December 2014

Dynamic Resource Management in Virtualized Data Centres

Gaston Keller
The University of Western Ontario

Supervisor
Hanan Lutfiyya
The University of Western Ontario

Graduate Program in Computer Science

A thesis submitted in partial fulfillment of the requirements for the degree in Doctor of Philosophy

DYNAMIC RESOURCE MANAGEMENT IN VIRTUALIZED DATA CENTRES

(Thesis format: Integrated Article)

by

Gastón Keller

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

Abstract

In the last decade, Cloud Computing has become a disruptive force in the computing landscape, changing the way in which software is designed, deployed and used over the world. Its adoption has been substantial and it is only expected to continue growing. The growth of this new model is supported by the proliferation of large-scale data centres, built for the express purpose of hosting cloud workloads. These data centres rely on systems virtualization to host multiple workloads per physical server, thus increasing their infrastructures' utilization and decreasing their power consumption. However, the owners of the cloud workloads expect their applications' demand to be satisfied at all times, and placing too many workloads in one physical server can risk meeting those service expectations. These and other management goals make the task of managing a cloud-supporting data centre a complex challenge, but one that needs to be addressed.

In this work, we address a few of the management challenges associated with dynamic resource management in virtualized data centres. We investigate the application of First Fit heuristics to the Virtual Machine Relocation problem (that is, the problem of migrating VMs away from stressed or overloaded hosts) and the effect that different heuristics have, as reflected in the performance metrics of the data centre. We also investigate how to pursue multiple goals in data centre management and propose a method to achieve precisely that by dynamically switching management strategies at runtime according to data centre state. In order to improve system scalability and decrease network management overhead, we propose architecting the management system as a topology-aware hierarchy of managing elements, which limits the flow of management data across the data centre. Finally, we address the challenge of managing multi-VM applications with placement constraints in data centres, while still trying to achieve high levels of resource utilization and client satisfaction.

Keywords: data center management, virtualized infrastructure management, cloud management, energy management, SLA management, application management.

Co-Authorship Statement

Each chapter of this thesis corresponds to a research paper in which the candidate was primary author. The candidate's supervisor, Dr. Hanan Lutfiyya, appears as co-author in every paper to indicate her participation in the research project, providing guidance both with the research itself and its write-up. Dr. Michael Bauer appears as co-author in some of the papers, too, having had a similar supervisory role as that of Dr. Lutfiyya.

An Analysis of First Fit Heuristics for VM Relocation

This work was primarily authored by the candidate, who developed the research idea, conducted the literature review, designed and implemented the proposed heuristics, designed and performed the experiments, and wrote the paper. Fellow PhD Candidate Michael Tighe contributed to the project providing feedback and assisting with experiment design and writing.

Switching Data Centre Management Strategies at Runtime

This work resulted from a collaboration between the candidate, fellow PhD Candidate Michael Tighe, and former MSc student Graham Foster. The candidate designed and implemented the three single-goal management strategies, as well as the Distance to Goals (Goal-DSS) meta-strategy. The candidate also contributed to the evaluation of the work and its write-up.

A Hierarchical, Topology-aware Approach to Dynamic VM Management

This work was primarily authored by the candidate, with assistance from fellow PhD Candidate Michael Tighe, who provided invaluable feedback and proof-reading.

Dynamic Management of Multi-VM Applications with Constraints

The candidate is the primary author of this work.

The first and only chapter of the Appendix, titled **DCSim**, describes the simulation framework of the same name that we developed over the years to evaluate algorithms and techniques for managing virtualized data centres. The simulator was primarily developed by fellow PhD Candidate Michael Tighe, with contributions from the candidate, mainly focused on the design of the first iteration of the simulator, the organization of the data centre in clusters and racks, as well as general design input throughout DCSim's lifetime. The work presented in this chapter is based on a publication co-authored by the candidate and expanded to cover the latest changes to the simulator.

Acknowledgements

"Gratitude is not only the greatest of virtues, but the parent of all others."
– Marcus Tullius Cicero

I would like to acknowledge here those people that have earned my outmost gratitude in these past years: my colleague and friend Michael Tighe, with whom I walked this arduous road; my friend Andrew Delong, a dedicated and honorable researcher like I know no other; my partner and editor Alex *"Julia"* Pontefract, who often moderates my harsh words; my family, a continent away, but always present; and finally the *"latinos"*, my safety net away from home.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computing today is in the midst of a paradigm shift. This shift is twofold: first, applications are no longer developed to be installed on individual users' computers, but to be installed on remote servers and accessed through the Internet; and second, the remote servers hosting these applications are typically not owned by the application providers themselves, but by data centre[1] providers renting their computing infrastructure on a pay-per-usage basis. This paradigm, known as *Cloud Computing*, has been widely adopted by the Information Technology (IT) world, and is expected to continue to grow in usage [1, 2, 3, 4]. For example, Cisco has predicted that by 2017 nearly two thirds of all workloads will be processed in the cloud [5].

Cloud Computing comes in three forms: *Software as a Service (SaaS)*, *Platform as a Service (PaaS)*, and *Infrastructure as a Service (IaaS)*. In SaaS, the (application) client pays for the use of an application. Clients do not purchase a license and install the application, but rather buy a subscription that enables them to access and use the application remotely through a web browser. Continuous application execution (i.e., availability), configuration and maintenance falls under the responsibility of the application provider, as well as the provision of any ancillary services, such as billing and security. Some well known examples of SaaS are Salesforce [6], Google Apps [7] and Basecamp [8].

In the PaaS flavour of Cloud Computing, the traded resource is a *software (or application) stack*. This resource consists of a set of software subsystems or components needed to perform a task without further external dependencies. A classic example of such a resource is the LAMP software stack, which consists of the GNU/Linux operating system, the Apache web server, the MySQL database management system, and the PHP programming language, and it is commonly used to deploy a web service. Examples of PaaS are Google App Engine [9], Microsoft Azure [10] and dotCloud Platform [11].

[1] A data centre can be thought of, in essence, as a collection of physical servers connected through a high speed, high bandwidth network.

Finally, in IaaS, clients rent computers – physical or virtual – and storage capacity from infrastructure providers. These computers are bare platforms consisting of little more than an operating system; clients have to build their desired software stack. In addition, clients are responsible for their computers' execution and maintenance, but in exchange for this increased level of responsibility they get more freedom and control over their platform than they would otherwise get under PaaS. Common examples of IaaS are Amazon Web Services [12], Google Compute Engine [9] and Microsoft Azure [10].

Regarding IaaS-type clouds, there exist three models to be aware of: *public*, *private* and *hybrid* clouds. This distinction does not respond to technical differences, but rather to business or functional ones; more specifically, to who has access to the cloud. A public cloud, for example, can be accessed by the general public. That is, a company owns the cloud and provides IaaS to the public. On the other hand, a private cloud can only be accessed by the company's employees. In this case, the cloud is for internal use only, not to be rented to the public. Lastly, a hybrid cloud represents a compromise between the previous two models: the company owns its own private cloud, but relies on public clouds for specific tasks or to handle unexpected increases in workload.

Our research focuses on the management of IaaS-type data centres (or clouds), either public or private. An IaaS-type data centre can be thought of, in essence, as a collection of physical servers connected through a high speed, high bandwidth network. Though real data centres are more complex than that, this simplified model will be used throughout this work.

Running a data centre incurs expenses (e.g., infrastructure acquisition, maintenance, administration, utilities), of which power consumption is one of the highest; studies have shown that data centres are high power consumers [13]. Given that physical servers are on average poorly utilized [14] and that a typical physical server can consume 50% to 70% of its peak power usage while idle [15], it is important for data centres to maximize their infrastructure's utilization, so as to make the most out of every powered on server and in doing so reduce power consumption. Thus, maximizing resource utilization is one of the main goals of a data centre management system.

One of the key features of Cloud Computing is the illusion of "infinite" resource availability, which guarantees clients that their workloads will be able to scale as much as needed any time. Data centres achieve this illusion through the use of *virtualization*. System virtualization is a software technique that enables the simultaneous execution of multiple computer systems in a single physical machine [16]. This feat is achieved by means of a hypervisor (or virtual machine monitor), which is a layer of software that runs directly on hardware. The hypervisor can create virtual machines (VM) within which operating systems can be installed, and allocates the underlying hardware resources to the VMs as an operating system would do (for

applications) in a non-virtualized environment. Virtual machines can be instantiated (and destroyed) quickly, enabling applications running in the data centre to scale their size on-demand (by adding or removing VMs) to meet their workloads' needs. It is this ability to host several computer systems per physical server paired with fast VM instantiation that enables data centres to simulate infinite resource capacity.

The use of virtualization results in an additional benefit: by co-locating multiple client workloads per host, data centres are able to increase their resource utilization. Moreover, by *oversubscribing* resources (i.e., promising more resources in total to the group of co-located VMs in a host than the host actually possesses), data centres can achieve a substantial increase in resource utilization. This strategy, however, poses a problem. Given the dynamic nature of workloads' resource demand [17], tightly loading a host can be problematic: if at any time the total resource demand of co-located VMs were to exceed the actual capacity of their host, one or more VMs would have their resource requirements unfulfilled, thus causing *Service Level Agreement (SLA)*[2] violations. Since these violations usually have a penalty attached to them (i.e., a monetary compensation to be paid by the data centre provider to the workload owner), it is important for data centres to minimize their occurrence. This requirement then defines another main goal in data centre management: minimizing SLA violations.

There may be many other management goals to consider, such as minimizing VM migrations or network overhead, as well as functional requirements for the management system itself, such as the ability to adapt to changing data centre conditions, pursue multiple management goals, or handle complex workload requirements (e.g., placement constraints, Quality of Service objectives), and also non-functional requirements, such as scalability, fault tolerance, and autonomy. This situation suggests that designing a management system for an IaaS-type data centre is not trivial and, as we will discuss below, there are many research challenges that need to be addressed.

In the next section, we provide some background by discussing the concept of *Virtual Machine Management* (with respect to data centres) and the operations it comprehends. In Section 1.2, we present four open research challenges in the area of dynamic management of virtualized data centres. Finally in Section 1.3, we present the research contributions of this work and provide an outline for this manuscript.

[2] A contract between the workload owner and the data centre provider specifying the conditions upon which the service is provided, the minimum service-level expectations (or guarantees), and the penalties (if any) associated to breaches in the contract.

1.1 Virtual Machine Management

A VM is a software implementation of a physical computer within which a computer system (i.e., operating system plus user space libraries and applications) can be deployed. VMs are subject to basic management operations, such as creation, suspension and termination. VMs can have their resource allocation modified dynamically, although the extent to which this operation is supported varies between systems virtualization technologies. In addition, VMs can be *live migrated*, that is, moved from one host to another (compatible) host, experiencing only minimal downtime in the process. These operations are considered *low-level* management operations.

VM management is typically associated with the execution of the following *high-level* management operations[3]:

1. **VM Placement.** When new application deployment requests arrive at the data centre, hosts must be selected to instantiate the VMs that compose the application.

2. **VM Relocation.** When a host is stressed (i.e., close to running out of spare resources to allocate to its VMs), one or more of its VMs must be (live) migrated away, so as to free resources. For each VM migration, a physical server must be found to become the new host of the migrated VM. We call this physical server the *target* host, while the stressed host is referred to as *source* host.

3. **VM Consolidation.** It is the process of relocating VMs in the data centre, so as to concentrate workloads into as few hosts as possible.

The VM Placement operation consists of mapping a set of VMs that are currently not instantiated in the data centre into a set of hosts. This can be done in several ways. The first and most basic approach is to allocate to each VM enough resources to satisfy their hosted workload's peak demand[4] and allocate those resources in an exclusive manner (i.e., resources allocated to one VM are not shared with any other VM co-located in the same host). The challenge of placing a VM would then be a matter of finding a host with enough spare resources to satisfy the VM's resource allocation (for peak demand). We will call this approach *Static Allocation Peak Demand*. (The term *static* is used to indicate that the mapping of VMs to hosts never changes.)

The benefit of the *Static Allocation Peak Demand* approach is that it is simple and straightforward, and that it satisfies the service guarantees in the workloads' SLAs by allocating

[3]Though resource reallocation (at host-level) and VM replication (i.e., the instantiation of a copy of a given VM) are sometimes brought under the umbrella of VM management, we will not consider them in this work.

[4]Assuming that we can learn in advance what the workload's peak demand is.

enough resources for the workloads to meet their peak demand. However, it is easy to see that this approach does not make for an efficient use of resources and can lead to high power consumption. As an example, consider the following scenario: there are two servers with a single quad-core CPU each and eight VMs to place, whose peak demand is one CPU core. According to this approach, we map four VMs into each server. Now, if the VMs are running at peak demand, the servers will present a resource utilization of 100%. However, if we assume an average demand of 50% for the VMs, which is considered to be a high average demand[5], the servers will only present a resource utilization of 50%. In other words, 50% of the potential of each server is wasted. A better placement would have seen all eight VMs mapped into one of the servers, keeping the second server powered off (or suspended), thus reducing power consumption.

An evident first improvement to the previous approach would then be to allocate VMs with enough resources to satisfy their hosted workload's average demand. However, if resources were allocated in an exclusive manner, the workloads would have their SLAs violated whenever their resource demand exceeded 50%. Therefore, resources should be allocated in a non-exclusive manner (i.e., resources are *oversubscribed*), meaning that workloads use only the amount of resources they need at each point in time, freeing the rest of their assigned (but unused) allocation for other co-located VMs to use if their current demand exceeds their given allocation. We will call this approach *Static Allocation Average Demand*.

This approach will work well most of the time, with the low demand of one VM allowing another VM to use resources in excess of its assigned allocation. This approach is more power efficient than *Static Allocation Peak Demand* and likely satisfies the service guarantees in the workloads' SLAs to a high degree. However, a high degree of satisfaction does not mean total satisfaction, and the data centre would incur SLA penalties in those occasions in which service guarantees are not met. As an example, consider the scenario described above, though now all eight VMs are mapped into one server. This arrangement will work well as long as the demand of each VM stays at or below 50%, or if any increase in resource demand is matched by a corresponding decrease in resource demand of the same or greater magnitude. However, the moment this equilibrium is broken, total demand (of the VMs) will exceed total available capacity (of the host) and SLA violations will ensue.

There is therefore a need for an approach that maps VMs into hosts, leveraging resource oversubscription and dynamic resource allocation to grant each VM the resources it needs at each point in time, and that is able to handle those instances in which the combined demand of co-located VMs exceeds the resource capacity of the host – we refer to these instances as *stress situations*. This latter problem could be dealt with by migrating one (or more) of the co-

[5]Data centre servers are estimated to see an average utilization of 5% to 20%. [14]

located VMs to a different host, thus freeing resources in the stressed host (to be allocated to the remaining VMs), and allowing the migrated VM(s) to continue running on their new host. The migrations could be performed *live* (i.e., without stopping the VMs[6]), causing minimal downtime to the VMs' workloads. We will call this approach *Dynamic Allocation*, to indicate that the mapping of VMs to hosts can be modified at runtime.

The Dynamic Allocation approach includes then not only VM Placement, but also VM Relocation. This approach addresses two challenges: first, selecting hosts in which to place incoming VMs (i.e., VMs not yet instantiated in the data centre), and second, selecting VMs to migrate away from stressed hosts and selecting *target* hosts in which to place those migrated VMs. In both situations, the set of available hosts includes hosts already loaded with VMs and also *empty* hosts (which may be powered on or not). This form of management provides much flexibility, but comes with an associated cost in terms of resource utilization overhead for the hosts involved in VM migrations and service disruption for the migrated VMs.

One disadvantage of Dynamic Allocation as discussed so far is that by relocating VMs in response to stress situations, this approach spreads load across the data centre, resulting after some time in servers hosting only a few VMs, thus lowering overall resource utilization. This issue can be addressed by performing VM Consolidation as part of Dynamic Allocation. VM Consolidation is a re-mapping process that aims to place all VMs in the data centre into as few hosts as possible. This process has an associated cost in terms of VM migrations, but it increases overall resource utilization and may empty hosts that can then be suspended (or powered off) to save power. In this way, Dynamic Allocation becomes a VM management approach consisting of three processes: VM Placement, VM Relocation and VM Consolidation.

Given the nature of our target environment (i.e., an IaaS-type data centre), where the number of client workloads is constantly changing and so is the resource demand of the workloads themselves [17], we focus our research on dynamic approaches to VM management.

1.2 Research Challenges

In this section, we describe some of the open research challenges associated with dynamic VM management in data centres; namely, *Virtual Machine Relocation* (Section 1.2.1), *Multi-goal Management Strategies* (Section 1.2.2), *Management Systems' Architecture* (Section 1.2.3), and *Multi-VM Application Management* (Section 1.2.4).

[6] VMs are actually *frozen* for a short period of time during live migration to complete the copy of the VMs' memory footprint.It is this freeze time what causes the minimal downtime experienced by the VMs.

1.2.1 Virtual Machine Relocation

In dynamic VM management, VMs are mapped and re-mapped into hosts according to current data centre state. The VM Relocation operation is part of that process. The VM Relocation problem can be summarized as follows: given a set of stressed hosts and a set of non-stressed host, find a sequence of VM migrations from hosts in the first set into hosts in the second set, so that the stress situations are terminated.

Formal Problem Definition

Let $H = \{h_1, \ldots, h_x\}$ be a set of hosts and $R = \{cpu, mem, bw\}$ be the set of resources available in a host. For each host $h \in H$, C_h^r denotes the resource capacity of host h, that is, the total amount of resource $r \in R$ available in h.

Let $V = \{v_1, \ldots, v_x\}$ be a set of VMs. For each VM $v \in V$, C_v^r denotes the resource needs of VM v, and $S_v^r(t)$ denotes the resource usage of VM v at time t, so that:

$$0 < S_v^r(t) \leq C_v^r$$

VMs are mapped or placed into hosts, however, a VM can only be mapped into a single host at any given time. We use the following formulas to express that:

$$p_{v,h}(t) = \begin{cases} 1 & \text{if } v \in V \text{ is mapped into } h \in H \text{ at time } t \\ 0 & \text{otherwise} \end{cases}$$

$$\text{if } \exists\, h_i \in H\colon\ p_{v,h_i}(t) = 1 \Rightarrow \forall h_j \in H \backslash h_i \ \ p_{v,h_j}(t) = 0$$

There is an additional placement rule to prevent mapping more VMs into a host than the host can sustain given its resource capacity. However, we first need to define how to calculate the resource usage of a host, $S_h^r(t)$, and its resource utilization, $U_h^r(t)$:

$$S_h^r(t) = \sum_{v \in V} S_v^r(t) * p_{v,h}(t)$$

$$U_h^r(t) = \frac{S_h^r(t)}{C_h^r}\,, \quad U_h^r(t) \in [0, 1]$$

The placement rule mentioned above can now be expressed as follows:

$$\forall h \in H : S_h^r(t) \leq C_h^r$$

which is equivalent to the following expression:

$$\forall h \in H : U_h^r(t) \leq 1$$

Hosts can be in *active* or *suspended* state. Hosts in active state have at least one VM mapped into them; otherwise, they are suspended to conserve power.

$$active_h(t) = \begin{cases} 1 & \text{if } N_h(t) > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$suspended_h(t) = \neg active_h(t)$$

where $N_h(t)$ denotes the number of VMs mapped into host $h \in H$ at time t.

Active hosts are further classified as *stressed* or *non-stressed*. A host is resource stressed (or in a resource stress situation) when the hosted VMs' combined demand for that resource reaches or exceeds the total capacity of the host. In this situation, the host is unable to satisfy the resource demand of the hosted VMs, negatively affecting their performance. A host with enough resources to satisfy the VMs' combined demand may still be considered stressed if the utilization level is so high that the host is unable to accommodate any further increases in the VMs' resource demand. A resource utilization threshold $\tau \in [0, 1]$ is used to determine whether a host is stressed or not.[7]

$$stressed_h(t) = \begin{cases} 1 & \text{if } U_h^r(t) \geq \tau \\ 0 & \text{otherwise} \end{cases}$$

$$non - stressed_h(t) = active_h(t) \wedge \neg stressed_h(t)$$

VMs can be *migrated* (i.e., transferred) from one host to another. A VM migration m is denoted by the ternary $< h_i, v, h_j >$, where $h_i, h_j \in H$ and $v \in V$. h_i is referred to as the *source* host, $p_{v,h_i}(t) = 1$, and h_j is referred to as the *target* host, $p_{v,h_j}(t+1) = 1$.

The VM Relocation problem can be succinctly described as follows: given a set of stressed hosts, a set of non-stressed hosts and a set of suspended hosts, find a set of VM migrations that will terminate the stress situations. The problem can be formally described as finding a set of VM migrations M with the following property:

$$M = \{m\colon p_{v,h_i}(t) = 1 \wedge stressed_{h_i}(t) \wedge p_{v,h_j}(t+1) = 1 \wedge \neg stressed_{h_i}(t+1) \wedge \neg stressed_{h_j}(t+1)\}$$

[7] Threshold value would depend on the physical servers' resource capacity and the data centre's business objectives.

while at time $t + 1$ all constraints previously defined are still satisfied.

The VM Relocation problem presents similarities with the *bin packing problem*[8], which is known to be NP-hard [19]: there is a set of VMs with different resource needs (*a set of items with their corresponding weights*) that have to be mapped into non-stressed hosts (*packed into bins*) so as to maximize the overall resource utilization of the set of hosts (*so as to minimize the number of bins used*). Given this similarity and the fact that greedy heuristics for the bin packing problem are known to achieve near optimal solutions (one such heuristic being *First Fit Decreasing* (FFD) [20, 21]), research has been conducted applying greedy heuristics for the bin packing problem to the VM Relocation problem. Most frequently, the heuristic used is FFD. This heuristic sorts the items in the set in decreasing order by weight, and one by one places each item in order into the first bin in which they fit. Applied to the VM Relocation problem, this heuristic would minimize the number of active hosts in the data centre.

However, there is a problem with this approach. While the goal of the bin packing problem is to place the items in the set into the least number of bins, the VM Relocation problem may have other goals to consider (or even prioritize) other than minimizing the number of active hosts; e.g., minimizing the number of SLA violations, or minimizing the number of migrations needed to solve the problem. Therefore, a heuristic that works in one context may not perform equally well in a different context.

It is important to determine whether the order in which heuristics for the VM Relocation problem consider VMs and hosts for migration has any effect in the long-term management goals of a data centre, as reflected in diverse metrics such as power consumption, SLA violations, and number of migrations.

1.2.2 Multi-goal Management Strategies

The management operations introduced in Section 1.1 can be carried out in many ways. We use the term *management policy* (or simply *policy*) to refer to any such implementation of a management operation. A *management strategy* is a set of policies that determines how each of the main management operations are carried out across the data centre. In this way, a management strategy defines the behaviour of the data centre management system.

Management strategies are designed with a given purpose, a management goal to strive for. This goal can be any number of things. The two most commonly studied goals in the literature are: (i) minimizing power consumption; and (ii) minimizing SLA violations. However, these two goals are often in conflict. In order to minimize power consumption, load has to be consolidated into as few hosts as possible and every host without VMs assigned to it should

[8] Though differences exist that make VM Relocation more complex [18].

be powered off or suspended. This approach does save power, but results in active hosts with high resource utilization, which increases the risk of having stress situations (as a consequence of sudden increases in VMs' resource demand), which may result in increased SLA violations. Conversely, minimizing SLA violations requires load to be spread across the data centre, leaving active hosts with a significant amount of spare resources available to handle sudden spikes in demand. This approach does minimize the risk of having stress situations (and thus SLA violations), but it results in many hosts being active, thus increasing power consumption.

Designing a management strategy to achieve both of these goals is difficult, as improving performance towards one goal typically results in degradation of performance towards the other goal. Thus, the design of management strategies tends to focus on defining a single goal to pursue, or on defining several goals, but prioritizing them in such a way that one goal is considered the primary goal and all others are considered secondary. Designing dual- or multi-goal management strategies remains an open problem.

1.2.3 Management Systems' Architecture

As mentioned before, a data centre can be thought of, in essence, as a collection of hosts connected through a high-speed, high-bandwidth network. A management system is required to administer this infrastructure, mapping VMs to hosts and powering hosts on and off as needed. The architecture of the management system determines important properties of the system, such as scalability, quality of management decisions, and degree of management overhead. Different architectures offer different trade-offs between these properties.

Centralized systems consist of a single, high-level manager governing over a collection of low-level managers, each assigned a managed element. Low-level managers gather monitoring data and send it to the central manager, which results in high bandwidth usage. The central manager has a global view of the system, thus being able to make management decisions that span the whole data centre. However, this concentration of management power is also the limiting factor on the scalability of the system.

Distributed systems, on the other hand, consist solely of a collection of low-level managers, each making decisions for its corresponding managed element. There is little (if any) information sharing between managers, so the data flow is minimal, but that also results in managers having a very limited (or localized) view of the system. The level of independence of the managers provides unlimited scalability to the system.

Decentralized (or hierarchical) systems try to strike a balance between the previous two architecture models, achieving better scalability than a centralized system, without having the limited system view of a distributed system. A hierarchical system is organized in levels. The

lowest level consists of managers assigned each to a managed element, and each level above that consists of managers assigned responsibility over a disjointed subset of the managers in the level below. Data is collected at every level of the hierarchy, but only a summary of the data makes it to the next level up, reducing overall data flow. Managers possess data about every manager and managed element enclosed in their area, but not beyond it. This enables them to make informed decisions concerning the data centre area under their control, and increases scalability by decentralizing management responsibilities.

Choosing what architecture model to use when designing a data centre management system is heavily influenced by the properties sought after in the management system. In addition, since the architecture of the systems affects the design of management policies and strategies, choosing a model requires careful consideration. Centralized systems (and centralized algorithms) tend to be easier to design than distributed ones, and it is easier as well to analyze and undestand their runtime behaviour, given that all management decisions have origin in a single manager, are based on a global view of the system, and encompass the whole data centre. However, given the large-scale of today's data centres (scale that is only expected to grow), systems' scalability is becoming increasingly important [22].

1.2.4 Multi-VM Application Management

Today, most applications deployed in the Cloud consist of multiple components. These components run in their own dedicated servers, but work together to provide a service. A prevalent example of this type of applications is a multi-tier web application, consisting of Web, Application and Database tiers, each tier hosted in a separate VM [23].

While managing single-VM applications in the data centre is a well-studied problem, managing multi-VM applications is not. Multi-VM applications can sometimes have, in addition to their basic resource requirements, a special set of requirements referred to as *placement constraints*. For example, an application may require some of its components to be co-located in the same host (or the same hardware rack or cluster) for performance reasons, while another application may require its components to be placed far apart in the data centre for high availability purposes.

These placement constraints significantly add to the complexity of VM management. Now, the management system not only has to address the challenge of meeting each application's resource requirements, but it also has to be cognizant of the location of every component of a multi-VM application and the rules that regulate their position with respect to each other.

1.3 Thesis Outline

In Chapter 2, we address the VM Relocation problem (Section 1.2.1). We hypothesize that the order in which VMs and hosts are considered for migration has an impact on data centre performance (as reflected by performance metrics such as power consumption and SLA violations), which is relevant given that some outcomes may be more desirable than others depending on data centres' current state or business goals. We design and evaluate a set of First Fit-based policies for VM Relocation, which prioritize VMs and hosts in different ways. We present simulation results showing that the policies achieve different levels of performance in various metrics according to data centre state (i.e., load-level).

Chapter 3 focuses on the issue of Multi-goal Management Strategies (Section 1.2.2). In order to pursue two opposing goals (namely, minimizing SLA violations and minimizing power consumption) with one management strategy, we propose the use of a *meta-strategy* that dynamically switches between two single-goal management strategies according to data centre state. We hypothesize that this arrangement may achieve a better balance between the two desired goals. We present three methods to dynamically switch management strategies, and evaluate these methods through simulation. Results suggest that dynamic strategy switching offers overall improved performance over single management strategies.

We address the issue of Management Systems' Architecture (Section 1.2.3) in Chapter 4, by proposing a hierarchical, topology-aware management system for large-scale data centres. We leverage the topology of the data centre network to create a hierarchy of (autonomic) managers. We define a set of aggregate metrics at various levels in the hierarchy to convey system state information to higher management levels, and define managers' responsibilities and interactions – expressed in the form of management policies and strategies. By making the hierarchy topology-aware, we limit the flow of management data across the data centre. We hypothesize that this organization will result in a more efficient use of the data centre, greatly reducing network traffic. The system is evaluated through simulation. The results confirm that management data flow across the data centre is greatly reduced.

In Chapter 5, we present our work on Multi-VM Application Management (Section 1.2.4). We propose two management strategies designed to manage multi-VM applications with placement constraints in data centres. These management strategies aim to increase infrastructure's utilization (and thus reduce power consumption) while at the same time minimizing SLA violations. While one of the management strategies enforces placement constraints at all times, the other allows for a temporary violation of placement constraints. The two management strategies are evaluated through simulation in multiple scenarios. Results suggest that both strategies are able to achieve high levels of infrastructure utilization and SLA achievement,

while satisfying applications' placement constraints, and that temporarily violating constraints does not provide any advantage, though it does add cost (in terms of degradation of application performance).

In Chapter 6, we present a discussion of our work, where we summarize our findings and consider their implication. We also enumerate a series of limitations that affect our work, and suggest potential directions of future work. We end the chapter by stating our conclusions.

Finally in Appendix A, we present DCSim (Data Centre Simulator), a tool that we developed to simulate virtualized data centres operating as Infrastructure as a Service (IaaS) clouds. The simulator is flexible and can be easily extended, enabling researchers to quickly evaluate data centre management algorithms and techniques at a speed and scale not possible in real environments. This tool is free and open source software (FOSS) and its code is available on GitHub [24].

Bibliography

[1] "IT cloud services at the crossroads: How IaaS/PaaS/SaaS business models are evolving," International Data Corporation, Mar. 2013.

[2] "Forecast: Public cloud services, worldwide, 2012-2018, 1q14 update," Gartner, Inc., Mar. 2014.

[3] "IBM global technology outlook: Cloud 2014: A more disruptive phase," International Business Machines Corp., 2014.

[4] "RightScale 2014 state of the cloud report," RightScale, Inc., 2014.

[5] "Cisco global cloud index: Forecast and methodology, 2012-2017," White Paper, Cisco, 2012.

[6] (2014) Salesforce.com. salesforce.com. [Online]. Available: http://www.salesforce.com/

[7] (2014) Google Apps for Business. Google Inc. [Online]. Available: http://www.google.com/enterprise/apps/business/

[8] (2014) Basecamp. Basecamp, LLC. [Online]. Available: https://basecamp.com/

[9] (2014) Google App Engine. Google Inc. [Online]. Available: https://cloud.google.com/products/app-engine/

[10] (2014) Microsoft Azure. Microsoft Corporation. [Online]. Available: https://azure.microsoft.com/

[11] (2014) dotCloud Platform. cloudControl, Inc. [Online]. Available: https://www.dotcloud.com/

[12] (2014) Amazon Web Services. Amazon.com, Inc. [Online]. Available: http://aws.amazon.com/

[13] R. Brown *et al.*, "Report to congress on server and data center energy efficiency: Public law 109-431," 2008.

[14] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[15] L. A. Barroso and U. Holzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[16] J. E. Smith and R. Nair, "The architecture of virtual machines," *Computer*, vol. 38, no. 5, pp. 32–38, 2005.

[17] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations," in *IM Proceedings, 2007 IEEE/IFIP Int. Symp. on*, 2007, pp. 119–128.

[18] C. Hyser, B. Mckee, R. Gardner, and B. J. Watson, "Autonomic virtual machine placement in the data center," HP Laboratories, Tech. Rep. HPL-2007-189, Dec. 2007.

[19] M. Gabay and S. Zaourar, "Variable size vector bin packing heuristics - Application to the machine reassignment problem," 2013. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00868016

[20] M. Yue, "A simple proof of the inequality FFD (L) $\leq$ 11/9 OPT (L) + 1, $\forall$ L for the FFD bin-packing algorithm," *Acta Mathematicae Applicatae Sinica (English Series)*, vol. 7, pp. 321–331, 1991, 10.1007/BF02009683. [Online]. Available: http://dx.doi.org/10.1007/BF02009683

[21] G. Dósa, "The tight bound of first fit decreasing bin-packing algorithm is FFD (I)$\leq$11/9 OPT (I)+6/9," in *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, ser. Lecture Notes in Computer Science, B. Chen, M. Paterson, and G. Zhang, Eds. Springer Berlin / Heidelberg, 2007, vol. 4614, pp. 1–11, 10.1007/978-3-540-74450-4_1. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74450-4_1

[22] H. Moens, J. Famaey, S. Latre, B. Dhoedt, and F. De Turck, "Design and evaluation of a hierarchical application placement algorithm in large scale clouds," in *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*. IEEE, 2011, pp. 137–144.

[23] (2014) AWS Reference Architectures. Amazon Web Services, Inc. [Online]. Available: http://aws.amazon.com/architecture/

[24] (2014) DCSim on GitHub. Distributed and Grid Systems (DiGS). [Online]. Available: https://github.com/digs-uwo/dcsim

Chapter 2

An Analysis of First Fit Heuristics for VM Relocation

In dynamic VM management, VMs are mapped and re-mapped into hosts according to current data centre state. The VM Relocation operation is part of that process and its responsibility is to deal with *stress situations*. The VM Relocation problem can be summarized as follows: given a set of stressed hosts and a set of non-stressed host, find a sequence of VM migrations from hosts in the first set into hosts in the second set, so that the stress situations are terminated.

The VM Relocation problem presents similarities with the bin packing problem[1], which is known to be NP-hard [3]. There is a set of VMs with varying resource demands (items with their associated weights) that have to be mapped into hosts (packed into bins) in such a way that the overall resource utilization is maximized (the number of bins used is minimized). It has been shown that greedy heuristics for the bin packing problem have near optimal solutions, one such heuristic being *First Fit Decreasing* (FFD) [4, 5]. This heuristic consists of sorting the items in decreasing order by *weight*, and sequentially placing them into the first bin in which they fit. Applied in the context of virtualized data centres, this heuristic would result in minimizing the number of active hosts.

However, while the goal of the bin packing problem is to place a set of items (VMs) in the least number of bins (hosts), the VM Relocation problem may have additional goals for which to strive, such as minimizing the number of Service Level Agreement (SLA) violations or minimizing the number of migrations used. These goals may even take priority over maximizing resource utilization, depending on the situation or the data centre's business goals. For that reason, an heuristic that works well under one situation or business strategy, may not work well under a different situation.

[0]This chapter is based on work published in [1].

[1]Though differences exist that make VM Relocation more complex [2].

The order in which a heuristic for the VM Relocation problem considers VMs and hosts can affect the final set of migrations produced. Therefore, different heuristics that prioritize VMs and/or hosts based on different criteria may produce better assignments (and achieve better long-term outcomes) when considering their particular goals, associated with the data centre's management strategy.

In this work, we evaluate and compare a set of relocation policies, all variations on a *First Fit* heuristic [6], where the difference between policies lies in the order in which VMs and hosts are considered as migration candidates and migration targets, respectively.

The remainder of this chapter is organized as follows: Section 2.1 discusses related work in the area, Section 2.2 describes our assumptions and the relocation policies designed, Section 2.3 presents experiments and results, and Section 2.4 presents conclusions and future work.

2.1 Related Work

The management of virtualized data centres presents several research challenges. The most basic or fundamental of them are how to implement the core management operations of VM Placement, VM Relocation and VM Consolidation (introduced in Section 1.1), and also how to perform resource reallocation at host-level (that is, dynamically reallocating resources to the VMs in a host, so as to meet their resource demand). In this section, we present some works in the literature that address these problems.

2.1.1 VM Placement

The problem of mapping VMs into hosts is referred to in the literature as VM Placement, static server consolidation, or simply resource allocation. Its difficulty resides in finding a mapping that ensures individual VMs are allocated enough resources to satisfy their demand, while at the same time maximizing resource utilization.

Bobroff et al. [7] implemented a First Fit Decreasing heuristic that periodically re-calculated the mapping of a set of VMs into a set of empty hosts, based on the VMs' forecasted demand. Their goal was to minimize the average number of active hosts, while providing probabilistic SLA guarantees for the VMs.

Cardosa et al. [8] developed a VM placement algorithm that leveraged the CPU allocation features `min` , `max` , and `shares` present in modern hypervisors to negotiate the tradeoff between VMs' performance (tied to CPU allocation) and overall power consumption.

Speitkamp and Bichler [9] proposed a static server consolidation approach that combined data analysis to characterize variations in real-world workload traces, and an LP-relaxation-

based heuristic to optimally map VMs into hosts.

Stillwell et al. [10] worked on mapping a set of static workloads into hosts, optimizing the VMs' resource allocation for performance and fairness. They proposed and evaluated an extensive set of algorithms, finally identifying a vector (or multi-dimensional) bin packing algorithm that achieved close to optimal solutions to the problem.

2.1.2 VM Relocation

The problem of determining which VM to migrate and to which host to migrate it when a stress situation occurs has also been studied. Khanna et al. [11] addressed the VM Relocation problem by developing a mathematical optimization model to select VMs and hosts for migration. They implemented an heuristic that sorted VMs in increasing order by CPU and memory utilization - to minimize migration costs - and sorted hosts in increasing order by residual capacity (i.e., available resources) - to maximize resource utilization. The authors did not consider any additional sorting strategies, nor the impact of their heuristic in the number of migrations issued.

Wood et al. [12] implemented in their management system Sandpiper a First Fit Decreasing heuristic that sorted hosts in increasing order by resource utilization. Their goal was not, however, to evaluate the efficiency of their heuristic, but to compare two mechanisms for VM monitoring.

Gmach et al. [13] developed a fuzzy logic-based controller as part of their proposed management system. The controller not only issued migrations when a host became stressed (VM Relocation), but also when a host became underutilized (Dynamic Server Consolidation). The target host for a migration was the least loaded host with enough resources to fit the VM. However, it was not clear how the VMs were selected for migration. The purpose of this work was to present the management system as a whole, thus the controller was not described in detail.

Beloglazov and Buyya [14] proposed algorithms and heuristics to deal both with stressed and underutilized hosts. In the case of stressed hosts, their algorithm selected for migration the VMs with the least memory allocation and selected as many VMs as needed to bring the hosts' CPU utilization down to an acceptable level. When hosts were underutilized, all their VMs were selected for migration. The target host selection was based on a Best Fit Decreasing heuristic: the migrating VMs were sorted in decreasing order by CPU utilization and placed in the host that provided the least increase in power consumption due to the allocation of the incoming VM. The use of a Best Fit approach makes target sorting strategies irrelevant at the cost of having to consider every single host in the data centre for each VM that has to be placed.

2.1.3 Resource Reallocation

On the topic of dynamic VM provisioning, Gmach et al. [15] developed and compared four different resource reallocation policies, with the goal of minimizing the number of active hosts while providing Quality of Service to the VMs. The authors concluded that work-conserving policies with dynamically set weights offered the best results.

Pokluda et al. [16] focused on dynamic memory management. They developed a policy-based framework that would reallocate memory among the VMs co-located in a host to meet the VMs' changing demand. The system could also trigger VM migrations were the local memory adjustments insufficient to deal with the stress situation.

2.2 VM Relocation

In this section we describe the assumptions and limitations we defined to work with the VM Relocation problem (Subsection 2.2.1), and the relocation policies we designed to test our hypothesis (Subsection 2.2.2).

2.2.1 Assumptions and Limitations

In order to limit the scope of the VM Relocation problem as defined in Section 1.2.1, we have defined the following assumptions and limitations:

1. **The set of physical servers is homogeneous (in terms of resource capacity and power consumption).**

2. **Hosts' utilization level calculation is based only on CPU utilization.**

3. **The virtual machines are independent from each other (i.e., no dependencies).**

Item 1 proposes that all physical servers be homogeneous in terms of their resource capacity and power consumption (effectively making all hosts equal under these two factors). Therefore, when selecting source or target hosts, resource capacity (as in total number of CPU cores, total memory, etc) and power consumption can be safely ignored in the decision process.

Item 2 proposes that CPU be the only resource considered when determining the utilization level of a host. Therefore, whether a host is stressed or not will depend solely on its CPU utilization level. A host could have all its memory allocated and in use, and still not be considered stressed (i.e., memory stress is not considered a stress situation). Nonetheless, memory and bandwidth needs are still taken into account when trying to place or relocate a VM, although

simply as a last minute check that the target host has enough resources to meet the needs of the incoming VM. In this study, both memory and bandwidth will be constant and the same values for all VMs, effectively reducing the number of dimensions in the problem.

Item 3 proposes that there be no dependencies between VMs. Dependencies may impose restrictions of the form *"virtual machine x must to be hosted together with virtual machine y"* or *"virtual machines x and y cannot be hosted in the same physical server (or even rack or cluster)."* Removing these restrictions simplifies the problem by making it easier to select VMs to relocate and target hosts for those relocations.

Although these assumptions and limitations do simplify the VM Relocation problem, the remaining problem is still challenging. Many factors still remain to be considered and prioritized when searching for relocations, such as size (CPU utilization level) of the VMs, utilization level of the source and target hosts before and after relocation, relocation overhead on the source and target hosts, number of concurrent relocations per host, and number of active hosts in the data centre.

2.2.2 Policies

The relocation policies are based on a First Fit heuristic [6]. This heuristic consists of taking the items that need to be placed, one at a time, and assigning them to the first bin in which they fit. The relocation policies differ from each other in the order in which they consider the VMs (items) and the target hosts (bins) for their migrations.

In order to distinguish between hosts, we use the active host classification introduced in Section 1.2.1, though refined as follows. Active hosts are classified as *stressed*, *partially-utilized* or *underutilized*, based on their resource utilization level. Two thresholds define the division between categories: $stress$ and $minUsage$ (high and low level thresholds, respectively). Since we only consider CPU to calculate the utilization level of a host (as per Item 2 in Section 2.2.1), the thresholds are renamed as $stress_{CPU}$ and $minUsage_{CPU}$. The categories are formalized as follows:

- **Underutilized**: hosts with CPU utilization in the range $[0, minUsage_{CPU}]$;
- **Partially-utilized**: hosts with CPU utilization in the range $(minUsage_{CPU}, stress_{CPU}]$;
- **Stressed**: hosts with CPU utilization in the range $(stress_{CPU}, 1]$.

Suspended hosts form a category of their own, **Suspended**.

The relocation policies have to find sets of VM relocations. Each relocation consists of three elements: a *source host*, a *candidate VM*, and a *target host*. Source hosts are selected

from among **Stressed** hosts, candidate VMs are selected from among the VMs hosted in the selected source hosts, and target hosts are selected from among **Suspended**, **Underutilized** and **Partially-utilized** hosts.

```
1:  classify hosts in categories
2:  filter and sort source hosts
3:  sort target hosts
4:  for each source host do
5:      filter and sort candidate VMs
6:      for each candidate VM do
7:          for each target host do
8:              try to migrate VM to host
9:              if success then
10:                 record migration
11:                 move on to next source host
12:             end if
13:         end for
14:     end for
15: end for
16: return list of migrations
```

Algorithm 1: First Fit heuristic.

As mentioned before, the relocation policies are based on a First Fit heuristic (see Algorithm 1). The first step (line 1) in the heuristic classifies the hosts in their respective categories. The second step (line 2) removes from among the source hosts those hosts that are currently involved in relocations. Once the relocations are completed, the utilization level of the hosts will change and the stress situations may cease to exist. If the stress situations persist, they will be addressed in the next cycle. The remaining hosts are then sorted in decreasing order by CPU utilization. This sorting process is common to all policies (except *Random*, introduced below).

The third step (line 3) sorts the target hosts. We defined three different ways of combining the categories from which a target host is to be selected:

1. **Increasing**: sort Partially-utilized and Underutilized hosts in increasing order by CPU utilization. Consider Underutilized, Partially-utilized and Suspended hosts, in that order;

2. **Decreasing**: sort Partially-utilized and Underutilized hosts in decreasing order by CPU utilization. Consider Partially-utilized, Underutilized and Suspended hosts, in that order;

3. **Mixed**: sort Partially-utilized hosts in increasing order by CPU utilization and Underutilized hosts in decreasing order by CPU utilization. Consider Partially-utilized, Underutilized and Suspended hosts, in that order.

Policies	VM Sorting	Target Sorting
FFDI	**D**ecreasing	**I**ncreasing
FFDD	**D**ecreasing	**D**ecreasing
FFDM	**D**ecreasing	**M**ixed
FFII	**I**ncreasing	**I**ncreasing
FFID	**I**ncreasing	**D**ecreasing
FFIM	**I**ncreasing	**M**ixed

Table 2.1: VM Relocation Policies

The fifth step (line 5) filters and sorts the VMs hosted in the selected source host. Only VMs with equal or greater CPU load than the CPU load by which the host is stressed are considered as migration candidates. If no VM satisfies this criteria, all the VMs are considered. This filtering is done to improve the chances of selecting for migration a VM that will bring the source host's utilization level below the $stress_{CPU}$ threshold. After filtering, the VMs are sorted in one of two ways:

1. **Decreasing**: sort VMs in decreasing order by CPU load;[2]
2. **Increasing**: sort VMs in increasing order by CPU load.

The eighth step (line 8) checks whether the selected target host has enough spare resources to accept the migration of the selected candidate VM. If this check turns positive, the migration is recorded (line 10) and the process moves on to the next source host (line 11). Be it noted that at most one migration is issued for each source host.

By exhaustively combining the two sorting strategies for migration candidates and the three sorting strategies for target hosts, we obtain six different relocation policies (shown in Table 2.1).

We implemented a seventh policy, *Random*, which randomizes the order in which source hosts, migration candidates and target hosts are considered. This policy was added as a benchmark.

A final note regarding host selection in these policies: a host cannot be selected both as migration source and target. Hosts can be source for one migration at a time, but can be target for several migrations concurrently (as long as they have enough resources to satisfy the resource requirements of all incoming VMs).

[2]Be it noted that CPU load is not the same as CPU utilization. The first term refers to the actual number of CPU shares in use, while the second term refers to the ratio of CPU shares in use over total allocated CPU shares.

2.3 Evaluation

We used the simulation framework DCSim [17, 18] to conduct our experiments. Though simulations do not capture the whole complexity of real-world environments, they allow for replicable testing environments (very important when comparing algorithms), enable large-scale experiments, and significantly reduce the real-time duration of experiments.

Subsection 2.3.1 discusses the configuration of the simulation environment, Subsection 2.3.2 describes the experiments' design, Subsection 2.3.3 lists and explains the reported metrics, and Subsection 2.3.4 presents the results.

2.3.1 Simulator Configuration

DCSim (Data Centre Simulator) is an extensible data centre simulation framework, designed to provide an easy framework for developing and experimenting with data centre management techniques and algorithms [18]. Several components in DCSim have to be extended or implemented to evaluate alternative management policies. We worked with DCSim, version 12.01, in these experiments.

In DCSim, a data centre has a VM Placement Policy to map VMs into hosts at creation time, a VM Consolidation Policy to perform server consolidation, and a VM Relocation Policy to relocate VMs away from stressed hosts.

In our experiments we use a simple VM Consolidation Policy that migrates VMs from underutilized hosts to partially-utilized or higher-loaded underutilized hosts. The use of a VM Consolidation policy was necessary to allow for stress situations to continue occurring throughout the simulation. Any solution to the VM Relocation problem, by definition, attempts to dissipate stress situations by migrating VMs away from stressed hosts. Unless the number of hosts in the set is so small that VMs are simply migrated in circles, the VM Relocation policy will unavoidably spread the load (VMs) across the data centre. Server consolidation policies exist to remedy this situation, and in this particular case, to better allow us to compare the VM Relocation policies introduced in Section 2.2.2.

Policies are also used in DCSim to manage the allocation (and reallocation) of resources among the hosted VMs. Static Resource Managers allocate resources to VMs upon placement in the host and do not alter the allocation at any further point. Dynamic Resource Managers, on the other hand, perform an initial allocation based on the resource request of the VMs and then dynamically adjust the allocation to match the VMs' resource utilization.

In our experiments, we use an oracle-like CPU Manager. This manager is a special kind of dynamic CPU Manager with perfect knowledge about the VMs' resource needs at any given time and can therefore compute a perfect allocation. The use of this CPU Manager allows us to

perform resource reallocation in real time to meet the resource needs of the VMs without over-provisioning them. Also, the oracle nature of this manager helps us to prevent ineffective CPU allocation policies from affecting (and obscuring) the results of the VM Relocation policies under study.

2.3.2 Experimental Design

We designed four sets of seven experiments to evaluate the relocation policies. Every experiment in a set used the same data centre configuration, but a different VM Relocation policy. Each experiment lasted 10 simulation days and was repeated 5 times. Results were averaged.

For the first set of experiments, the data centre was configured with 100 hosts, and was set to run the VM Relocation process every 10 minutes and the VM Consolidation process every 24 hours. The data centre hosted 300 VMs, initially provisioned to meet their peak demand. (When provisioned for peak demand, a host can only fit 3 VMs, so 300 is the maximum number of VMs that can be placed in this data centre.)

The second, third and fourth sets of experiments utilized the same data centre configuration, but hosted 400, 452[3] and 500 VMs, respectively, which were initially provisioned to meet their average demand.

The hosts in these experiments had 4 CPU cores with 1000 CPU shares each (for a total of 4000 CPU shares per host) and 8GB of memory. Bandwidth and storage were not considered in these experiments. Each host was restricted to at most two concurrent outgoing migrations. The $stress_{CPU}$ threshold was set at 85% and the $minUsage_{CPU}$ threshold at 50%. There were 400 CPU shares reserved by the CPU Manager to cover migration overhead (200 shares * 2 concurrent migrations), thus effectively reducing the CPU shares available for allocation to 3600.

Each VM was attached to a separate instance of one of four Application Models, with an equal number of VMs attached to each type of model. All four Application Models worked in a similar way, but used different workload traces as input. The first two used the ClarkNet HTTP trace and the EPA HTTP trace available from the Internet Traffic Archive [19]. The second two used the Google Cluster Data trace [20]. Google Cluster Data is divided into four job types, from which we extracted the first two as individual workload traces. For each workload trace, we used the total number of incoming requests in 100 second intervals as the workload level, which was normalized to the range [0, 1]. Workload traces that were shorter than the simulation time were looped, and each virtual machine started the trace at a randomly chosen offset time value to differentiate them from each other. The virtual machines were placed in

[3] 452 VMs were used instead of 450, so as to have an equal number of VMs associated to each of the four available Application Models.

the hosts by the VM Placement Policy in a randomized order, with a different random order for each execution of the simulation.

VMs were limited to a maximum CPU need of 1000 shares (equal to 1 core of the hosts described above), calculated using the formula $100 + 900 * workload$. In the experiment in which VMs' initial allocation was set to *peak*, the allocation was 1000 CPU shares. When it was set to *average*, the allocation was the average workload level of the trace to which the VM was attached. Memory needs for the VMs were fixed at 1GB, and bandwidth and storage were not considered.

2.3.3 Metrics

In order to compare the relocation policies, we used the following set of metrics provided by DCSim:

- **Active Hosts (Hosts):** The average number of hosts powered on. The higher the value, the more physical hosts are being used to run the workload.
- **Average Active Host Utilization (Host Util.):** The average CPU utilization of all powered on hosts. The higher the value, the more efficiently resources are being used.
- **Power Consumption (Power):** Power consumption is calculated for each host, and the total kilowatt-hours consumed during the simulation are reported. Suspended hosts are assumed to have negligible power consumption. Hosts' power consumption is modelled as a linear function that defines a fixed power consumption when the CPU is idle and scales linearly with CPU utilization:

$$250\ watts + 250\ watts * host.utilization \tag{2.1}$$

- **Dropped Requests (SLA).** The average number of requests that had to be dropped due to VMs not having their resource demand met. It is used as a measure of SLA violation.
- **Number of Migrations (Migs.):** The number of VM migrations triggered during the simulation. Typically, a lower value is preferable, since fewer migrations means less network overhead.

2.3.4 Results and Discussion

The results of the experiments are presented in Tables 2.2, 2.3, 2.4 and 2.5. Results were averaged over 5 repetitions of each experiment. The first day of simulation time was discarded

to eliminate the influence of the initial VM placement and allow the system to stabilize before collecting metrics. Figure 2.1 shows the results of the second experiment.

Policies	Hosts	Host Util.	Power	SLA	Migs.
FFDI	70.2	61.5%	6,142.9	0.6%	480
FFDD	64.8	63.5%	6,002.7	0.9%	681
FFDM	65.6	62.8%	6,046.5	0.8%	577
FFII	79.7	63.3%	6,029.7	0.6%	530
FFID	61.5	67.6%	5,715.3	1.7%	1,755
FFIM	64.3	64.3%	5,953.5	0.8%	766
Random	72.9	53.3%	6,760.6	0.7%	909

Table 2.2: Exp. 1 – 300 VMs

Policies	Hosts	Host Util.	Power	SLA	Migs.
FFDI	88.6	65.3%	7,879.6	0.6%	478
FFDD	84.7	67.5%	7,696.4	0.8%	737
FFDM	86.4	66.4%	7,781.8	0.7%	585
FFII	82.9	68.8%	7,610.8	0.7%	814
FFID	78.0	72.5%	7,300.6	1.7%	2,315
FFIM	84.7	67.8%	7,684.2	0.7%	826
Random	93.0	59.2%	8,397.6	0.7%	1,015

Table 2.3: Exp. 2 – 400 VMs

Policies	Hosts	Host Util.	Power	SLA	Migs.
FFDI	99.6	65.1%	8,917.2	0.6%	513
FFDD	95.6	67.4%	8,699.5	0.8%	865
FFDM	97.8	66.3%	8,810.5	0.7%	669
FFII	92.8	69.4%	8,550.0	0.8%	960
FFID	88.4	72.3%	8,262.0	1.6%	2,527
FFIM	95.6	67.8%	8,682.7	0.7%	929
Random	99.3	65.2%	8,917.2	0.6%	652

Table 2.4: Exp. 3 – 452 VMs

Policy FFDI used consistently the most hosts, consumed the most power, and achieved the lowest host utilization. On the other hand, it also achieved the best service (i.e., lowest percentage of dropped requests) and required the least number of migrations.

Policy FFID behaved opposite to FFDI. It achieved consistently the highest host utilization and used the least hosts; consequently, it also reported the lowest power consumption. However, it paid the price dropping the most requests and triggering the most migrations.

Policies	Hosts	Host Util.	Power	SLA	Migs.
FFDI	100.0	71.4%	9259.0	1.1%	1,242
FFDD	100.0	71.4%	9255.4	1.2%	1,540
FFDM	100.0	71.4%	9255.9	1.2%	832
FFII	100.0	71.4%	9258.2	1.1%	1,665
FFID	97.5	72.6%	9110.4	1.7%	3,114
FFIM	100.0	71.3%	9254.5	1.2%	1,371
Random	100.0	71.4%	9258.4	1.1%	1,324

Table 2.5: Exp. 4 – 500 VMs

Policies FFII, FFIM and FFDD achieved average results (between FFDI and FFID), not excelling under any particular metric. FFDM followed one step behind in terms of using less hosts, less power and achieving higher utilization, but it performed fewer migrations in doing so.

The Random policy behaved – predictably – random. It did not excel in any category, but it did not trigger the most migrations and it did achieve good service.

This results show that no one policy scored best in every metric, which is to be expected when considering conflicting goals, such as minimizing the number of active hosts and minimizing the number of dropped requests.

The experiments also show that the policies succeeded to different extents depending on the scenario and the metrics observed. For example, policy FFDM used consistently more hosts and consumed more power than policies FFII and FFIM. However, in the last experiment, FFDM achieved similar results under both metrics, while issueing 50-60% the number of migrations.

With regard to the sorting strategies used, we can make a few observations. By sorting VMs in decreasing order (policies FFD), the first VMs to be considered for migration (and likely migrated) are high-load VMs. Consequently, source hosts see a significant drop in their CPU utilization (up to 25% if the VM was using one whole core) and more hosts are activated due to the difficulty of finding target hosts that can receive a high-load VM.

On the other hand, by sorting VMs in increasing order (policies FFI), low-load VMs[4] are considered first. These VMs have a smaller impact on the source hosts' utilization (thus keeping utilization high) and its likely easier to find a partially-utilized host that can receive them. The downside of this situation is that since source hosts are kept highly utilized and partially-utilized hosts see an increase in their utilization with the arrival of migrated VMs, both source and target hosts are at a higher risk of becoming stressed again in the near future, therefore

[4]Though in most cases, these VMs would have enough load to terminate the stress situation; otherwise, they would have been filtered out.

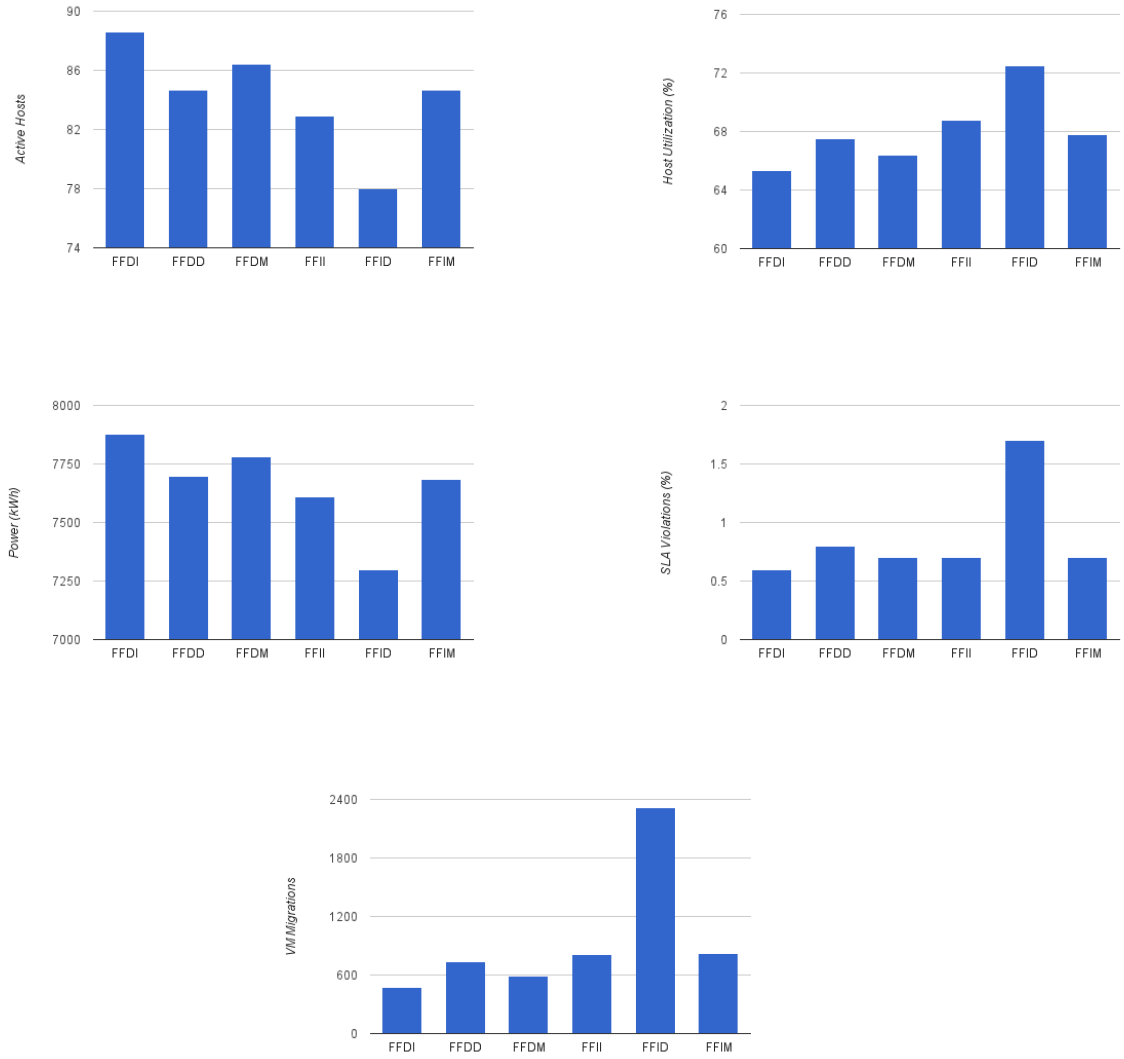

Figure 2.1: Exp. 2 – 400 VMs

causing more migrations in the long run.

From the viewpoint of the target hosts sorting strategies, an anomaly is observed. Policies FFDI and FFII are the most liberal, looking for suitable target hosts among the lowest utilized hosts first, while policies FFDD and FFID are the most conservative, trying to migrate VMs to highly utilized hosts first. Finally, policies FFDM and FFIM represent a compromise, searching for suitable hosts among the lowest partially-utilized hosts first, thus trying to keep an overall high utilization without being too conservative. This observation appears to hold true for the FFD policies and FFID. However, policies FFII and FFIM seem to have switched roles, the former using less hosts and achieving higher utilization than the latter.

Another interesting observation is that the highest host utilization achieved in these experiments was only 72.6% when the stress threshold (and therefore the utilization goal) was set to 85%. A careful analysis of the restrictions and resource allocation policies in place inform us that the maximum host utilization that the management policies could have achieved is 76.5%. This conclusion follows from two premises. First, that the maximum CPU shares available for allocation in a host is 3600 (as indicated in Section 2.3.2). Second, that the CPU Manager Oracle allocates CPU to VMs in such a way that the VMs' utilization is 85% of their CPU allocation. Therefore, even if a host has 3600 CPU shares allocated, only 3060 would normally be in use, which is equivalent to a utilization of 76.5%. Therefore, even if utilization could actually increase up to 90% (i.e., using the 3600 CPU shares allocated), it would be completely dependent upon the workload of the hosted VMs; the management policies would not be able to force the utilization higher by placing more VMs into the host.

This work presents a series of shortcomings. Some of these shortcomings were purposefully introduced as assumptions or limitations to limit the scope of the VM Relocation problem (see Section 2.2.1). Other shortcomings were imposed by the simulation framework.[5]

First of all, the cost (or overhead) of VM migrations. Migrations impose an overhead in both source and target hosts. In this experiments, only CPU overhead was accounted for (the CPU Manager considered a fixed overhead of 200 CPU shares per migration in both hosts). Memory and bandwidth overhead was ignored at the hosts, and bandwidth overhead on the networking infrastructure was equally neglected, assuming a dedicated network for migrations.

Second, no delay or cost was associated to the host activation and suspension processes, enabling migrations to suspended hosts to start immediately.

Third, DCSim provided an advantage that would not hold true in a real-world scenario: the data centre management system (and policies) possessed perfect knowledge of the utilization levels of hosts and VMs at all times. In a real system, status information would be sent periodically from hosts to management system, with two consequences: first, the information would

[5]Several of these shortcomings have been addressed already in a newer version of DCSim.

not be always up-to-date, and second, the status update messages would introduce an overhead in the networking infrastructure, whose extent would depend on the amount of information transferred and the frequency of such transmissions.

Finally, we do not compare our policies directly with First Fit heuristics proposed by other researchers. However, it is easy to recognize similarities between some of our heuristics and those of other authors: for example, policy FFID matches the heuristic described in [11], while policy FFDI matches the heuristic described in [12] (see Section 2.1 for more details on these heuristics).

2.4 Conclusions and Future Work

The goal of this work was to determine whether changing the order in which a relocation policy considered the candidate VMs and target hosts for migration resulted in better outcomes (in terms of data centre performance metrics), depending on the situation or the data centre's business goals.

The experimental results (see Section 2.3) showed that no one policy scored best in every metric, and that the policies succeeded to different extents depending on the scenario and the metrics observed.

These results demonstrate that one single policy will not satisfy all goals and that by tweaking the VM and host sorting strategies better trade-offs can be achieved. Most importantly, the results suggest that dynamically switching between policies may offer better overall results.

As future work, we plan to focus on removing the assumptions and limitations introduced in Section 2.2.1, as well as the shortcomings described in Section 2.3.4, and evaluating the merits of dynamically switching between policies. Another avenue of research could be to look into fuzzy logic-based controllers for the tasks of target host and candidate VM selection.

Bibliography

[1] G. Keller, M. Tighe, H. Lutfiyya, and M. Bauer, "An analysis of first fit heuristics for the virtual machine relocation problem," in *Network and Service Management (CNSM), 2012 8th International Conference on*. IEEE, Oct. 2012, pp. 406–413.

[2] C. Hyser, B. Mckee, R. Gardner, and B. J. Watson, "Autonomic virtual machine placement in the data center," HP Laboratories, Tech. Rep. HPL-2007-189, Dec. 2007.

[3] M. Gabay and S. Zaourar, "Variable size vector bin packing heuristics - Application to the machine reassignment problem," 2013. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00868016

[4] M. Yue, "A simple proof of the inequality FFD (L) ≤ 11/9 OPT (L) + 1, ∀ L for the FFD bin-packing algorithm," *Acta Mathematicae Applicatae Sinica (English Series)*, vol. 7, pp. 321–331, 1991, 10.1007/BF02009683. [Online]. Available: http://dx.doi.org/10.1007/BF02009683

[5] G. Dósa, "The tight bound of first fit decreasing bin-packing algorithm is FFD (I)≤11/9 OPT (I)+6/9," in *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, ser. Lecture Notes in Computer Science, B. Chen, M. Paterson, and G. Zhang, Eds. Springer Berlin / Heidelberg, 2007, vol. 4614, pp. 1–11, 10.1007/978-3-540-74450-4_1. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74450-4_1

[6] M. Y. Kao, *Encyclopedia of Algorithms*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[7] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations," in *IM Proceedings, 2007 IEEE/IFIP Int. Symp. on*, 2007, pp. 119–128.

[8] M. Cardosa, M. R. Korupolu, and A. Singh, "Shares and utilities based power consolidation in virtualized server environments," in *IM Proceedings, 2009 IEEE/IFIP Int. Symp. on*, 2009.

[9] B. Speitkamp and M. Bichler, "A mathematical programming approach for server consolidation problems in virtualized data centers," *IEEE TSC*, vol. 3, no. 4, pp. 266 –278, 2010.

[10] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource allocation algorithms for virtualized service hosting platforms," *J. Parallel Distrib. Comput.*, vol. 70, no. 9, pp. 962–974, Sep. 2010.

[11] G. Khanna, K. Beaty, G. Kar, and A. Kochut, "Application performance management in virtualized server environments," in *NOMS Proceedings, 2006 IEEE/IFIP*, 2006.

[12] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *NSDI Proceedings, 4th Symp. on*, Cambridge, MA, USA, Apr. 2007, pp. 229–242.

[13] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper, "An integrated approach to resource pool management: Policies, efficiency and quality metrics," in *38th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, June 2008.

[14] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency Computat.: Pract. Exper.*, pp. 1–24, 2011.

[15] D. Gmach, J. Rolia, and L. Cherkasova, "Satisfying service level objectives in a self-managing resource pool," HP Laboratories Palo Alto, Tech. Rep. HPL-2009-170, Jul. 2009.

[16] A. Pokluda, G. Keller, and H. Lutfiyya, "Managing dynamic memory allocations in a cloud through golondrina," in *Systems and Virtualization Management (SVM), 2010 4th International DMTF Academic Alliance Workshop on*. IEEE, Oct. 2010.

[17] M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya, "DCSim: A data centre simulation tool for evaluating dynamic virtualized resource management," in *Network and Service Management (CNSM), 2012 8th International Conference on*. IEEE, Oct. 2012, pp. 385–392.

[18] (2014) DCSim on GitHub. Distributed and Grid Systems (DiGS). [Online]. Available: https://github.com/digs-uwo/dcsim

[19] (2014) The Internet Traffic Archive. [Online]. Available: http://ita.ee.lbl.gov/

[20] (2014) Google Cluster Data. Google Inc. [Online]. Available: http://code.google.com/p/googleclusterdata/

Chapter 3

Switching Data Centre Management Strategies at Runtime

As discussed in Chapter 1, managing a virtualized data centre is a complex task and there are multiple goals that a management system could pursue. A *management strategy* is a set of policies that determines how each of the main VM Management operations are carried out across the data centre. In this way, a management strategy defines the behaviour of the data centre management system.

In this work, we focus on two of the most commonly studied management goals in the area: (i) minimizing power consumption; and (ii) minimizing Service Level Agreement (SLA) violations. These goals are often in conflict. Minimizing power consumption is usually approached by reducing the number of hosts in use (and thus powered on). This is achieved by placing as many VMs on a single host as possible. However, sudden increases in workload are more likely to result in a shortage of resources (i.e., a stress situation) and therefore lead to a high number of SLA violations. Conversely, minimizing SLA violations typically requires VMs to be spread across more hosts, often each having a significant amount of unused resources available to handle spikes in demand. This, however, results in higher power consumption. Designing a management strategy to achieve both of these goals is therefore difficult, as improving performance towards one goal typically results in degradation of performance towards the other. Design of management strategies often focuses on achieving a single goal, or on prioritizing goals such that a single goal is considered the primary goal and others are considered secondary, e.g., [2, 3, 4, 5].

Within a dynamic environment there may be times when one management strategy is more appropriate than another. We propose an approach to dynamically switch between management strategies where each has a primary focus on a single goal; in this case, one strategy to minimize

[0]This chapter is based on work published in [1].

SLA violations and another to minimize power consumption. By doing so, we aim to achieve better performance in attaining both goals. The main contributions of this work are three novel methods of dynamically switching between single-goal management strategies, and a method of comparing the performance of strategies that aim to achieve more than one goal.

The remainder of this chapter is organized as follows: Section 3.1 reviews related work in this area, Sections 3.2 and 3.3 describe the management strategies and strategy switching approaches we explored, respectively. Section 3.4 presents experiments and evaluation, and finally, Section 3.5 concludes and discusses future work.

3.1 Related Work

Several works in the area have approached the problem of placing VMs in a data centre statically. These works assume VMs' service demand to be static, or to be variable but exhibit a periodic cycle, and perform a one-time mapping (i.e., a static allocation) of a set of VMs into a set of empty hosts by different methods. Cardosa et al. [6] relied on a Best Fit heuristic that leveraged the CPU allocation features `min`, `max`, and `shares` present in modern hypervisors. Speitkamp and Bichler [7] showed that the problem could be reduced to the multidimensional bin packing problem, which is NP-hard, and then proposed a Linear Programming relaxation-based heuristic, combined with data analysis to characterize variations in workload traces. Stillwell et al. [8] framed the problem as a constrained optimization problem and formulated it as a Mixed Integer Linear Program, which required exponential time to calculate an exact solution. They then proposed and evaluated an extensive set of algorithms, finally settling for a vector bin packing algorithm that achieved close to optimal solutions to the problem.

Other static approaches have considered variable and aperiodic service demand, but have addressed the problem by statically allocating VMs to hosts and periodically re-calculating the complete mapping. Such is the case of Bobroff et al. [3], who used a First Fit Decreasing heuristic for the task. In most cases, these static approaches aimed to minimize the average number of active hosts, or to increase host or data centre utilization, and in that way minimize power consumption.

Research in dynamic management of virtualized data centres focuses on one of the three management operations (i.e., VM Placement, VM Relocation or VM Consolidation), or a combination of them. Most solutions focus on pursuing a single goal, or on pursuing multiple goals, but prioritizing one goal over all others. Wood et al. [4] addressed the VM Relocation problem using a First Fit Decreasing heuristic that spread load across hosts and thus aimed to reduce SLA violations. Keller et al. [9] studied variants of a First Fit heuristic to address the VM Relocation problem, showing that the order in which VMs and hosts are considered for migration

impacts data centre performance metrics, such as power consumption and SLA violations.

Several works address VM Relocation and VM Consolidation together. In most cases, they focus primarily on minimizing power consumption and then on minimizing SLA violations. Khanna et al. [2] implemented a First Fit heuristic that migrated the least loaded VMs (in terms of CPU and memory usage) into the highest loaded hosts. Verma et al. [5] relied on a First Fit Decreasing heuristic that placed VMs in the most power efficient servers first. Beloglazov and Buyya [10] proposed a Best Fit Decreasing heuristic that selected for migration the VMs with the smallest memory footprint and placed them in the host that provided the least increase in power consumption.

The main difference between this work and the cited ones is that the solution proposed here focuses on pursuing at the same time two different (and opposing) goals – namely minimizing SLA violations and minimizing power consumption – without prioritizing one goal over the other.

As indicated in [8], optimization models have limited applicability in this context, given that calculating a solution can take several hours – e.g., [8] notes that finding an exact solution to a small problem instance took under an hour. For this reason, most works in the area turn to heuristics. As shown above, much of the existing work on dynamic management uses some form of First Fit heuristics, though occasionally alternatives are proposed, such as fuzzy logic-based controllers, e.g., [11].

Some heuristics make use of cost functions derived from optimization models to solve a problem. One approach is to combine multiple objectives (i.e., minimizing power consumption and minimizing SLA violations) into a single objective function, assigning weights to each objective, and use said function to choose VMs and target hosts for migration. The challenge with this approach resides in choosing appropriate weights. If both objectives are equally important, then the same weights could be applied to each objective. However, the effectiveness of such an approach depends on the level of load in the data centre, which changes dynamically. For example, when there is little load, using equal weights may result in higher power consumption than needed. In that situation, a heavier weight should be associated to the power objective, so as to consolidate load and reduce power consumption. This suggests that the weights should be variable and adaptable to current data centre state.

Alternatively, cost could be assigned to SLA violations and power consumption and the objective would be to minimize cost. This approach presents two challenges. First, assigning constant cost would be ineffective. For example, the cost of power consumption varies as the day progresses. Second, reducing SLA violations to a monetary cost may result on the system causing many SLA violations (if power costs are high), introducing the possibility of alienating clients who may take their business elsewhere.

3.2 Management Strategies

This section presents three management strategies that are representative of those found in the literature. The strategies presented assume frequent monitoring. Calculations are performed on monitored values over a sliding window of time, referred to as the *monitoring window*.

3.2.1 Terminology

This section presents the terms and metrics used in the description of management strategies.

- **SLA Violation:** An SLA violation occurs when resources required by a VM are not available to it, as this situation leads to a degradation in performance. The percentage of required CPU not available in the SLA violation is denoted by s.

- **Data Centre Utilization:** The overall utilization of the data centre is calculated as the percentage of total CPU capacity in the data centre that is currently in use.

- **CPU Shares:** We quantify the capacity of a CPU as *CPU shares*, where each CPU core has a specific number of shares which represents its computing power. In our work, the number of shares assigned to each core is based on its frequency, with 1GHz = 1000 shares.

- **Power Efficiency:** For a host, h, the power efficiency, p_h, is the amount of processing being performed per watt of power. This is measured in CPU-shares-per-watt (cpu/watt). The calculation of the power efficiency of a single host is presented in Equation 3.1:

$$p_h = \frac{cpuInUse_h}{powerConsumption_h} \tag{3.1}$$

  where $cpuInUse_h$ is the number of CPU shares currently in use across all cores in the host, and $powerConsumption_h$ is the current power consumption in watts of the host. As an active host machine consumes a significant amount of power even when under little or no CPU load (i.e., very low power efficiency) increased host utilization corresponds with increased power efficiency for that host. This metric is used to calculate the power efficiency for the entire data centre, p_{dc}, calculated as

$$p_{dc} = \frac{\sum_{h \in hosts} cpuInUse_h}{\sum_{h \in hosts} powerConsumption_h} \tag{3.2}$$

such that *hosts* is the collection of all hosts in the data centre.

- **Maximum Power Efficiency:** This metric represents the best power efficiency a host can achieve, calculated as the power efficiency of the host at maximum CPU utilization.
- **Optimal Power Efficiency:** Optimal Power Efficiency, p_{dc}^{opt}, represents the best possible power efficiency achievable at the data centre level, given the current workload and set of host machines available. The best power efficiency would be achieved by placing VMs in such a way that each host is 100% utilized, with the most power efficient hosts being filled first. We first calculate the total CPU-in-use across the data centre. We order the available hosts by maximum power efficiency, and allocate the CPU-in-use to hosts such that each host is allocated 100% of its CPU capacity. We calculate p_{dc}^{opt} to be the power efficiency of the data centre given this allocation.

3.2.2 Host Classification

Each time a management operation takes place, hosts are classified into categories based on their power state: `on`, `suspended` or `off`. Powered on hosts are further classified as *stressed*, *partially-utilized* or *underutilized*, based on their CPU utilization level. Hosts may transition between these states based either on changes in workload of the hosted VMs, or migrations performed by the management operations. Two threshold values are used for categorization: $stress_{CPU}$ and $minUsage_{CPU}$. Classification is based on the hosts average CPU utilization over the last monitoring window (measurements collected every 2 minutes over a sliding window of size 5). Categories are defined as follows:

- **Stressed**: hosts with average CPU utilization in the range $(stress_{CPU}, 1]$;
- **Partially-utilized**: hosts with average CPU utilization in the range $(minUsage_{CPU}, stress_{CPU}]$;
- **Underutilized**: hosts with average CPU utilization in the range $[0, minUsage_{CPU}]$;
- **Empty**: hosts that do not currently have any VM assigned to them. Hosts in `suspended` or `off` power state are included in this category.

It should be noted that VM Relocation policies make the determination of whether a host is stressed in a slightly different way based on how the most recent measurements of host utilization are considered. This *stress check* may mark as partially-utilized a host that is considered stressed under this classification, or vice versa. More information on this issue is presented in Section 3.2.3.

3.2.3 Power and SLA Strategies

Power and SLA are *single-goal* strategies, which means that all management decisions are geared towards achieving a single, primary goal. Single-goal strategies may pursue secondary goals, but always give them lower priority than the primary goal.

In the next subsections, we will describe the VM Placement, VM Relocation and VM Consolidation policies that comprise these two strategies. Much of the existing work on dynamic management uses some form of First Fit heuristics. The work described in Stillwell et al. [8] (for static workloads) and Keller et al. (for dynamic workloads) [9] studied variants of First Fit heuristics to find that they work best in practice. The Power and SLA strategies are based on such heuristics and are representative of other work on dynamic resource management.

The strategies use different values for the $stress_{CPU}$ threshold: the Power strategy uses 95% and the SLA strategy used 85%. The lower threshold for the SLA strategy allows for additional resources to be available for workload variations. Both strategies use the $minUsage_{CPU}$ threshold at 60%.

VM Placement

This management operation runs each time a new VM creation request is received, and selects a host in which to instantiate the VM. The VM Placement policy for the Power strategy (see Algorithm 2) first classifies *hosts* in their respective categories (line 3): stressed (z), partially-utilized (p), underutilized (u) and empty (e). The policy then sorts each host category (lines 4-5): p and u are sorted in decreasing order first by maximum power efficiency and then by CPU utilization, and e is sorted in decreasing order first by maximum power efficiency and then by power state. This sorting method ensures that the placement focuses on power efficiency over any other considerations. The policy then builds a list of *target* hosts by concatenating p', u' and e' (line 6). Finally, following a First Fit approach, the policy assigns the VM to the first host in *target* with enough capacity to host the VM (lines 7-12). The method `hasCapacity(VM)` checks whether the host can meet the resource requirements indicated in the VM creation request (line 8) without the host becoming stressed.

The VM Placement policy for the SLA strategy differs from the Power strategy's policy in the way p and u are sorted: p is sorted in increasing order first by CPU utilization and then by maximum power efficiency and u is sorted in decreasing order first by CPU utilization and then by maximum power efficiency. This sorting method ensures that the placement focuses on spreading load across the hosts, leaving spare resources to handle spikes in resource demand, over any other considerations.

```
1: Input: VM
2: Output: –
3: z, p, u, e = classifyHosts(hosts)
4: p′, u′ = sortPowerEffThenUtil(p, u)
5: e′ = sortPowerEffThenState(e)
6: target = concatenate(p′, u′, e′)
7: for host in target do
8:    if host.hasCapacity(VM) then
9:       host.deploy(VM)
10:      break
11:   end if
12: end for
```

Algorithm 2: Power strategy's VM Placement policy.

VM Relocation

This management operation runs frequently over short intervals of time, so as to detect stress situations as soon as possible. For both strategies, the interval is set to 10 minutes. This operation determines which hosts are experiencing a stress situation and attempts to resolve the situations by migrating one VM from each stressed host to a non-stressed host. The VM Relocation policy for the Power strategy (see Algorithm 3) first classifies *hosts* in their respective categories (line 1), performing a *stress check* on all hosts to determine whether or not they are stressed. The policy determines that a host is stressed if its CPU utilization has remained above the $stress_{CPU}$ threshold all of the time over the last CPU load monitoring window. The resulting host categories are: stressed (z), partially-utilized (p), underutilized (u) and empty (e). The policy then sorts each host category (line 2-4): z is sorted in decreasing order by CPU utilization, p and u are sorted in decreasing order first by maximum power efficiency and then by CPU utilization, and e is sorted in decreasing order first by maximum power efficiency and then by power state. The policy then builds a list of *target* hosts by concatenating p', u' and e' (line 6). Following a First Fit heuristic, the policy selects one VM from each host h in *source* and a corresponding *host* in *target* to which to migrate the VM (lines 7-22). For each host h in *source*, the policy filters out the VMs with less CPU load than the CPU load by which h is stressed and sorts the remaining VMs in increasing order by CPU load (line 8). If the list of remaining VMs is empty, all VMs are considered and sorted in decreasing order by CPU load. The method `migrate(h,VM,host)` initiates a migration (line 13).

The VM Relocation policy for the SLA strategy differs from the Power strategy's policy in the way p and u are sorted: p is sorted in increasing order first by CPU utilization and then by maximum power efficiency and u is sorted in decreasing order first by CPU utilization and then by maximum power efficiency. In addition, the policy performs a different stress check: a

```
1:  z, p, u, e = classifyHosts(hosts)
2:  z' = sortUtil(z)
3:  p', u' = sortPowerEffThenUtil(p, u)
4:  e' = sortPowerEffThenState(e)
5:  source = z'
6:  target = concatenate(p', u', e')
7:  for h in source do
8:      vms = filterAndSort(h.vms)
9:      success = FALSE
10:     for VM in vms do
11:         for host in target do
12:             if host.hasCapacity(VM) then
13:                 migrate(h, VM, host)
14:                 success = TRUE
15:                 break
16:             end if
17:         end for
18:         if success then
19:             break
20:         end if
21:     end for
22: end for
```

Algorithm 3: Power strategy's VM Relocation policy.

host is stressed if its last two monitored CPU load values are above the $stress_{CPU}$ threshold or its average CPU utilization over the last CPU load monitoring window exceeds $stress_{CPU}$.

VM Consolidation

This management operation runs less frequently than VM Relocation, given that its purpose is to consolidate the load that VM Placement and VM Relocation have spread across the data centre. The interval is set to 4 hours for the SLA strategy (as shown in [11]) and to 1 hour for the Power strategy, aiming in the latter case to achieve and sustain a higher degree of consolidation. This operation consolidates load in the data centre by migrating VMs away from underutilized hosts (and suspending or powering them off) and into partially-utilized hosts. The VM Consolidation policy for the Power strategy (see Algorithm 4) first classifies *hosts* in their respective categories (line 1): stressed (z), partially-utilized (p), underutilized (u), and empty (e), and powers off e (line 2). The policy then sorts p and u in decreasing order first by maximum power efficiency and then by CPU utilization (line 3) and builds a list of *target* hosts by concatenating p' and u' (line 4). Afterwards, the policy sorts u again, but this time in increasing order first by power efficiency and then by CPU utilization, and uses that list as *source* (line 5). Following a First Fit heuristic, the policy attempts to vacate every host h in *source* by migrating their VMs into *hosts* in *target* (lines 6-16). For each host h in *source*, the policy sorts its VMs in decreasing order first by overall resource capacity (memory, number of CPU cores, core capacity) and then by CPU load (line 7). Given that *source* and *target* are not disjunct, measures have to be taken to avoid using a host both as source and target for migrations.

The VM Consolidation policy for the SLA strategy differs from the Power strategy's policy in the way p and u are sorted: first, p is sorted in increasing order first by CPU utilization and then by maximum power efficiency and u is sorted in decreasing order first by CPU utilization and then by maximum power efficiency, and then, u is sorted in increasing order by CPU utilization.

3.2.4 Hybrid Strategy

We designed a *dual-goal* strategy as a combination of the Power and SLA strategies; the Hybrid strategy consists of the VM Placement and VM Relocation policies of the SLA strategy and the VM Consolidation policy of the Power strategy. Furthermore, the stress check performed by the VM Relocation policy represents a compromise between the checks of SLA and Power: it determines that a host is stressed only if its average CPU utilization over the last monitoring window exceeds the $stress_{CPU}$ threshold. The thresholds $stress_{CPU}$ and $minUsage_{CPU}$ were set

```
1: z, p, u, e = classifyHosts(hosts)
2: powerOff(e)
3: p', u' = sortPowerEffThenUtil(p, u)
4: target = concatenate(p', u')
5: source = sortPowerEffThenUtil(u)
6: for h in source do
7:    vms = sort(h.vms)
8:    for VM in vms do
9:       for host in target do
10:         if host.hasCapacity(VM) then
11:            migrate(h, VM, host)
12:            break
13:         end if
14:      end for
15:   end for
16: end for
```

Algorithm 4: Power strategy's VM Consolidation policy.

to 90% and 60% respectively.

3.3 Dynamic Strategy Switching

Dynamic Strategy Switching (DSS) refers to changing between strategies at run-time in response to changing data centre state. DSS periodically performs an evaluation of data centre metrics monitored between executions to determine if the strategy currently in use (the *active strategy*) should be changed. In this section, we present three different DSS meta-strategies.

3.3.1 SP-DSS

The SLA-Power Thresholds (SP-DSS) meta-strategy uses the SLA violation (s) and power efficiency ratio (per) metrics to evaluate whether the active strategy should be switched. The power efficiency ratio is calculated as the ratio of optimal power efficiency (p_{dc}^{opt}) to current power efficiency (p_{dc}) over the last hour. A strategy switch is triggered when the metric related to the goal of the active strategy (i.e., s for the SLA strategy, per for the Power strategy) is below a normal (i.e., acceptable) threshold (s_{norm} or per_{norm}), while the metric related to the inactive strategy exceeds a high threshold (s_{high} or per_{high}). See Algorithm 5 for details. Switching strategies in this manner allows the data centre to respond to a situation in which performance in one metric has deteriorated, by activating the strategy that focuses on optimizing it.

1: **if** $activeStrategy == Power\_Strategy$ **then**
2: **if** $per < per_{norm}$ & $s > s_{high}$ **then**
3: Switch to $SLA\_Strategy$
4: **end if**
5: **else if** $activeStrategy == SLA\_Strategy$ **then**
6: **if** $s < s_{norm}$ & $per > per_{high}$ **then**
7: Switch to $Power\_Strategy$
8: **end if**
9: **end if**

Algorithm 5: SP-DSS Switching Conditions.

3.3.2 Goal-DSS

We define two goals, $s = 0\%$ and $p_{dc} = p_{dc}^{opt}$, to evaluate performance with respect to the s and p metrics. By calculating the distance to these goals, it is possible to determine towards which goal the system is performing worst and thus switch to the strategy that would improve achievement of that goal. The Distance to Goals (Goal-DSS) meta-strategy is based on this principle. Evaluating whether or not the active strategy should be switched requires the calculation of two metrics that represent the distance to those goals. These are presented in Equations 3.3 and 3.4.

$$slaDist = \frac{s}{s_{worst}} \tag{3.3}$$

where s is the current SLA violation percentage and s_{worst} is an operator-defined parameter that indicates the worst acceptable SLA violation percentage, and

$$powerDist = 1 - \frac{p_{dc} - p_{worst}}{p_{dc}^{opt} - p_{worst}} \tag{3.4}$$

$$p_{worst} = p_{dc}^{opt} * p_C \tag{3.5}$$

where p_{worst} is the worst acceptable power efficiency, and p_C is an operator-defined parameter that indicates how large a deviation from the optimal power efficiency is acceptable.

Calculating the distances in this manner is necessary in order to equate values of s with values of p, based on the parameters s_{worst} and p_C. These two values are considered equivalent in terms of distance to their respective goals. At each iteration of the strategy switching mechanism, the strategy for which the corresponding distance is greater is selected to become active. See Algorithm 6 for details.

```
1: if activeStrategy == Power_Strategy then
2:     if slaDist > powerDist then
3:         Switch to SLA_Strategy
4:     end if
5: else if activeStrategy == SLA_Strategy then
6:     if powerDist > slaDist then
7:         Switch to Power_Strategy
8:     end if
9: end if
```

Algorithm 6: Goal-DSS Switching Conditions.

3.3.3 Util-DSS

Through experimentation, two key situations in which one strategy had an advantage over the other became apparent. When overall data centre utilization is growing, increasing the stress on host machines, the SLA strategy is more effective as it places greater emphasis on preventing SLA violations. Conversely, when utilization is decreasing or stable, thus increasing the likelihood of hosts becoming underutilized, the Power strategy is more effective as it can quickly make changes to conserve power. Data centre utilization is defined as the percentage of CPU shares in use across the entire data centre.

The Data Centre Utilization Trends (Util-DSS) meta-strategy is designed to exploit this pattern. It uses the rate of change of overall data centre utilization, m, to determine appropriate times to switch strategies. Measurements of the overall data centre utilization are taken at regular intervals. Linear regression over the last n data centre utilization measurements provides the rate of change, m, over a window of time. The value m_{SLA} defines a threshold for m over which a switch is made to the SLA strategy. Similarly, the value m_{Power} defines a threshold for m under which the Power strategy is set to be active. See Algorithm 7 for details.

```
1: if activeStrategy == Power_Strategy then
2:     if m > m_sla then
3:         Switch to SLA_Strategy
4:     end if
5: else if activeStrategy == SLA_Strategy then
6:     if m < m_power then
7:         Switch to Power_Strategy
8:     end if
9: end if
```

Algorithm 7: Util-DSS Switching Conditions.

3.4 Evaluation

This section presents our experimental approach, results and discussion.

3.4.1 Strategy Evaluation and Comparison

In order to evaluate the effectiveness of the strategies, two metrics are used: power efficiency (p) and SLA violation (s). Comparing strategies based only on the use of these two metrics is problematic. If one strategy were to perform well with respect to SLA violations at the expense of power, and another performed well with respect to power at the expense of SLA violations, it is difficult to conclude which strategy is preferable. The decision depends in part upon the relative change in each area as well as the importance placed on each metric by the data centre operators based on their business objectives, the relative costs of power and SLA violations and the potential for lost revenue due to poor application behaviour.

In order to determine whether DSS can offer improved results over a single strategy, we propose a method of evaluating the performance of a strategy based on experimental results. We use the SLA and Power strategies as benchmarks, with their SLA violation and power efficiency results serving as baseline measurements with which to evaluate other strategies. The SLA strategy provides the bounds for the best SLA violation value ($s_{best} = s_{SLA}$) and the worst power efficiency ($p_{worst} = p_{SLA}$), while the Power strategy provides the worst SLA violation ($s_{worst} = s_{Power}$) and best power efficiency ($p_{best} = p_{Power}$). Values from a candidate strategy, i, are then normalized using these bounds to produce the normalized vector, v_i, represented by $[s_{norm}, p_{norm}]$. The values s_{norm} and p_{norm} are defined in Equation 3.6.

$$\begin{aligned} s_{norm} &= \frac{(s_i - s_{best})}{(s_{worst} - s_{best})} \\ p_{norm} &= \frac{(p_{best} - p_i)}{(p_{best} - p_{worst})} \\ v_i &= (s_{norm}, p_{norm}) \end{aligned} \tag{3.6}$$

where p_{norm} is the normalized power efficiency and s_{norm} is the normalized SLA violation.

Note that $p_{best} > p_{worst}$, but $s_{best} < s_{worst}$, so the normalization equations differ to reflect this. Once we have the normalized vector, v_i, we calculate its L^2-norm, $|v_i|$, and use this as an overall score ($score_i$) for the candidate strategy.

$$score_i = |v_i| = \sqrt{s_{norm}^2 + p_{norm}^2} \tag{3.7}$$

where a smaller score is considered better, as it represents a smaller distance to the *best* bounds of each metric (defined by s_{best} and p_{best}). The SLA and Power strategies always achieve a score of 1 by definition, as they achieve the *best* score in one metric and the *worst* in the other.

Scores less than 1 indicate that overall performance of the candidate strategy has improved relative to the baseline strategies.

Note that this score is only valid for a single experiment in which all factors except for the active management strategy remain constant. In our work, we vary the workload pattern experienced by the data centre. As such, the baselines and score must be calculated separately for each workload pattern. The average final score across all experiments can then be used to evaluate the strategy. We use this method to evaluate and compare competing management strategies.

3.4.2 Experimental Setup

We conduct our experimentation by simulation using DCSim [12]. Our simulated data centre consists of 200 host machines, of which there are an equal number of two types: *small* and *large*. The *small* host is modelled after the HP ProLiant DL380G5, with 2 dual-core 3GHz CPUs and 8 GB of memory. The *large* host is modelled after the HP ProLiant DL160G5, with 2 quad-core 2.5GHz CPUs and 16GB of memory. Cores in the *large* host have 2500 CPU shares, and cores in the *small* host have 3000 CPU shares. The power consumption of both hosts is calculated using results from the SPECPower benchmark [13]. The maximum power efficiency of the *large* host (85.84 cpu/watt) is roughly double that of the *small* host (46.51 cpu/watt).

Three VM sizes are created: *small* requires 1 virtual core with at least 1500 CPU shares and 512MB of memory, *medium* requires 1 virtual core with at least 2500 CPU shares and 512MB of memory, and *large* requires 2 virtual cores with at least 2500 CPU shares each and 1GB of memory.

Hosts are modelled to use a work-conserving CPU scheduler, as available in major virtualization technologies. That is, any CPU shares not used by a VM can be used by another. No maximum cap on CPU is set for VMs. In the case of CPU contention, VMs are assigned shares in a round-robin fashion until all shares have been allocated. No dynamic voltage and frequency scaling (DVFS) is considered. Memory is statically allocated and not overcommitted.

During a VM migration, an SLA violation of 10% of CPU utilization is added to migrating VMs, and an additional CPU overhead of 10% of the migrating VMs CPU utilization is added to both the source and target host [10].

Measurements of metrics used by management policies, such as host CPU utilization and SLA violation, are drawn from each host every 2 minutes and evaluated by the policy over a sliding window of 5 measurements.

Strategy	Param.	Value
SP-DSS	per_{norm}	0.004
SP-DSS	per_{high}	0.006
SP-DSS	s_{norm}	1.15
SP-DSS	s_{high}	1.3
Goal-DSS	s_{worst}	0.01
Goal-DSS	p_C	0.83
Util-DSS	m_{SLA}	0.00255
Util-DSS	m_{power}	0.00255

Table 3.1: DSS Tuning Parameters

3.4.3 Workload

A data centre experiences a highly dynamic workload, driven by VM arrivals and departures, as well as dynamic workloads and resource requirements of VMs. We generate random *workload patterns* to evaluate our strategies, where a workload pattern consists of a set of VMs with specific start and stop times, each with dynamic trace-driven resource requirements. Each VM is driven by one of 5 individual traces: the *ClarkNet*, *EPA*, and *SDSC* traces [14], and two different job types from the *Google Cluster Data* trace [15]. The normalized rate of incoming requests, in 100 second intervals, is calculated for each trace. The request rates are used to define the current workload of each VM, with the CPU resource requirements of the VM calculated as a linear function of the current rate. Each VM starts its trace at a randomly selected offset time.

The number of VMs within the data centre is also varied dynamically to simulate the arrival and departure of VMs. A base of 600 VMs is created within the first 40 hours and remain running throughout the entire experiment, to maintain a reasonable minimum level of load. After 2 simulated days, new VMs begin to arrive at a changing rate, and terminate after about 1 day. The arrival rates are generated such that on a fixed interval of once per day, the total number of VMs in the data centre is equal to a randomly generated number uniformly distributed between 600 and 1600. The maximum number of VMs, 1600, was chosen because beyond that point, the SLA strategy is forced to deny admission of some incoming VMs due to insufficient available resources. This continues for 10 simulated days at which point the experiment terminates. Data from the first 2 days of simulation are discarded to allow the simulation to stabilize before recording results.

3.4.4 Strategy Switching Tuning Parameters

Each DSS strategy has some tuning parameters that must be configured to provide the best possible results, as described in Section 3.3. The first of these is the frequency with which the strategy switching algorithm is run. We evaluated the meta-strategies over multiple frequency values and found 1 hour to be an appropriate frequency for evaluating a strategy switch. Each DSS strategy looks at a set of data centre metrics sampled by a monitor over a certain window size: SP-DSS and Goal-DSS sample every 5 minutes and use a window size of 6 samples; Util-DSS samples every 20 minutes and uses a window size of 6. Util-DSS uses a longer monitoring frequency and window size in order to ignore minor fluctuations in data centre utilization and focus on longer term trends. This helps identify periods of real change in overall utilization, and avoid thrashing between strategies. For the remaining DSS tuning parameters, each combination of values was evaluated over a set of 5 randomly generated workload patterns, and the values that resulted in the best *score* were chosen. Table 3.1 contains the values of the best performing of all parameters defined in Section 3.3.

3.4.5 Metrics

- **Average Active Host Utilization (Host Util.):** the average CPU utilization of powered on hosts.
- **Power Consumed (Power):** the total power consumed by all hosts, measured in kWh.
- **Power Efficiency (PwrEff):** is p_{dc} over the entire simulation.
- **SLA Violation (SLA):** s over the entire simulation.
- **Number of Migrations (Migrations):** the number of VM migrations triggered by the management strategies.
- **Number of Strategy Switches (Switches):** the number of times that the active strategy was changed.

3.4.6 Results and Discussion

The results of the experiments are presented in Table 3.2. Each management strategy was evaluated with the same set of 100 randomly generated workload patterns. Each experiment was repeated only once per workload pattern, as the simulation is deterministic. Results were averaged across all workload patterns. In addition to reporting the metrics listed in Section 3.4.5, we report the normalized SLA and power values for each strategy, as well as the *score*.

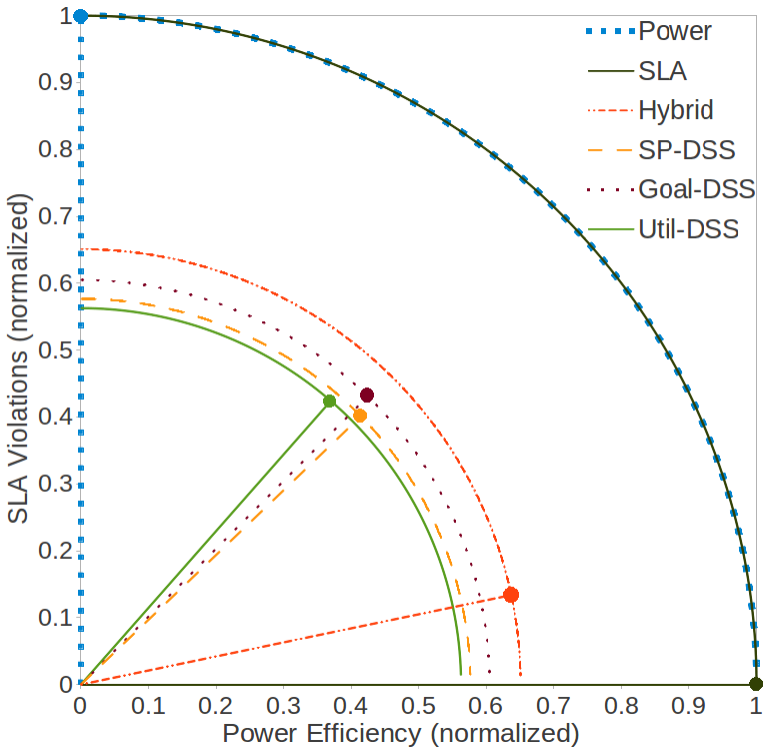

Figure 3.1: Strategy Scores

Figure 3.1 presents a graphical representation of the scores. The benchmark strategies (SLA and Power) both achieve a score of 1, by the definition of the score in Section 3.4.1. The angle of the line from the origin to each point gives an indication of how fairly the strategy behaved towards each goal, with a 45 degree angle representing a perfect balance between SLA and power.

Analysis of Variance was performed on the score results, as well as paired t-tests for each pair of management strategies. The resulting scores for each management strategy were found to be significantly different from each other.

All three DSS meta-strategies, as well as Hybrid, achieved better scores than the single-goal SLA and Power strategies. Util-DSS achieved the lowest score, followed by SP-DSS, then Goal-DSS, and finally Hybrid. The meta-strategies improved the score by about 40% when compared to Power and SLA, and by about 7-12% when compared to Hybrid. Util-DSS and SP-DSS each slightly favoured one of the goals, with Util-DSS favouring power and SP-DSS favouring SLA. Goal-DSS behaved fairly towards both goals. Hybrid, on the other hand, was considerably more skewed towards SLA than power, potentially limiting its usefulness in a practical application. The improved overall performance, as well as the balanced treatment of each goal, may therefore favour the selection of DSS over Hybrid. Among the meta-strategies, Util-DSS showed to be the most effective, though Goal-DSS was the most balanced.

Strategies	Host Util.	Power	PwrEff	SLA	Migrations	Switches	s_{norm}	p_{norm}	Score
SLA	75%	5,488	60.6	0.033%	15,818	–	0.0	1.0	1.0
Power	88%	4,384	75.2	0.474%	24,378	–	1.0	0.0	1.0
Hybrid	81%	5,049	65.9	0.092%	14,643	–	0.135	0.636	0.651
SP-DSS	80%	4,840	69.7	0.198%	18,608	20	0.360	0.452	0.588
Goal-DSS	81%	4,821	69.0	0.222%	19,448	56	0.430	0.425	0.607
Util-DSS	82%	4,778	69.8	0.220%	19,580	30	0.425	0.373	0.576

Table 3.2: Results per Strategy

All meta-strategies triggered 31 to 33% more migrations than the Hybrid strategy. While migration overhead was taken into consideration and reflected in the SLA violation and host utilization metrics, further work investigating the effect of migrations on networking should be conducted to determine if this migration count is acceptable. The increase in migration count from Hybrid to DSS is likely a side-effect of switching between strategies with different *stress* thresholds. The Power strategy efficiently pushes the utilization of a large number of hosts to a high value, just below its *stress* threshold. A switch to the SLA strategy at this point causes a large number of hosts to be considered stressed, as its *stress* threshold is below the current utilization achieved by the Power strategy. Thus, a spike in migrations is triggered. This also causes a spike in SLA violations due to migration overhead. It may be possible to introduce a mechanism to mitigate this effect and thus lower the DSS meta- strategy migration count. Such a mechanism may also result in an overall better *score* for the meta-strategies.

SP-DSS switched strategies the least number of times, followed by Util-DSS and Goal-DSS. This may be an indication that Util-DSS and Goal-DSS performed some strategy switches that did not contribute to improving performance towards the intended goals (possibly exhibiting a thrashing behaviour), and should be investigated.

3.5 Conclusions and Future Work

The development of data centre management strategies that can simultaneously pursue opposing goals, such as maximizing power efficiency and minimizing SLA violations, is a difficult task. In this work, we proposed dynamically switching between two strategies, each designed to achieve a single goal, to better adapt to changing data centre conditions. We developed three *meta-strategies* to perform dynamic strategy switching, and evaluated them through simulation. The meta-strategies improve overall performance by about 40% when compared to either of the single-goal strategies, and by 7-12% when compared to a hybrid strategy designed to pursue both goals simultaneously.

There are several directions for future work. Regarding DSS, the meta-strategies' behaviour when switching between strategies could be improved, so as to avoid spikes in migrations. DSS could also be applied separately to subsets of hosts, such as individual racks or clusters. Finally, threshold values and tuning parameters could be learned rather than fixed.

Networking overhead was not considered in this work. In the future, we intend to develop networking metrics to be used in our evaluations of management policies and strategies. Another topic of interest is the inclusion of constraints and affinity rules to help in determining the placement of VMs on hosts, as discussed by Gulati et al. [16].

Currently, our work relies only on CPU measurements to determine the level of load of

a host or VM, with other resources used only as a constraint on placement. In the future, load calculation should take into consideration memory and bandwidth, in addition to CPU (e.g.,[17]).

Future work could incorporate forecasting of VMs' resource demand, such as done by Bobroff et al. [3]. This would have an effect in the detection of stress situations, and in VM and host selection for migration.

Finally, this work essentially assumes a central manager for all decision making. Other architectural models could be explored, such as that presented by Zhu et al. [18].

Bibliography

[1] G. Foster, G. Keller, M. Tighe, H. Lutfiyya, and M. Bauer, "The Right Tool for the Job: Switching data centre management strategies at runtime," in *Integrated Network Management (IM), 2013 IFIP/IEEE International Symposium on*, May 2013.

[2] G. Khanna, K. Beaty, G. Kar, and A. Kochut, "Application performance management in virtualized server environments," in *NOMS Proceedings, 2006 IEEE/IFIP*, 2006.

[3] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations," in *IM Proceedings, 2007 IEEE/IFIP Int. Symp. on*, 2007, pp. 119–128.

[4] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *NSDI Proceedings, 4th Symp. on*, Cambridge, MA, USA, Apr. 2007, pp. 229–242.

[5] A. Verma, P. Ahuja, and A. Neogi, "pmapper: power and migration cost aware application placement in virtualized systems," in *Proceedings of the 9th ACM/IFIP/USENIX Int. Conf. on Middleware*, 2008.

[6] M. Cardosa, M. R. Korupolu, and A. Singh, "Shares and utilities based power consolidation in virtualized server environments," in *IM Proceedings, 2009 IEEE/IFIP Int. Symp. on*, 2009.

[7] B. Speitkamp and M. Bichler, "A mathematical programming approach for server consolidation problems in virtualized data centers," *IEEE TSC*, vol. 3, no. 4, pp. 266 –278, 2010.

[8] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource allocation algorithms for virtualized service hosting platforms," *J. Parallel Distrib. Comput.*, vol. 70, no. 9, pp. 962–974, Sep. 2010.

[9] G. Keller, M. Tighe, H. Lutfiyya, and M. Bauer, "An analysis of first fit heuristics for the virtual machine relocation problem," in *Network and Service Management (CNSM), 2012 8th International Conference on*. IEEE, Oct. 2012, pp. 406–413.

[10] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency Computat.: Pract. Exper.*, pp. 1–24, 2011.

[11] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper, "An integrated approach to resource pool management: Policies, efficiency and quality metrics," in *38th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, June 2008.

[12] M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya, "DCSim: A data centre simulation tool for evaluating dynamic virtualized resource management," in *Network and Service Management (CNSM), 2012 8th International Conference on*. IEEE, Oct. 2012, pp. 385–392.

[13] (2014) SPECpower_ssj2008. Standard Performance Evaluation Corporation. [Online]. Available: http://www.spec.org/power_ssj2008/

[14] (2014) The Internet Traffic Archive. [Online]. Available: http://ita.ee.lbl.gov/

[15] (2014) Google Cluster Data. Google Inc. [Online]. Available: http://code.google.com/p/googleclusterdata/

[16] A. Gulati, G. Shanmuganathan, A. Holler, C. Waldspurger, M. Ji, and X. Zhu, "Vmware distributed resource management: design, implementation, and lessons learned," *VMware Technical Journal*, vol. 1, no. 1, 2012.

[17] E. Arzuaga and D. R. Kaeli, "Quantifying load imbalance on virtualized enterprise servers," in *Perf. Eng. Proceedings, 1st WOSP/SIPEW Int. Conf. on*, 2010.

[18] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D. Gmach, R. Gardner, T. Christian, and L. Cherkasova, "1000 islands: Integrated capacity and workload management for the next generation data center," in *Proceedings of the 2008 International Conference on Autonomic Computing (ICAC'08)*, Chicago, IL, USA, Jun. 2008, pp. 172–181.

Chapter 4

A Hierarchical, Topology-aware Approach to Dynamic VM Management

As mentioned earlier, a data centre can be thought of, in essence, as a collection of hosts connected through a high-speed, high-bandwidth network. A management system is required to administer this infrastructure, mapping VMs to hosts and powering hosts on and off as needed. The architecture of the management system determines important properties of the system, such as scalability, quality of management decisions, and degree of management overhead. Different architectures offer different trade-offs between these properties.

Most resource management systems are designed for a cluster, do not appear to scale beyond a single cluster and are typically centralized [2]. The assumption of a centralized manager is often reflected in proposed dynamic resource management systems (e.g., [3, 4, 5]). These systems, however, may not be able to cope with the scale of large data centres so as to satisfy realistic demands [6, 7]. In addition, by treating the entire data centre as a single pool of resources, management data (including communication and VM migrations) is sent across the network without considering the overhead, resulting in an inefficient use of data centre resources.

In this work, we investigate a hierarchical approach to data centre management. Hierarchical approaches offer increased scalability at the expense of having only a partial view of the system [8]. We divide the management domain into *scopes* of management, each encompassing a subset of managed elements, and encapsulating management data and management actions within itself. We leverage the topology of the data centre network to define scopes; the scopes are *host*, *rack*, *cluster* and *data centre*. By making the hierarchy topology-aware, we limit the flow of management data across the data centre. We hypothesize that this organization will result in a more efficient use of the data centre, greatly reducing network traffic.

[0]This chapter is based on work published in [1].

Challenges in using a hierarchical approach to dynamic data centre management include the following: (i) Defining a set of metrics to represent the system state of racks and clusters. A cluster manager could send information about all of its hosts to the data center manager, but that would represent a large amount of data to be transmitted and analyzed; (ii) Determining the responsibilities of managers and the interactions between them. Both challenges are addressed in this work.

The remainder of this chapter is organized as follows: Section 4.1 reviews related work in this area, Section 4.2 discusses the proposed management system, and Section 4.3 describes the management strategies and policies implemented. Section 4.4 presents and discusses experimental results, and Section 4.5 concludes and suggests directions of future work.

4.1 Related Work

Most work on data centre management focuses on centralized approaches to dynamic VM management. In these works, a single, central manager makes decisions using global knowledge of the state of every component in the data centre. Some of these works assume VM resource demand to be static and perform a static allocation of a set of VMs into a set of empty hosts, using best fit heuristics, linear programming or vector bin packing algorithms [9, 10, 3]. Other works, however, do not make this assumption (i.e., work with dynamic VM demand) and use fuzzy logic-based controllers, first fit heuristics or greedy hill-climbing techniques to perform dynamic reallocation of host resources to VMs [11, 4, 5, 2].

Some research has been done, nonetheless, on hierarchical approaches to data centre management. Zhu et al. [12] proposed the use of a hierarchy of resource controllers: a node controller performed resource reallocation among co-located workloads in a host, a pod controller migrated workloads between hosts in its pod (to prevent stress situations), and a pod set controller migrated workloads between pods to improve their individual performance. The system relied on a variety of techniques, such as fuzzy logic-based controllers, genetic algorithms and trace-based analysis. Feller et al. [7] developed a VM management framework for private IaaS clouds that consisted of a 3-level hierarchy of managers (host, cluster and data centre levels) for increased scalability, and offered fault-tolerance through replication and failure recovery mechanisms. Moens et al. [6] proposed the use of a hierarchical management system for clouds, where nodes were organized in a B-Tree structure. Their goal was to investigate how a centralized application placement algorithm could be adapted to run on a hierarchical context. In a follow-up work, Moens et al. [8] presented a scalable framework to build and manage hierarchical management systems.

Our work differs from the current literature in that we propose to leverage the topology

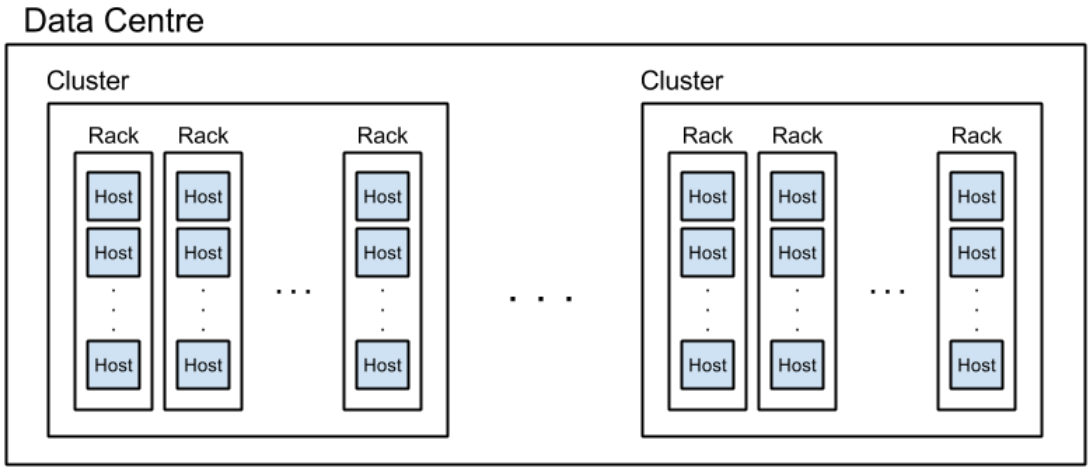

Figure 4.1: Target data centre infrastructure and management scopes.

of the data centre network to build the management system's hierarchy, thus limiting the flow of management data across the network. We also focus on the definition of aggregate metrics to convey system state information across management levels. In addition, we develop protocols (expressed as management strategies and policies) for the managers at different levels to collaborate in the execution of management operations, such as placement and consolidation.

4.2 Hierarchical Management

In this section we describe the target data centre infrastructure (Section 4.2.1) and the proposed management system's architecture (Section 4.2.2). We also discuss the collection of monitoring data, the definition of aggregate metrics, and the communication of status data (Section 4.2.3).

4.2.1 Data Centre Infrastructure

The target infrastructure consists of a collection of clusters, each cluster being a collection of racks, and each rack a collection of physical servers (see Figure 4.1). Virtual machines (VMs) are hosted in physical servers (or hosts), where they are allocated resources to run.

There are two networks in the data centre: the *data network* and the *management network*. The data network is used by VMs for applications' communication, while the management network is reserved for management data (i.e., communication and VM migrations). Both networks have the same architecture. The hosts in a rack are connected to the networks through two switches – one per network – placed inside the rack. The racks in a cluster are connected to each other through a cluster-level switch, and the cluster-level switches are connected to a

central switch at data centre-level. From now on, when we talk about the *data centre network*, we will be referring to the management network.

4.2.2 System Architecture

The idea behind using a hierarchical approach to management is to logically group sets of hosts and manage each set as a unit, encapsulating detailed monitoring and management actions within the unit as much as possible. In addition, the elements at each level are grouped again to form the elements of the level above. These results in the creation of *scopes* of management, which limits the spread of management data across the data centre. Monitoring data is collected within each scope and a summary of the data is shared with the upper-level management. VM migrations, as much as possible, are also restricted to their own scope.

We decided to leverage the organization of the data centre network by using hosts, racks and clusters to delimit management scopes. In addition, the data centre as a whole forms a large, all-encompassing management scope.

Each management scope has an *autonomic manager* associated to it. Each manager has a set of policies and a knowledge base. Events received by the manager may trigger the execution of one or more policies, and the execution of policies may result in actions or further events being triggered. The knowledge base is updated with information sent to the manager in the form of events and by the policies' execution. Host managers form the lowest level of the hierarchy (Level 0). Above the host managers are rack managers (Level 1) and above rack managers are cluster managers (Level 2). The data centre manager forms the top of the hierarchy (Level 3).

Communication between managers occurs both periodically (monitoring) and as needed (in response to events). Messages travel vertically up and down the hierarchy and, less frequently, horizontally. (Management communication is covered in more detail in Sections 4.2.3 and 4.3.)

4.2.3 Monitoring Data, Aggregate Metrics and Status Updates

Host managers periodically collect monitoring data from hosts and (hosted) VMs. The collected data includes resource utilization (CPU, memory, network bandwidth and storage) of each VM and of the host itself, host power consumption, and the number of current incoming and outgoing VM migrations. This information is packaged and sent to the manager of the rack to which the host belongs in the form of a status update.

Rack managers process the status updates sent by the various host managers within their scope and update their knowledge base with the information received. In time, the rack managers process the information in their knowledge base and create a status update message to send to their corresponding cluster manager. The status update includes the number of active,

suspended and powered off hosts in the rack, the rack's power consumption (calculated as the sum of the hosts' power consumption, plus the power consumption of the two switches in the rack), and the amount of spare resources available in the least loaded active host in the rack (*max spare capacity* metric).

The purpose of the *max spare capacity* metric is to convey, in a single metric, the largest number of spare resources available in one of the rack's hosts. In this way, when a decision has to be made regarding the placement or migration of a VM, it is possible to determine whether a rack will have enough space available – in any of its active hosts – to fit the VM. In addition, the *max spare capacity* metric was designed to be a single, unitless value. This is achieved by defining a standard VM size measure (similar to defining a standard measure of "volume") and calculating the number of standard VMs that could fit in the given spare capacity. For example, if a host currently had 3 free CPU cores and 5 GB of memory and the standard VM size were 1 CPU core and 2 GB of memory, then the host's *max spare capacity* metric would have a value of 2.5.

The status updates sent by the managers of the racks that form a cluster are received and processed by that cluster's manager. The information contained in the status updates is used to update the cluster manager's knowledge base. The cluster manager also composes periodic status updates, which are sent to the data centre manager. These status updates include the number of active racks, the cluster's power consumption, the amount of spare resources available in the least loaded active host in the cluster (*max spare capacity* metric), and the number of inactive hosts in the rack with the most active hosts (*min inactive hosts* metric).

The purpose of the *min inactive hosts* metric is to be able to identify the cluster that contains the rack closest to becoming completely active (i.e., all its hosts are active), so that if a new host must be started, it is started in that rack. (See Assumption #3 in Section 4.3.1 for more details.)

The frequency with which host managers send status updates is higher than that with which rack managers send status updates, which in turn is higher than that of cluster managers.

4.3 Management Strategies and Policies

Managers are responsible for performing management operations within their assigned scope and for interacting with other managers when collaboration is necessary. In this section, we describe the management strategies and policies designed to perform the following data centre management operations:

- **VM Placement.** When a new VM creation request arrives in the data centre, a host must be selected to instantiate the VM.

- **VM Relocation.** When a host is stressed (i.e., close to running out of spare resources to allocate to its VMs), one or more of its VMs must be migrated away, so as to free resources. For each VM migration, a physical server must be found to become the new host of the migrated VM – we call this physical server the target host.

- **VM Consolidation.** The process of relocating VMs in the data centre, so as to concentrate the VMs into as few hosts as possible.

As explained in Section 4.2.2, there are four levels of management in the system: host-level (Level 0), rack-level (Level 1), cluster-level (Level 2), and data centre-level (Level 3). The management strategies consist of policies running at one or more levels. Managers in the same level run the same set of policies.

4.3.1 Assumptions

Several assumptions inform the design of the management strategies in the system. These assumptions are listed and described next.

Assumption 1. CPU utilization alone provides a good enough indicator of host power consumption. Thus, power consumption can be calculated based on CPU utilization.

Assumption 2. Network switch power consumption is constant and independent of network traffic.

Assumption 3. Computing entities work more efficiently, in terms of power consumption, when operating at high levels of resource utilization.

The last assumption rests in two observations: first, hosts are more power efficient the higher their utilization[1], and second, racks and clusters consume power, due to their networking infrastructure, with independence of the load of their hosts. From these observations, it follows that by consolidating load in the least number of hosts, racks and clusters, power consumption can be reduced. With that in mind, the management strategies focus on increasing the utilization level of hosts, racks and clusters and choose (placement and migration) target hosts accordingly.

4.3.2 VM Placement

The VM Placement strategy consists of policies running at three levels: rack, cluster and data centre. New VM creation requests are received by the data centre manager, who must choose a cluster to which to forward the request (actually, the request is forwarded to the cluster's

[1] A server can consume between 50% and 70% of its peak power usage when idle [13].

manager). The cluster manager in turn must choose a rack (within the cluster) to which to forward the request. The selected rack's manager, upon reception of the VM creation request, selects one of its hosts to instantiate the VM and sends the appropriate message to the host.

When a manager receives an Application (or VM) Placement request, the following procedure is followed:

1. If there are no active computing entities in its management scope, the manager selects the most power efficient entity[2], activates it, and forwards the VM Placement request to said entity.

2. If there is only one active entity, the manager checks if said entity can accommodate the VM (i.e., has enough spare or inactive resources). If so, it forwards the request to the entity; otherwise, the manager activates the next most power efficient entity.

3. If there are several active entities, the manager searches for a target entity among subsets of equally power efficient entities, trying first to identify the entity with the most spare capacity (biggest *max spare capacity* value). If said capacity is not enough to accommodate the VM, then the manager searches for the entity with the most loaded rack (smallest *min inactive host* value).

4. If after parsing the list of active entities none has been identified as a feasible target, then the manager activates the next most power efficient entity left.

At rack-level, the Placement policy implements a greedy algorithm to select the first non-stressed host with enough spare capacity to take the VM without exceeding a given threshold – host's target utilization threshold. (This policy is described in detail in [14].)

4.3.3 VM Relocation

The VM Relocation strategy, like the VM Placement strategy, consists of three policies, running at Levels 1, 2 and 3. In contrast to the VM Placement process, the VM Relocation process always begins at rack-level. A rack manager detects that one of its hosts is stressed and tries to migrate one of the host's VMs to another host in the rack. If a suitable target host is found, the VM migration is issued and the relocation process terminated. However, if no target host is found within the scope, then a search for a suitable target host beyond the local scope is started.

At rack-level, the VM Relocation process consists of two steps. The first step consists of a greedy algorithm that attempts to migrate a VM away from its stressed host and into a non-stressed host. (This policy is described in detail in [14].) If that fails, the manager requests

[2]The entity that can perform the most processing per watt of power, measured as CPU-shares-per-watt.

assistance from the cluster-level manager to find a suitable target host in a different rack. For that purpose, the rack manager selects a VM and sends its information to the cluster manager in the form of a VM Relocation request.

A cluster manager can receive a VM Relocation request from two different sources: a local rack manager or the data centre manager. In the first case, the manager of one of the racks in the cluster requests assistance to migrate a VM away from the rack. In the second case, the data centre manager requests assistance on behalf of another cluster manager to migrate a VM into the (local) cluster.

The process followed by the cluster manager to find a suitable target rack to which to migrate the VM is similar to the VM Placement process. The only difference is that additional measures are taken to avoid selecting as target the rack whose manager sent the VM Relocation request – if said request was indeed sent by a local rack manager.

If all of the racks in the cluster are active and no suitable target rack is found, then the manager forwards the VM Relocation request to the data centre manager (if the request had come from a local rack manager), or the manager sends a *request reject* message to the data centre manager (if the request had come from the data centre manager).

At data center-level, the VM Relocation process is similar to the VM Placement process, with additional measures taken to avoid selecting as relocation target the cluster that sent the VM Relocation request. If all the clusters in the data centre are active and no suitable target cluster is found, then the data centre manager sends a *request reject* message directly to the rack manager that had sent the original VM Relocation request.

4.3.4 VM Consolidation

The VM Consolidation strategy consists of a single policy running at rack-level (Level 1). This means that the consolidation process concerns itself with the hosts and VMs contained within a rack; no VM migrations occur between racks due to consolidation.

This policy implements the VM Consolidation process using a greedy algorithm. VMs are migrated out of underutilized hosts and into hosts with higher utilization. (This policy is described in detail in [14].)

4.4 Evaluation

We evaluate our work through experimentation with the simulation tool DCSim [15, 16]. The experimental setup is described in detail in Section 4.4.1, while the experimental design is described in Section 4.4.2. Section 4.4.3 describes the metrics collected and Section 4.4.4

presents and discusses the results.

4.4.1 Experimental Setup

Our simulated data centre consists of 5 homogeneous clusters, each containing 4 racks, and each rack containing 10 hosts. Three clusters consist of hosts modelled after the HP ProLiant DL380G5, with 2 dual-core 3GHz CPUs (3000 CPU shares per core) and 8 GB of memory. The other two clusters consists of hosts modelled after the HP ProLiant DL160G5, with 2 quad-core 2.5GHz CPUs (2500 CPU shares per core) and 16GB of memory. Hosts are modelled to use a work-conserving CPU scheduler, which means that CPU shares not used by one VM can be used by another VM. Caps on CPU usage are not supported. If the total CPU demand of co-located VMs exceeds the CPU capacity of their host, CPU shares are distributed among the VMs in a fair-share manner. Memory is statically allocated and is *not* overcommitted.

In this work, we consider a host to be stressed if its CPU utilization exceeds 90% (non-stressed otherwise), while a host with CPU utilization below 60% is considered underutilized. Hosts' target utilization is set at 85%.

Network switches were modelled after the power measurement study by Mahadevan et al. [17]. Rack switches have 48 1-Gpbs ports and exhibit a power consumption of 102 Watts. Cluster and data centre switches have 48 1-Gpbs ports and exhibit a power consumption of 656 Watts.

We define three different VM sizes: *small* requires 1 virtual core with a minimum of 1500 CPU shares and 512MB of memory, *medium* requires 1 virtual core with a minimum of 2500 CPU shares and 512MB of memory, and *large* requires 2 virtual cores with a minimum of 2500 CPU shares each and 1GB of memory. Note that these are requested, maximum resource requirements. VMs are initially placed into hosts based on these values and attempts are made to guarantee them when required. However, once VMs are running, they are mapped into hosts and allocated resources based on their actual resource *usage*, not request.

To simulate workload for the experiments, we model a set of interactive applications running within the data centre, each application running inside a single VM. Applications are assigned a VM type in such a way to create an equal number of all three VM types. Real workload traces are used to simulate dynamic resource demand for the applications. Each application is assigned a trace built from one of five sources: the *ClarkNet*, *EPA*, and *SDSC* traces [18], and two different job types from the *Google Cluster Data* trace [19]. For each source trace, its normalized request rate is calculated, in 100-second intervals. Applications' CPU demand is calculated through the use of a queuing model and the current request rate. To prevent all applications powered by the same trace from exhibiting identical behaviour, each

application starts reading its trace at a randomly selected offset time.

The set of applications can be divided into *base load* and *additional load*. Applications in the *base load* arrive in the data centre at the beginning of the simulation and remain running throughout the entire experiment. During this setup period, metrics are not recorded. Once the setup period is over and the *base load* is running in the data centre, metric recording starts and applications in the *additional load* may start arriving. These additional applications may arrive at varying rates and terminate after a certain length of time. A specific, randomly-generated instance of application arrivals, departures, and trace offset times is referred to as a *workload pattern*, and is entirely repeatable through specifying the random seed used to generate it. We use a set of 10 different *workload patterns* to evaluate each experiment, and all presented results are averaged across these 10 *workload patterns*.

4.4.2 Experimental Design

In these experiments we compare our proposed hierarchical, topology-aware management system (referred to as **Hierarchical**) with an existing centralized management system (referred to as **Centralized**) that treats the whole data centre as a single pool of resources. The centralized system uses a balanced management strategy – *Hybrid Strategy*, described in detail in [14] – that achieves reasonable performance in terms of both power consumption and SLA violations. Furthermore, we use a version of the centralized system where the VM Relocation policy is triggered in a reactive fashion instead of periodically. That is, hosts are checked for stress every time they send a status update and the VM Relocation policy is triggered upon stress detection. In that way, the system responds faster to stress situations and the policy is not invoked when no host is stressed. This improved version of the management system was presented in [20].

We design three experiments: two with a constant number of applications deployed in the data centre and one with a variable number of applications. Experiments 1 and 2 consist of a base load of about 800 and 1440 applications, respectively, that arrive in the data centre within the first 4 and 6 days of simulation, respectively. In both cases, there is no additional load. Experiment 3 consists of a base load of about 800 applications that arrive in the data centre within the first 4 days of simulation, and an additional load that consists of applications arriving in the data centre at a rate of 10 applications per hour over days 6, 7, 9 and 10. These applications terminate after about 4 days. The total number of applications in the system varies between 800 and 1520. In all cases, metrics are recorded for a total of 8 simulated days.

In these experiments, host managers are configured to send status updates every 2 minutes, and rack and cluster managers every 5 minutes.

4.4.3 Metrics

- **Active Hosts (Hosts):** The average number of hosts in the *On* state. The higher the value, the more physical hosts are being used to run the workload.
- **Active Racks (Racks):** The average number of racks in the *On* state.
- **Average Active Host Utilization (Host Util.):** The average CPU utilization of all hosts in the *On* state. The higher the value, the more efficiently resources are being used.
- **Power Consumption (Power):** Power consumption is calculated for each host and switch, and the total kilowatt-hours consumed during the simulation are reported. Hosts' power consumption is calculated using results from the SPECPower benchmark [21], and is based on CPU utilization. Switches' power consumption is assumed constant and independent of network traffic – as long as the switch is powered on – and is based on the power benchmarking study by Mahadevan et al. [17].
- **SLA Achievement (SLA):** SLA Achievement is the percentage of time in which the SLA conditions of an application are met. We define the SLA of an application in terms of an upper threshold on its response time. When the response time exceeds this threshold, SLA is considered violated. Response time exceeding the threshold is a consequence of under provisioning CPU resources to the application, caused by contention with other VMs on a host.
- **Number of Migrations (Migrations):** The number of VM migrations triggered during the simulation. These are further divided into *intrarack*, *intracluster* and *intercluster*. Typically, a lower value is preferable, since fewer migrations means less network overhead.

4.4.4 Results and Discussion

The results of the experiments are presented in Table 4.1 and Figure 4.2. As we can see, **Hierarchical** achieves 7% to 9% lower *Average Active Host Utilization* than **Centralized**, which results in more hosts and racks having to be powered on, thus increasing *Power Consumption* in the data centre by 6% to 9%. On the other hand, the lower host utilization results in 1.5% higher *SLA Achievement*. In addition, **Hierarchical** issues 48% to 56% fewer VM migrations than **Centralized**, and 97% to 99% of those migrations occur within the rack scope, as opposed to **Centralized** with only 6% to 9% of migrations issued within a rack. Figure 4.3 shows the results of the third experiment, and is representative of all experiments conducted.

Exp.	System	Host Util.	Hosts	Racks	Power	SLA	Migrations	Intrarack	Intracluster	Intercluster
1	**Hierarchical**	78.5%	79.7	11.4	5,414.6	99.7%	78,651.6	99.5%	0.4%	0.0%
1	**Centralized**	87.5%	67.4	7.5	4,956.6	98.3%	180,644.0	9.0%	20.1%	70.8%
2	**Hierarchical**	79.1%	174.2	19.9	9,736.6	99.5%	183,470.9	98.9%	0.8%	0.1%
2	**Centralized**	87.4%	152.4	16.1	8,935.1	98.0%	375,262.7	6.0%	20.4%	73.5%
3	**Hierarchical**	80.2%	144.1	17.3	8,374.4	99.5%	168,099.8	97.9%	1.5%	0.4%
3	**Centralized**	87.5%	129.6	13.7	7,859.2	98.0%	328,190.7	6.5%	20.5%	72.2%

Table 4.1: Hierarchical vs Centralized Results

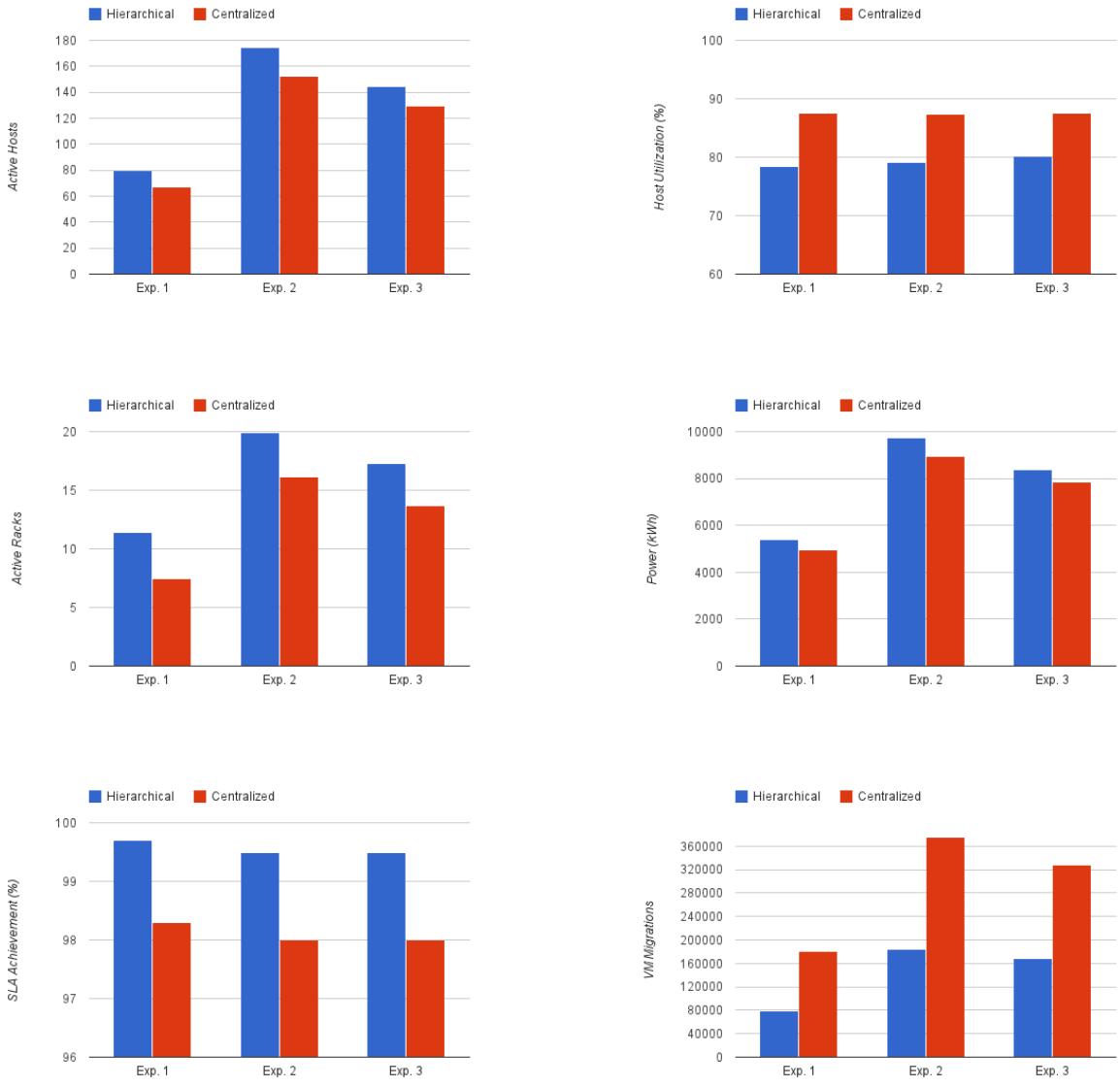

Figure 4.2: Hierarchical vs Centralized Results

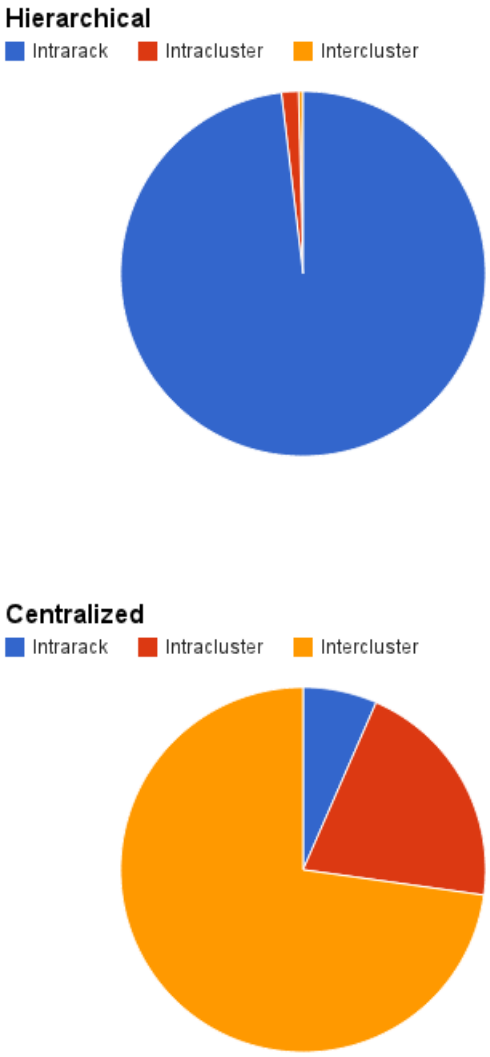

Figure 4.3: Exp. 3 – VM Migrations per Type

As we noted early on, minimizing power consumption, just as maximizing SLA achievement, is an important goal in data centre management. From that perspective, **Hierarchical** does not perform as well as **Centralized**, given its larger use of data centre infrastructure. However, the reason why **Hierarchical** does not achieve better host utilization is that its strategies focus on restricting migrations to the rack scope, only migrating VMs outside of a rack when there is absolutely no other option. When **Hierarchical**'s policies try to find a target host for a migration, they only consider the hosts in the same rack as the source host, thus having very limited options. However, this approach results in very few VMs ever traversing the network between racks or even clusters. If we consider that the smallest VM in these experiments had a memory footprint of 512 MB and that the VM Relocation strategies of both **Hierarchical** and **Centralized** always try to migrate the smallest VM available, given the number of intracluster and intercluster migrations, we can calculate that **Hierarchical** moves at least 1.6 TB of management data (i.e., migrated VMs' memory footprint) through the network (outside the rack scope), while **Centralized** moves at least 82.1 TB of data. This represents a significant reduction in network overhead, potentially outweighing the increase in power consumption.

4.5 Conclusions and Future Work

In this work we presented a hierarchical, topology-aware management system for large-scale data centres. We defined a set of aggregate metrics to convey system state information to higher management levels, and defined communication protocols between management elements – expressed in the form of management strategies and policies – to carry on data centre management operations. By leveraging the data centre network topology to create the hierarchy, the flow of management data (VM migrations and communications) is encapsulated to a high extent within the lowest management scope (i.e., the rack), thus reducing the overhead in the data centre management network.

This work could be extended in several ways. One such way would be to extend the scope of the VM Consolidation operation to Levels 2 and 3, for it to work in a similar fashion as VM Relocation does. Another extension would be to eliminate Management Level 3 (i.e., the data centre manager) and organize all cluster managers in a peer-to-peer structure. Yet another extension would be to modify all three management operations to support applications composed of multiple VMs. It could also be of interest to study how to dynamically adjust the frequency of status updates based on workload behaviour (i.e., resource demand variability) or the arrival rate of applications. Finally, it may sometimes be convenient to migrate a VM away from its hosting rack, so as to break a cycle of continuous migrations. Determining if and when to do this makes for an interesting challenge.

Bibliography

[1] G. Keller, M. Tighe, H. Lutfiyya, and M. Bauer, "A hierarchical, topology-aware approach to dynamic data centre management," in *Network Operations and Management Symposium Workshops (NOMS), 2014. IEEE/IFIP*, May 2014.

[2] A. Gulati, G. Shanmuganathan, A. Holler, C. Waldspurger, M. Ji, and X. Zhu, "Vmware distributed resource management: design, implementation, and lessons learned," *VMware Technical Journal*, vol. 1, no. 1, 2012.

[3] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource allocation algorithms for virtualized service hosting platforms," *J. Parallel Distrib. Comput.*, vol. 70, no. 9, pp. 962–974, Sep. 2010.

[4] G. Keller, M. Tighe, H. Lutfiyya, and M. Bauer, "An analysis of first fit heuristics for the virtual machine relocation problem," in *Network and Service Management (CNSM), 2012 8th International Conference on*. IEEE, Oct. 2012, pp. 406–413.

[5] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Gener. Comput. Syst.*, vol. 28, no. 5, 2012.

[6] H. Moens, J. Famaey, S. Latre, B. Dhoedt, and F. De Turck, "Design and evaluation of a hierarchical application placement algorithm in large scale clouds," in *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*. IEEE, 2011, pp. 137–144.

[7] E. Feller, L. Rilling, and C. Morin, "Snooze: A scalable and autonomic virtual machine management framework for private clouds," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE Computer Society, may 2012, pp. 482–489.

[8] H. Moens and F. De Turck, "A scalable approach for structuring large-scale hierarchical cloud management systems," in *Network and Service Management (CNSM), 2013 9th International Conference on*. IEEE, Oct 2013, pp. 1–8.

[9] M. Cardosa, M. R. Korupolu, and A. Singh, "Shares and utilities based power consolidation in virtualized server environments," in *IM Proceedings, 2009 IEEE/IFIP Int. Symp. on*, 2009.

[10] B. Speitkamp and M. Bichler, "A mathematical programming approach for server consolidation problems in virtualized data centers," *IEEE TSC*, vol. 3, no. 4, pp. 266 –278, 2010.

[11] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper, "An integrated approach to resource pool management: Policies, efficiency and quality metrics," in *38th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, June 2008.

[12] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D. Gmach, R. Gardner, T. Christian, and L. Cherkasova, "1000 islands: Integrated capacity and workload management for the next generation data center," in *Proceedings of the 2008 International Conference on Autonomic Computing (ICAC'08)*, Chicago, IL, USA, Jun. 2008, pp. 172–181.

[13] L. A. Barroso and U. Holzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[14] G. Foster, G. Keller, M. Tighe, H. Lutfiyya, and M. Bauer, "The Right Tool for the Job: Switching data centre management strategies at runtime," in *Integrated Network Management (IM), 2013 IFIP/IEEE International Symposium on*, May 2013.

[15] M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya, "Towards an improved data centre simulation with DCSim," in *Network and Service Management (CNSM), 2013 9th International Conference on*. IEEE, Oct. 2013, pp. 364–372.

[16] (2014) DCSim on GitHub. Distributed and Grid Systems (DiGS). [Online]. Available: https://github.com/digs-uwo/dcsim

[17] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan, "A power benchmarking framework for network devices," in *Proceedings of the 8th International IFIP-TC 6 Networking Conference*. Springer-Verlag, 2009, pp. 795–808.

[18] (2014) The Internet Traffic Archive. [Online]. Available: http://ita.ee.lbl.gov/

[19] (2014) Google Cluster Data. Google Inc. [Online]. Available: http://code.google.com/p/googleclusterdata/

[20] M. Tighe, G. Keller, H. Lutfiyya, and M. Bauer, "A Distributed Approach to Dynamic VM Management," in *Network and Service Management (CNSM), 2013 9th International Conference on*. IEEE, Oct 2013.

[21] (2014) SPECpower_ssj2008. Standard Performance Evaluation Corporation. [Online]. Available: http://www.spec.org/power_ssj2008/

Chapter 5

Dynamic Management of Multi-VM Applications with Constraints

While deploying and managing single-VM applications in a data centre is a well studied problem [2, 3, 4, 5, 6, 7, 8, 9, 10, 11], managing multi-VM applications is not. A multi-VM application is an application that consists of multiple components working together to provide a service, where each component runs in its own dedicated VM. A common example of a multi-VM application is a 3-tier web application, consisting of web, application and database tiers, where each tier is hosted on a separate server [12].

Multi-VM applications may require an IaaS provider to meet certain placement constraints, which could be specified in an SLA. For example, an application may require some of its components to be co-located in the same host (or the same rack) for performance reasons, while another application may require its components to be placed far apart for high availability purposes. In this way, the data centre management system is faced with the challenge of meeting these placement constraints, in addition to the challenge of meeting each application's resource demand at every point in time.

In this work, we investigate how to manage multi-VM applications in data centres, so as to increase infrastructure utilization while keeping SLA violations low, and satisfying application placement constraints. In addition, we explore temporarily violating placement constraints through management actions to evaluate its effect on overall data centre power consumption and SLA satisfaction.

The remainder of this chapter is organized as follows: Section 5.1 discusses related work in the area, Section 5.2 describes the application placement constraints we considered, Section 5.3 presents the two approaches we developed to manage multi-VM applications in the data centre, Section 5.4 presents our evaluation and discusses the results, and finally Section 5.5

[0]This chapter is based on work published in [1].

states our conclusions and suggests directions of future work.

5.1 Related Work

There is considerable work that deals with the deployment and management of single-VM applications in data centres, using a variety of techniques ranging from greedy heuristics (First Fit, Best Bit, hill-climbing, etc.) and genetic algorithms to integer linear programming and fuzzy-logic [2, 3, 4, 5, 6, 7, 8, 9, 10].

There is also considerable work that focuses on the placement of virtual networks or virtual infrastructures (also referred to as Virtual Data Centres) in data centres (e.g., [13, 14, 15]). These virtual infrastructures consist not only of VMs, but also switches, routers and links connecting those VMs and having requirements of their own, such as bandwidth or delay. Our work, however, focuses on mapping application components running inside VMs to hosts in the data centre. Multiple application components can be mapped to the same host (and sometimes are required to in this work), which is usually not possible when mapping virtual networks.

There is work, however, that does focus on managing multi-VM applications in data centres. Gulati et al. [16] presented a high-level overview of VMware's Distributed Resource Scheduler (DRS), which is used to map VMs into hosts and to periodically perform load balancing. DRS allows users to specify VM-to-VM and VM-to-Host *affinity* and *anti-affinity* rules (or constraints) in their deployments. VM-to-VM anti-affinity rules are respected at all times, while VM-to-VM affinity rules are respected during load-balancing, but may be violated during initial placement. The system described in this work relies on a centralized architecture and does not scale beyond a single cluster. Our approach, on the other hand, relies on a hierarchical architecture with the express purpose of scaling across multiple clusters. We adopt their definitions of (VM-to-VM) *affinity* and *anti-affinity* constraints for our own work.

Shrivastava et al. [17] addressed the issue of managing multi-tier applications in virtualized data centres. More specifically, they focused on the problem of finding new target hosts for VMs that had to be migrated away from their current host due to resource stress. They proposed an approach that considered both the data centre network topology and the communication dependencies between application components when making VM migration decisions, with the goal of minimizing the resulting data centre network traffic (due to inter-VM communication) after the VM migrations had been completed. They proposed a Best Fit heuristic that aimed to minimize the cost of each migration, calculated as the total delay introduced in the communication between the migrated VM and any other VM with which it communicated. Their proposed algorithm relocated *overloaded* VMs (though it is unclear what constituted an overloaded VM). In contrast, our work does not treat VMs as overloaded, but rather treats hosts

as stressed; as a consequence, we are not forced to migrate specific VMs, but rather we select which VM to migrate so as to optimize a given goal. In addition, given that we seek to place all VMs of an application within a single rack, our solution minimizes communication traffic in the data centre network by default.

Shi et al. [18] also worked on placing sets of VMs with placement constraints in a data centre, with the goal of maximizing the data centre provider's revenue. They defined three constraint types: *full*, all VMs in the set must be placed in the data centre or none; *anti-affinity*, all VMs in the set must be placed in different hosts; and *security*, all VMs in the set must be placed in hosts that do not host VMs from other sets. VM sets can have one of these constraint types, no constraints at all, or the combination *full* + *anti-affinity* or *full* + *security*. They proposed an Integer Linear Programming formulation that achieved optimal solutions, but was time consuming and unscalable. They also proposed a Firt Fit Decreasing heuristic for multi-dimensional bin packing that achieved suboptimal solutions, but was fast to compute. In our work, we consider the *full* constraint implicitly; in other words, all applications have to be placed completely or not at all. On the other hand, we do not consider the *security* constraint, which requires VMs to be placed separately from the rest of the workloads in the data centre, thus greatly simplifying their placement. In addition, Shi et al. apply constraints to all the VMs in a set, while we consider constraints to be applied to individual VMs in a set. Finally, our work does not only address placement, but also relocation and consolidation.

Finally, Tighe and Bauer [19] worked on integrating application autoscaling with multi-VM application management in data centres, with the aim of satisfying the management goals of both application owners and data centre provider. In addition, in order to reduce inter-VM communication latency, their solution sought to place all components of an application within a single rack whenever possible, though it would violate this constraint if necessary. Their work does not consider any other form of placement constraint between application components, while our work did not consider application autoscaling.

5.2 Application Placement Constraints

A placement constraint is a restriction specified by the application owner to indicate the way in which the components of an application should be placed with respect to each other in the data centre.[1] The purpose of a constraint is usually to improve an application's performance or reliability.

In this work, we consider three types of constraints:

[1]Placement constraints can also specify an application component's need for specific hardware or software, but that type of constraint is beyond the scope of this work.

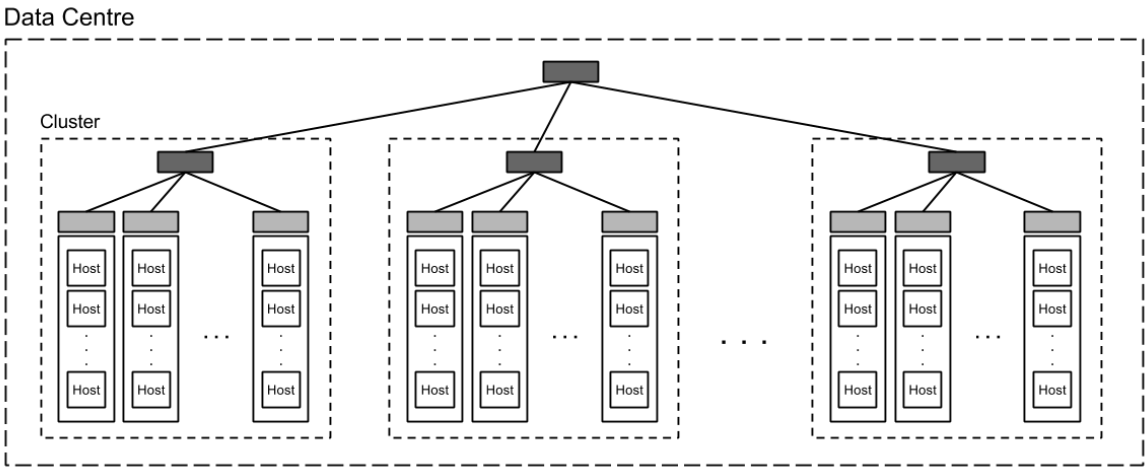

Figure 5.1: Data Centre Infrastructure

1) **Single-rack:** all components of an application should be placed within a single rack.

2) **Affinity:** application components related by affinity should be placed in the same host.

3) **Anti-affinity:** application components related by anti-affinity should be placed in different hosts.

The motivation behind the *single-rack* constraint is that by spreading an application's components across multiple racks, the communication paths between components become longer (see Figure 5.1), and each additional link that needs to be traversed introduces delays in the communication (i.e., increases network latency), thus degrading application performance. In order to prevent this problem, all components of an application should be placed within a single rack. By default, we treat every application submitted to the data centre as affected by this constraint.

The *affinity* and *anti-affinity* constraints are adopted from the work of Gulati et al. [16]. We say that an application component affected neither by *affinity* nor *anti-affinity* is *neutral*. A neutral application component may, however, communicate with other components in the application.

Finally, though the constraints are defined in terms of application components, given the one-to-one mapping of application component to hosting VM, we say that the constraints apply to the VMs or the application components interchangeably.

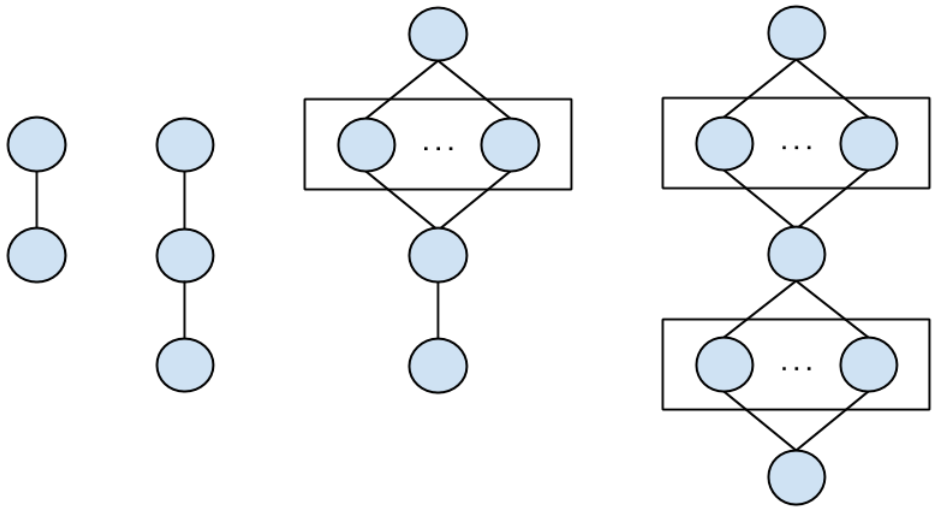

Figure 5.2: Application Templates. Nodes represent application components and edges represent communication links. Boxes indicate that multiple instances of the same component may exist.

5.2.1 Application Templates

For this work, we created a series of *templates* from which to create all applications to be deployed in the data centre (see Figure 5.2). These templates model *interactive applications* in the form of multi-tiered web applications. The templates specify the communication paths between application components and whether there can be multiple instances of any component. From this information, placement constraints can be inferred as follows:

- if a component can have multiple instances, then all instances of the component are subject to the *anti-affinity* constraint;
- if a component can have only one instance, then the component is *neutral*; however, if two *neutral* components communicate with each other, then those components are actually constrained by *affinity*.

One limitation of these templates is that they only allow for an application component to be affected by *affinity* or *anti-affinity*, but not both.

5.2.2 Application Representation

Given an application template, the application components can be grouped according to their constraint type. Thus, an application can be represented as a collection of *affinity sets*, *anti-affinity sets*, and a single *neutral set*, where each *affinity* and *anti-affinity set* consists of a single

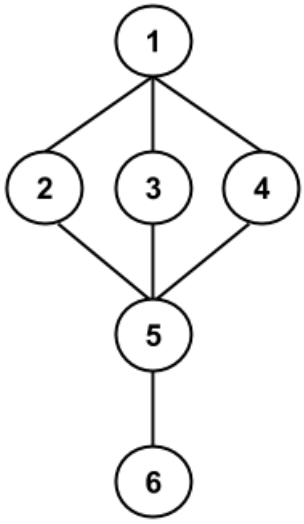

Figure 5.3: Sample Application

group of components related by the associated constraint. This representation of an application is used by the management strategies.

For example, consider the sample application in Figure 5.3, modelled after one of the templates in Figure 5.2. This application can be represented as follows:

- Affinity sets: { { 5 , 6 } }
- Anti-affinity sets: { { 2 , 3 , 4 } }
- Neutral set: { 1 }

5.3 Management Strategies

In the context of dynamic VM management, there are three main operations:

- **Placement.** When a new VM has to be instantiated in the data centre, a physical server must be selected to host the VM.
- **Relocation.** When a host becomes stressed (i.e., close to running out of spare resources to allocate to its VMs), one or more of its VMs must be migrated away, so as to free resources locally. For each VM migration, a physical server must be found to become the new host of the migrated VM. This physical server is referred to as *target host*.
- **Consolidation.** The process of relocating VMs in the data centre, so as to concentrate the VMs into as few hosts as possible.

A management strategy defines the behaviour of the data centre management system and is designed to pursue specific management goals. It consists of a set of policies that specify how each of the main management operations are carried out across the data centre. In this section, we describe two management strategies: one that respects applications' placement constraints at all times and another that may violate constraints under certain conditions.

5.3.1 Data Centre Organization and System Architecture

Before describing the management strategies, it is necessary to define the organization of the target data centre and the architecture of the management system for which the strategies were designed.

The target infrastructure consists of a collection of clusters, each cluster being a collection of racks, and each rack a collection of physical servers. Two networks provide connectivity throughout the data centre: the *data network* and the *management network*. The former is used by the client applications, while the latter is reserved for the management system. Both networks have the same architecture. The hosts in a rack are connected to the networks through two switches – one per network – placed inside the rack. The racks in a cluster are connected to each other through a cluster-level switch, and the cluster-level switches are connected to a central switch at data centre-level. Figure 5.1 shows the architecture just described (except that, for simplicity, only one network is shown in the figure).

Each computational entity in the data centre (i.e., hosts, racks, clusters, and the data centre itself) has an associated *autonomic manager*. Managers have a set of policies and a knowledge base. Managers can receive events, which may trigger the execution of zero or more policies, which in turn may trigger management actions or additional events. The knowledge base is updated with monitoring data sent to the manager (as an event) or through policies' execution. Managers communicate with each other periodically (e.g., monitoring) and aperiodically (in response to events).

The management system is organized as a hierarchy of autonomic managers. There are four levels of management: host-level (Level 0), rack-level (Level 1), cluster-level (Level 2), and data centre-level (Level 3). The management strategies consist of policies running at one or more levels. Managers at the same level use the same set of policies.

5.3.2 Management Strategy: Enforced Constraints (MS-EC)

This management strategy was designed to operate at the granularity-level of applications, that is, the strategy maps applications to clusters or racks instead of mapping individual application components or VMs. However, at rack-level, the policies do operate at a lower granularity-

level, mapping (and re-mapping) VMs to hosts as needed, always respecting the VMs' placement constraints as defined by the application they belong to.

Placement

The Placement management operation is carried on by three policies, running at rack-, cluster-, and data centre-level, respectively. When managers receive a *Placement request* (i.e., a request to place a new application), they execute their associated Placement policy. Policies search for candidate entities to which to forward the request (or send VM instantiation events at the lowest level); if none can be found, the request is rejected, and the upper level manager has to start a new search.

When the Data Centre Manager receives a Placement request (or a Placement reject message – see below), its Placement policy parses the list of clusters, removing those that do not meet the hardware requirements of the application, sorts the clusters in decreasing order by power efficiency (i.e., processing per watt of power), and divides them in active (or powered on) and inactive clusters. If there is no active cluster, one is powered on and selected. If there is only one active cluster, the policy checks if the cluster has enough spare resources to host the application. If there are multiple active clusters, the policy checks each subset of equally power efficient clusters, searching for the cluster with the least loaded rack (i.e., least active hosts) and that can host the application, or the cluster with the most active racks, but that still has racks to activate. If no active cluster was identified, the next inactive cluster in line is powered on and selected; if there are no inactive clusters, the Placement request is rejected. If a suitable target cluster was found, the Placement request is forwarded to the cluster's manager.

When a Cluster Manager receives a Placement request, its Placement policy starts by separating active from inactive racks. If there is no active rack, one is powered on and selected. If there is one active rack, the policy checks if the rack has enough spare resources to host the application. If there are multiple active racks, the policy searches the entire list of active racks to identify the rack that can host the application and would activate the least number of additional hosts in so doing; if several such racks exist, the most loaded one (i.e., most active hosts) is selected. If no active rack was found, the next inactive rack in line is powered on and selected; if there are no inactive racks, a Placement reject message is sent to the Level 3 manager. If a suitable target rack was found, the Placement request is forwarded to said rack's manager.

Finally, when a Rack Manager receives a Placement request, its policy first classifies and sorts the available hosts in the rack. It then tries to place all the VMs of the application. First, for each affinity set, the policy tries to map all the VMs in the set into a single host. Second, for each anti-affinity set, the policy tries to map each VM in the set into a different host. Third,

the policy takes the set of neutral VMs and maps the VM to whichever host can take them. In every step, the intention is to maximize the CPU utilization of the hosts in the rack without exceeding a given threshold – hosts' target utilization threshold.

Relocation

Just like the Placement operation, Relocation is achieved through the combined work of three policies, used by managers at rack-, cluster-, and data centre-level. In contrast, the Relocation operation starts at Level 1 and often does not require the involvement of upper management levels. In this operation, managers try to solve stress situations within their scope by performing migrations between computational entities under their control. It is only when a manager cannot deal with a stress situation on its own that the manager requests assistance from its upper level manager.

```
1:  s, p, u, e = classifyHosts(hosts)
2:  p', u' = sortByCpuUtil(p, u)
3:  e' = sortByPowerState(e)
4:  targets = concatenate(p', u', e')
5:  n, x, a = classifyVms(stressed)
6:  if processNeutralVms(n, targets) then
7:      migrateVm()
8:      return true
9:  end if
10: if processAntiAffinityVms(x, targets) then
11:     migrateVm()
12:     return true
13: end if
14: if processAffinityVms(a, targets) then
15:     migrateVm()
16:     return true
17: end if
18: return false
```

Algorithm 8: MS-EC Relocation policy at rack-level – Internal process.

At rack-level, Relocation is a two-step process. When a Rack Manager detects that one of its hosts is stressed, the Relocation policy starts its *internal* relocation process (see Algorithm 8), by which it tries to migrate a VM from the stressed host to a non-stressed host in the rack. The policy first classifies the available *hosts* in the rack as stressed (s), partially-utilized (p), underutilized (u) or empty (e), and sorts them as follows (lines 1-4): p is sorted in increasing order by CPU utilization, u is sorted in decreasing order by CPU utilization, and e is sorted in decreasing order by power state (i.e., *on*, *suspended*, *off*). It then classifies the VMs in the

stressed host (line 5) in three groups according to their placement constraint type – neutral (n), anti-affinity (x) and affinity (a) – and considers each group in order using a *greedy* approach (lines 6-17). First, the policy tries to find the least loaded *neutral* VM that still has enough load to terminate the stress situation and that can be taken by another host in the rack (line 6). If that fails, the policy repeats the process with the group of *anti-affinity* VMs (line 10), checking in addition that no VM is selected to be migrated to a host that is already hosting a VM from the same anti-affinity set (i.e., a VM hosting the same application component). If that step also fails, the policy considers at last the group of *affinity* VMs (line 14), first grouping the VMs into their affinity sets, and then trying to find the smallest affinity set that could be taken by another host in the rack. The first of these three steps that can find a suitable migration issues said migration and terminates the relocation process.

```
1:  l, s = classifyVms(stressed)
2:  VM = processVms(l)
3:  if VM == null then
4:      VM = processVms(s)
5:  end if
6:  if VM != null then
7:      send(AppMigRequest(VM.getApp()), manager)
8:      return true
9:  else
10:     return false
11: end if
```

Algorithm 9: MS-EC Relocation policy at rack-level – External process.

When the *internal* relocation process fails to find a suitable migration, the policy starts its *external* relocation process (see Algorithm 9), so as to migrate an entire application to a different rack. First, the policy divides the VMs in the *stressed* host into two groups, *large* (l) and *small* (s), according to whether the VMs have enough CPU load (i.e., amount of CPU under consumption) to terminate the stress situation or not (line 1). The VMs in the *large* group are processed first (line 2), searching for the VM that belongs to the smallest application (i.e., the application with the least number of components), and selecting the least loaded VM if there is a tie. If no VM is selected, the process is repeated with the VMs in the *small* group (line 4), searching for the VM that belongs to the smallest application, and selecting the most loaded VM if there is a tie. If a suitable VM is found, the Rack Manager requests assistance from its corresponding Cluster Manager (line 7) to find a target rack to which to migrate the application that the chosen VM belongs to. If the Cluster Manager fails to find a suitable placement for the migrated application in the cluster, it will in turn request assistance from the Data Centre Manager to migrate the application to a different cluster in the data centre.

The Relocation policies at Levels 2 and 3 are similar to the Placement policies at those levels, with minor differences: when the policies consider an active rack or cluster as potential migration target, they first verify that the computational entity is not the sender of the migration request, and if it is, the computational entity is skipped.

Consolidation

In contrast with the previous two operations, Consolidation only happens at rack-level. This operation occurs periodically and what the policy attempts to achieve is to empty and power off underutilized hosts by migrating their VMs to hosts with higher utilization. However, all migrations occur within the scope of the rack; in other words, no VM is migrated between racks as a result of a consolidation process. By limiting this operation to the rack scope, we reduce overhead on the management network.

When the rack-level policy is invoked, it starts by classifying the hosts and making two lists: the *sources* list contains all underutilized hosts, while the *targets* list contains all non-stressed hosts. The first list is sorted in increasing order by CPU utilization and the second is sorted in decreasing order by CPU utilization. For each host in the *sources* list, the policy tries to migrate all its VMs into hosts in the *targets* list, starting with *affinity* VMs, then *anti-affinity* VMs, and finally *neutral* VMs, always respecting the constraints in the same way the Relocation policy at rack-level does. VMs are processed in this order according to constraint type, so as to attempt to place the more restrictive VMs first and fail early. If suitable migrations could be found for all VMs in the source host, then the migrations are issued and the host is marked to be powered off once the migrations are completed. Otherwise, no VM is migrated away from this host.

5.3.3 Management Strategy: Relaxed Constraints (MS-RC)

This management strategy differs from **MS-EC** in how the Relocation operation is handled; more specifically, the *external* step of the relocation process.

The Relocation policy at rack-level performs the *internal* relocation process as described for **MS-EC**. However, during the *external* relocation process (see Algorithm 10), this policy does not look to migrate an entire application, but rather to migrate a single VM. This approach should offer the benefit of terminating stress situations while performing fewer VM migrations, though it requires the temporary violation of placement constraints.

The policy sorts the VMs in the stressed host in increasing order by CPU load (though it ignores the VMs with not enough load to terminate the stress situation) (line 1), and traverses the list (lines 2-13), searching for the first single-VM application it can find (lines 3-6). In case no such VM is found in the list, the policy also identifies the first non-affinity VM (*na*)

```
1:  vms = sortByCpuLoad(stressed)
2:  for VM in vms do
3:     if VM.getAppSize() == 1 then
4:        send(AppMigRequest(VM.getApp()), manager)
5:        return true
6:     end if
7:     if na == null and VM.getConstraintType() != affinity then
8:        na = VM
9:     end if
10:    if a == null and VM.getConstraintType() == affinity then
11:       a = VM
12:    end if
13: end for
14: if na != null then
15:    send(VmMigRequest(na), manager)
16:    return true
17: else if a != null then
18:    send(VmMigRequest(a), manager)
19:    return true
20: end if
21: return false
```

Algorithm 10: MS-RC Relocation policy at rack-level – External process.

(i.e., *neutral* or *anti-affinity*) (lines 7-9) and the first *affinity* VM (*a*) (lines 10-12). If a single-VM application is found, the Rack Manager requests assistance from its corresponding Cluster Manager to migrate away the application (line 4). Otherwise, the Rack Manager requests assistance from the Cluster Manager to migrate away the non-affinity VM identified (line 15), or if no such VM was identified, to migrate away the *affinity* VM (line 18).

It is easy to see that if a single-VM is found and migrated, no constraints are violated. However, if a VM that is part of a larger application is chosen for migration, then this action will violate the *single-rack* constraint. In addition, if the migrated VM is the one related by *affinity*, then the system will violate this other constraint as well.

The Relocation policy was not only modified to allow for the violation of placement constraints, but also to correct these situations. Given an application that has one or more of its VMs hosted remotely, the Rack Manager contacts the corresponding remote Rack Managers to request current resource consumption information about the VMs and see if the VMs can be hosted back in the local rack, respecting the placement constraints of the application. If a suitable local target host can be found for any of the VMs, the migration is issued. This procedure is invoked every hour, starting one hour after a VM was migrated away, and it continues until the VM is recovered.

5.4 Evaluation

We evaluate the two management strategies proposed in Section 5.3 through simulations using DCSim [20, 21], a tool designed to simulate a multi-tenant, virtualized data centre. In this section, we describe the experimental setup and design, list and explain the metrics used for evaluation, and discuss the results obtained.

5.4.1 Experimental Setup

We created a simulated infrastructure consisting of 5 clusters, with each cluster containing 4 racks, and each rack containing 10 hosts. The hosts were modelled after the HP ProLiant DL160G5 server, with 2 quad-core 2.5GHz CPUs (2500 CPU shares per core) and 16GB of memory. As many hypervisors nowadays, the hosts make use of a work-conserving CPU scheduler, which means that CPU shares not used by one VM can be used by another VM. However, CPU caps are not supported. If the total CPU demand of co-located VMs exceeds the CPU capacity of their host, CPU shares are divided among VMs in a fair-share manner. Memory is statically allocated and is *not* overcommitted. Network switches were modelled after the power measurement study conducted by Mahadevan et al. [22]. Rack switches have 48 1-Gpbs ports and have a power consumption of 102 Watts. Cluster and data centre switches have 48 1-Gpbs ports and consume 656 Watts.

We defined three different *VM sizes*:

1) 1 virtual core of 1500 CPU shares, 512MB RAM

2) 1 virtual core of 2400 CPU shares, 1024MB RAM

3) 2 virtual core of 2400 CPU shares, 1024MB RAM

Note that these are maximum resource requirements. At runtime, however, VMs are allocated enough resources to meet their current demand, not their maximum requirements.

We created five different application types based on the templates defined in Section 5.2.1. At the start of the simulation, applications are randomly assigned a VM size, which determines the resource requirements of their components (and in that way, the size of the VMs that host those components). We did not allow for *application elasticity*, meaning that once an application is deployed in the data centre, the number of its components stays fixed. The application types created are shown in Table 5.1 with their respective configuration.

The application model used is that of an interactive, multi-tiered web application. In this model, a number of clients issue requests to an application, wait for a response, and issue

App. Type	Task Id	Service Time (s)	Visit Ratio	#Instances
1	1	0.03	1	1
2	1	0.02	1	1
	2	0.015	1	1
3	1	0.02	1	1
	2	0.015	1	1
	3	0.015	1	1
4	1	0.01	1	1
	2	0.02	$\#Instances/2$	2..4
	3	0.008	1	1
	4	0.007	1	1
5	1	0.01	1	1
	2	0.04	$\#Instances/4$	4..6
	3	0.01	1	1
	4	0.02	$\#Instances/2$	2..3
	5	0.01	1	1

Table 5.1: Applications

follow-up requests. Applications are modeled as a closed queueing network, solved with Mean Value Analysis (MVA). Applications have an associated *think time*, which is the time clients wait between receiving a response to a request and producing a follow-up request, and a *workload*, which is the number of clients currently using the application. Workloads change over time, according to trace data from an input file. Individual tasks – the term used in DCSim to refer to application components – have their own configuration parameters: *service time* indicates the time it takes for the task to process a request, while *visit ratio* indicates the number of times the task is invoked by a single request. If a task instance does not have its resource demand met (due to its host being stressed), its service time is incremented to account for processor queueing, which would impact the application's response time, potentially causing SLA violations. When there are multiple instances of a task, the load (i.e., the requests) is shared equally between the instances.

All applications were configured with a think time of 4 seconds and were randomly assigned a trace built from one of three sources: *ClarkNet*, *EPA* or *SDSC* [23]. Each of these real workload traces was processed and normalized request rates calculated, in 100-second intervals. These values were used to indicate the number of clients using the application over time. The normalized workloads were scaled so that, when the number of clients was at its peak, the application's response time was 0.9 seconds (just below the 1-second response time threshold associated with SLA violations – see Section 5.4.3). Each application was assigned a random offset to start reading its associated trace, so as to prevent applications with the same trace from

exhibiting synchronized behaviour.

5.4.2 Experimental Design

To evaluate the management strategies, we ran each strategy under four different scenarios. Each scenario consisted of a subset of the five application types described in the previous section. Each number in the set corresponds to an application type listed in Table 5.1. The four scenarios used were the following:

1) **Set A:** { 2 }

2) **Set B:** { 2, 3 }

3) **Set C:** { 2, 3, 4 }

4) **Set D:** { 2, 3, 4, 5 }

The set of applications to submit to the data centre consisted, in every scenario, of 1,200 applications divided equally between the available application types. All generated applications were submitted to the data centre within the first 5 days of simulation at a rate of 10 applications per hour. During this period metrics were not recorded. The system was given 1 additional day to stabilize itself before recording metrics. After that time, metrics were collected for 7 days, and then the simulation was terminated.

We define a *workload pattern* as a randomly-generated sequence of application submission times and random offsets, which is reproducible by specifying the random seed used to generate the pattern. For this experiment, we generated five different workload patterns and ran each scenario once per pattern. Results were averaged across scenarios.

A second experiment was conducted with slighly different scenarios. While the scenarios in the first experiment consisted solely of multi-VM applications, every scenario in the second experiment included single-VM applications (that is, the scenarios included Application Type 1). The purpose of this experiment was to evaluate how the management strategies performed when *unconstrained* applications were included. The four scenarios used were the following:

1) **Set A':** { 1, 2 }

2) **Set B':** { 1, 2, 3 }

3) **Set C':** { 1, 2, 3, 4 }

4) **Set D':** { 1, 2, 3, 4, 5 }

Finally, host managers were configured to send status updates every 2 minutes, and rack and cluster managers to do so every 5 minutes. Hosts were considered stressed when their CPU utilization exceeded 90% (non-stressed otherwise), while they were considered underutilized when their CPU utilization fell below 60%. The target utilization was set at 85%.

5.4.3 Metrics

- **Active Hosts (Hosts):** The average number of hosts powered on. The higher the value, the more physical hosts are being used to run the workload.

- **Average Active Host Utilization (H. Util.):** The average CPU and memory (MEM) utilization of all powered on hosts. The higher the value, the more efficiently resources are being used.

- **Power Consumption (Power):** Power consumption is calculated for each host and switch, and the total kilowatt-hours consumed during the simulation are reported. Hosts' power consumption is calculated using results from the SPECPower benchmark [24], and is based on CPU utilization. Switches' power consumption is assumed constant and independent of network traffic – as long as the switch is powered on – and is based on the power benchmarking study by Mahadevan et al. [22].

- **SLA Achievement (SLA):** SLA Achievement is the percentage of time in which the SLA conditions are met. We define the SLA for an application as an upper threshold on its response time – set at 1.0 seconds. While the response time stays below the threshold, we consider the SLA *satisfied*; otherwise, the SLA is *violated*. Response times exceeding the threshold are a consequence of underprovisioned CPU resources to a VM (or application component), as a consequence of CPU contention with other VMs on the host.

- **Number of Migrations (Migrations):** The number of VM migrations triggered during the simulation (due to both Relocation and Consolidation). Typically, a lower value is preferable, since fewer migrations means less network overhead.

- **Applications Deployed (Apps.):** The number of applications successfully deployed in the data centre. Since application submissions can be rejected, this number may be lower than the total number of submissions.

- **VMs Instantiated (VMs):** The total number of VMs instantiated in the data centre.

- **Spread Penalty (Spread):** The Spread Penalty is calculated as the amount of time, measured in hours, that an application had its components distributed across multiple racks. This metric indicates the extent to which the *single-rack* constraint (defined in Section 5.2) has been violated by the management system.[2] We report the mean Spread Penalty (**Mean**) calculated over all the applications in the data centre, as well as the percentage of application with non-zero Spread Penalty (**Apps**), calculated over the total number of applications in the data centre.

5.4.4 Results and Discussion

The results of the first experiment (presented in Table 5.2) show that both strategies use similar number of hosts and achieve equally high host resource utilization. Power consumption and SLA achievement metrics are also very close in both strategies. Regarding migrations, the strategies issue similar numbers – in two scenarios, the difference is negligible, and in the other two, the difference is 6% and 9%, respectively. In other words, **MS-RC** does not provide any major advantage over **MS-EC**. On the contrary, **MS-RC** suffers the disadvantage that, by temporarily violating constraints, it degrades the performance of the affected applications. In this experiment, we see that about 70% of the applications deployed in the data centre have the *single-rack* placement constraint violated at least once during their lifetime (see column **Spread (Apps)**). In addition, the average amount of time that affected applications spend with their components spread across multiple racks (over their 7-day lifetime) varies between 23.6 and 42.8 hours.

The results of the second experiment (presented in Table 5.3 and Figure 5.4) allow for the same observations to be made with regard to active hosts, host resource utilization, power consumption and SLA achievement. However, when we look at the number of migrations, we see that **MS-RC** consistently issues more migrations than **MS-EC** (8.6% to 41.4% more). As for the spread penalty, the percentage of applications with violated constraints varies between 20.7% and 51.4%, and the average amount of time applications are affected by this situation varies between 5.7 and 15.9 hours.

A few additional observations can be made by looking at the results of both experiments. First, both strategies issue fewer migrations in the second experiment than in the first one (up to 40% less) – with the exception of **MS-RC** in the fourth scenario. Second, **MS-RC** violates constraints less often in the second experiment (20.7% to 51.4% affected applications against about 70%) and the amount of time applications remain with their constraints vio-

[2] Since DCSim is not able to simulate application performance degradation due to network congestion, we use this metric to approximate the extent to which applications are adversely affected by having their components spread over the data centre.

Strategy	App Set	Hosts	H. Util. (CPU)	H. Util. (MEM)	Power	SLA	Migrations	Apps.	VMs	Spread (Mean)	Spread (Apps)
MS-EC	**A**	139.8	69.0	83.1	6,737.8	99.7252	62,914.8	1,116.0	2,232.0	-	-
MS-EC	**B**	172.1	67.5	84.2	7,839.6	99.6568	75,832.6	1,115.6	2,785.0	-	-
MS-EC	**C**	198.9	73.1	92.7	8,936.3	99.2466	81,636.2	1,004.0	3,555.6	-	-
MS-EC	**D**	197.9	71.5	94.3	8,860.6	99.3208	55,056.4	842.0	3,667.8	-	-
MS-RC	**A**	132.0	71.9	86.8	6,512.5	99.6554	61,957.8	1,101.6	2,203.6	39.8	69.6
MS-RC	**B**	160.0	70.9	89.0	7,490.9	99.5673	69,727.8	1,094.3	2,735.0	42.4	72.1
MS-RC	**C**	189.4	72.8	92.5	8,589.1	99.2654	79,184.6	954.6	3,384.6	28.2	68.7
MS-RC	**D**	198.9	71.9	95.5	8,905.1	99.0890	61,693.8	847.6	3,720.2	23.8	70.3

Table 5.2: Exp. 1 – Multi-VM Applications

Strategy	App Set	Hosts	H. Util. (CPU)	H. Util. (MEM)	Power	SLA	Migrations	Apps.	VMs	Spread (Mean)	Spread (Apps)
MS-EC	**A'**	106.3	70.9	81.8	5,576.0	99.7638	35,121.0	1,116.4	1,671.4	-	-
MS-EC	**B'**	139.1	69.2	84.1	6,713.0	99.6990	48,624.4	1,116.8	2,246.0	-	-
MS-EC	**C'**	193.0	72.0	88.3	8,701.4	99.5938	61,417.0	1,104.2	3,285.4	-	-
MS-EC	**D'**	197.8	71.8	92.6	8,869.8	99.6196	47,498.8	962.2	3,568.0	-	-
MS-RC	**A'**	104.7	71.8	82.9	5,533.6	99.7828	38,981.8	1,114.6	1,669.0	6.4	22.5
MS-RC	**B'**	135.4	70.6	85.9	6,608.7	99.6916	53,352.0	1,109.4	2,232.8	16.0	40.9
MS-RC	**C'**	186.3	72.8	89.4	8,477.1	99.4846	76,976.0	1,073.6	3,204.8	11.7	46.4
MS-RC	**D'**	196.1	72.2	93.1	8,817.0	99.4866	65,477.0	955.2	3,558.8	13.5	52.0

Table 5.3: Exp. 2 – Single- and Multi-VM Applications

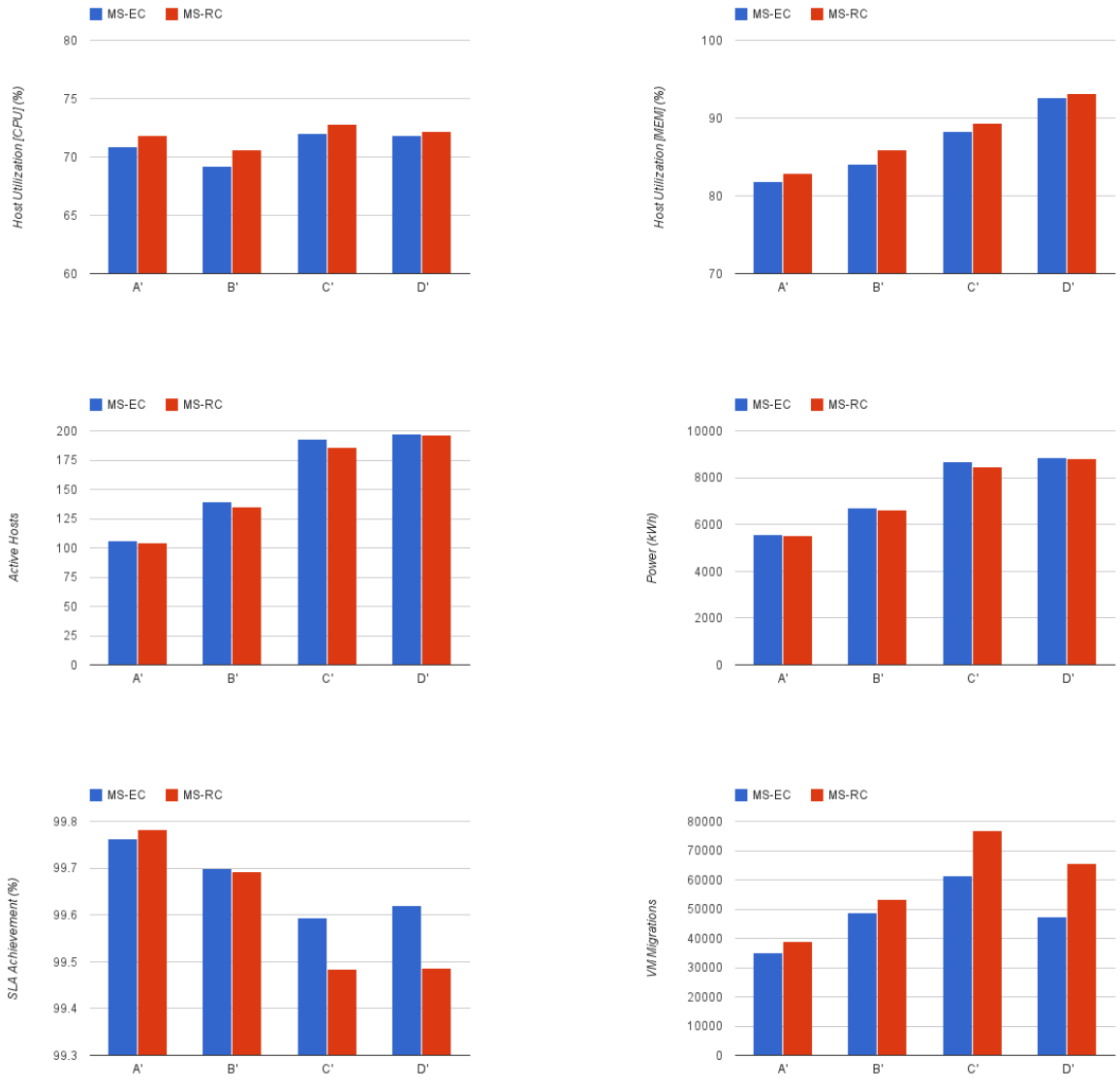

Figure 5.4: Exp. 2 – Single- and Multi-VM Applications

lated is considerably smaller. Both of these observations can be explained by the presence of single-VM applications in the second experiment. Regarding the smaller number of migrations, both strategies always try to prioritize smaller applications during Relocation: **MS-EC** always choses the application with the fewest components available, so whenever there is a single-VM application among the candidates, that application will be relocated; and **MS-RC**, before violating a constraint, checks whether a single-VM can be found in the stressed host, so as to relocate that application instead. This latter behaviour of **MS-RC** also explains the decrease in the number of applications affected by violated constraints.

5.5 Conclusions and Future Work

In this paper, we addressed the issue of managing multi-VM application with placement constraints in data centres. We developed a management strategy for a hierarchical management system to place, relocate and consolidate this type of applications, satisfying at all times the applications' placement constraints. In addition, we developed a variant of the original management strategy to allow for constraints to be temporarily violated. The experiments showed that the first management strategy could satisfactorily deal with applications' placement constraints, while at the same time achieving high levels of resource utilization and SLA achievement. While the second strategy performed equally well with regard to resource utilization and SLA achievement, it tended to issue more migrations. In addition, the second strategy causes application performance degradation when violating placement constraints. Therefore, the second strategy provides no advantages over the first strategy, while adding an extra cost.

There are several ways in which this work could be extended. The most immediate one would be to expand the set of available application templates, be it with multi-tier applications or other types of applications altogether. A more general approach to the task of generating multi-VM applications would be to do so randomly, that is, to randomly generate the number of components in an application and the connections between them. One potential difficulty with this approach is that the generated applications could be far from any real-world application architecture and in that way be of little usefulness in practice.

Another direction to explore is to allow for *application elasticity*, that is, for applications to change their size (i.e., number of components) at runtime to match current service demand. In this situation, would enforcing constraints at all times still be possible (and desirable), or would violating constraints become a necessity? In other words, would the conclusions reached in this study still hold in the new context? This idea goes in line with the latest work from Tighe and Bauer [19].

Another issue to explore is violating other placement constraints besides the *single-rack*

one. This work considers violating the *single-rack* constraint during relocation. What if the *affinity* or *anti-affinity* constraints were to be temporarily violated during (internal) relocation or consolidation? What effect would this management approach have in resource utilization, power consumption, SLA achievement, and number of migrations? This approach would require the definition of appropriate penalties for these constraint violations.

Finally, it would be interesting to investigate using some form of relocation to assist the Placement operation. In this work, when an application is submitted to the data centre, the Placement operation tries to map the application components to hosts in a single rack. If that fails, the application submission is rejected. However, it would be interesting to consider, after the failure of a straightforward deployment, moving VMs around so as to make room for the incoming application instead of rejecting the submission outright.

Bibliography

[1] G. Keller and H. Lutfiyya, "Dynamic management of applications with constraints in virtualized data centres," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, May 2015, (accepted to appear).

[2] G. Khanna, K. Beaty, G. Kar, and A. Kochut, "Application performance management in virtualized server environments," in *NOMS Proceedings, 2006 IEEE/IFIP*, 2006.

[3] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations," in *IM Proceedings, 2007 IEEE/IFIP Int. Symp. on*, 2007, pp. 119–128.

[4] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper, "An integrated approach to resource pool management: Policies, efficiency and quality metrics," in *38th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, June 2008.

[5] A. Verma, P. Ahuja, and A. Neogi, "pmapper: power and migration cost aware application placement in virtualized systems," in *Proceedings of the 9th ACM/IFIP/USENIX Int. Conf. on Middleware*, 2008.

[6] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D. Gmach, R. Gardner, T. Christian, and L. Cherkasova, "1000 islands: Integrated capacity and workload management for the next generation data center," in *Proceedings of the 2008 International Conference on Autonomic Computing (ICAC'08)*, Chicago, IL, USA, Jun. 2008, pp. 172–181.

[7] M. Cardosa, M. R. Korupolu, and A. Singh, "Shares and utilities based power consolidation in virtualized server environments," in *IM Proceedings, 2009 IEEE/IFIP Int. Symp. on*, 2009.

[8] B. Speitkamp and M. Bichler, "A mathematical programming approach for server consolidation problems in virtualized data centers," *IEEE TSC*, vol. 3, no. 4, pp. 266 –278, 2010.

[9] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource allocation algorithms for virtualized service hosting platforms," *J. Parallel Distrib. Comput.*, vol. 70, no. 9, pp. 962–974, Sep. 2010.

[10] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Gener. Comput. Syst.*, vol. 28, no. 5, 2012.

[11] G. Keller, M. Tighe, H. Lutfiyya, and M. Bauer, "An analysis of first fit heuristics for the virtual machine relocation problem," in *Network and Service Management (CNSM), 2012 8th International Conference on*. IEEE, Oct. 2012, pp. 406–413.

[12] (2014) AWS Reference Architectures. Amazon Web Services, Inc. [Online]. Available: http://aws.amazon.com/architecture/

[13] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees," in *Proceedings of the 6th International COnference*. ACM, 2010, p. 15.

[14] M. G. Rabbani, R. P. Esteves, M. Podlesny, G. Simon, L. Z. Granville, and R. Boutaba, "On tackling virtual data center embedding problem," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013, pp. 177–184.

[15] M. Zhani, Q. Zhang, G. Simon, and R. Boutaba, "Vdc planner: Dynamic migration-aware virtual data center embedding for clouds," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, 2013, pp. 18–25.

[16] A. Gulati, G. Shanmuganathan, A. Holler, C. Waldspurger, M. Ji, and X. Zhu, "Vmware distributed resource management: design, implementation, and lessons learned," *VMware Technical Journal*, vol. 1, no. 1, 2012.

[17] V. Shrivastava, P. Zerfos, K.-W. Lee, H. Jamjoom, Y.-H. Liu, and S. Banerjee, "Application-aware virtual machine migration in data centers," in *INFOCOM, 2011 Proceedings IEEE*, 2011, pp. 66–70.

[18] L. Shi, B. Butler, D. Botvich, and B. Jennings, "Provisioning of requests for virtual machine sets with placement constraints in iaas clouds," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013, pp. 499–505.

[19] M. Tighe, “Advances in dynamic virtualized cloud management,” Ph.D. dissertation, The University of Western Ontario, 2014.

[20] M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya, “Towards an improved data centre simulation with DCSim,” in *Network and Service Management (CNSM), 2013 9th International Conference on*. IEEE, Oct. 2013, pp. 364–372.

[21] (2014) DCSim on GitHub. Distributed and Grid Systems (DiGS). [Online]. Available: https://github.com/digs-uwo/dcsim

[22] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan, “A power benchmarking framework for network devices,” in *Proceedings of the 8th International IFIP-TC 6 Networking Conference*. Springer-Verlag, 2009, pp. 795–808.

[23] (2014) The Internet Traffic Archive. [Online]. Available: http://ita.ee.lbl.gov/

[24] (2014) SPECpower_ssj2008. Standard Performance Evaluation Corporation. [Online]. Available: http://www.spec.org/power_ssj2008/

Chapter 6

Conclusion

Cloud Computing is changing the way in which software is being designed, deployed and used. Adoption of this paradigm has already been substantial [1] and it is predicted that by 2017 nearly two thirds of all workloads will be processed in the cloud [2]. These workloads running in the cloud are actually hosted in virtualized data centres, making of these data centres complex systems to manage.

This work addressed the issue of dynamic VM management in IaaS-type data centres. More specifically, we addressed the following research challenges:

- **Virtual Machine Relocation:** *does changing the order in which VMs and hosts are considered for migration have any effect on the long-term management goals of a data centre?*
- **Multi-goal Management Strategies:** *is it possible to design a management strategy to both minimize SLA violations and minimize power consumption, while giving both goals equal priority?*
- **Management Systems' Architecture:** *is it possible to design a scalable management system that makes efficient use of the data centre's resources (e.g., CPU, network, power)?*
- **Multi-VM Application Management:** *how can we manage multi-VM applications, so as to satisfy their placement constraints and at the same time maximize resource utilization and minimize SLA violations?*

6.1 Discussion

In Chapter 2, we addressed the VM Relocation problem. We designed a set of policies that implemented the VM Relocation operation, where all the policies were based on the same

First Fit heuristic and differed from each other in the order they considered VMs and hosts for migration. We evaluated the policies under different load-levels and compared the results in terms of active hosts, host utilization, power consumption, dropped requests (SLA violation), and number of migrations. We determined that no one policy scored best in every metric; moreover, the policies succeeded to different extents depending on data centre state (i.e., load-level) and the metrics considered. From this work follow a couple of implications: first, policies should be customized according to data centres' management goals, so as to achieve the most desirable trade-offs; and second, no single policy will be able to meet all management goals or achieve consistent results under different data centre conditions. This last point suggests that a management system might be better served by a set of policies (that perform the same operation) that can be swapped based on data centre state.

In Chapter 3, we focused on the issue of Multi-goal Management Strategies. We proposed the use of a *meta-strategy*, that is, a mechanism that dynamically switched between two base management strategies (each one designed to pursue one of the desired goals) according to data centre state. We developed three different meta-strategies, and evaluated them through simulation, jointly with the base single-goal SLA and Power strategies, and a hybrid strategy designed to pursue both goals simultaneously. The experiments showed that dynamic strategy switching (i.e., the meta-strategies) improved overall performance (in terms of power consumption and SLA violations) compared to all three single management strategies. This work suggests that pursuing multiple goals (even opposed ones) with equal intensity is achievable through the use of multiple management strategies. Moreover, changing management strategies at runtime based on data centre state makes for a more adaptable management system.

In Chapter 4, we addressed the issue of Management Systems' Architecture. We designed a hierarchical management system, leveraging the topology of the data centre network to organize the hierarchy of autonomic managers. We defined aggregate metrics to convey system state information across management levels, and defined managers' responsibilities and interactions through the implementation of management policies. The system was evaluated through simulation, jointly with the centralized system presented in Chapter 3. Thought the hierarchical management system could not make as efficient a use of CPU and power as the centralized system did, the hierarchical system managed to issue about half the number of migrations and to greatly reduce the flow of management data across the data centre network, thus considerably reducing network overhead. This work shows that hierarchical, topology-aware systems are highly effective at limiting the flow of management data (VM migrations, monitoring, etc.) in data centres. In addition, it shows that the approach of treating a data centre as a single pool of resources makes for an inefficient use of its network.

Finally in Chapter 5, we addressed the issue of Multi-VM Application Management. We

developed a couple of strategies for a hierarchical system to manage multi-VM applications with placement constraints in data centres, while aiming to increase resource utilization (thus reducing power consumption) and decrease SLA violations. One of the management strategies enforced placement constraints at all times, while the other allowed for the constraints to be temporarily violated during VM Relocation. The strategies were evaluated under various application sets consisting of applications with different placement constraints. Both strategies achieved high levels of resource utilization and SLA achievement, suggesting that violating constraints was not worth the cost (in terms of application performance degradation). This research indicates that it is possible to handle multi-VM applications with placement constraints and still achieve high levels of resource utilization and SLA achievement. It is worth noticing, though, that the management task is easier to accomplish when the application set includes unconstrained applications.

6.2 Limitations

Some limitations present in our earlier works where removed in subsequent works (e.g., working only with homogeneous test environments). However, other limitations still remain. In this section, we discuss some of those limitations, in no particular order.

First, our work relies solely on CPU measurements to determine the level of load of a host or VM (though all resources are considered when trying to map a VM to a host). For example, hosts are classified during management operations according to their CPU utilization. It might be desirable to expand our definition of load to encompass other resources, so as to avoid, for example, trying to consolidate an underutilized hosts that has its memory fully allocated (and would therefore result in an expensive operation in terms of network overhead). On a related note, we also use CPU utilization only to determine stress situations in hosts. However, this approach does not constitute a problem because CPU is the only resource that we oversubscribe and hence can cause SLA violations if the total CPU demand of co-located VMs exceeds the total CPU capacity of the host.

Second, the process of migrating a VM imposes an overhead in both source and target hosts. This overhead is accounted for by penalizing the application running inside the migrating VM: the application loses 10% of its processing capacity during migration. In other words, during migration, a VM is only able to use 90% of its CPU allocation for work processing; the other 10% is lost as migration overhead. Neither memory nor bandwidth overhead is considered at the hosts due to migration.

Third, we do not measure or account for network traffic in the data centre. That is, we do not know the load of links and devices on the network. As a consequence, VM migrations are

completed without delay (which may not happen in real environments). On a related note, our management strategies seem to issue large numbers of migrations. While migration overhead is considered and reflected in the SLA violations and host utilization metrics, we are unaware whether this migration count is acceptable or even possible to sustain.

Fourth, we do not do forecasting of VMs' resource demand, which could have an effect in the detection of stress situations and in the selection of VMs and hosts for migration.

Fifth, our test environments may not be representative of high-end data centres nowadays. However, it is difficult to get information about workload characteristics or hardware specifications from industry.

Sixth, the simulation framework used in our experiments, DCSim, is not able to simulate application performance degradation due to network congestion. It is for this reason that a *Spread Penalty* metric was included in the experiments of Chapter 5, so as to circumvent this limitation.

Seventh, our work was evaluated solely through simulation. This is perhaps one of the biggest limitations of our work, and also one of the most difficult to overcome. Though it would be highly informative and extremely useful to evaluate our work in a real data centre, obtaining access to a data centre and acquiring a diverse set of traces is difficult. As a consequence, most work in the area is evaluated through simulation (e.g., [3, 4, 5]).

6.3 Future Work

Dynamic VM Management in data centres is a vast area of research, so there are many directions in which this work could be extended. In this section, we discuss a series of issues that could be investigated.

6.3.1 Hierarchical Management

Some of the most interesting issues stem from our work on hierarchical management (Chapter 4). One such issue has to do with determining how fast monitoring data becomes stale and whether that is a problem at all. In this work, we defined aggregate metrics to convey system state information to higher management levels so that decisions could be made. The system state conveyed in those metrics could change as result of a management decision, making said data invalid or *stale*, and any further decisions made using that data could compromise the quality of the decision.

Another issue is how to detect *VM migration thrashing*. We proposed a hierarchical management system, which was designed – jointly with its policies and strategies – to be topology-

aware, making decisions that would minimize the flow of management data and thus reduce the overhead on the data centre management network. As a result, most VM migrations happen within the scope of a rack – 95% to 99% of the migrations, according to our experiments. This behaviour does reduce the overhead on the management network, but it may also cause an increase in the number of migrations issued when all the hosts in a rack present high resource utilization. It may be the case that an stressed host migrates a VM to another host in the rack, which soon after becomes stressed itself.

One task that we left pending was extending the scope of the VM Consolidation operation beyond the rack. Both VM Placement and VM Relocation involved policies at every level in the hierarchy; VM Consolidation did not. A further issue to explore regarding VM Consolidation would be the use of optimization techniques, such as Integer Programming or genetic algorithms, at rack-level. Given the small number of hosts and VMs enclosed in a rack, it may be possible to obtain optimized solutions in a reasonable amount of time.

Another issue to explore is further modifying the architecture of the management system: take the hierarchy, remove its highest management level (i.e., the single data centre manager), and organize all cluster managers in a peer-to-peer fashion. These change would increase data flow (due to cluster-to-cluster communication not needed before) and could degrade the quality of management decisions (lightly, if at all), but would not affect the scalability of the system and it would rid the system from a single point of failure.

6.3.2 Placement Constraints

In Chapter 5, we explored the issue of managing multi-VM applications with placement constraints. This work, however, considered only interactive, multi-tiered web applications. It would be interesting to expand the repertoire of application templates to include different application architectures.

The placement constraints considered in this work included *single-rack*, *affinity* and *anti-affinity*. However, placement constraints can also be used to specify an application component's need for specific hardware or software. This kind of constraints would signify an additional level of challenge during VM Management.

Finally, it would be interesting to investigate using some form of relocation to assist the Placement operation. In this work, when an application is submitted to the data centre, the Placement operation tries to map the application components to hosts in a single rack. If that fails, the application submission is rejected. However, it would be interesting to consider, after the failure of a straightforward deployment, moving VMs around so as to make room for the incoming application instead of rejecting the submission outright.

6.3.3 Complementary Work

There are also some issues to explore that would complement our work. The first has to do with the resource utilization thresholds that we use to determine whether a host is stressed or underutilized. These thresholds are normalized values in the range [0..1], independent of the resource capacity of the hosts to which they are applied. These results in hosts being deemed stressed when they still have more spare capacity than a host only deemed partially utilized, or in hosts being marked as underutilized while hosting more load than another host marked as partially utilized. For this reason, utilization thresholds cannot be independent of the resource capacity of the hosts.

Speaking of utilization thresholds, these values are defined based on hosts' CPU utilization. Our policies focus mainly on CPU utilization to determine whether a host is stressed or underutilized, and to select VMs for migration or hosts as migration targets. Other resources, such as memory, bandwidth and storage, are mostly ignored, only considered when we have to determine whether a host has enough spare resources to accept a new VM.

Another issue to explore would be that of selecting VMs for migration in such a way to reduce per VM (or application) penalty. When VMs are migrated or their resource demand is not met, they accrue a penalty, which is used to measure their level of SLA achievement or compliance. In order to reduce this *application penalty*, when searching for VMs to relocate, we could choose to migrate the least penalized VM (that is still useful to migrate). This could improve the overall SLA achievement of the management system.

6.3.4 New Approaches to VM Management

There are some other issues to explore that would mean somewhat of a departure from our current line of work. One of it is to consider Quality of Service classes for the applications that are deployed to the data centre. This categorization could be leveraged in any number of ways. For example, when having a stress situation, policies could always select for migration VMs from the lowest class, so as to not disrupt the performance of higher-class VMs, or instead of issuing a migration, policies could take resources away from lower-class VMs and allocate them to higher-class VMs.

The other issue to explore is the implementation of a hybrid approach to VM Management, combining dynamic management with an offline module that would create mappings of VMs into hosts based on complementary resource demand traces. The idea would be to logically divide the set of hosts in the data centre in two groups: *sandbox* and *stable*. The membership of these groups would be dynamic, meaning that a host could belong to one group and then to the other. When a VM (or application) is submitted to the data centre, the VM is dynamically

mapped to a host in the sandbox. From the moment the VM is placed, its resource demand is monitored and stored as a trace. After a period of time, once enough history is known of the VM, the VM is moved out of the sandbox and into a host in the stable group. This new mapping is developed by the offline module, leveraging VMs' history and using some optimization technique, such as Integer Programming or genetic algorithms. If at a later point the VM's resource demand moves considerably away from its predicted values, the system would migrate the VM away of its host in the stable group and back into a host in the sandbox, where a new trace would start to be built. This idea is inspired in the work of Gmach et al. [4].

6.4 Conclusion

This work has addressed the issue of Virtual Machine Management in IaaS-type data centres from four different directions and made contributions in each. First, we explored a variety of First Fit heuristics applied to the VM Relocation problem and determined that a single policy will not be able to satisfy all management goals or perform consistently under different data centre conditions. Second, we worked on pursuing multiple management goals with equal intensity, which we achieved by dynamically switching management strategies according to data centre state. Third, we designed a hierarchical management system that, by virtue of being organized to match the topology of the data centre network, could greatly reduce network overhead. Fourth, we developed management strategies to handle multi-VM applications with placement constraints and still achieve high levels of resource utilization and SLA achievement.

Bibliography

[1] "RightScale 2014 state of the cloud report," RightScale, Inc., 2014.

[2] "Cisco global cloud index: Forecast and methodology, 2012-2017," White Paper, Cisco, 2012.

[3] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations," in *IM Proceedings, 2007 IEEE/IFIP Int. Symp. on*, 2007, pp. 119–128.

[4] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper, "An integrated approach to resource pool management: Policies, efficiency and quality metrics," in *38th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, June 2008.

[5] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Gener. Comput. Syst.*, vol. 28, no. 5, 2012.

Appendix A

DCSim

The scale of data centres providing Cloud services continues to increase, with thousands to tens-of-thousands of servers to manage. This presents a unique challenge to researchers developing methods and algorithms for management, as the scale of the target environment precludes the use of a physical testbed. As such, simulation is commonly used for the evaluation of management techniques. Simulation also helps researchers quickly evaluate data centre management algorithms and techniques at a speed and scale not possible with a real implementation.

DCSim (Data Centre Simulator) [2, 3] is a simulation tool for simulating a virtualized data centre operating as an Infrastructure as a Service (IaaS) cloud. To support our research and to provide tools that other researchers can leverage in their work, we have developed DCSim. Its features include event handling and message passing, mechanisms for event callbacks and sequencing, new components to simplify the creation of management systems and to model the communication between them, and a more complete model of the structure of a data centre including racks and clusters. We also introduce classes to help streamline the creation of new experiments, new output options and metrics, and a visualization tool to help provide a new perspective on the behaviour of data centre management methods and systems. Finally, we continue to focus on providing an extensible platform for researchers to extend and adapt to suit their own work.

The remainder of this chapter is organized as follows: Section A.1 presents related work in data centre simulation, Section A.2 describes the architecture, core features and new additions to DCSim, Section A.3 gives some detail on how to configure and run experiments with DCSim, Section A.4 provides an evaluation of the simulator through a demonstration of its use, and Section A.5 presents some conclusions and future work.

[0]This chapter is based on work published in [1].

A.1 Related Work

There are a small set of existing simulation tools available, each with their own strengths, weaknesses, and target environments. GreenCloud [4] is designed to evaluate the energy costs of operating a data centre. It is a packet-level simulator built as an extension to Ns-2 [5], and provides a detailed model of communication hardware and power consumption of each element of the data centre. It does not, however, include modelling of virtualization. MDCSim [6] is designed to simulate a large-scale data centre running a three-tiered web application. It focuses only on evaluating the configuration of each tier, measuring both power and performance metrics. As with GreenCloud, it does not consider virtualization. Furthermore, it is built on a commercial product and is not publicly available.

GDCSim (Green Data Centre Simulator) [7] aims to help researchers fine-tune the interactions between management systems and the physical layout of the data centre, including thermal and cooling interactions with workload placement. This tool does not consider multiple tenants of the data centre, nor does it consider virtualization.

CloudSim [8] simulates a virtualized data centre, with multiple clients operating VMs. However, it implements an HPC-style workload, with *Cloudlets* (jobs) submitted by users to VMs for processing. It can be manipulated to simulate an interactive, continuous workload such as a web server [9], but it lacks a real model of such an application. An extension of CloudSim, NetworkCloudSim [10], considers communication costs between VMs performing parallel computations, but again focuses on HPC-style workloads rather than interactive workloads. Additionally, our work on DCSim adds data centre organization components such as racks and clusters not present in CloudSim.

SimWare [11] targets the modelling of data centre cooling and power costs, including the impact of server fan power consumption as related to the temperature of the data centre, and air travel time from CRACs to servers. Their simulated client workload is based on traces of HPC systems, rather than interactive applications.

DCSim [2, 3] models a virtualized data centre providing IaaS to multiple tenants, with a focus on transactional, continuous workloads, and models such an application using a basic queuing model. It can model replicated VMs sharing incoming workload, as well as dependencies between VMs that are part of a multi-tiered application. It also provides metrics to gauge SLA achievement, power consumption, and other performance metrics that serve to evaluate a data centre management approach or system. Furthermore, DCSim is designed to be easily extended, implementing new features and functionality.

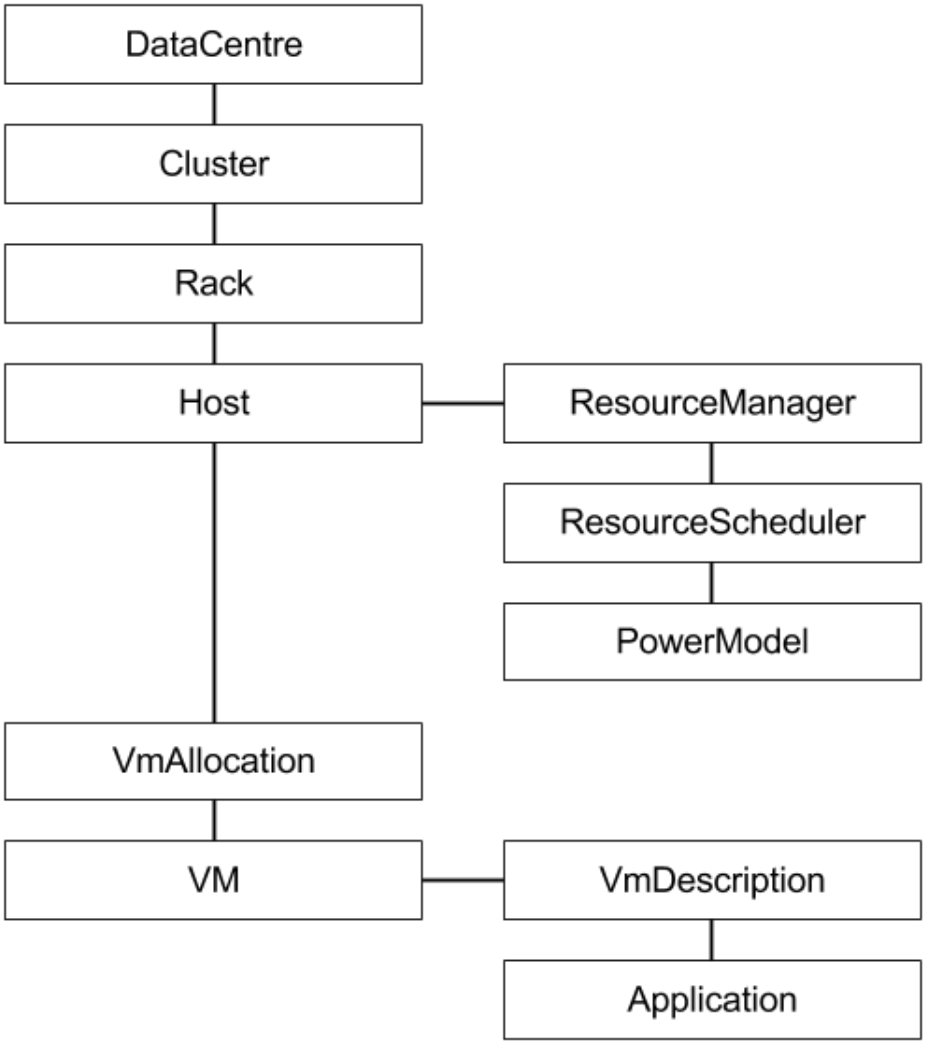

Figure A.1: Data Centre Model

A.2 DCSim Architecture & Components

DCSim is an event-based simulation tool, written in the Java programming language. It is designed to be easily extended, so as to support research in the area of data centre management. Figure A.1 gives a high-level overview of the basic data centre model implemented by DCSim. The remainder of this section outlines the components and underlying mechanisms that drive DCSim.

A.2.1 Simulation Engine

As DCSim is intended to be a simulation platform that can be extended to suit the needs of a particular area of research, it is useful to take a look at the mechanism by which DCSim advances through simulated time in order to help gauge the feasibility of possible extensions. Algorithm 11 outlines a simplified version of the main simulation loop.

```
1:  simTime = 0
2:  while eventQueue not empty && simTime ≤ duration do
3:      scheduleResources()
4:      postScheduling()
5:      e = peek(eventQueue)
6:      simTime = e.getTime()
7:      advanceSimulation(simTime)
8:      updateMetrics()
9:      performLogging()
10:     while eventQueue not empty && peek(eventQueue).getTime() = simTime do
11:         e = pop(eventQueue)
12:         handleEvent(e)
13:     end while
14: end while
```

Algorithm 11: Main Simulation Loop

The *eventQueue* contains all future events that must be executed, *simTime* records the current simulation time, and *duration* is the length of the simulation (in simulation time). The outer loop is responsible for advancing in simulation time to the next scheduled event(s). Changes to data centre state only occur via events; in-between events, the state of the data centre is static. The first phase of the loop uses the current data centre state to *schedule resources* (line 3), in which an allocation of resources/second for each VM is calculated (see Section A.2.6). CPU scheduling is based on current application demands, in a fair-share manner up to the maximum capacity of the host processor. Next, the post-scheduling hook (line 4) allows data centre components can create new events or move existing events based on dynamic resource scheduling. This allows the simulation of operations and processes that exhibit variable runtime based on available resources. This feature is included primarily for the planned future development of variable VM migration times (due to changes in available network bandwidth), and batch/HPC type jobs whose runtime is based on dynamically scheduled CPU. We then check for the simulation time of the next event (which may have changed based on processing in postScheduling()), and *advance* the simulation to the time of the next event using the calculated resource scheduling (lines 6 & 7). Simulation metrics are then updated (see Section A.3.2) and logging is performed. The inner loop (lines 10 to 13) executes all events that take

place at the current simulation time. The process is then repeated to advance to the next set of events.

A.2.2 Events

As DCSim is an event-driven simulation, all actions, operations and state changes in the simulation are triggered by an *Event*. Events are also used for communication between data centre elements and management components. The basic properties of an event are the simulator component(s) that will receive the event, and the time at which to execute it. Events are ordered such that, in the case of multiple events being executed at the same simulation time, they are executed in the order in which they were sent. Any component can send an event to another component, or to itself in order to trigger some functionality at a specific time in the future. The basic *Event* class is abstract, with specific event types implementing any behaviour or storing any data they require.

There are a number of hooks and methods which can be used to add additional functionality to an event. Pre-execution and post-execution methods can be implemented to perform operations before and after the event is executed, such as logging event details. An event callback can also be registered with an event, allowing one or more objects to be notified once an event has been executed. In some cases, an event can cause several other events to be generated in order to complete an operation, which may require the post-execution and callback methods to be triggered only after the complete sequence of events has been executed. To accomplish this, events can be strung together in a sequence. For example, instructing a host (i.e., a physical server; see Section A.2.3) to boot up involves one event sent to the host, and another event sent by the host to itself some time later indicating the completion of the operation – hosts take time to boot. These events are added in sequence together, allowing a management component to receive a callback only once the full boot up operation has completed.

A special event subclass, *MessageEvent*, can be used for communication between components by extending it with any additional functionality required. MessageEvent automatically keeps track of the number of messages of each specific subclass that are sent during the simulation. Finally, a special type of event, called the *RepeatingEvent*, can be used to trigger repeated executions of the event on a regular interval.

A.2.3 VMs, Hosts, Racks & Clusters

DCSim uses a series of abstractions to organize the architecture of a data centre. These abstractions are *VM*, *Host*, *Rack*, *Cluster* and *DataCentre*. In DCSim, a data centre consists of a collection of clusters, each cluster being a collection of racks, and each rack a collection of

hosts. Both Cluster and Rack are designed to be homogeneous collections (in terms of their composing elements), but DataCentre may be an heterogeneous collection.

VM

A VM in DCSim represents a virtual machine running a single application (or application component). The properties and requirements of a VM are defined in its *VMDescription*, which is used to create an instance of the VM. The VMDescription defines the number of virtual cores and the amount of CPU, memory, bandwidth and storage resources requested. In its present state, DCSim allocates memory, bandwidth and storage statically to a VM, in the full requested amount – they are not oversubscribed. CPU resources, however, do not need to be fully allocated, allowing a host's CPU to be oversubscribed. Once a VM is created and started on a host, its CPU requirements are driven dynamically by the needs of the application it is running (See Section A.2.5 for details on applications in DCSim).

In order to perform dynamic management of VMs in a data centre, VMs must be moved from one host to another using VM *live migration*. This mechanism allows a VM to be moved between physical servers with minimal downtime. DCSim supports simulating live migration, and calculates the time to migrate a VM based on available bandwidth and VM memory size.

Host

A host represents a physical machine in the data centre, capable of running VMs. Its physical properties are defined by the following set of attributes: the number of CPUs; the number of cores per CPU; core capacity; memory capacity; network capacity; storage capacity; and a power model. Core capacity is defined in terms of CPU Units, where one CPU unit is equivalent to 1MHz of processor speed (e.g., a 2.4GHz processor has 2400 CPU units). The power model defines how much power the host consumes at a given CPU utilization level, and is calculated using results from the SPECPower benchmark [12]. The resource utilization of the host at any given time is calculated as the sum of the resources in use by the set of VMs it is hosting, including its privileged domain. A host can be in one of three states: `on`, `off`, or `suspended`. VMs are only given resources to run their applications when the host is in the `on` state. The host consumes some small amount of power when in the `suspended` state, and no power when `off`. Transition times between states can be defined in the simulation configuration file.

Rack

A rack represents a collection of hosts in the data centre. This collection is homogeneous; that is, all hosts in the rack are of the same type. A rack has a given number of *slots* that can be filled

with hosts, and this number may vary between racks. A rack counts also with two switches to which every host in the rack is connected.

Cluster

A cluster represents a collection of racks in the data centre. This collection too is homogeneous. The number of racks per cluster is not fixed, so different clusters can have different numbers of composing elements. The cluster also contains two collections of switches, one for the data network and one for the management network (more information on networks in the next section).

A.2.4 Data Centre Network

In DCSim, a data centre has two different networks: a data network and a management network. The first is used to meet the communication needs of the hosted VMs, while the second is used for the internal management of the data centre. VM migrations make use of the management network as do status update messages or migration requests exchanged between management entities.

A network consists of nodes and edges, namely, *NetworkElement* objects and *Link* objects. A Link has a certain bandwidth capacity and it connects two NetworkElement objects. There are two types of NetworkElement: *NetworkCard* and *Switch*.

Every Host has two network cards, one for each network. These network cards are connected through links to their corresponding switch in the rack (two switches per rack, one per network). At cluster-level, two network arrangements are possible: one, every rack in the cluster is connected to a single switch (per network), which is referred to as *main switch* and requires as many ports as there are racks in the cluster; and two, there is a two-level hierarchy of switches (per network), where racks are connected to low-level switches and low-level switches are connected to a single high-level switch (referred to as *main switch*). At data centre-level, there is a central switch (per network) to which each cluster's main switch is connected.

A.2.5 Application Model

Applications in DCSim are modeled after interactive, multi-tiered web application. In this model, a number of clients issue requests to an application, wait for a response, and then issue follow-up requests. An application consists of a set of tasks, and each task can have multiple identical instances. When there are multiple instances of a task, the incoming load

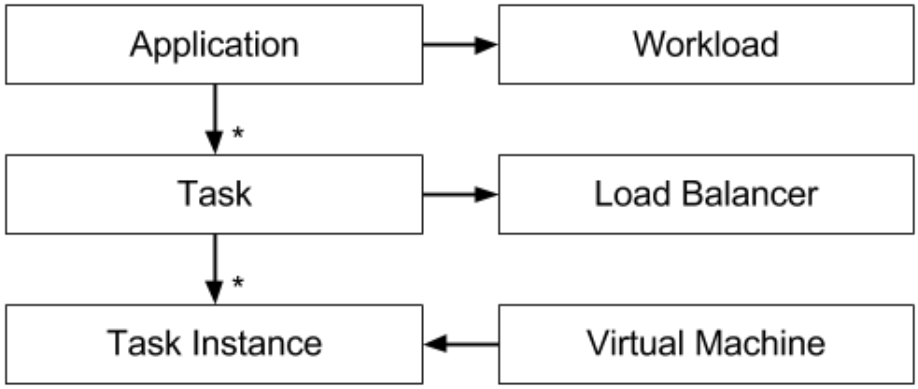

Figure A.2: Application Model

(i.e., requests) is shared equally between the instances. Task instances are mapped into VMs in a one-to-one mapping. (See Figure A.2 for an overview of the application model.)

Applications are modeled as a closed queueing network, solved with Mean Value Analysis (MVA). Applications have an associated *think time*, which is the time clients wait between receiving a response to a request and producing a follow-up request, and a *workload*, which is the number of clients currently using the application. Workloads can change at discrete points in time during the simulation, according to trace data from an input file or using a random number generator.

Tasks have a set of parameters that need to be defined: *service time* indicates the time it takes for the task to process a request, *visit ratio* indicates the number of times the task is invoked by a single request, *resource size* is the amount of resources allocated to the task, and finally *default* and *maximum* indicate the base and maximum number of task instances.

Tasks have resource requirements; these specify an amount of CPU, memory, bandwidth and storage. The last three resources are considered static and the specific quantities have to be available (i.e., unallocated) in a host for a task instance (inside a VM) to be successfully deployed in the host. The CPU resource, however, is dynamic, meaning that the CPU demand of a task instance changes over time according to the application's workload, and the total CPU requirement does not need to be allocated to the task instance at all times. Moreover, it is possible for the CPU demand of a task instance to be unsatisfied at some point in time due to CPU contention with other task instances co-located on the same host. If this happens, the task instance's service time is incremented to account for processor queueing, which in turn affects the application's response time, and may cause SLA violations (see below).

A workload trace file consists of a sequence of value pairs, where the first component is a discrete point in time and the second component is a normalized value in the range [0, 1]. The

normalized values are scaled up to match the size of the associated application. The scaling factor can be calculated by DCSim by specifying the maximum desired utilization of a task instance or the maximum desired response time.

It is possible to associate a Service Level Agreement (SLA) to an application. This feature takes the form of an upper threshold on the application's response time or throughput. DCSim keeps track of the amount of time the SLA conditions for an application were satisfied (or achieved). SLAs can also define penalties to be calculated during periods of SLA violation (i.e., unmet conditions).

A.2.6 Resource Managers & Scheduling

Host resources in DCSim are managed by a *Resource Manager* component on each host. The resource manager is responsible for allocating and deallocating resources for VMs, keeping track of the total amount of resource allocated, and deciding whether or not the host is capable of running a given VM. The resource manager is an abstract class and must be extended to provide the desired functionality. The default resource manager allocates memory, bandwidth and storage statically, with no oversubscription. CPU is oversubscribed, allocating to VMs as much CPU as they request, although they may not actually receive it if the host's CPU becomes overloaded.

While the resource manager handles allocations, the resource scheduler handles the scheduling of dynamic resources, such as CPU, based on current demand. At present, the resource scheduler schedules only CPU, although it could be extended to dynamically calculate usage of other resources as well, such as bandwidth. Other resources are simply given their full allocation, as determined by the resource manager. During the *schedule resources* phase of the main simulation loop (see Section A.2.1), the resource scheduler for each host calculates the amount of resources/second that each VM is given. In the case of CPU, this would be the number of CPU units given to each VM. It does so in an fair-share manner, giving each VM a chance to receive an equal amount of CPU, up to the total CPU required by its application at the current time. CPU not used by one VM can be used by another, and any CPU amount required by a VM over and above the capacity of the host is not scheduled, resulting in application performance degradation.

A.2.7 Autonomic Managers & Policies

Our development of DCSim is focused on providing tools to support research on virtualized data centre management. The *Autonomic Manager* (AM) and related components provide a framework to allow quick development of new management systems, while taking care of

some of the messaging and event handling details of DCSim automatically. The AM acts as a container for a set of *Capabilities* and *Policies*. A capability is simply a data storage object, which provides methods for the policies to access the stored data. For example, the *HostManager* capability provides a reference to a host that is managed by an AM possessing the capability. This capability can be used by a policy that is designed to manage a host in the data centre. Policies are installed into AM, and implement the actual management logic. A policy can only be installed in an AM that possesses the capabilities that the policy requires to function.

Policies can be triggered on a regular interval, or in response to events sent to the AM by another policy or component. In order to design a policy that executes on a regular interval, we simply create a *Policy* class that defines an `execute()` method, and pass the time interval to the AM when installing the policy. To trigger a policy on the arrival of a specific event class, we simply define an `execute(ConcreteEvent e)` method, and the AM will automatically detect that the policy accepts this event class, and call the policy whenever an event of this class is received.

AMs do not need to be attached to any other component, and can simply run detached from the physical data centre infrastructure. However, they can also be attached to host objects, to indicate that the AM is running on that host. When this configuration is used, the AM will only execute when the host is in the `on` power state.

Within this framework, it is a quick and simple process to define new policies, capabilities and events to build a desired management system or test a management algorithm.

A.2.8 Management Actions

Common management operations performed within a simulation can be encapsulated in a *Management Action*. DCSim currently features management actions for instantiating a new VM, migrating a VM, replicating a VM within an application task, and shutting down a host. Additional management actions can be created by extending an abstract class. It is possible to build a set of actions which can be executed either concurrently, in sequence, or in any combinations of the two. If a sequence of management actions is executed, the preceding management actions must complete before subsequent ones can execute. This includes the case where some management actions, such as VM migration, may take some time to complete.

A.2.9 Metrics

DCSim includes a mechanism for recording metrics of interest in order to evaluate management systems and algorithms through simulation. The class *MetricCollection* represents a set

of related metrics and is responsible for the calculation of those metrics. The class *SimulationMetrics* consists of several default MetricCollection objects, such as *HostMetrics* and *ClusterMetrics* which collect infrastructure related metrics, and users can expand this collection adding their own custom MetricCollection objects.

A.3 Configuring and Using DCSim

In this section we describe some of what is required to configure and run DCSim.

A.3.1 Workloads

As discussed earlier in Section A.2.5, the *Workload* component is responsible for specifying a dynamic workload level for applications running in the simulated data centre. In our simulations, we use normalized workload traces built from 5 real web server traces: the *ClarkNet*, *EPA*, and *SDSC* traces [13], and two different job types from the *Google Cluster Data* trace [14]. To ensure that VMs do not exhibit identical behaviour, we always start the trace for each VM at a randomly selected offset time.

In a data centre, the set of VMs is not static; VMs continuously arrive and depart the data centre. DCSim allows for this behaviour to be simulated, that is, the continual arrival of new applications (see Section A.2.5) to the data centre, which are submitted by sending an *Application Placement* event containing a description of the application to deploy. Applications have a lifespan chosen randomly from a specified distribution, after which they terminate. This helps model not only changes in individual VM resource requirements, but also changes in overall data centre utilization over the course of a single simulation.

A.3.2 Default Metrics

DCSim provides a number of useful metrics in order to help judge the performance of data centre management systems and algorithms, including the following:

Average Active Host Utilization

The average CPU and memory utilization of all hosts that are currently in the on state.

Average Data Centre Utilization

The average CPU and memory utilization of the data centre as a whole – treating all CPU and memory resources as one big pool.

Max, Min, and Average Active Hosts

The maximum, minimum, and average number of hosts in the on state at once.

Max, Min, and Average Active Racks and Clusters

If racks and clusters are defined in an experiment, the maximum, minimum, and average number of racks and clusters in the on state at once are calculated. Racks and clusters are considered in the on state when any of their hosts are active.

Number of Migrations

The number of migrations triggered during the simulation, by each management component that triggers migrations. Migrations are further broken down into migrations within a rack, between racks in the same cluster, and between clusters.

SLA Achievement

This metric represents the percentage of time in which Service Level Agreement (SLA) conditions are met (e.g., applications' response time stays below the threshold specified in the SLA). DCSim reports the following metrics related to SLA Achievement: mean and standard deviation; maximum and minimum; 95th, 75th, 50th, and 25th percentiles; the number of application with 99%, 95%, 90%, and less than 90% achievement. Penalties can be defined in a per application basis, so as to be calculated during SLA violation.

CPU Underprovisioning

This is the amount of CPU demand that could not be satisfied. When a VM has its CPU demand unmet, the difference between CPU demand and CPU scheduled is recorded and at the end a total percentage is calculated and reported.

Power Consumption

Power consumption is calculated for each host, and the total kilowatt-hours consumed during the simulation are reported. If racks and clusters are defined in an experiment, a second set of power consumption metrics is calculated taking into account both hosts and network switches.

Message Counts

The number of message sent, for each subclass of MessageEvent used during the simulation.

Spawned, Total, Shutdown, and Failed Applications

A number of application-related metrics are collected: created (*spawned*), successfully deployed (*total*), completed (*shutdown*), and unsuccessfully deployed (*failed*).

Max, Min, and Mean Response Time and Throughput

The maximum, minimum, and mean application response time and throughput values.

Max, Min, Mean and Total VMs

The maximum, minimum, and mean number of VMs running at once in the data centre. The total number of VMs that were instantiated during the simulation is also reported.

A.3.3 Performing Experiments with DCSim

In order to make configuring and performing experiments with the simulator as clean and easy as possible, DCSim provides a set of helper classes for performing simulations. The *SimulationTask* class encapsulates a single simulation configuration, allowing the user to configure the simulator by implementing the `setup()` method, while taking care of the details of running the simulation automatically. Simulation name, and duration can be specified, as well as a period of time to wait before recording metrics. Finally, a seed for random number generation can be passed to the SimulationTask to be used to generate any random elements, such as workload configurations. This provides repeatable experiments, which is convenient both for debugging and for comparing management systems and algorithms. Once the simulation task has been run, a collection of metrics recorded during the simulation is returned.

In order to run several simulations, either sequentially or concurrently, SimulationTask objects can be added to a *SimulationExecutor*. The simulation executor handles spawning threads for individual simulation tasks, waiting for all tasks to complete, and returning the resulting metric collections from each task.

A.3.4 Output & Logging

DCSim uses the logging library Apache log4j [15]. By default, only basic output is printed to the console, with other options available for more detailed logging (at the expense of processing time required for logging I/O). The DCSim configuration file contains several options specifying different logging output:

Enable Detailed Console

This will cause detailed, human-readable data on the execution of the simulation to be outputted to the console. This includes data on each host and VM at every step in simulation, as well as data on management operations such as VM migration.

Enable Console Log File

Console output will also be written to a log file.

Enable Simulation Logging

Individual, detailed data on the execution of the simulation (the same data as enabled with the *Enable Detailed Console* option), will be written to a separate log file for each simulation task run, even if several simulation task objects are executed concurrently.

Enable Trace

This will enable a machine-readable version of the detailed simulation data, for use in graphing or visualizations.

A.3.5 Visualization Tool

When developing and evaluating data centre management techniques, it can be extremely helpful to have a tool to visualize what is happening within the simulated data centre. We have developed a visualization tool that makes use of the machine-readable trace output of DCSim to provide a set of graphs describing the simulation run in detail. Furthermore, it includes an animation, allowing the state of Hosts and VMs in the data centre to be viewed as the simulation time progresses. Host and VM resource utilization are presented, and VM migrations and new instantiations are clearly shown. This allows the researcher to visually see how a management system or algorithm is operating, and to gain new insight into its behaviour.

A.4 Evaluation

In this section we demonstrate how DCSim can be used to implement and evaluate a management system, and use three different (though similar) management systems as working examples. We first describe the elements of these management systems (such as autonomic manager capabilities, policies, and events), discuss the changes that were made from one system to the next, and later compare the systems through simulation.

A.4.1 Data Centre Infrastructure

The target infrastructure consists of a collection of hosts and a DataCentre abstraction that contains all of the hosts. Each host has an associated Autonomic Manager (AM), as does the data centre. In the next sections we will discuss the capabilities of these managers and their associated policies.

A.4.2 Management Systems - Common Elements

Each host in the data centre has an AM associated with it. This manager possesses a capability, namely *HostManager*, that acts as a knowledge base for the manager, storing all relevant management information that the policies may need to successfully execute. One such policy is the *HostMonitoringPolicy*, which upon invocation collects the current status information of the host (resources in use or allocated, power consumption, number of incoming and outgoing VM migrations, etc.), packages the information in a *HostStatusEvent* message, and sends the message to the data centre's AM. The *HostMonitoringPolicy* requires the *HostManager* capability, so as to be able to access the host and collect the necessary status information.

Another policy installed in every host's AM is the *HostOperationsPolicy*. This policy defines the behaviour of the manager upon receiving the events *InstantiateVmEvent*, *MigrationEvent* and *ShutdownVmEvent*. These events trigger the allocation of the resources requested for the VM in the host, start a migration process, and stop and deallocate a VM, respectively.

At installation time, the *HostMonitoringPolicy* is configured to be triggered every 5 minutes. This behaviour is achieved by creating a *RepeatingPolicyExecutionEvent* with a periodicity of 5 minutes and specifying the host's AM as intended target. When the manager receives the event (once every 5 minutes), it triggers the associated policy.

The data centre's AM possesses the *HostPoolManager* capability, which serves to store information about a collection of hosts (in this case, all the hosts in the data centre). In the following sections we will discuss the policies that are installed in this AM.

A.4.3 Static Management System

The Static Management System allocates VMs in the data centre according to their expected peak resource demand, allocating to each incoming VM the total resources requested at creation time and never modifying that allocation. This is achieved through a single management policy, which is installed in the data centre's AM. This policy is a *VM Placement policy*, which defines how to perform the mapping of incoming VMs to hosts. Every time a *VmPlacementEvent* is received, the data centre's AM invokes the VM Placement policy. This policy

implements a greedy algorithm to place the incoming VM in the first host that has enough resources available to fit the VM without oversubscribing resources. If one such hosts is found, then the search is terminated and an *InstantiateVmEvent* is sent to the host. Otherwise, the VM Placement fails and the client request is rejected. The policy relies on the manager's *HostPoolManager* capability to get status information about all the hosts.

Another policy installed in the data centre's manager is the *HostStatusPolicy*. This policy is invoked every time a *HostStatusEvent* is received. The policy stores the new host status information in a data structure in the *HostPoolManager* capability of the data centre AM.

A.4.4 Dynamic Periodic Management System

The Dynamic Periodic Management System maps VMs into hosts based on their current resource needs. Resources such as memory, bandwidth and storage are statically allocated and never change, but the CPU is oversubscribed, therefore allowing the system to map more VMs to a host than is possible with the Static Management System.

Like the Static Management System, the *VM Placement policy* installed in the data centre's AM is invoked upon reception of a *VmPlacementEvent*. This policy is similar to the one used in the Static Management System, but since this system leverages CPU oversubscription, the policy does not require the hosts to have unallocated CPU for the incoming VM, but the policy rather checks how much CPU is actually in use in the host, and if there is enough CPU not in use, then the VM can be mapped into the host. As mentioned before, the system maps VMs into hosts based on the VMs' current resource needs. At creation time, the requested resources are taken as the current resource needs of the VM.

By oversubscribing resources, the management system can increase the resource utilization of the hosts, and therefore of the data centre as a whole. However, this strategy increases the risk of hosts becoming *stressed*. A *stress situations* occurs when the combined demand of the VMs co-located in a host exceeds the resource capacity of the host. When this happens, one or more VMs have to be migrated to another host, so as to free resources locally to satisfy the resource demand of the remaining VMs.

The management system uses a *VM Relocation policy* to determine which VMs to migrate away from a stressed host and to choose a new host for the migrating VMs. The policy is configured at installation time to run periodically every 10 minutes. When invoked, the policy first checks the set of hosts to determine which, if any, are stressed. For each stressed host, the policy follows a greedy algorithm to select VMs for migration and to find target hosts in which to place the migrated VMs.

The management system also uses a *VM Consolidation policy* to periodically consolidate

VMs in the data centre, attempting to minimize the number of physical servers that need to be powered on to host VMs. This policy is installed in the data centre's AM and is configured to be invoked every hour. Upon invocation, the policy uses a greedy algorithm to migrate VMs away of underutilized hosts and into hosts with higher resource utilization. Hosts that are emptied of VMs are then suspended or powered off, to conserve power.

The same *HostStatusPolicy* used in the Static Management System is used here to process *HostStatusEvent* messages and maintain up-to-date status information about the hosts in the data centre.

A.4.5 Dynamic Reactive Management System

The Dynamic Reactive Management System is very similar to the Dynamic Periodic Management System, except that it triggers its *VM Relocation policy* on demand rather than periodically. The *VM Relocation policy* itself is essentially the same, with minor changes implemented to allow the policy to run as frequently as required rather than periodically.

The Reactive system attempts to detect stress situations and trigger VM migrations as soon as possible, so as to reduce the SLA violations suffered by VMs co-located in stressed hosts. In order to achieve this behaviour, a new *HostStatusPolicy* (i.e., different from the corresponding policy from the Dynamic Periodic Management System) is necessary. This policy, known as *ReactiveHostStatusPolicy*, is still invoked upon receipt of a *HostStatusEvent* and is still responsible for updating hosts' status information. However, once the status information of the host associated with the event is updated, the policy issues a *VmRelocationEvent* so as to invoke the *VM Relocation policy*.

Upon invocation, the new *VM Relocation policy* first queries the *VmRelocationEvent* to obtain identification information of the host whose status information was recently updated. The policy then performs a stress check on the host. If the host is stressed, the policy looks for VMs to migrate away from the host and for target hosts to receive the migrated VMs. If the host is not stressed, the policy terminates its execution.

A.4.6 Experimental Setup and Design

The simulated data centre for these experiments consists of 200 hosts, divided equally between two types: *small* and *large*. The *small* host is modelled after the HP ProLiant DL380G5, with 2 dual-core 3GHz CPUs and 8 GB of memory. The *large* host is modelled after the HP ProLiant DL160G5, with 2 quad-core 2.5GHz CPUs and 16GB of memory. The different types of host have different power efficiency, which is calculated as *CPU capacity / power consumption at*

100% utilization. The power efficiency of the *large* host is 85.84 cpu/watt, while the power efficiency of the *small* host is 46.51 cpu/watt.

We use three types of VMs in these experiments. The *small* VM requires 1 virtual core with 1500 CPU units (minimum), plus 512MB of memory. The *medium* VM requires 1 virtual core with 2500 CPU units (minimum), plus 512MB of memory. The *large* VM requires 2 virtual cores with 2500 CPU units each (minimum), plus 1GB of memory. These descriptions correspond to the resource requirements of the VMs at creation time. Once a VM is running in the data centre, further placement and allocation considerations are made based on the actual resource *usage* of the VM. These experiments include an equal number of each type of VM.

The experiments are configured to create 600 VMs in the first 40 hours of simulation. These VMs remain throughout the entire experiment, so as to maintain a minimum level of load in the data centre. In the third day of simulation, new VMs begin to arrive; they do so at a changing rate and last for about a day. The total number of VMs in the data centre changes daily, using randomly chosen values uniformly distributed between 600 and 1600. This second set of VMs provides for a dynamic load in the data centre.

We use the term *workload pattern* to refer to a randomly generated collection of VM instances with arrival, departure, and trace offset times. A *workload pattern* can be repeated by providing the random seed with which it was first generated. We generate 10 different *workload patterns* and evaluate each management system under each of these *workload patterns*. The experiments have a duration of 10 simulated days, though only the last 8 days of simulation are recorded; the first 2 days are discarded to allow for the system to stabilize before recording results. Results are averaged across *workload patterns*.

A.4.7 Results and Discussion

Table A.1 presents the results for each management system. We can see that the Static Management System achieved the lowest host utilization by far, which translated also into the highest power consumption. However, given that VMs are statically allocated their total resource request (enough to meet their peak demand), the management system avoids CPU underprovisioning completely. It should be noted, however, that such a conservative approach to resource allocation resulted in an elevated percentage of failed placements, while the other management systems were able to accept every VM creation request.

Both Dynamic Management Systems achieved similar results, with Periodic showing slightly higher host utilization (and therefore less power consumption) and Reactive lowering CPU underprovisioning by about 40%. However, Reactive's reduction of CPU underprovisioning was achieved by triggering VM migrations as soon as hosts became stressed, which resulted in a

Systems	Host Util. (CPU)	Power	CPU Underprov.	Migrations	Failed Placements
Static	46%	7,221kWh	–	0	24%
Periodic	80%	5,056kWh	0.109%	10261	0%
Reactive	79%	5,121kWh	0.059%	12508	0%

Table A.1: Management Systems Comparison

20% increase in the total number of VM migrations issued.

A.5 Conclusions and Future Work

Developing and evaluating data centre management techniques on the scale that they are ultimately required to perform at presents a significant challenge. As such, most work turns to simulation tools for their experimentation. We have presented DCSim (Data Centre Simulator), an extensible simulation tool for simulating a virtualized data centre operating as an Infrastructure as a Service (IaaS) cloud. This tool allows researchers to quickly evaluate data centre management algorithms and techniques. We have presented an example use-case of the simulator, comparing three different VM management systems, to demonstrate the usefulness of the simulation results.

A number of additional features are planned for DCSim. An HPC/batch style application model should be included, as data centres typically host both interactive and HPC workloads. VM migrations are an important aspect to dynamic VM management, and their overhead needs to be considered in as accurate a manner as possible. We plan to include a more detailed modelling of migration bandwidth, and the impact of multiple simultaneous migrations on both migration time and SLA metrics, using our new model of data centre networking. Finally, the thermal state of the data centre should be considered and used to calculate cooling costs, as cooling power represents a significant cost for data centre operations.

Bibliography

[1] M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya, "Towards an improved data centre simulation with DCSim," in *Network and Service Management (CNSM), 2013 9th International Conference on*. IEEE, Oct. 2013, pp. 364–372.

[2] (2014) DCSim on GitHub. Distributed and Grid Systems (DiGS). [Online]. Available: https://github.com/digs-uwo/dcsim

[3] M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya, "DCSim: A data centre simulation tool for evaluating dynamic virtualized resource management," in *Network and Service Management (CNSM), 2012 8th International Conference on*. IEEE, Oct. 2012, pp. 385–392.

[4] D. Kliazovich, P. Bouvry, Y. Audzevich, and S. Khan, "Greencloud: A packet-level simulator of energy-aware cloud computing data centers," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, 2010.

[5] (2014) ns-2. [Online]. Available: http://nsnam.isi.edu/nsnam/

[6] S.-H. Lim, B. Sharma, G. Nam, E. K. Kim, and C. Das, "Mdcsim: A multi-tier data center simulation, platform," in *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, 2009.

[7] S. Gupta, R. Gilbert, A. Banerjee, Z. Abbasi, T. Mukherjee, and G. Varsamopoulos, "Gdcsim: A tool for analyzing green data center design and resource management techniques," in *Green Computing Conference and Workshops (IGCC), 2011 International*, 2011.

[8] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[9] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, 2010.

[10] S. Garg and R. Buyya, "Networkcloudsim: Modelling parallel applications in cloud simulations," in *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, 2011, pp. 105–113.

[11] S. Yeo and H. H. Lee, "SimWare: A holistic warehouse-scale computer simulator," *Computer*, vol. 45, no. 9, pp. 48–55, 2012.

[12] (2014) SPECpower_ssj2008. Standard Performance Evaluation Corporation. [Online]. Available: http://www.spec.org/power_ssj2008/

[13] (2014) The Internet Traffic Archive. [Online]. Available: http://ita.ee.lbl.gov/

[14] (2014) Google Apps for Business. Google Inc. [Online]. Available: http://www.google.com/enterprise/apps/business/

[15] (2014) Apache Log4j. The Apache Software Foundation. [Online]. Available: http://logging.apache.org/log4j/

Curriculum Vitae

Name:	Gastón Keller
Post-Secondary Education and Degrees:	Universidad Nacional del Sur Bahía Blanca, Buenos Aires, Argentina 2001-2007 B.Sc. Honors Computer Science
	The University of Western Ontario London, Ontario, Canada 2007-2009 M.Sc. Computer Science
	The University of Western Ontario London, Ontario, Canada 2009-2014 Ph.D. Computer Science
Honours and Awards:	OGS 2013-2014
Related Work Experience:	Teaching Assistant The University of Western Ontario 2007-2013

Selected Publications:

G. Keller, M. Tighe, H. Lutfiyya, and M. Bauer. An analysis of first fit heuristics for the virtual machine relocation problem. In *Network and Service Management (CNSM), 2012 8th International Conference on*, pages 406–413. IEEE, October 2012.

G. Foster, G. Keller, M. Tighe, H. Lutfiyya, and M. Bauer. The Right Tool for the Job: Switching data centre management strategies at runtime. In *Integrated Network Management (IM), 2013 IFIP/IEEE International Symposium on*, May 2013.

M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya. Towards an improved data centre simulation with DCSim. In *Network and Service Management (CNSM), 2013 9th International Conference on*, pages 364–372. IEEE, October 2013.

G. Keller, M. Tighe, H. Lutfiyya, and M. Bauer. A hierarchical, topology-aware approach to dynamic data centre management. In *Network Operations and Management Symposium Workshops (NOMS), 2014. IEEE/IFIP*, May 2014. *Best paper award.*

G. Keller and H. Lutfiyya. Dynamic Management of Applications with Constraints in Virtualized Data Centres. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, May 2015. *Accepted to appear.*