Dynamic Management of Applications with Constraints in Virtualized Data Centres

Gastón Keller and Hanan Lutfiyya
Department of Computer Science
The University of Western Ontario
London, Canada
{gkeller2|hanan}@csd.uwo.ca

***Abstract*—Managing single-VM applications in data centres is a well-studied problem, managing multi-VM applications is not. Multi-VM applications usually have, in addition to their basic resource requirements, a special set of requirements that the data centre provider has to satisfy as well. We refer to these requirements as *placement constraints*. We propose two management strategies designed to manage multi-VM applications with placement constraints in data centres. These management strategies aim to increase data centre's resource utilization (and thus reduce power consumption) while at the same time minimizing SLA violations. One of the management strategies enforces placement constraints at all times, while 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 and it does add cost.**

I. Introduction

Infrastructure-as-a-Service (IaaS) clouds are powered by large-scale data centres that rent their infrastructure (to application owners) on a pay-per-use basis. The business of these providers lies on maximizing their infrastructure's utilization while minimizing data centre expenses, such as power consumption [1]. Still, their business goals have to be balanced with their clients' service-level expectations, usually specified in the form of Service Level Agreements (SLA). Achieving this balance can be difficult and so it has become the focus of much research [2], [3], [4], [5], [6], [7].

Systems virtualization enables IaaS providers to increase their infrastructure's utilization by co-locating multiple computing systems per physical server (or host), each system encapsulated in its own virtual machine (VM). Moreover, resources can be *oversubscribed* (i.e., more resources can be promised in total to the group of co-located VMs in a host than the host actually possesses). However, applications tend to present a rather dynamic resource demand [8], making application co-location a risky affair: if the total resource demand of co-located applications were to exceed the available resources in a host, some applications would have their resource needs unmet, causing SLA violations. By leveraging VM live migration, an application could be relocated to a different host, reducing resource utilization (and demand) locally, but avoiding SLA violations. This approach of mapping and re-mapping VMs to hosts as needed is known as *dynamic VM management*.

Deploying and managing single-VM applications in a data centre is a well studied problem [9], [8], [2], [10], [3], [4], [5], [11], [6], [12]. However, 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 server. 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 [13].

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 paper, 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 paper is organized as follows: Section II discusses related work in the area, Section III describes the application placement constraints we considered, Section IV presents the two approaches we developed to manage multi-VM applications in data centres, Section V presents our evaluation and discusses the results, and finally Section VI states our conclusions and suggests directions of future work.

II. 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 [9], [8], [2], [10], [3], [4], [5], [11], [6], [12].

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., [14], [15], [16]). 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. [17] 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. [18] 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. [19] 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.

III. 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) **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 between components suffers from increased network latency, thus degrading the application's 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. [17]. 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.

A. 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

[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.

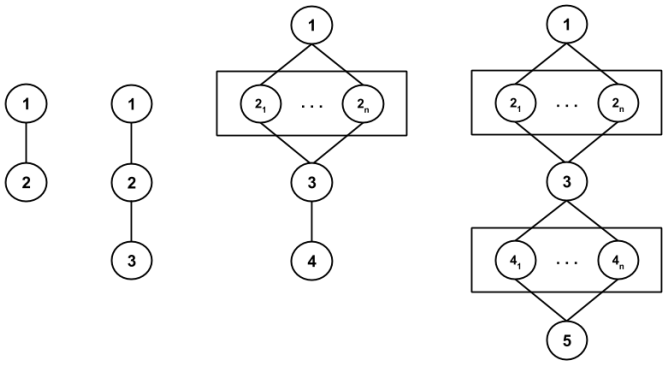

Fig. 1. Application Templates. Nodes represent application components and edges represent communication links. Boxes indicate that multiple instances of the same component may exist.

Figure 1). 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.

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

For example, consider a sample application modelled after the third template in Figure 1. Such an application would be represented as follows:

- Affinity sets: $\{ \{ 3 , 4 \} \}$
- Anti-affinity sets: $\{ \{ 2_1 , ... , 2_n \} \}$
- Neutral set: $\{ 1 \}$

IV. Management Strategies

In the context of dynamic VM management, there are three main operations: (i) the **Placement** operation selects hosts in which to instantiate new VMs; (ii) the **Relocation** operation migrates VMs away from *stressed* hosts (i.e., hosts that are close to running out of spare resources to allocate to its VMs) and into non-stressed hosts; and (iii) the **Consolidation** operation relocates 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.

A. Data Centre Organization and System Architecture

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.

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.

B. 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.

1) 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.

2) 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 1: 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 1), 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.

When the *internal* relocation process fails to find a suitable migration, the policy starts its *external* relocation process (see Algorithm 2), 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.

```
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 2: MS-EC Relocation policy at rack-level – External process.

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.

3) 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.

C. 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 3), 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.

```
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 3: MS-RC Relocation policy at rack-level – External process.

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) (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.

V. Evaluation

We evaluate the two management strategies proposed in Section IV 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.

A. 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 III-A. 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 I with their respective configuration.

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 I
Applications

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 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 V-C). 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.

B. 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 I. 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%.

C. 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 (**Host 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.

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 III) has been violated by the management system. 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. 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.

D. Results and Discussion

The results of the first experiment (presented in Table II) 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

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 II
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 III
EXP. 2: SINGLE- AND MULTI-VM APPLICATIONS

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 III) 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 violated 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 give priority to small applications during Relocation: **MS-EC** selects for relocation 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.

VI. 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.

However, the conclusions reached in this study assume no *application elasticity* (i.e., the ability of an application to change the number of its components at runtime to match current service demand). If we were to remove this restriction, would enforcing constraints at all times still be possible (and desirable), or would violating constraints become a necessity? In other words, would our conclusions still hold? Other issues to explore include the expansion of the set of available application templates and the violation of other placement constraints besides *single-rack*, such as *affinity* or *anti-affinity*.

ACKNOWLEDGEMENTS

We thank the National Sciences and Engineering Research Council of Canada (NSERC) and the Government of Ontario for their support.

REFERENCES

[1] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, pp. 7–18, 2010.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] (2014) AWS Reference Architectures. Amazon Web Services, Inc. [Online]. Available: http://aws.amazon.com/architecture/

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[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/