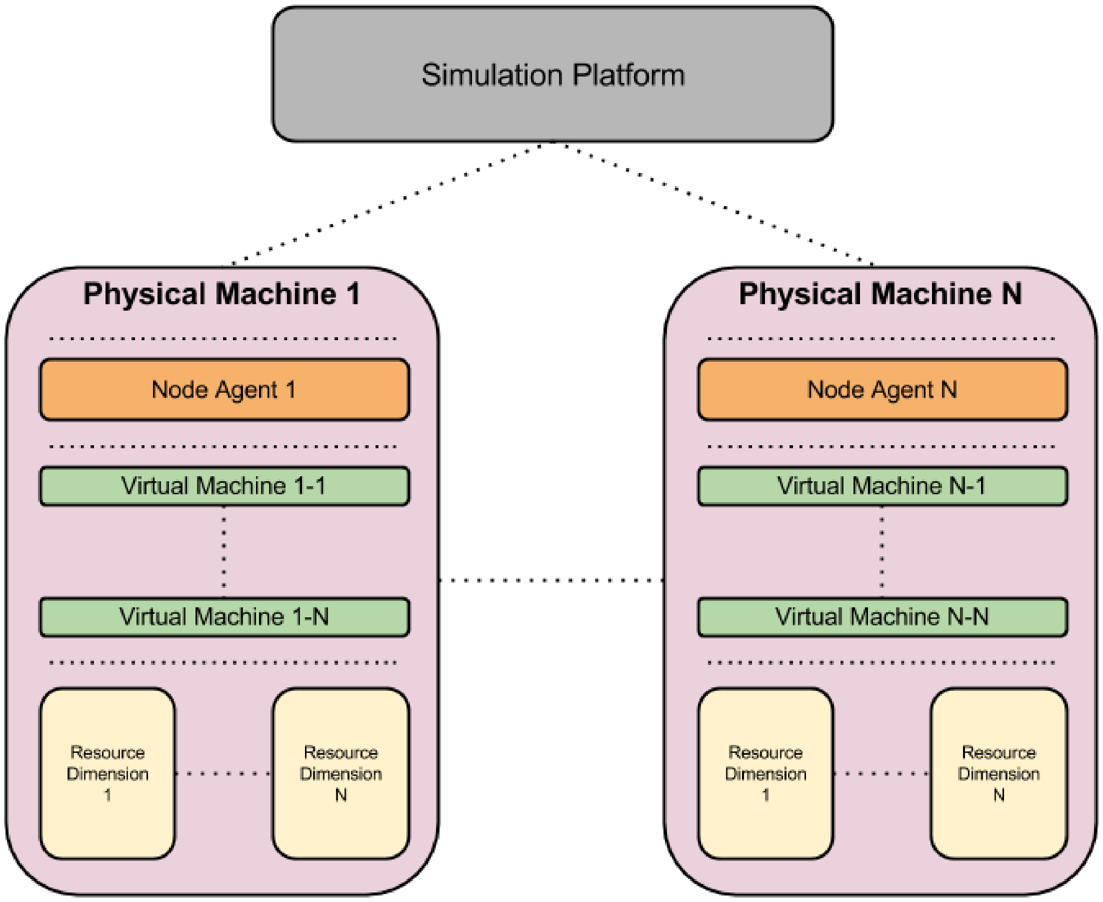

Figure 5.2: Abstract view of the data center in terms of virtual and physical machines

approach enables the analysis of performance at any given point across simulations.

5.1.2 Simulation Settings and Performance Metrics

The simulation platform, described in previous subsection, is given control under the provision of each resource consolidation manager in fully isolated yet identical runs. For each manager the simulation is run for 2000 steps with a data center consisting of 1000 virtual machines with changing resource requirements and a virtually unlimited number of physical machines. In this context, the amount of computational resources in the data center is virtually unlimited. More specifically, each manager can start and utilize new physical machines as needed throughout a simulation run. This feature is beneficial in order to observe each managers resource utilization, performance and the number of employed physical machines. Furthermore, the amount of virtual machines in the data center stay constant throughout each run in order to prevent the sudden substantial changes caused by mass virtual machine arrivals or departures. Each manager is tested by conducting 10 Monte Carlo runs with different random generation seeds for each run. The seeds are set to identical across the same runs of different managers in order to gather the data under exact same conditions. In addition, this allows for a complete comparison at any given step across managers in the data center.

The available resource dimensions in the data center is defined as CPU, memory and bandwidth. As described in Chapter 4, a service level agreement is assumed to occur upon the violation of the higher threshold on each one of these three resource dimensions. In this manner, it is important to set the thresholds based on the nature of the resource dimension. For both CPU and bandwidth, the resource thresholds are set as 60% in order to reflect on the knee utilizations and service channel counts [53] due to their load dependent nature. On the other hand, the resource threshold for memory is set as 90% since memory doesn't suffer from the limitations encountered with CPU and bandwidth. The remaining 10% is present in order to prevent starvation. Each simulation run starts from a point where the resources in the data center are distributed among the virtual machines using the First Fit method in order to ensure that the starting conditions are identical across the same Monte Carlo runs of different managers.

Each manager is analysed with respect to four generally accepted performance metrics of *physical machine usage*, *service level agreement violations*, *virtual machine migrations* and *configuration time*. Physical machine usage measures the extent of efficiency of resource consolidation managers with respect to resource utilization. Therefore, less physical machine usage is deemed better given that the service level agreements are avoided as much as possible. The service level agreement violation performance metric indicates the number of such violations throughout each simulation run. In this context, less service level agreement violations are better as well as suggesting that the manager is capable of delivering the resources to application environments without interruption. It is important to note that the definition of service level agreement violations used in this work is rather strict in comparison to an actual cloud setting. In the simulation platform, it is assumed that the service level agreements of all the hosted virtual machines are violated when the usage on a resource dimension exceeds 80% of what is available. In this context, the service level agreement violations metric is a count of actual violations and the situations that may result in a such violation. The amount of virtual machine migrations encountered after each simulation step after computing new virtual machine to physical machine migration mappings is another important performance metric considered in this work. Although, the overhead of such migrations are not considered in the scope of this thesis, this performance metric indicates the efficiency of the manager as avoiding the unnecessary extra burden introduced on the data center resources. In this manner, less number of migrations are deemed better and more efficient. Finally, configuration times represent the amount of time taken when computing the new virtual machine to physical machine mappings in the data center. Naturally, quicker configuration times are deemed to be better as it increases responsiveness. In addition, this metric is associated with the scalability of the manager. Slow configuration times may result in stale solutions, which, in turn, result in unnecessary migrations and service level agreement violations.

5.2 Simulation Results

It is important to make a distinction between the actual MCDA methods and the IMPROMPTU managers that employ these methods since these managers implement the MCDA methods within the design of the IMPROMPTU model, as we as, according to the nature of the problem. Therefore, in association with the MCDA methods used by their node agents, these eight managers will henceforth be referred to as AHPM (AHP Manager), SMAM (SMART Manager), PROM (PROMETHEE II Manager), ELEM (ELECTRE III Manager), PAMM (PAMSSEM II Manager), ORSM (ORESTE Manager), RGMM (REGIME Manager) and QFXM (QUALIFLEX Manager). In addition, DDRM (Distributed Dynamic Random Manager) is used to provide a worst case benchmark for the MCDA managers in use. Furthermore, DDRM performance measures are excluded in the comparison and only referred when a manager performs worse than the benchmark set by DDRM. Table 5.1 summarizes the performance metrics achieved by all managers.

Figure 5.3 represents the average number of physical machines used under all resource consolidation managers. In literature, the amount of physical machine usage is directly associated with the efficiency of the manager since it can be considered as more cost efficient in regards to hardware, cooling, power consumption, co-location, etc. The results show that QFXM uses the least number of physical machines with the average count and the standard deviation pair of $(109.026, 3.012)$ respectively. QFXM

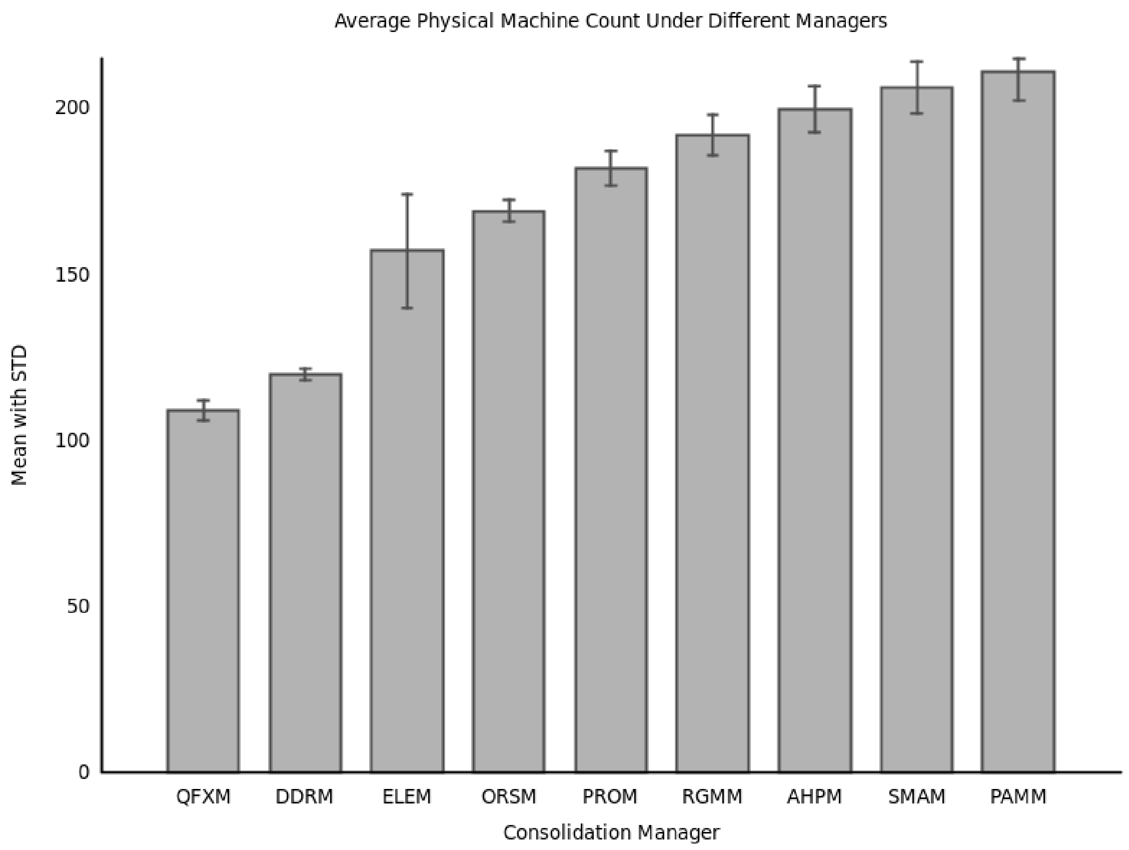

Figure 5.3: The Average Physical Machine Count Under Different Managers

is followed by ELEM with $(157.029, 17.172)$. It is important to note here that QFXM uses about 50 less physical machines in order to host 1000 virtual machines when compared to the second most efficient manager ELEM. The reason behind this is that the filtering technique used in QFXM significantly reduces the granularity of the decisions made since only the candidates with the most resource consumption or availability is considered. This, in turn, results in tight packing of virtual machines. However, the efficiency in using less physical machines in likely to introduce performance issues in respect to service level agreement violations, and virtual machine migrations. ELEM is followed by ORSM, PROM, RGMM, AHPM and SMAM respectively. The amount of physical machines used by these managers increase about by 10 units in each case. The manager with the most loose packing behaviour is PAMM with $(211.015, 8.514)$. In the context of standard deviations, it is beneficial to analyse the managers in three groups as (1) low, (2) intermediate and (3) high deviation. Both QFXM and ORSM belong to the first group of low deviation with standard deviation measure of 3. The

Table 5.1: Managers' Performances over 10 Simulated Runs.

Manager	**PM Usage**		**SLA Violation**		**VM Migration**		**Config. Time**	
	Mean	**SD**	**Mean**	**SD**	**Mean**	**SD**	**Mean**	**SD**
QFXM	109.02	3.01	504.72	24.76	94.17	4.59	0.079	0.002
DDRM	119.83	1.54	330.61	19.34	94.45	2.37	0.006	-
ELEM	157.02	17.17	132.25	17.17	25.88	1.91	0.024	0.003
ORSM	169.05	3.24	71.65	18.57	25.53	3.55	1.288	0.170
PROM	181.91	5.13	62.57	18.38	19.92	2.55	0.631	0.067
RGMM	191.97	6.17	53.09	18.45	19.52	2.63	0.816	0.083
AHPM	199.80	6.80	48.26	17.80	19.27	2.59	0.288	0.030
SMAM	206.33	7.70	44.65	18.06	18.71	2.47	0.003	-
PAMM	211.01	8.51	43.51	17.87	18.61	2.42	0.643	0.077

reason behind this can be considered as the granularity of the decisions made by these managers. That is, the ability of differentiating between small differences in resource usages or availability is significantly lower than the rest of the managers. This, in turn, results in more consistent behaviour with respect to the amount of physical machines in use. PROM, RGMM, AHPM, SMAM and PAMM all belong to the group of intermediate deviation with a standard deviation measure between 5 and 8. In the last group, ELEM has the highest standard deviation value of 17. The reason behind this is that the ELECTRE III method employed by this manager considers preference, indifference and veto thresholds which introduces fine granularity when making decisions. In addition, the distillation process carried out at the end of the method in order to yield a total pre-order may result in inconsistencies.

Figure 5.4 represents the average number of service level agreement violations encountered throughout the simulation run under all managers. Physical machine number under use is a performance metric that is solely associated with the service provider. However, in the client-centric view of the cloud, end user metrics also become important. Most obviously, the top performance metric associated with the end user is service level agreement violations. Naturally, the manager that can limit these violations can be considered as more efficient since the service provider can deliver the computational resource that were agreed upon. In this context, this metric becomes important for both the service provider and the client. The results show that PAMM encounters the least amount of violations with count and standard deviation pair of $(43.517, 17.870)$. PAMM is closely followed by SMAM and AHPM with $(44.652, 18.064)$ and $(48.266, 17.802)$ respectively. More specifically, it is

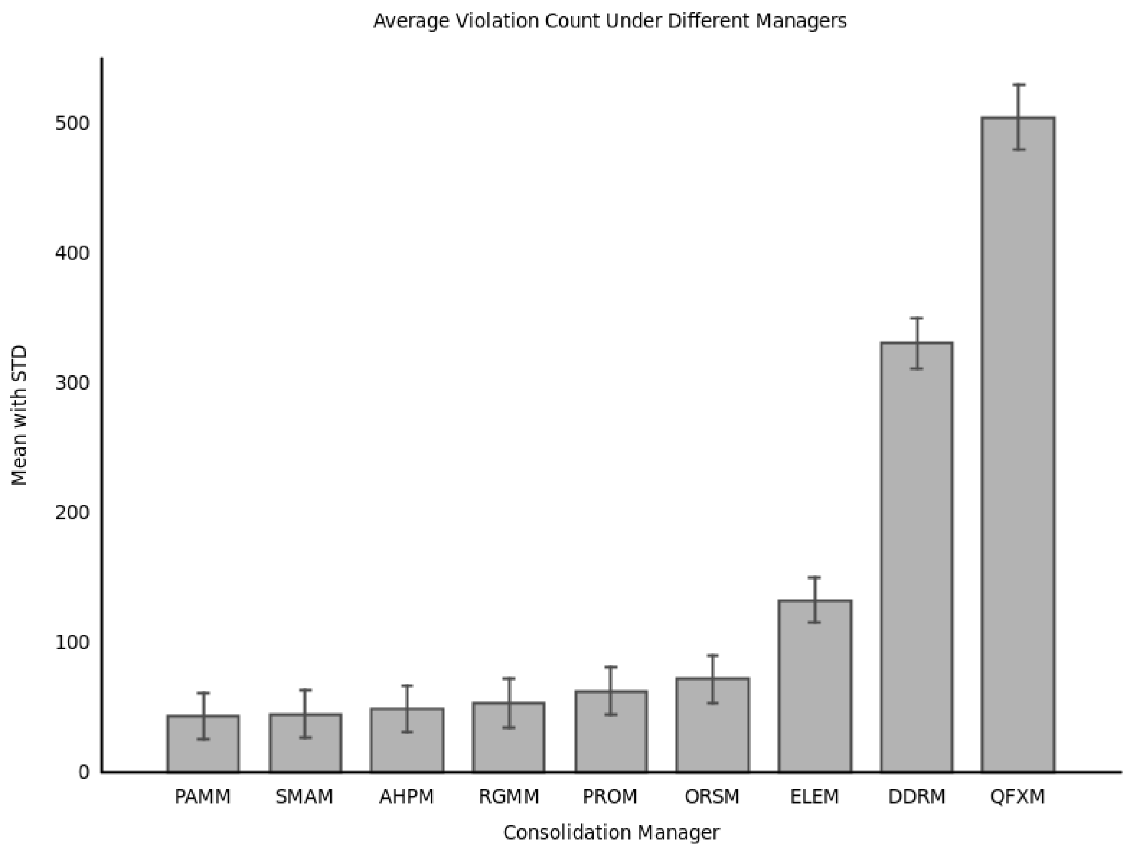

Figure 5.4: The Average SLA Violation Count Under Different Managers

important to note that SMAM uses about 5 less physical machines when compared to PAMM and is able to encounter only a single additional service level agreement violation. In the context of service level agreement violations, it is beneficial analyse the managers under three groups of (1) low, (2) intermediate and (3) high number of encountered violations. The first three managers analysed above belongs to the first group. The intermediate group consists of RGMM, PROM and ORSM with values of $(53.097, 18.453)$, $(62.574, 18.385)$ and $(71.653, 18.575)$ respectively. The last group consists of ELEM and QFXM with the values of $(132.25, 17.172)$ and $(504.726, 24.763)$ respectively. The reason behind the high number of service level agreement violations is the tight packing of virtual machines in the context of resource allocation. Carefully readers will notice that these two managers use the least number of physical machines visible in Figure 5.3. Furthermore, the results also show that the amount of physical machines used by the managers have a direct effect on the amount of service level agreement violations. The order of managers with respect to both performance

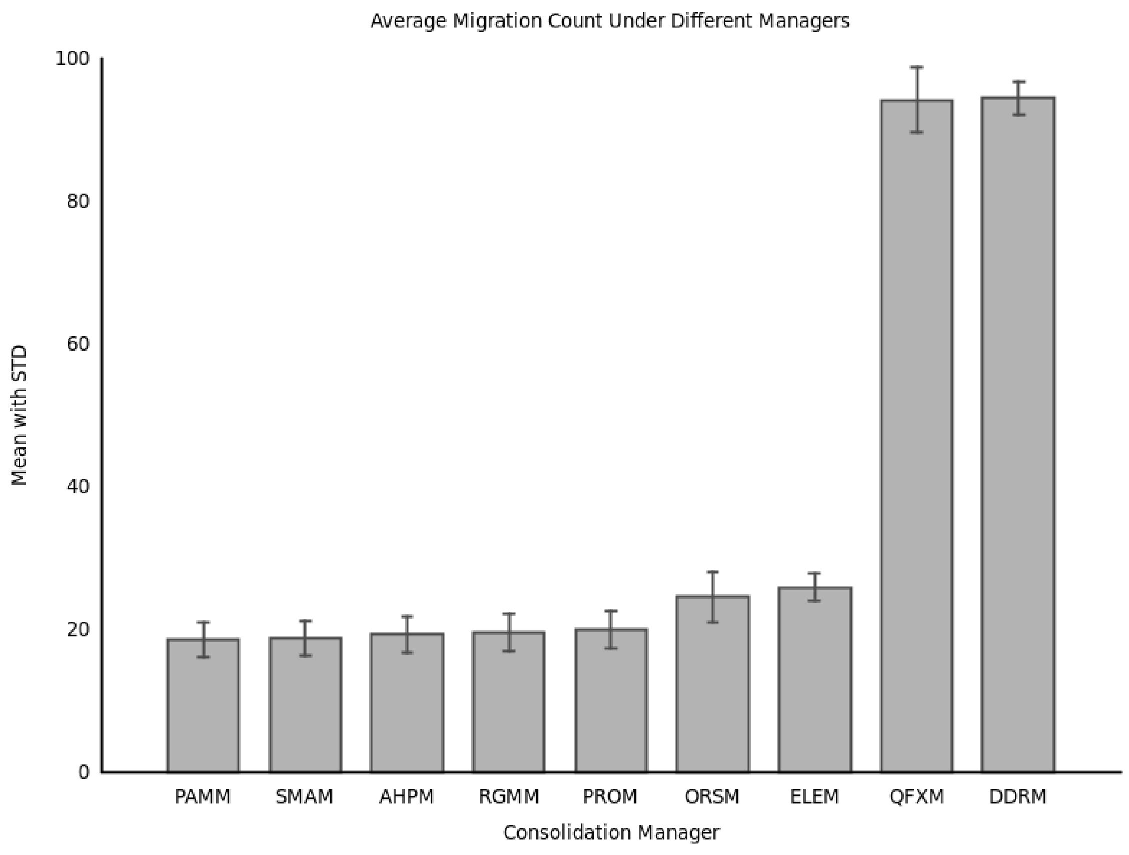

Figure 5.5: The Average Migration Count Under Different Managers

metrics is the complete opposite of each other. Furthermore, QFXM causes significantly more violations than DDRM which denotes the exceptionally bad performance achieved by QFXM. Naturally, if a manager performs worse than the random method under the measure of service level agreement violations, it is safe to conclude that QFXM is not an efficient manager.

Figure 5.5 represents the average number of virtual machine migrations throughout the the simulation run under all managers. Similarly to the relation between service level agreement violations and physical machines under use, the order of managers with respect to virtual machine migrations is the complete opposite of physical machines under use. However, the grouping of the managers differ. In the group of low migrations, the manager with the least amount of migrations is PAMM, followed by SMAM, AHPM, RGMM and PROM with the values of $(18.614, 2.425)$, $(18.712, 2.470)$, $(19.271, 2.597)$, $(19.525, 2.634)$ and $(19.926, 2.558)$ respectively. The second group of intermediate values consist of ORSM and ELEM with the values of

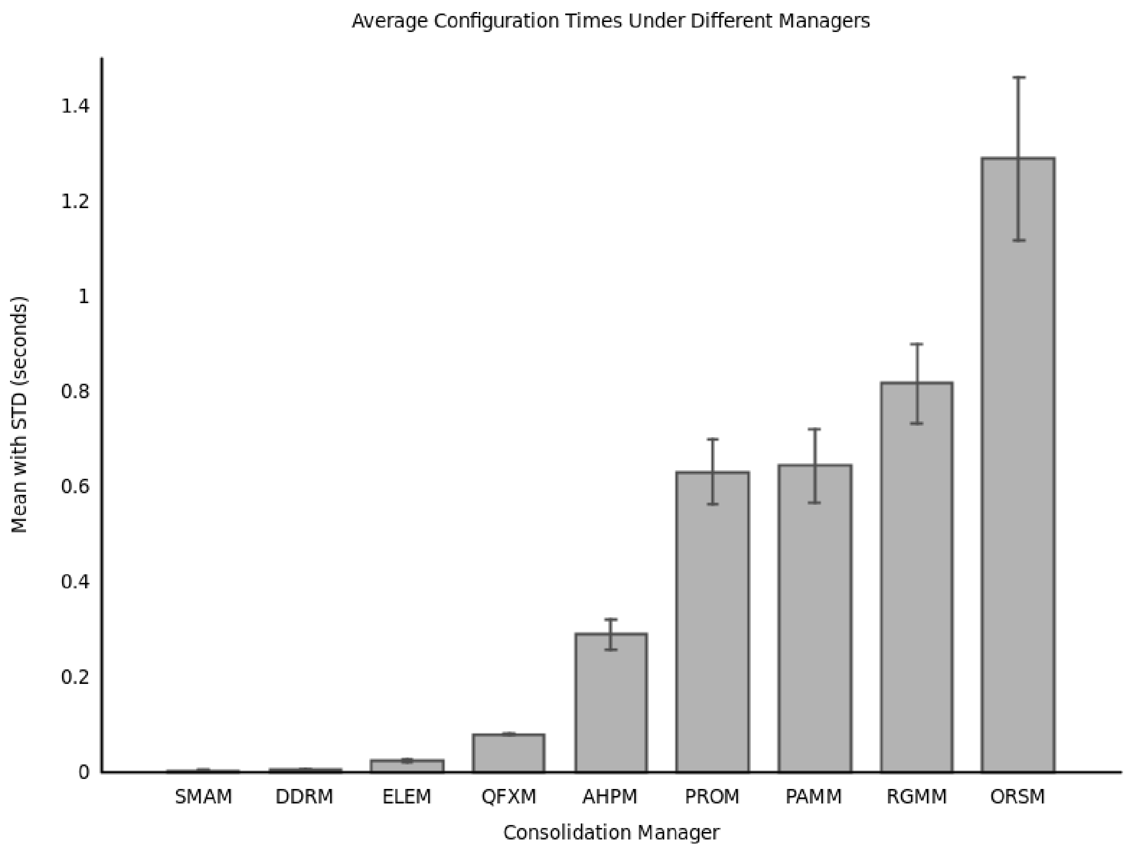

Figure 5.6: The Average Configuration Times in Seconds Under Different Managers

$(24.532, 3.551)$ and $(25.884, 1.911)$ respectively. The results show that the manager with the most number number of virtual machine migrations is QFXM with the values of $(94.175, 4.599)$. In this context, the order of managers with respect to virtual machine migrations and physical machine usage is the complete opposite of each other. Finally, the standard deviation value is consistent among all eight managers.

Figure 5.6 represents the average time taken to compute new virtual machine to physical machine mappings throughout the simulation run under all managers. Specifically, this performance metric is associated with the scalability of the resource consolidation manager. SMAM has the fastest configuration times with 0.004 seconds on average. This value shows that the decision making process carried out by SMAM is almost instant due to the simple utility function used in the decision making process. In this context, SMAM can be considered highly scalable with respect to the ability of dynamically allocating resources quickly in very large data center settings. SMAM is followed by ELEM with the average of 0.025 seconds. QFXM averages 0.079 seconds as a result of the filtration process of alternatives. QFXM employs QUALIFLEX method which considers all $n!$ possible permutations of given n alternatives. The filtration process limits the number of alternatives to 5. Therefore, the solution space is limited to $5! = 120$ permutations which can be computed in a reasonable amount of time. Next in the order is AHPM with the average time of 0.288 seconds. The most time consuming computation in the decision making process employed by AHPM is the construction of the hierarchical structure and producing a ranking of the alternatives at each level of this hierarchy for each step taken throughout the simulation run. In this context, AHPM deals with more pairwise comparisons than QFXM. PROM and PAMM achieve very similar configuration times with 0.631 and 0.643 seconds on average. The main reason behind this is that both decision making methods use the same exploitation procedure. It is important to note here that the main source of lengthy configuration times occur because of the various exploitation procedures conducted by the outranking based managers. In this context, PROM employs the light weight exploitation procedure of PROMETHEE II where positive and negative outranking flows are combined to yield a net outranking flow. RGMM achieves the second longest average configuration times with 0.817 seconds. RGMM employs a simple pairwise comparison process in the aggregation procedure. However, REGIME method leaves the node agents with a set of relations between alternatives. In other words, the node agents has to go through a separate process in order to come up with a total pre-order. Firstly, the node agents considers each alternative under the condition of being non-dominated by the rest of the alternatives and forms an associated set. Secondly, the node agents picks a random alternative from the set of non-dominated alternatives. It is important to note that this exploitation procedure is computationally complex and requires more time when compared to ones employed by PROM and PAMM. The manager with the longest average configuration time is ORSM with 1.289 seconds. ORSM employs the same exploitation procedure as RGMM which contributes to making the computation longer. On the other hand, the aggregation procedure of ORSM is more complex than RGMM due to the various steps taken in ORESTE method such as evaluating pre-orders of both alternatives and criteria, ranking them using a projection matrix and finally converting these projections into a global rank matrix.

Figure 5.7 represents the average CPU utilization achieved in the data center under all managers. The ascending order of managers with respect to CPU utilization in the data center can be given as: PAMM, AHPM, SMAM, RGMM, PROM, ELEM,

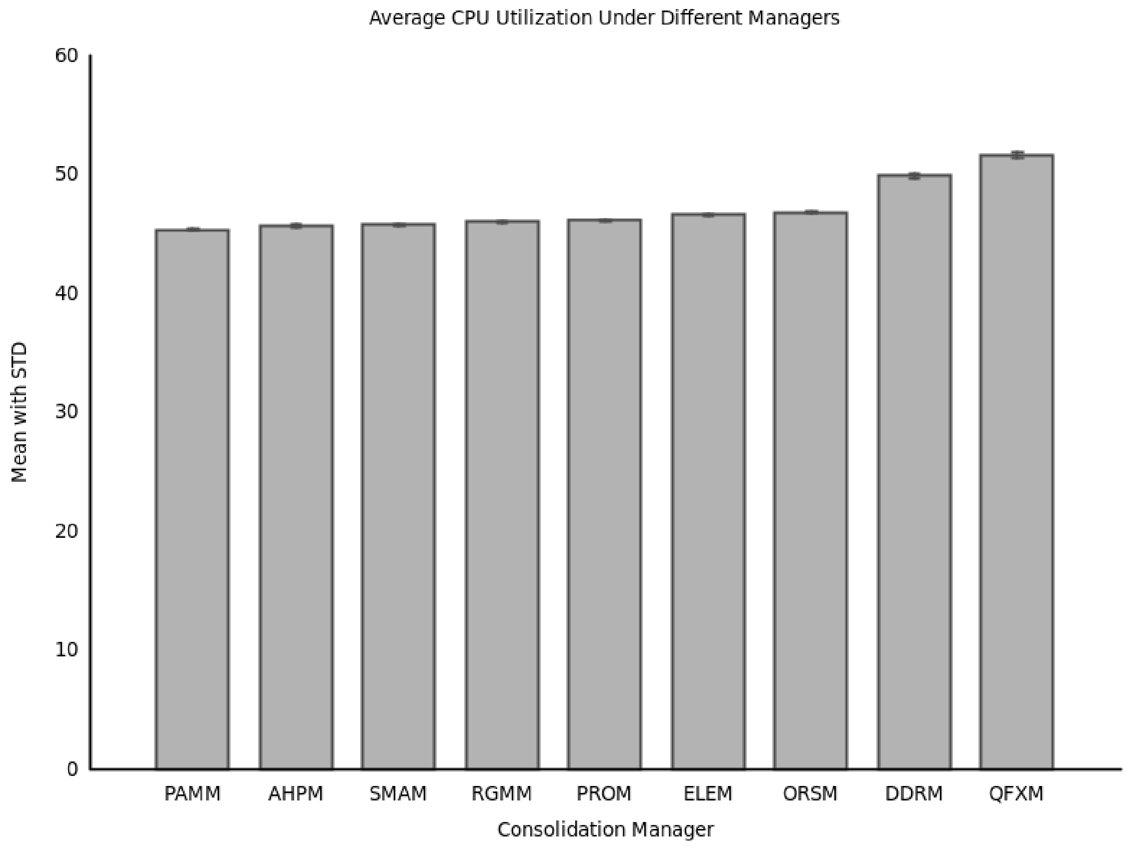

Figure 5.7: The Average CPU Utilization Under Different Managers

and ORSM. It is important to remind here that the higher threshold for the CPU usage was set at 60%. The first seven managers utilize the CPU between 45.374% and 46.804%. The difference in the values is arguably negligible. QFXM stand out from this group with 51.591% CPU utilization. Figure 5.8 represents the average memory utilization achieved in the data center under all managers. The ascending order of managers with respect to memory utilization in the data center is the same as the order given above. In this resource dimension, the higher threshold was set at 90%. Therefore, the expected utilization on this resource dimension is higher than CPU. The first seven managers utilize the memory between 56.923% and 58.119% respectively. As in the case of CPU utilization, QFXM separates from this group by achieving 62.283% utilization. Finally, Figure 5.9 represents the average bandwidth utilization achieved in the data center under all managers. The ascending order managers with respect to this resource dimension is the same as the first two resource dimensions. Furthermore, the higher threshold was set at 60%. The first seven

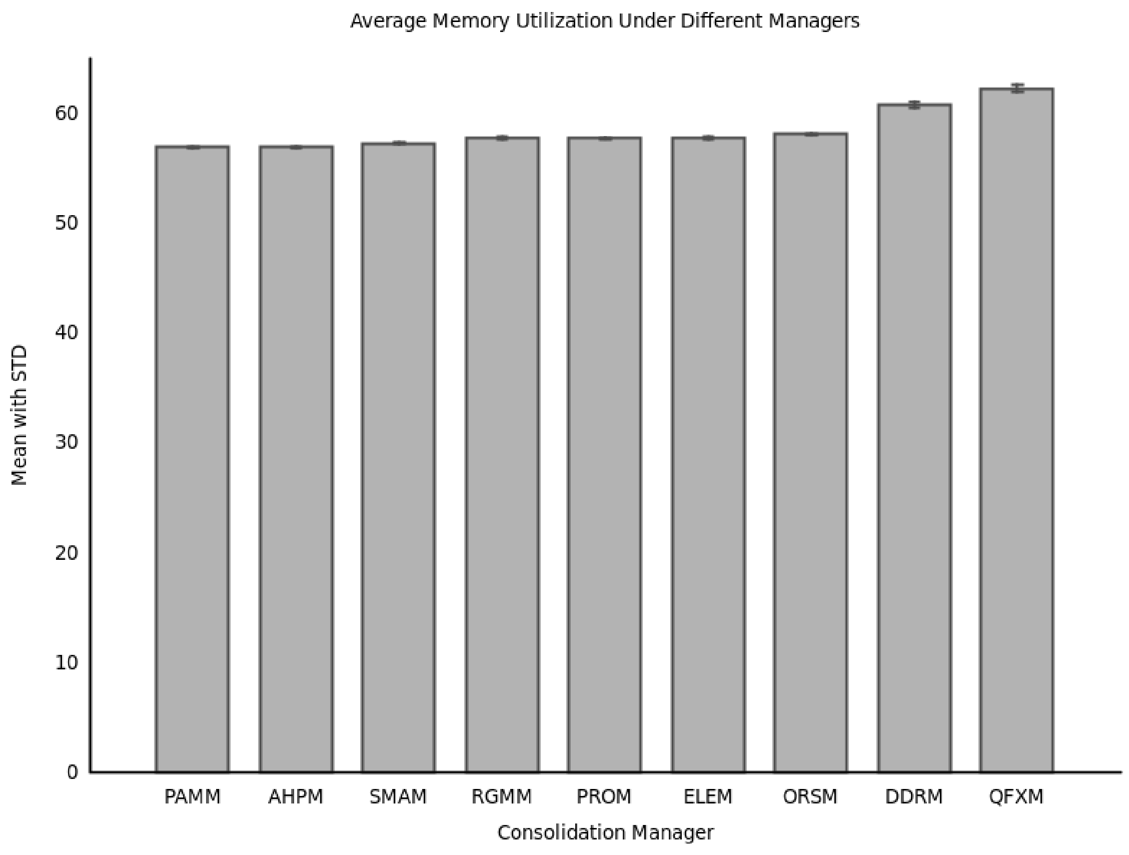

Figure 5.8: The Average Memory Utilization Under Different Managers

managers utilize bandwidth between 45.260% and 46.725%. QFXM achieves 50.735% utilization on this resource dimension. The main reason behind QFXM achieving the most amount of resource utilization under all dimensions is the tight virtual machine packing done by the manager. QFXM uses the least number of physical machines with the average of 109.026, whereas PAMM uses the most amount with 211.015. Naturally, these results comply with the CPU, memory and bandwidth utilizations achieved by the managers.

Until this point in the analysis of the results, we have provided the values of performance metrics achieved by each manager and the rankings with respect to the associated measures. In addition, we have suggested reasons for these measures briefly. In order to fully reveal the reasons behind differences and similarities in performance metrics of the eight managers, it is necessary to investigate the mathematics and the underlying algorithms deeply. In this context, let us investigate the managers in the following four groups : (1) PROM, ELEM and PAMM, (2) ORSM and RGMM, (3)

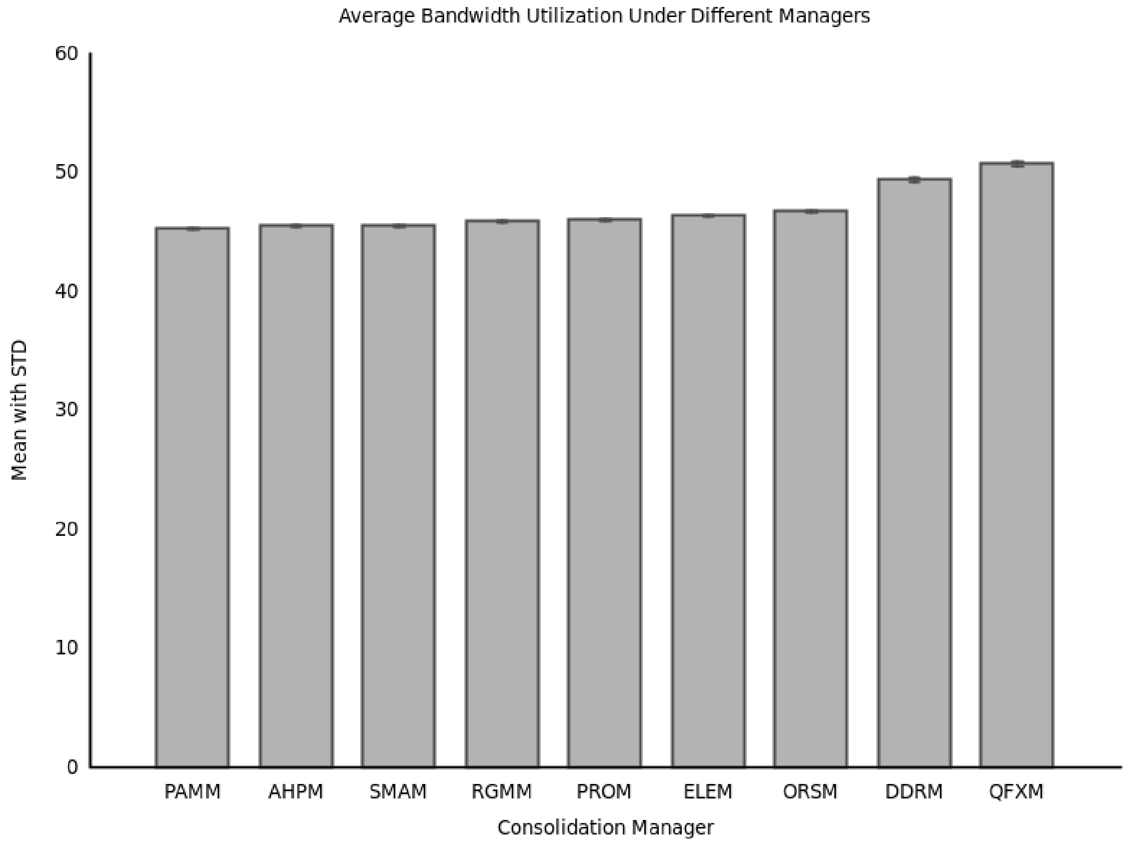

Figure 5.9: The Average Bandwidth Utilization Under Different Managers

AHPM and SMAM and finally (4) QFXM.

PROM, ELEM and PAMM employ three outranking methods of PROMETHEE II, ELECTRE III and PAMSSEM II respectively. Among these three managers, PAMM is of significant importance due to its implementation of PAMSSEM II method which is a hybrid of PROMETHEE II and ELECTRE III outranking methods. In each of these methods, the decision making process has two stages, aggregation and exploitation. PAMSSEM II inherits the aggregation procedure carried out by ELECTRE II method where the concordance matrix is constructed using various thresholds such as preference, indifference and veto. In the exploitation phase, PAMSSEM II method uses the exploitation procedure carried out by PROMETHEE II method where a net outranking flow for each alternative is computed in order to provide the final ranking in the form of a complete pre-order. Therefore, the main difference in the performance metrics achieved by PAMM and ELEM lie in their implementation of the exploitation procedure. ELECTRE III method computes two pre-orders

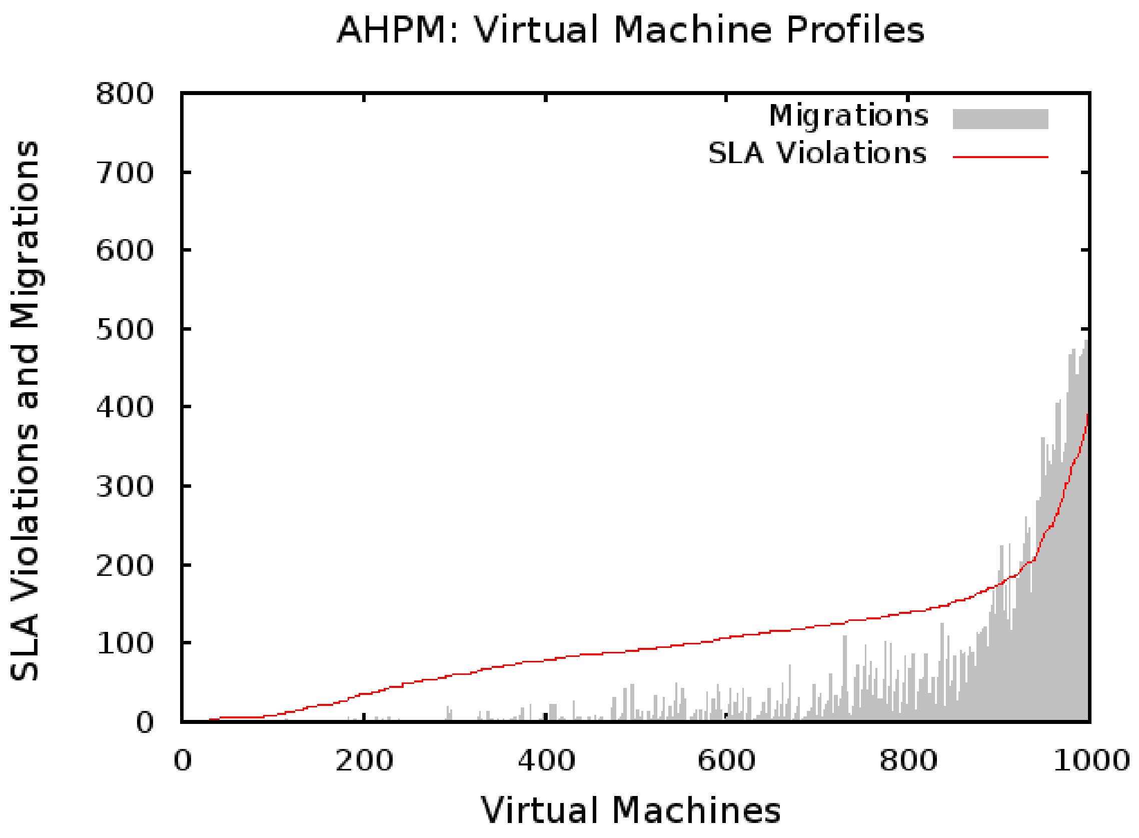

Figure 5.10: AHPM Virtual Machine Profiles

called ascending distillation and descending distillation. The ascending distillation is formed by aggregating the entries in the concordance matrix in order to find the least preferred alternative. When such an alternative is found, it is removed from the concordance matrix. The process is repeated until no alternatives remain in the concordance matrix. The order of this removal process results in the ascending distillation. On the other hand, the exact opposite procedure is implemented in the case of descending distillation where the most preferred alternative is removed from the concordance matrix in each step. Finally, the two distillations are intersected in order form pre-order that is consistent with both distillations. It is important to note that the result may possibly be a partial pre-order with incomparabilities and inconsistencies between alternatives. In order to further asses the differences between the two methods, it is necessary to look in the virtual machine migrations carried out by these managers. PAMM and ELEM migrated 96 same virtual machines out of 100. The difference in the consistency of virtual machine migrations increase when

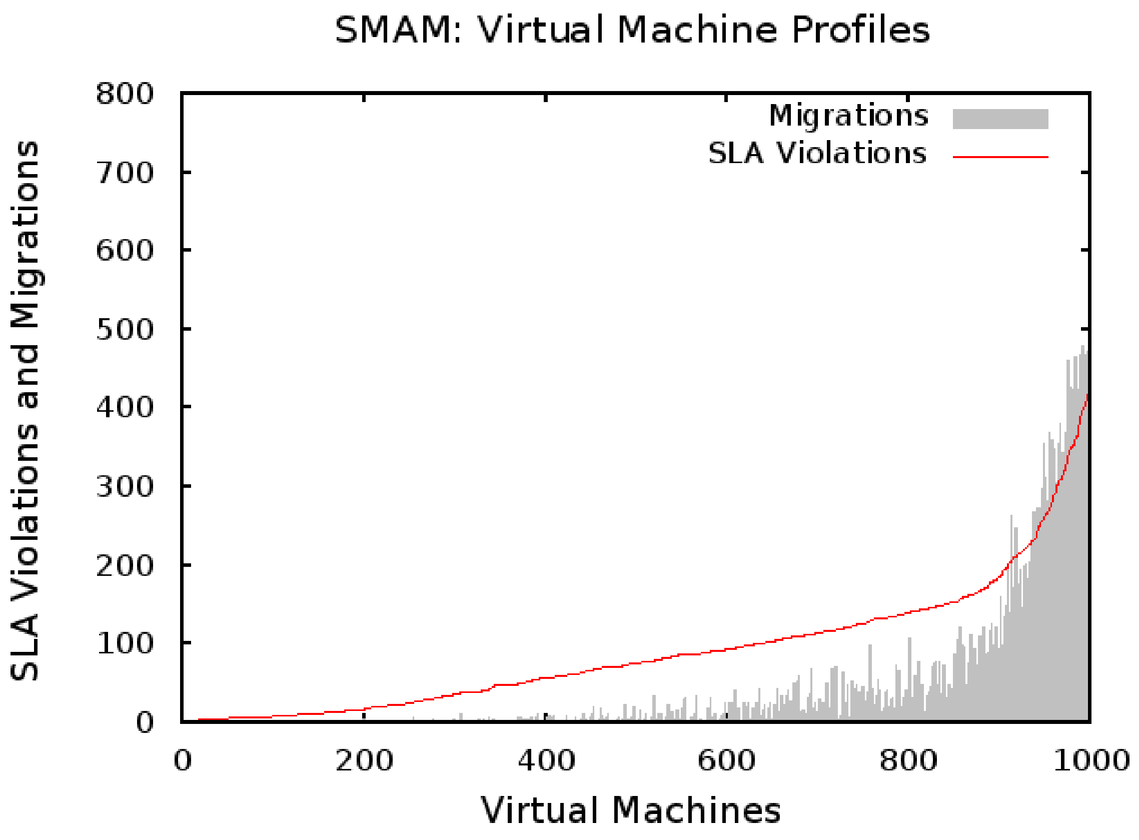

Figure 5.11: SMAM Virtual Machine Profiles

the top 500 most migrated virtual machines are considered as only 459 of these are the same.

The main reason behind the performance differences in PAMM and PROM lie in the different aggregation procedures employed by PAMSSEM II and PROMETHEE II outranking methods respectively. Although the calculation of aggregate indices in PROMETHEE II and the construction of the concordance matrix in PAMSEMM II (fundamentally ELECTRE III) display major similarities, the usage of thresholds when suggesting assertions between alternatives causes the source of differences. In both PAMSEMM II and ELECTRE III an assertion is deemed invalid if they are below certain thresholds. In addition, the veto threshold can invalidate assertions if they fall under a given value for a given criteria. This can be explained in detail as follows. Let us consider an alternative that outranks another alternative on every single criteria by a great extent except for a single criterion. In the cases where, the performance of the alternative falls under the veto threshold, the assertion is deemed

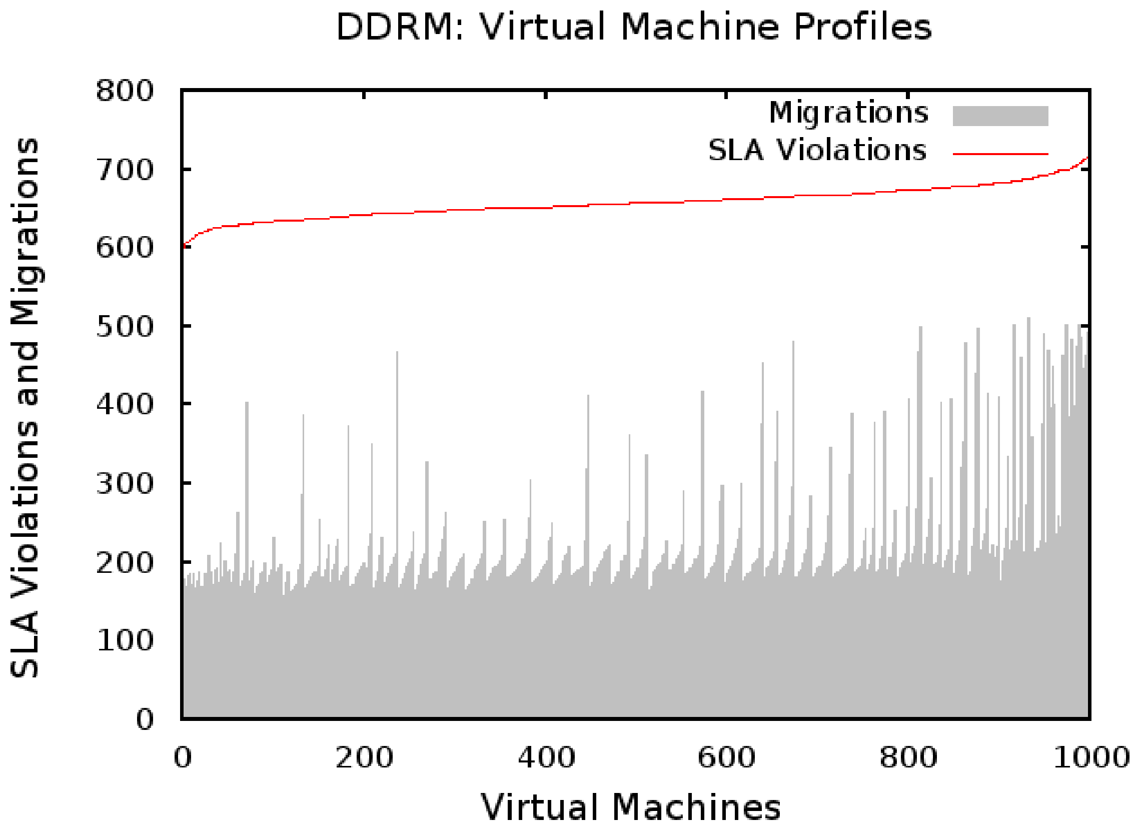

Figure 5.12: DDRM Virtual Machine Profiles

invalid regardless of the other criteria that suggest in the favour of this assertion. Therefore, PAMM and ELEM can encounter inconsistent assertions due to the veto threshold which is absent in PROM. In this regard, it is important to stress the importance of setting these thresholds correctly. PAMM and PROM migrated the same 97 out of 100 and 455 out of 500 virtual machines respectively which denotes a significant percent of commonality in the decisions made.

The performance differences between PROM and ELEM can be explained following a similar logic. These two managers employ different procedures in both aggregation and exploitation phases. Despite the differences in the underlying algorithms, 97 out of 100 and 462 out of 500 most migrated virtual machines are common. Based on the comparisons between PROM, ELEM and PAMM, it can be concluded that the main difference in performance lie in the usage and the absence of veto thresholds. In this context, PROMETHEE II method produces stronger assertions between alternatives when the performance differences of alternatives are considerably small.

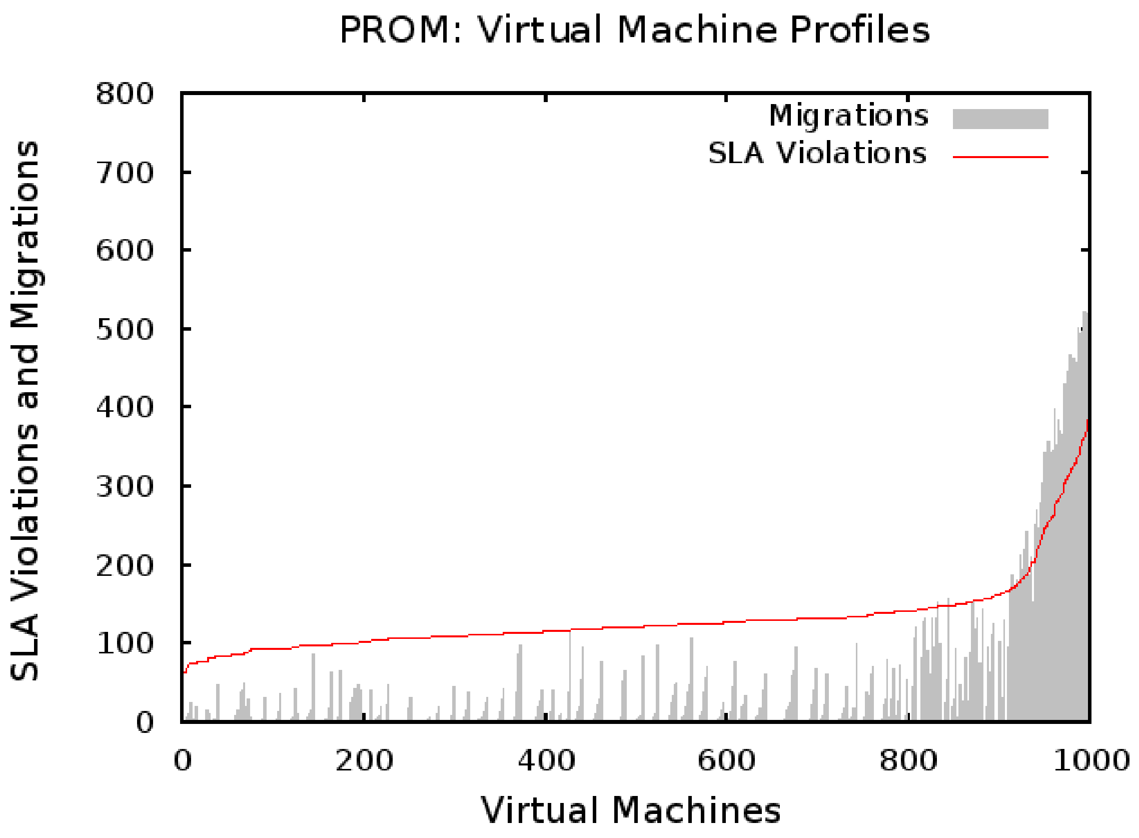

Figure 5.13: PROM Virtual Machine Profiles

This, in turn, results the different physical machine selections which further impact the performance metrics achieved by these three managers.

In the second group of outranking methods, RGMM and ORST differ in their aggregation procedures employed by REGIME and ORESTE methods respectively. REGIME method focuses on the sign of the difference in the performance values of alternatives during the pairwise comparisons. This, in turn, reduces the sensitivity of the assertion when the performance differences are large. Let us consider a three alternative setting where the first and the second outrank the third alternative by 10 and 50 units respectively. Although the second alternative should be deemed the clear winner in this case, REGIME method is indifferent between the two alternatives. The assertion is strictly based on the sign of the evaluation difference where preference is assigned when the sign is positive, preference in the opposite direction is assigned when the sign is negative and finally indifference when the evaluation difference is zero. In ORESTE method, the aggregation procedure starts by ranking each criteria

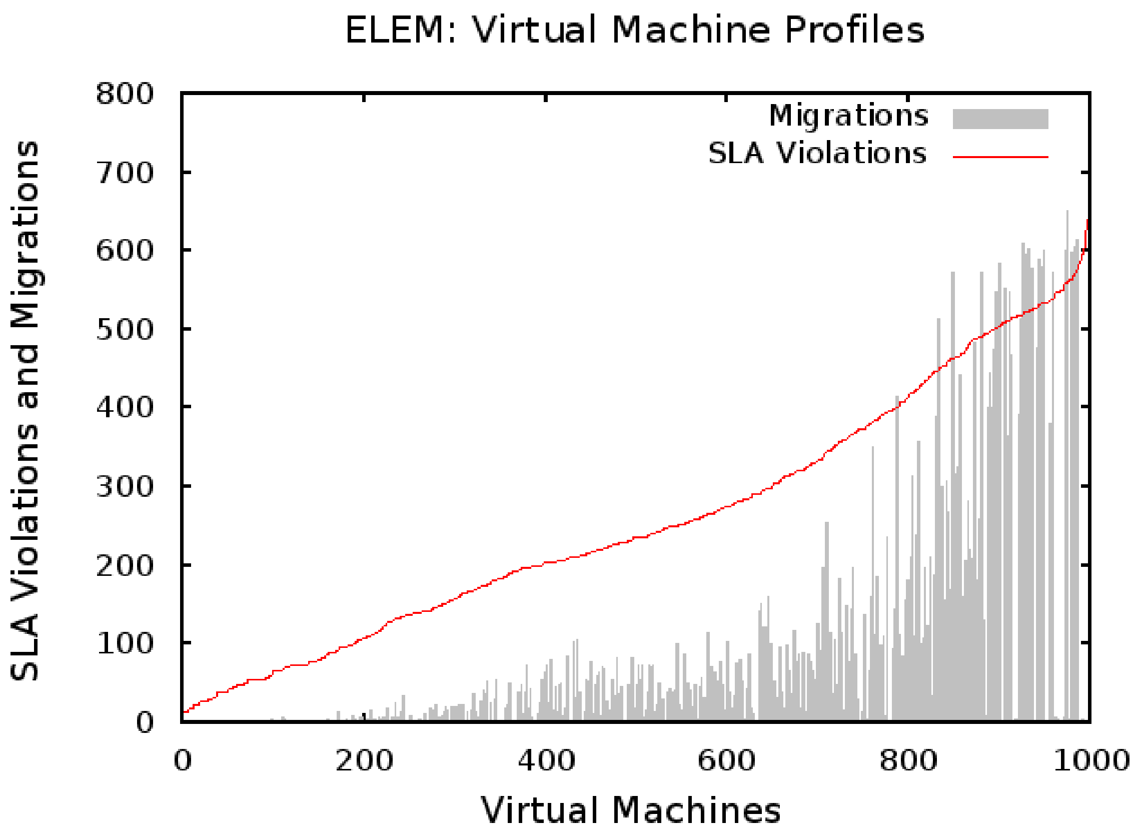

Figure 5.14: ELEM Virtual Machine Profiles

as well as the alternatives. These aggregations form pre-orders which are then used to construct a projection matrix. Furthermore, the projection matrix is aggregated in order to convert these projections into a global ranking matrix. Clearly, the aggregation procedures performed by these two methods are extremely different. However, both of these methods do not suggest a total pre-order of the alternatives as they leave the decision maker with preference relations among alternatives. In this regard, RGMM and ORST implement a similar exploitation procedure to PAMM where a set of non-dominated alternatives is formed. The winner is chosen at random from this set of alternatives. Despite the differences in the aggregation procedures, 95 out of 100 and 414 out of 500 most migrated virtual machines are common.

AHPM and SMAM employ single synthesizing criteria methods of AHP and SMART respectively. Therefore, it is necessary to investigate the difference of these two managers together. SMART method implemented by SMAM uses an approach much like a weighted sum utility function. The performance evaluations of each

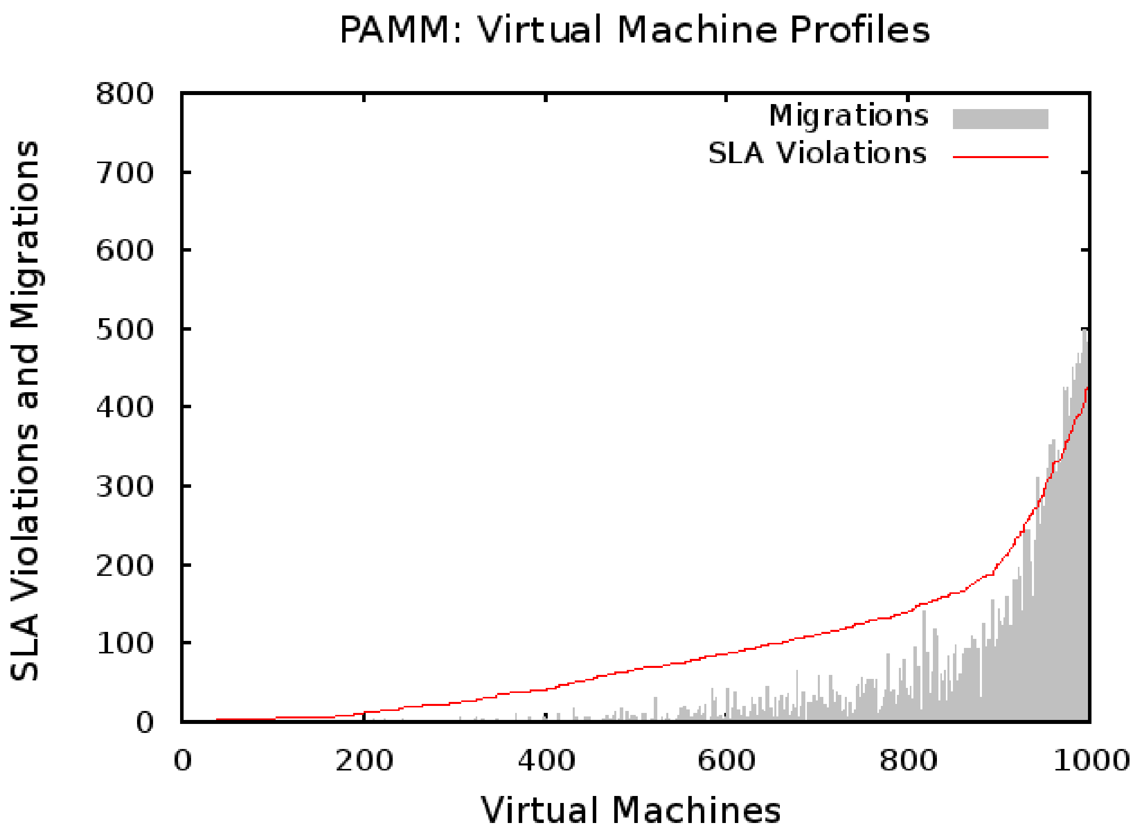

Figure 5.15: PAMM Virtual Machine Profiles

alternative is converted into a single value where no pairwise comparisons are necessary. The linear utility function uses in SMAM ensures the smaller differences between alternatives produce small distinctions regardless of the distance between the most preferred and the least preferred alternatives. On the other hand, AHP method implemented by AHPM forms a hierarchical structure with respect to design of IMPROMTU. The alternatives are evaluated at each level in a pairwise manner by modulating the strength of preference between the values 1/9 and 9 before generating aggregate evaluations. In comparison to SMAM, this results in clear differences in alternatives at hand. Despite the completely different methods of reducing various performance measures into a single performance measure, 97 out of 100 and 461 out of 500 most migrated virtual machines are common. In this context, the process of choosing physical machines with respect to the offered available resources also slightly differ.

QFXM is excluded from grouping with other multiple criteria decision analysis

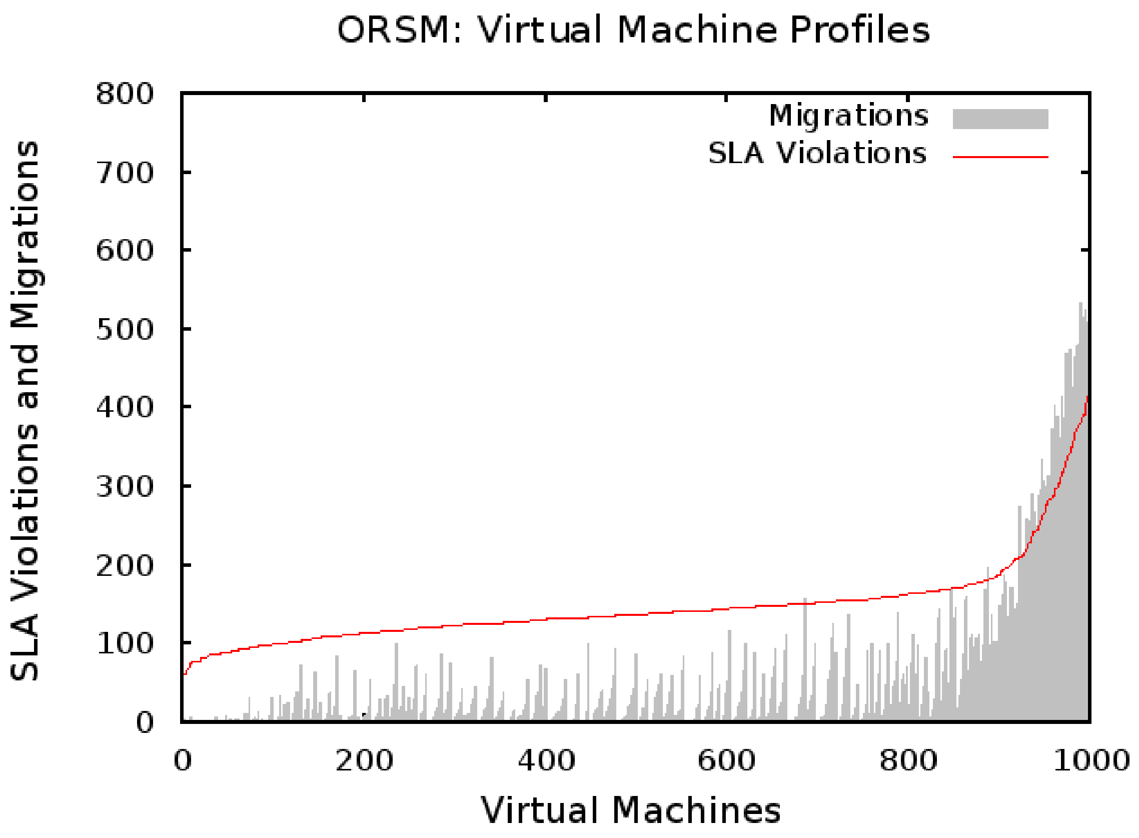

Figure 5.16: ORSM Virtual Machine Profiles

methods due to its computationally intensive characteristics that limit the decision making process. QFXM employs QUALIFLEX method which can be seen as the aggregation of permutation evaluations. In the considerably large simulation setting used in this study, evaluating all possible permutations of alternatives at each step is infeasible. In particular, QUALIFLEX method treats each one of $n!$ permutations as a pre-order. The ranking in these pre-orders are compared in a pairwise manner in order to asses the performance of a given permutation. QFXM uses a filtration mechanism in order to reduce the number of alternatives analysed by the QUALIFLEX method. Firstly, the alternatives are sorted lexicographically. From the sorted set of alternatives, the top 5 alternatives with the highest performance measures are picked to be analysed by QUALIFLEX. The filtration process introduces cases of inconsistency as the lexicographical sorting alternatives with conflicting criteria is not the most sound option. This claim is supported when the most migrated virtual machines among all managers are considered. In the case where the virtual machine migrations

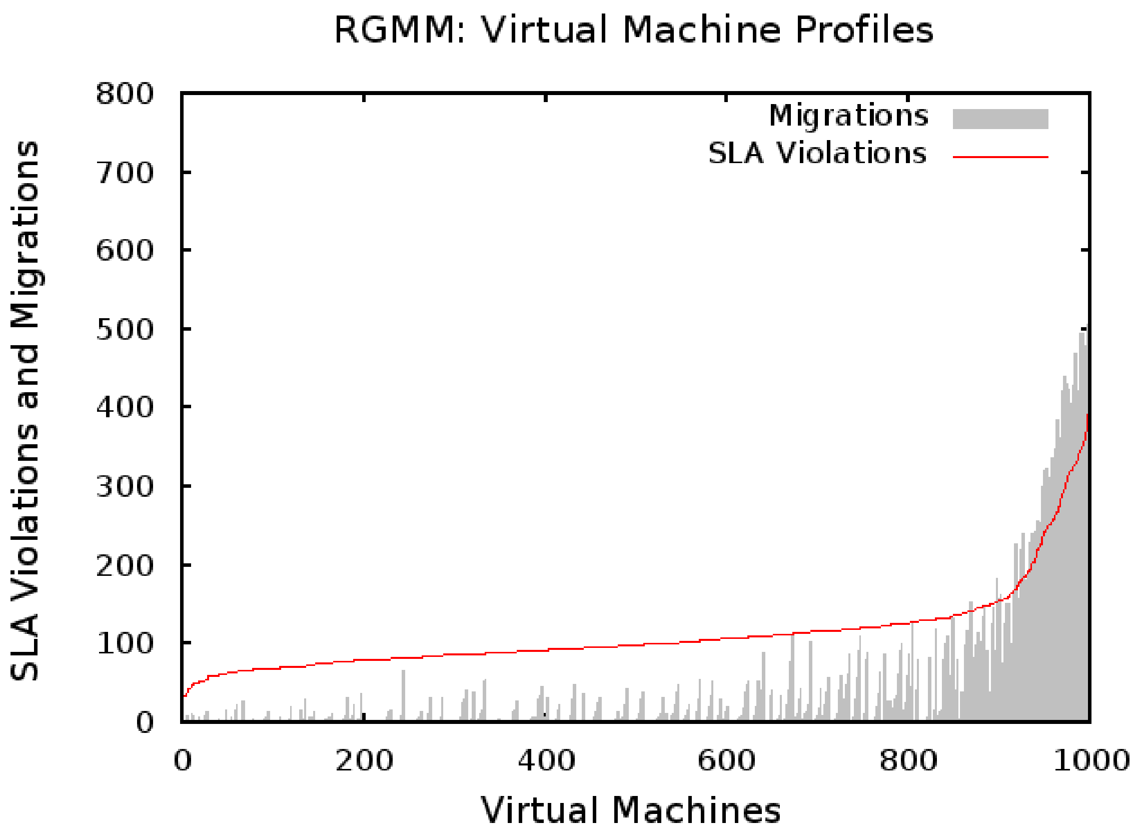

Figure 5.17: RGMM Virtual Machine Profiles

of seven managers excluding QFXM is considered 90 out of 100 and 382 out of 500 most migrated virtual machines are common. However, in the case where QFXM is included in the group these values decrease drastically to 37 out of 100 and 274 our of 500. Another interesting finding regarding these numbers is that the QFXM especially performs poorly when choosing the top 100 virtual machines.

5.3 Summary

IMPROMPTU model was first suggested and analysed under the CDFF, DDR, IMP-1 and IMP-2 managers which employ Centralized First Fit, Distributed Dynamic Random, PROMETHEE II and PROMETHEE II with influence criteria respectively in the former studies [80]. IMPROMPTU showed that the distributed reactive approach could be a valid alternative to the existing centralized architectures as it proposed a scalable system, as well as, triggering less service level agreement violations by

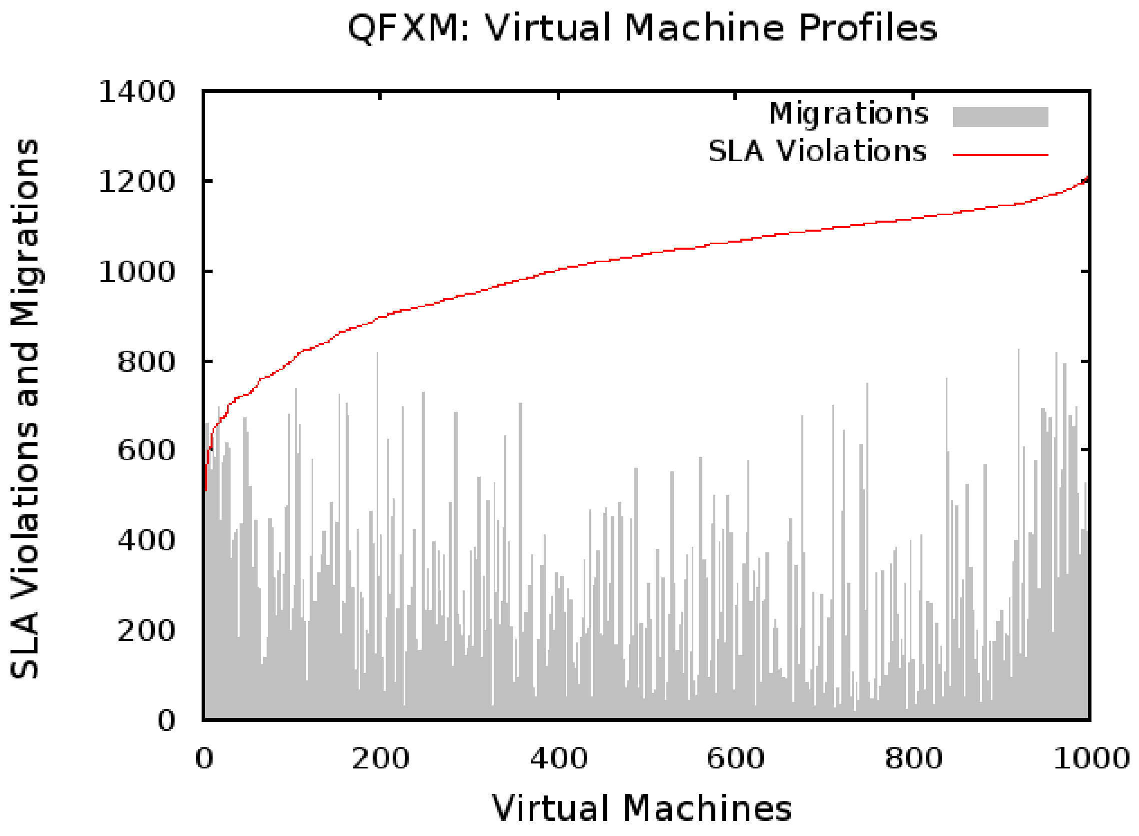

Figure 5.18: QFXM Virtual Machine Profiles

smoothly distributing the overall resource utilizations within the data center. The previous work left the need for investigating a comprehensive set of multiple criteria decision analysis methods within the context of dynamic resource allocation. This analysis, in turn, answers whether better performing MCDA methods exist and highlights the properties of good MCDA methods in the domain of dynamic resource allocation.

The data collected and analysed from the simulation runs can be further assessed, however using an MCDA method to decide on the ranking of a set of MCDA methods is ironic. Furthermore, such assessments can yield contradictory results. Therefore, this process is not a viable option. However, evaluation of these methods is possible with out mathematical procedures when the performance measures are prioritized. In this context, we can refer to the two most valuable performance measures as configuration time and the amount of migrations.

The main reason behind adopting a distributed architecture for the resource allocation management is to achieve a responsive system. In an environment where the workloads of application environments are changing rapidly and continuously, a virtual machine to physical machine mapping is only valid and of use, if it can be computed faster than the change in workloads. In the majority of the previous work, authors provide elegant but computationally complex methods to calculate optimal mappings. From a practical point of view, these configurations are stale and hold no importance. Therefore, configuration time can be regarded as having the power of a veto threshold. In addition, the amount of migrations carried out by a manager is also equally important due to the practical limitations. Constantly migrating virtual machines within the data center imposes additional computational burden that results in critical performance issues. On the other hand, physical machine usage and the amount of service level agreement violations are not bound by practical limitations. A service provider may prefer to employ extra physical machines and possibly negotiate less strict penalties in the case of violations. In this regard, these two measures solely depend on the preferences of the service providers. Therefore, assuming that the amount of service level agreement violations and the amount of migrations are of equal importance, the ranking of most preferred to least preferred managers can be given as follows: (1) SMAM, (2) AHPM, (3) ELEM. In this sense, it is important to note that configuration times over 0.4 seconds are assumed to yield stale solutions.

The analysis performance measures leave open questions for further research. The standard deviation values appear to be related to the mathematical methods employed by the managers, which are outlined in the previous subsection. However, further research is necessary to pin point the source of these differences. Finally, the virtual machine profiles represented in Figures 5.10−5.18 suggest that the majority of the migrations and service level agreements violations are caused by the problematic virtual machines. Particularly, this finding suggests the possible improvements of managers if the problematic and the non-problematic virtual machines can be blended in a more efficient way. Furthermore, isolating such virtual machines can be considered as the last resort for the sake of other virtual machines hosted in the data center.

Chapter 6

Conclusion and Future Work

In this thesis, we have extended the existing IMPROMPTU model with six outranking based and two single synthesizing criterion based MCDA approaches. Firstly, we have provided a detailed introduction to the distributed systems. In this context, a chronological order of milestone achievements are listed with various associated examples. The idea of billing clients for computational services is an idea that dates back to 1960s. However, due to the insufficient operating system, semi-conductor and communication technologies, this business centric view of distributed could not be realized for decades. Although, virtualization not being a new paradigm, it lies at the core of cloud computing. The ability to host, suspend and migrate virtual machines within data centers enables the resource allocation process to be dynamic and autonomous. An efficient dynamic resource consolidation manager can minimize service level agreement violations and virtual machine migrations, as well as, maximizing resource utilization without any human interaction. In addition, calculating virtual machine to physical machine mappings in an environment where the conditions are continuously changing calls for quick configuration times that are beyond human capabilities. In this context, dynamic resource consolidation emerges as a necessity in the current state of cloud computing.

This work provides a detailed literature survey on the existing studies on the problem domain. The problem of dynamic resource allocation management in data centers have been investigated mainly under two categories where the new configurations in the data center are calculated in a centralized or a distributed manner. The centralized methods employ a global arbiter which attempts to maximize a given utility function. These solutions can be efficient in small-scale data center settings, however, they suffer from scalability and feasibility issues in realistic large-scale settings. In addition, new optimal configurations generally take minutes to compute, which results in stale solutions. On the other hand, distributed approaches attempt to distribute the task of resource allocation among the responsible units. These methods are inherently resilient to scalability and feasibility issues. However, some of these approaches lack the responsiveness needed for the dynamic resource allocation or their usage is limited to Map-Reduce applications.

We have also outlined the MCAP based MCDA methods under two categories, single synthesizing criterion and outranking based methods. Single synthesizing criterion methods aim to reduce the different criteria used in the decision making process into a single comprehensive index by employing some rules, procedures and mathematical formulae. In this context, we have provided formal definitions of Simple Multi-Attribute Rating Technique (SMART) and Analytic Hierarchy Process (AHP). On the other hand, outranking based MCDA methods follow a successive pairwise comparison of alternative actions in isolation to produce a complete pre-order. This thesis overviews the following six outranking based MCDA methods, PROMETHEE II, ELECTRE III, PAMSSEM II, ORESTE, REGIME and QUALIFLEX. Moreover, we have provided details and explanations regarding each method.

This thesis provides an overview of IMPROMPTU which is a threshold based reactive resource consolidation for clouds. IMPROMPTU distributes the responsibility of resource allocation management among autonomous node agents that are tightly coupled with a unique physical machine in the data center. The node agents monitor the resource usage on each physical machine and migrate virtual machines to be hosted elsewhere in the data center when the conditions impose. This in turn, results in a reactive behaviour where each node agent attempts to keep the associated physical machine within desired states. The reaction process calls for the following two decisions, choosing a virtual machine for migration and choosing a physical machine in order to be the new host for this virtual machine. In this context, the next course of action is based on the MCDA method employed by the node agents. The details about the implementation of these methods are also given in detail.

Furthermore, we have analysed the performance of eight MCDA based managers in-detail using the results gathered from simulation runs. These managers are investigated in terms of physical machine usage, amount of migrations, amount of service level agreement violations and configuration times. In this context, this thesis is particularly significant as a comprehensive set of managers provide valuable information denoting the properties of good MCDA methods for this problem domain.

From the author's point of view, the most important criteria regarding the qualities of a manager are virtual machine migrations and configuration times due to practical limitations. In this context, an efficient manager needs to compute new virtual machine to physical machine mappings as these configurations are only valid when they are computed faster than changing workloads. On the other hand, migrating virtual machines imposes additional computational burden on the data center. Furthermore, the results show that the majority of the migrations and service level agreement violations are caused by specific problematic virtual machines. In the context of industry application, SMART is the most feasible candidate amongst the analysed managers in this thesis. The results show that the dynamic resource allocation manager employing SMART (SMAM) tends to find a well balanced spot between resource usage and VM migrations. In addition, the configuration times are efficient enough to adjust to rapidly changing workloads. Finally, compared to the rest of the managers, SMART is the least challenging to implement. This, in turn, makes SMAM basic yet elegant.

The work outlined in this thesis can be extended by improvements to the current state of the system by: (1) conducting a comprehensive sensitivity analysis in order to further calibrate the managers in use. The managers that employ Single synthesizing were ranked as the top managers according to our study. This leaves the question of whether calibrating and fine tuning outranking methods would have an effect on this ranking. (2) introducing additional resource dimensions and modelling their behaviour. This would be beneficial to asses the performance of managers under more criteria. In addition, it would answer the question of whether adding more criteria increases the efficiency of the manager. (3) implementing a blending procedure for problematic virtual machines with the non-problematic ones in a more efficient way by possibly using a better method for load forecasting and reinforcement learning. (4) isolating or penalising the problematic virtual machines as a majority of the SLA violations are caused by the same VMs. Under the ideal isolation setting, the managers may possibly need less migrations, which, in turn, would have an impact on efficiency of resource management. (5) considering the cost of migrations and placement conflicts. The simulation platform needs a layer to model the computational burden of VM migrations. Also the question of whether placement conflicts may occur and how this would impact the migration is left unanswered. (6) adding a proper network layer simulation to better understand the implications of VM migrations and communication between nodes. In the current simulation platform, a global entity is assumed to know the state of every physical and virtual machine in the data center.

In a real life application, this may not be possible as availability metrics are needed to be streamed between nodes. (7) implementing an improved simulation platform to support these enhancements, and finally (8) testing the proposed model on an actual physical cloud setting. Realistically, running a real implementation on cloud with limited number of physical machines would not be the most beneficial test scenario. From a statistical point of view, having a large number of nodes would result in more consistent data. However, running tests on a data center with 1000 physical machines is very unlikely to be feasible. From author's point of view, improving the simulation platform is more important than running tests on a physical test bed with limited resources.

Bibliography

[1] Janet Abbate. *Inventing the Internet.* MIT Press, 2000.

[2] Shubham Agrawal, Sumit Kumar Bose, and Srikanth Sundarrajan. Grouping Genetic Algorithm for Solving the Server Consolidation Problem with Conflicts. In *ACM/SIGEVO GEC'09*, pages 1–8, 2009.

[3] J. Almeida, V. Almeida, D. Ardagna, C. Francalanci, and M. Trubian. Resource Management in the Autonomic Service-Oriented Architecture. In *ICAC '06: IEEE International Conference on Autonomic Computing*, pages 84–92, 2006.

[4] Gregory R. Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming.* Addison-Wesley, 2000.

[5] E. Arianyan, D. Maleki, A. Yari, and I. Arianyan. Efficient resource allocation in cloud data centers through genetic algorithm. In *Telecommunications (IST), 2012 Sixth International Symposium on*, pages 566–570, 2012.

[6] Ed Barbeau. Perron's Result and a Decision on Admissions Tests. *Mathematics Magazine*, 59(1):12—22, 1986.

[7] Mohamed N. Bennani and Daniel A. Menasce. Resource Allocation for Autonomic Data Centers using Analytic Performance Models. In *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, pages 229–240, 2005.

[8] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic Placement of Virtual Machines for Managing SLA Violations. In *IM '07: 10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 119–128, 2007.

[9] D. Bouyssou. Some Remarks on the Notion of Compensation in MCDM. *European Journal of Operational Research*, 26:150—160, 1986.

[10] J. P. Brans. *L'Ingéniérie de la Décision. Elaboration d'Instruments d'Aide à la Décision. Méthode PROMETHEE',*. PhD thesis, Université Laval, Québec, Canada, 1982.

[11] J. P. Brans, B. Mareschal, and P. Vincke. How to Select and How to Rank Projects : The PROMETHEE Method for MCDM. *European Journal of Operations Research*, 24:228—238.

[12] J. P. Brans, B. Mareschal, and P. Vincke. PROMETHEE : A new family of outranking methods in MCDM. In *IFORS'84*, 1984.

[13] Jean-Pierre Brans and Bertrand Mareschal. Promethee methods. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 163—186. Springer New York.

[14] J.P Brans, B. Mareschal, and P. Vincke. A Preference Ranking Organisation Method : The PROMETHEE Method for MCDM. *Management Science*, 31(6):647—656.

[15] D.M. Chess, A. Segal, I. Whalley, and S.R. White. Unity: Experiences with a Prototype Autonomic Computing System. In *2004. Proceedings. International Conference on Autonomic Computing*, pages 140–147, 2004.

[16] T.C. Chieu and Hoi Chan. Dynamic resource allocation via distributed decisions in cloud environment. In *e-Business Engineering (ICEBE), 2011 IEEE 8th International Conference on*, pages 125–130, 2011.

[17] A. Chipperfield, J. Whidborne, and P. Fleming. Evolutionary Algorithms and Simulated Annealing for MCDM. In *Multicriteria Decision Making: Advances in MCDM Models, Algorithms, Theory and Applications*, pages 16.1—16.32. Kluwer Academic Publishers, 1999.

[18] CHOCO. Choco library, 2013. [Online; accessed 14-October-2013].

[19] R. Das, J.O. Kephart, I.N. Whalley, and P. Vytas. Towards Commercialization of Utility-based Resource Allocation. In *ICAC '06: IEEE International Conference on Autonomic Computing*, pages 287–290, 2006.

[20] distributed.net. distributed.net:project rc5, 2013. [Online; accessed 14-October-2013].

[21] Shlomi Dolev. *Self-Stabilization.* MIT Press, 2000.

[22] X. Dutreilh, N. Rivierre, A. Moreau, J. Malenfant, and I. Truck. From data center resource allocation to control theory and back. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 410–417, 2010.

[23] James Dyer. Maut—multiattribute utility theory. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 265—292. Springer New York.

[24] W. Edwards. How to Use Multiattribute Measurement for Social Decision Making. *IEEE Transactions on Systems, Man and Cybernetics*, 5:326—340, 1977.

[25] Eugen Feller, Cyril Rohr, David Margery, and Christine Morin. Energy Management in IaaS Clouds: A Holistic Approach. In *IEEE CLOUD'12*, pages 204–212, 2012.

[26] Jacques Ferber. *Multi-Agent System: An Introduction to Distributed Artificial Intelligence.* Harlow: Addison Wesley Longman, 1999.

[27] José Figueira, Vincent Mousseau, and Bernard Roy. Electre methods. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 133—153. Springer New York.

[28] Folding@home. Folding@home, http://folding.stanford.edu/home/. [Online; accessed 14-October-2013].

[29] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *GCE: Grid Computing Environments Workshop*, pages 1–10, 2008.

[30] M.E. Frincu and C. Craciun. Multi-objective meta-heuristics for scheduling applications with high availability requirements and cost constraints in multi-cloud environments. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 267–274, 2011.

[31] Sukumar Ghosh. *Distributed Systems: An Algorithmic Approach.* Chapman & Hall/CRC, November 2006.

[32] Bill Godfrey. A Primer on Distributed Computing. http://www.bacchae.co.uk/docs/dist.html, Retreived 2013.

[33] S. Greco, B. Matarazzo, and R. Slowinski. The Use of Rough Sets and Fuzzy Sets in MCDM. In *Multicriteria Decision Making: Advances in MCDM Models, Algorithms, Theory and Applications*, pages 14.1—14.59. Kluwer Academic Publishers, 1999.

[34] Adel Guitouni, Jean-Marc Martel, Micheline Blanger, and Charles Hunter. Multiple Criteria Courses of Action Selection. *Military Operations Research*, 13(1):35–50, 2008.

[35] Fabien Hermenier, Xavier Lorca, Jean M. Menaud, Gilles Muller, and Julia Lawall. Entropy: A Consolidation Manager for Clusters. In *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 41–50, 2009.

[36] Edwin Hinloopen, Peter Nijkamp, and Piet Rietveld. Qualitative Discrete Multiple Criteria Choice Models in Regional Planning. *Regional Science and Urban Economics*, 13(1):77–102, 1983.

[37] History-Computer. Lisp of john mccarthy, 2013. [Online; accessed 14-October-2013].

[38] Brian Hopkins. Is Moore's Law Still Valid? [Online; accessed 14-October-2013].

[39] IBM. Cloud computing with Linux, http://www.ibm.com/developerworks/linux/library/l-cloud-computing/index.html. [Online; accessed 14-October-2013].

[40] IBM. Autonomic Computing: IBM's Perspective on the State of Information Technology. Technical report, IBM Research, 2011.

[41] Intel. Moore's law and intel innovation, 2013. [Online; accessed 14-October-2013].

[42] E. Jacquet-Lagrèze and Y. Siskos. Assessing a Set of Additive Utility Functions fo Multicriteria Decision Making: The UTA Method. *European Journal of Operational Research*, 10(2):151—164, 1982.

[43] Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *Computer*, 36:41–50, January 2003.

[44] G. Khanna, K. Beaty, G. Kar, and A. Kochut. Application Performance Management in Virtualized Server Environments. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 373–381, 2006.

[45] Aaron Kimball. Problem Solving on Large-Scale Clusters, http://code.google.com/edu/submissions/mapreduce-minilecture/listing.html, Retreived 2011.

[46] A. Kochut. On Impact of Dynamic Virtual Machine Reallocation on Data Center Efficiency. In *Modeling, Analysis and Simulation of Computers and Telecommunication Systems, 2008. MASCOTS 2008. IEEE International Symposium on*, pages 1 –8, 2008.

[47] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

[48] J. M. Martel and B. Matarazzo. Other outranking approaches. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 197—259. Springer New York.

[49] M. Maurer, I. Brandic, and R. Sakellariou. Self-adaptive and resource-efficient sla enactment for cloud computing infrastructures. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 368–375, 2012.

[50] Aaron McConnell, Gerard Parr, Sally McClean, Philip Morrow, and Bryan Scotney. A sla-compliant cloud resource allocation framework for n-tier applications. In *Cloud Networking (CLOUDNET), 2012 IEEE 1st International Conference on*, pages 41–45, 2012.

[51] Daniel A. Menasce and Mohamed N. Bennani. Autonomic Virtualized Environments. In *ICAS: Proceedings of the International Conference on Autonomic and Autonomous Systems*, pages 28–37. IEEE Computer Society, 2006.

[52] Patrick Meyer and Marc Roubens. Choice, ranking and sorting in fuzzy multiple criteria decision aid. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 471—503. Springer New York.

[53] Cary Millsap. Thinking Clearly About Performance, Part 2. *Communications of the ACM*, 53(10):39–45, 2010.

[54] David Peleg. *Distributed Computing: A Locality-Sensitive Approach.* SIAM, 2000.

[55] P. Perny and J.Ch. Pomerol. Use of Artificial Intelligence in MCDM. In *Multi-criteria Decision Making: Advances in MCDM Models, Algorithms, Theory and Applications*, pages 15.1—15.43. Kluwer Academic Publishers, 1999.

[56] Marc Roubens. Preference Relations on Actions and Criteria in Multiple Criteria Decision Making. *EJOR*, 10:51–55, 1982.

[57] B. Roy and V. Mousseau. A Theoretical Framework for Analysing the Notion of Relative Importance of Criteria. *Journal of Multi Criteria Decision Analysis*, 5:145—159, 1996.

[58] Bernard Roy. Paradigms and challenges. In *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 3—24. Springer New York.

[59] Bernard Roy. Classement et Choix en Présence de Points de Vue Multiples (La Méthode ELECTRE. *RIRO*, 8:57—75, 1968.

[60] Bernard Roy. The Outranking Approach and the Foundations of ELECTRE. *Theory and Decision*, 31:49—73, 1991.

[61] Thomas Saaty. A Scaling Method for Priorities in Hierarchical Structures. *Journal of Mathematical Psychology*, 15:234—280, 1977.

[62] Thomas Saaty. *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation.* McGraw-Hill, 1980.

[63] Thomas Saaty. Ranking According to Perron: A New Insight. *Mathematics Magazine*, 60(4):211—213, 1987.

[64] SETI@home. SETI@home, http://setiathome.berkeley.edu. [Online; accessed 14-October-2013].

[65] R. Slowinski. *Intelligent Decision Support: Handbook of Applications and Advances of the Rough Sets Theory.* Kluwer Academic Publishers, 1992.

[66] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems: Principles and Paradigms.* Pearson Prentice Hall, 2006.

[67] G. Tesauro, R. Das, W.E. Walsh, and J.O. Kephart. Utility-Function-Driven Resource Allocation in Autonomic Systems. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pages 342–343, 2005.

[68] G. Tesauro, N.K. Jong, R. Das, and M.N. Bennani. A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation. In *ICAC '06: Proceedings of the 2006 IEEE International Conference on Autonomic Computing*, pages 65–73. IEEE Computer Society, 2006.

[69] Gerald Tesauro. Online Resource Allocation Using Decompositional Reinforcement Learning. In *AAAI'05: Proceedings of the 20th National Conference on Artificial Intelligence*, pages 886–891. AAAI Press, 2005.

[70] Hien Nguyen Van and Frederic Dang Tran. Autonomic Virtual Resource Management for Service Hosting Platforms. In *ICES '09: Proceedings of the International Conference on Software Enginnering Workshop on Software Engineering Challenges of Cloud Computing*, pages 1–8. IEEE Computer Society, 2009.

[71] P. Vincke. L'aide Multicritère à la Dècision. *Éditions de l'Université de Bruxelles.*

[72] William E. Walsh, Gerald Tesauro, Jeffrey O. Kephart, and Rajarshi Das. Utility Functions in Autonomic Systems. In *ICAC '04: Proceedings of the First International Conference on Autonomic Computing*, pages 70–77. IEEE Computer Society, 2004.

[73] Rui Wang and Nagarajan Kandasamy. A Distributed Control Framework for Performance Management of Virtualized Computing Environments: Some Preliminary Results. In *ACDC'09*, pages 7–12, 2009.

[74] XiaoYing Wang, DongJun Lan, Gang Wang, Xing Fang, Meng Ye, Ying Chen, and QingBo Wang. Appliance-Based Autonomic Provisioning Framework for Virtualized Outsourcing Data Center. In *Autonomic Computing, 2007. ICAC '07. Fourth International Conference on*, pages 29–29, 2007.

[75] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Black-Box and Gray-Box Strategies for Virtual Machine Migration. In *4th USENIX Symposium on Networked Systems Design and Implementation*, pages 229–242, 2007.

[76] Michael Wooldridge. *An Introduction Multiagent Systems.* John Wiley and Sons, 2009.

[77] Zhen Xiao, Weijia Song, and Qi Chen. Dynamic resource allocation using virtual machines for cloud computing environment. *Parallel and Distributed Systems, IEEE Transactions on*, 24(6):1107–1117, 2013.

[78] Wang Xing-Wei, Wang Xue-yi, and Huang Min. A resource allocation method based on the limited english combinatorial auction under cloud computing environment. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*, pages 905–909, 2012.

[79] Yağız Onat Yazır. On the Virtues and Liabilities of ConfiDNS: Can Simple Tactics Overcome Deep Insecurities? Master's thesis, University of Victoria, 2007.

[80] Yağız Onat Yazır. *Multiple Criteria Decision Analysis in Autonomous Computing: A Study on Independent and Coordinated Self-Management.* PhD thesis, University of Victoria, 2011.

[81] Yağız Onat Yazır, Yağmur Akbulut, Roozbeh Farahbod, Adel Guitouni, Stephen William Neville, Sudhakar Ganti, and Yvonne Coady. Autonomous Resource Consolidation Management in Clouds Using IMPROMPTU Extensions. In *IEEE CLOUD'12*, pages 614–621, 2012.

[82] Yağız Onat Yazır, 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 *IEEE CLOUD: 2010 IEEE 3rd International Conference on Cloud Computing*, 2010.

[83] Yağız Onat Yazır, Chris Matthews, Roozbeh Farahbod, Stephen William Neville, Adel Guitouni, Sudhakar Ganti, and Yvonne Coady. Dynamic Resource Allocation in Computing Clouds Using Distributed Multiple Criteria Decision Analysis. In *IEEE CLOUD'10*, pages 91–98, 2010.

[84] Z.I.M. Yusoh and Maolin Tang. A penalty-based grouping genetic algorithm for multiple composite saas components clustering in cloud. In *Systems, Man, and*

Cybernetics (SMC), 2012 IEEE International Conference on, pages 1396–1401, 2012.

[85] Zhongni Zheng, Rui Wang, Hai Zhong, and Xuejie Zhang. An approach for cloud resource scheduling based on parallel genetic algorithm. In *Computer Research and Development (ICCRD), 2011 3rd International Conference on*, volume 2, pages 444–447, 2011.