DCSim: A Data Centre Simulation Tool for Evaluating Dynamic Virtualized Resource Management

Michael Tighe, Gaston Keller, Michael Bauer, Hanan Lutfiyya
Department of Computer Science
The University of Western Ontario
London, ON, N6A 5B7, CANADA
Email:{mtighe2;gkeller2;bauer;hanan}@csd.uwo.ca

***Abstract*—Computing today is shifting from hosting services in servers owned by individual organizations to data centres providing resources to a number of organizations on a shared infrastructure. Managing such a data centre presents a unique set of goals and challenges. Through the use of virtualization, multiple users can run isolated virtual machines (VMs) on a single physical host, allowing for a higher server utilization. By consolidating VMs onto fewer physical hosts, infrastructure costs can be reduced in terms of the number of servers required, power consumption, and maintenance. To meet constantly changing workload levels, running VMs may need to be migrated (moved) to another physical host. Algorithms to perform dynamic VM reallocation, as well as dynamic resource provisioning on a single host, are open research problems. Experimenting with such algorithms on the data centre scale is impractical. Thus, there is a need for simulation tools to allow rapid development and evaluation of data centre management techniques. We present DCSim, an extensible simulation framework for simulating a data centre hosting an Infrastructure as a Service cloud. We evaluate the scalability of DCSim, and demonstrate its usefulness in evaluating VM management techniques.**

***Keywords*-Cloud, Data Centre, Simulator, Virtualization, Infrastructure as a Service**

I. Introduction

One of the biggest trends in computing today is the movement of processing from servers owned and operated by individual organizations to large-scale data centres, providing on-demand computing resources for many organizations on a shared infrastructure. An Infrastructure as a Service (IaaS) Cloud allows users to provision servers as needed, paying only for what they require. Even large services, such as Netflix, are hosted by the likes of Amazon's EC2 Cloud service [1]. For an organization, moving computing into an IaaS cloud promises to reduce infrastructure costs, improve scalability, and offload the responsibility of server administration and maintenance.

The provider of the IaaS faces its own set of challenges in managing its data centre. Resources should be used in the most efficient way possible in order to reduce operating costs. Typically, a server makes poor use of its resources, operating at far below its maximum capacity. Since even an efficient server can consume 50% of its peak power usage when idle [2], underutilized servers represent a significant waste of power. Virtualization can be employed in order to help remedy this situation. Virtualization is a technology that allows multiple Virtual Machines (VMs) to run on a single host system, providing the illusion that each has control of a physical machine. By packing many VMs onto a single host, its resources can be more fully utilized, thus saving on power, infrastructure, and maintenance costs. Furthermore, a host's resources can be *overcommitted*, such that more resources are promised to each VM than are actually available. This can enable significantly higher utilization, but if one or more VMs exhibit an increase in their resource needs, it may lead to a situation where the host does not have enough resources to satisfy all of the demand. In this case, a VM must be migrated (moved) to a different host that has enough available resources to satisfy the VM's demand. Determining which VMs should be migrated, when, and to which host, is a non-trivial task. Not only must they meet the needs of client VMs, but they must balance them against power and cost considerations, adding further complexity.

Algorithms for dynamic resource management in the data centre have proven difficult to evaluate due to the scale and complexity of the infrastructure on which they are intended to run. As such, simulation is becoming accepted as a means of rapidly evaluating new techniques at a speed and scale not possible with real implementations. Once a technique has been evaluated and fine-tuned using a simulation, further experimentation can be performed using a real infrastructure, albeit very likely on a much smaller scale.

There is currently a lack of easily customizable and extensible simulation tools that model a virtualized, multi-tenant data centre. Furthermore, there is a need for a tool that provides an application model to simulate the interactions and dependencies between VMs working together as a single service (e.g. a multi-tiered web application). Even if management algorithms only treat VMs as 'black boxes', with no knowledge of their applications, it is still important to model these applications within the simulation to drive VM behaviour and resource utilization. That being said, some IaaS providers (such as Amazon EC2 [3]) offer advanced services, such as auto-scaling and dynamic load balancing, thus presenting an even stronger case for a detailed application model within the simulation tool. Other features of virtualization, such as a *work conserving* CPU scheduler used in modern hypervisors, resource allocation and VM migration and replication must also be available. Finally, host power states (*on*, *off*, *suspended*) must be modelled with appropriate transition times between states. We present a new simulator, DCSim (Data Centre Simulator), designed specifically to address these requirements. DCSim is an extensible simulation framework designed to study VM management in a data centre providing an IaaS Cloud.

The remainder of the paper is organized as follows: Section II presents related work in data centre and cloud simulation. Section III presents the architecture and components of DCSim. Section IV presents the metrics that DCSim collects to evaluate the performance of the data centre. Section V presents three experiments designed to evaluate the scalability and usefulness of DCSim. Finally, Section VI presents conclusions and a list of future work.

II. Related Work

GreenCloud [4] is a packet-level data centre simulator designed to evaluate the energy costs of data centre operation. Built as an extension to the network simulator Ns-2 [5], it utilizes a highly detailed simulation of network communication. As a low level network simulation, it executes slower than event-based simulations. Although it has a detailed workload model, it does not include any modelling of virtualization. As such, it is not suitable for virtualized resource management research.

MDCSim [6] is a data centre simulation platform designed to simulate large scale, multi-tier data centres. It focuses on data centre architecture and cluster configuration, measuring both performance and power metrics. The simulator models a data centre running a three-tiered web application (web, application and database tiers), with the ability to modify and evaluate the configuration of each tier. Again, virtualization is not considered, nor are multiple tenants of the data centre. Also, it is built using a commercial product and is therefore not publicly available.

GDCSim (Green Data Centre Simulator) [7] aims to simulate both the management and physical design of a data centre, examining the interactions and relationships between the two. The goal is to fine-tune the interactions between management algorithms and the physical layout of the data centre, such as thermal and cooling interactions with workload placement. Resource management considers HPC (High Performance Computing) job placement and scheduling, power modes, and cooling settings. Transactional workloads (such as a web server) are modelled using a single workload, load-balanced across the data centre. Multiple tenants of the data centre and virtualization technology are not considered.

CloudSim [8] is a toolkit for simulating a data centre hosting a virtualized IaaS Cloud. Multiple users can create VMs within the data centre. It also provides the ability to simulate multiple data centres operating as a federation, that are capable of coordinating resource allocation. CloudSim implements an HPC-style workload, with *Cloudlets* (jobs) submitted by users to VMs for processing. It can be used to simulate a transactional, continuous workload such as a web server or other service [9], but it lacks a detailed model of such an application.

DCSim differs from GreenCloud, MDCSim, and GDCSim in that it is focused on a virtualized data centre providing IaaS to any multiple tenants, similar to CloudSim. It differs from CloudSim in that it focuses on transactional, continuous workloads. As such, DCSim provides the additional capability of modelling replicated VMs sharing incoming workload as well as dependencies between VMs that are part of a multi-tiered application. SLA achievement can also be more directly and easily measured and available to management elements within the simulation.

III. DCSim Architecture & Components

DCSim (Data Centre Simulator) is an extensible data centre simulator implemented in Java, designed to provide an easy framework for developing and experimenting with data centre management techniques and algorithms. It is an event-driven simulator, simulating a data centre offering IaaS to multiple clients. It focuses on modelling transactional, continuous workloads (such as a web server), but can be extended to model other workloads as well. Figure 1 outlines the main components of DCSim. The primary class is the DataCentre, which contains Hosts, VMs, and various management components and policies.

DataCentre consists of a set of Host machines, which themselves contain a set of VMs. DataCentre and Host instances are governed by a set of policies and managers, which determine how they operate within the simulation and provide extensible and customizable points to insert new management algorithms and techniques. Default implementations of all abstract classes are available, so that the simulator user can pick and choose which components to extend for their specific research needs. The components of DCSim are described in detail throughout the rest of this section.

A. Hosts and Virtual Machines

In DCSim, a data centre consists of a set of interconnected physical host machines governed by a set of management policies. The purpose of the data centre is to host a set of Virtual Machines (VMs) on its physical hosts, each with its own dynamically changing resource needs driven by an external workload. We typically consider VMs to be running continuous, transactional servers, such as a web server. The basic details about each VM, such as its initial

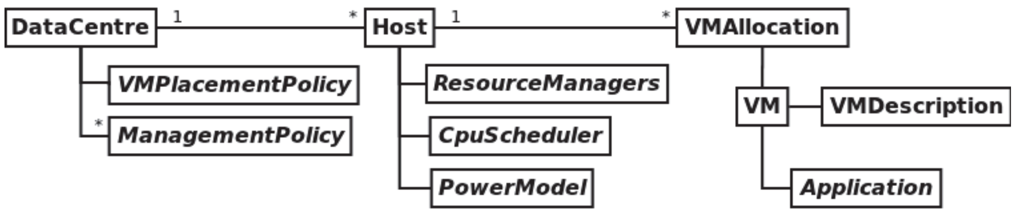

Figure 1: DCSim Architecture

resource requirements, are contained within the VMDescription. Each host can have several VMs sharing its resources, all contributing to the overall utilization of the host. Hosts also have a Virtual Machine Manager (VMM). In a real system, the VMM (or *hypervisor*) handles the operation of the virtualized server and provides the ability to manage a host's VMs. In DCSim, the resource utilization of the VMM is modelled using a special VM that is present on each host.

VM Allocation: Each host maintains a collection of VMAllocations, which represent a set of resources allocated to a single VM. In the case of a migration of a VM from a *source* host to a *target* host, the target host creates a new VMAllocation to reserve resources for the incoming VM, but the VM does not actually reside in the new VMAllocation until the migration is complete. Once the migration is complete, the VM is moved and its original VMAllocation on the source host is deallocated.

Overcommitting Resources: In order to make the most use of a host's resources, a host may promise more resources to its set of VMs than it actually possesses, in the hope that the VMs will not require their full requested allocation at the same time. Overcommitting of CPU is supported by major virtualization technologies (such as KVM [10], Xen [11] and VMWare ESX [12]), and as such, is supported in DCSim.

Host Utilization: If a host's resources are being highly utilized, at or above some threshold (say, 85% of available resources), then the host is considered *stressed*. A *stressed* host is in an undesirable state, as any increase in resource requirements of a hosted VM (due to an increase in workload level) could result in one or more VMs not being allocated enough resource to handle their workload. In a real-world scenario, this would be equivalent to dropped requests or a compromised response time, leading to SLA (Service Level Agreement) violations.

On the other hand, a host with very few of its resources in use (say, less than 50%) is considered *underutilized*. This is also an undesirable situation, as it may be possible to run the set of VMs in a smaller number of hosts, thus allowing some hosts to be suspended to conserve power. Management policies can be developed to handle such situations, as described in Section III-C.

Host State: Each host may be in one of three primary states: *On*, *Off*, or *Suspended*, as well as transitional states between those three. A host in the *On* state operates normally. When a host is *Off* or *Suspended*, any VMs it is hosting will not be allowed to execute. Furthermore, in the *Suspended* state, a host consumes a small amount of power, and in the *Off* state, a host host consumes no power. The time required to transition between each state is configurable.

Power Consumption: Each host contains a PowerModel which defines how much power it consumes given its current CPU utilization. DCSim uses the SPECpower benchmark [13] data to build power models, which provides power consumption levels of real servers in 10% CPU utilization intervals. We use these values and calculate intermediary values using linear interpolation.

B. Resource Managers & CPU Scheduling

On each host, its CPU, memory, bandwidth and storage are each managed by a separate ResourceManager (CpuManager, MemoryManager, BandwidthManager, and StorageManager, respectively). The resource managers' task is to manage the allocation of resources to VMs. For example, when a VM wants to migrate to a host, it is the responsibility of the resource managers to determine if enough resource is available, and to allocate it for the incoming VM.

CPU is allocated in *shares*, with each core having a specified number of shares available, depending on the relative computing capacity of the processor. Within the simulation, CPU is treated as a single scalar value, with the total CPU capacity being the sum of the capacity of all of the CPU cores available on the host. This is a reasonable simplification as modern virtualization technologies perform load balancing of VMs across processors to make full use of all available capacity. Memory and storage are allocated in MB, and bandwidth in Mb/s.

Resource managers can allocate a static amount of resource to a VM, change the allocation of resources dynamically, as well as handle resource overcommitting, to suit particular management policies or goals. All four resource managers can be extended to suit the needs of a specific experiment.

While the CpuManager handles allocation of CPU, the CPUScheduler is responsible for actually scheduling the running of VMs on the host. There are several ways in which VMs can be scheduled to execute. The most basic method is to only allow a VM to use at most its allocated number of CPU shares. This can lead to a waste of CPU resources, as any shares left unused by a VM cannot be used by another. The more common method is to allow other VMs to make use of these unused shares. This is referred to as a *work conserving* scheduler. DCSim currently implements a fair-share CPU scheduler, which gives each VM an equal opportunity to use CPU resources. This is similar to the scheduling method used by KVM [10] (which uses the Linux scheduler).

There is also a physical upper limit on the amount of CPU a VM can use. Each VM is created with a specific number of *virtual cores*, which the operating system of the VM interprets as real physical CPU cores. It is therefore not possible for a VM to consume more CPU than the full capacity of the number of virtual cores it contains, as this would require parallel execution of a single virtual core on more than one physical core.

C. Data Centre Management Policies

DCSim provides the ability to create Management Policies to manage hosts and VMs to achieve the goals of the data centre. DCSim requires that the data centre has at least one policy: the VMPlacementPolicy. The VM placement policy controls how a VM is initially placed in the data centre. That is, the first time that the VM is instantiated, this policy determines on which host it should be run. DCSim provides a simple policy, *Most Loaded Host First*, which chooses the host with the highest utilization that still has space for the incoming VM, without pushing it into the *stressed* utilization range.

Other management policies can be created by extending the ManagementPolicy abstract class, and executed during the simulation at a regular interval. For example, a policy could be created to dynamically reallocate VMs to relieve stress situations or consolidate load onto fewer hosts. When a host becomes *stressed*, one or more VMs may need to be migrated to another host to ensure that there are resources available to meet the demand of each VM. Similarily, when a host becomes *underutilized*, its VMs could be migrated away so that it can be suspended to conserve power. Determining which VM to migrate, when, and to which host, is similar to the m-dimensional knapsack problem, which is NP-hard [14], except it must be performed continuously on a dynamic system, and consider other constraints such as VM dependencies, migration bandwidth considerations, and balancing power savings with potential SLA violations.

The development and evaluation of data centre management policies to perform tasks such as VM reallocation is the primary motivator behind the development of DCSim. Any number of management policies may be active within the simulation, allowing for experimentation with a combination of several policies that can cooperate or even compete with each other for the achievement of their goals.

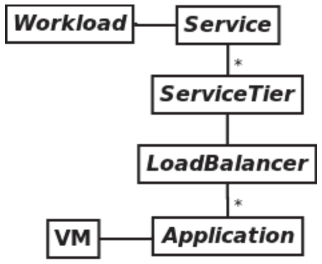

Figure 2: DCSim Application Model

D. Application Model

The resource needs of each VM in DCSim are driven dynamically by an Application, which varies the level of resources required by the VM to simulate a real workload. The Application class is abstract and can be extended to implement different types of applications, but the primary application model implemented in DCSim mimics a continuous, transactional workload, such as a web server. Figure 2 outlines the basic components of the application model. Each component of the application model can be extended to implement the desired behaviour.

At the highest level, the Service class encapsulates all of the elements of the application model for a single service running in the cloud. A service can be, for example, a multi-tiered web application, with multiple applications running in separate VMs in each tier, serving requests from an external workload. Service contains an instance of the Workload class, which specifies a time varying level of incoming *work*, which in the case of our web application example, can be thought of as *requests*. The number of requests (work units) per second changes dynamically based on trace inputs, which specify the workload level on a fixed time interval (for example, every 100 seconds). Workload values from traces are normalized to the range [0, 1], so that they may be easily scaled to the desired size but still maintain the relative changes in workload.

Service contains one or more ServiceTier objects, which represent tiers of the application and are specified in consecutive order. A single tier consists of a LoadBalancer and one or more identical Application instances. Each instance of the Application runs on a single VM, which can be located on any host in the data centre. Incoming work from the workload is sent to the first tier, and shared by the load balancer amongst all of the applications within the tier. Within each individual application, the incoming requests are translated into an amount of resources that are required to satisfy them. If there are not enough resources available to the VM running the application, some requests will be dropped, which is considered an SLA violation. The successfully completed work of the first tier (possibly less than the incoming work from the workload if the VMs in the tier are underprovisioned) feeds the incoming work of the second tier, and so on. The final tier returns work to the Workload which records the number of requests completed. The number of VMs in each tier (running application instances) and their placement in the data centre can be adapted dynamically by VM management policies, which were discussed earlier. Any number of services can be defined and instantiated to run within the data centre.

E. Migration and Replication

VMs in DCSim can be both migrated to another host and replicated to split incoming work between multiple VMs. To perform a migration, the memory of a VM must be copied from the source host to the target host. As such, the time to complete a migration is calculated as the amount of memory in use by the VM divided by the available bandwidth for migration. A CPU overhead is added to the VMM running in both the source and target host for the duration of the migration, which can be configured as desired. For our current work, we assume a CPU overhead of 10% of the current utilization of the migrating VM. Additionally, a 10% performance penalty is attributed to the VM and recorded as an SLA violation [15]. See Section IV for more information about the SLA Violation metric. The 10% SLA penalty does not, however, affect the amount of work completed by the VM. The work is considered delayed, hence the SLA violation, but is still completed and forwarded to any subsequent service tiers.

Replication of a VM can be performed to split the incoming workload of a service tier. When a new replica is created, the application running on the VM is added to the tier's load balancer and begins receiving a share of the workload. When a replica is shut down, the application is removed from the load balancer. Note that each application has a fixed resource overhead in addition to the resources required to process incoming work. As such, adding additional replicas to a service tier has the negative consequence of increasing the total overhead of the tier.

IV. DCSim Metrics

The following section outlines the metrics currently being computed by DCSim for each simulation.

SLA Violation: A Service Level Agreement (SLA) defines some quality of service that the cloud client expects to receive from the cloud provider, such as a response time below a certain threshold. DCSim reports the percentage of total incoming work in which SLA was violated. When a VM requires more resources than is available to it, some incoming work cannot be completed. DCSim considers this to be an SLA violation, and the amount of work not completed is added to the total SLA violation amount. Additionally, while a VM is in the process of migrating to another host, an SLA violation of 10% of completed work is recorded to account for the performance degradation experienced by migrating VMs.

Active Hosts: DCSim records the minimum, maximum, and average number of hosts that are *active* (hosting at least one VM) at any given time during the simulation. This can provide some insight as to how well a dynamic VM reallocation policy is consolidating load, with smaller number typically considered more desirable.

Host-hours: Host-hours is the combined total of the active time of every host in the simulation. That is, if 10 hosts were active for 30 simulation minutes each, then 5 host-hours were used. This gives a combined measure of the number of hosts that were required to meet the workload demand throughout the entire simulation run.

Active Host Utilization: One of the goals of VM consolidation is to increase the overall utilization of each physical host, so that resources are not wasted. DCSim measures the CPU utilization of all hosts that are currently in the *On* state. The higher the average utilization, the more efficiently resources are being used.

Number of Migrations: Dynamic VM reallocation is achieved by migrating a VM from one host to another. Unfortunately, in terms of resource consumption, migrations are not free. It is therefore important to keep track of the number of migrations that a given algorithm triggers. If similar results can be achieved by one algorithm with fewer migrations than another, then the former algorithm may be considered superior.

Power Consumption: Reducing power consumption has become a top priority for data centre operators, both for cost reduction and environmental motivations. Power consumption is calculated for each host, as described in Section III-A, and the total kilowatt-hours consumed during the simulation are reported.

Simulation and Algorithm Running Time: DCSim reports on the time it took to run the simulation; as well as the average, maximum, and minimum execution time of management policies. This facilitates comparing the overhead of different algorithms.

V. Evaluation

Three experiments were run to evaluate DCSim. The first was intended to evaluate the scalability and overhead of the simulator. The second and third demonstrate DCSim's capabilities and usefulness in evaluating VM management policies and techniques.

A. Data Centre Configuration

All three experiments share some common configuration characteristics.

Hosts: Two different real-life servers were modelled for use in the simulation: the HP ProLiant DL160G5 and the ProLiant DL360G5. Both have 2 CPUs with 4 cores each (8 cores total), and 16GB of memory. The DL160G5's

Size	1	2	3	4
# Cores	1	1	1	2
Core Capacity	1500	2500	3000	3000
Memory	512MB	1GB	1GB	1GB

Table I: VM Sizes

	1	2	3	4	5	6
# Hosts	100	1000	10000	1000	1000	1000
# VMs	400	4000	40000	5000	6000	7000
Time	9s	130s	3597s	189s	256s	318s

Table II: Scalability & Overhead Experiment

processor runs at 2.5GHz and the DL360G5's processor runs at 3GHz, and as such their core capacities have been set to 2500 and 3000 shares, respectively. Their power consumption has been modelled using data from the SPECpower [13] benchmark. Both hosts are configured to have a 10Gb/s Ethernet connection, and a 1TB hard disk. The VMM of each host is configured to consume 300 CPU shares.

Host resource managers are configured such that CPU can be overcommitted, and the other three resources are allocated statically. A fair-share CPU scheduler was used, as described in Section III-B. Hosts are considered *stressed* when their CPU utilization is greater than 85%, and *underutilized* when below 50%.

VMs: Four different VM sizes have been defined, and are described in Table I. All four VM sizes require 100Mb/s of bandwidth and 1GB of storage.

Each VM runs an Application modelling a single, independent web server, fed by its own Workload instance. Each workload uses one of the following 5 traces as its input: the ClarkNet HTTP trace, EPA HTTP trace, or SDSC HTTP trace from the Internet Traffic Archive [16], or two job types from the Google Cluster Data trace [17]. Workload traces shorter than the duration of the experiment were looped, and each application's workload began its trace at a randomly chosen offset time. Each application requires 1 CPU share to complete 1 request, plus a fixed overhead of 200 shares. The incoming workloads were scaled such that at their peak value, the total CPU consumption of the application would be equal to the size of the VM (as defined in Table I). Memory, bandwidth and storage usage remained fixed. VMs were initially placed in random order using the *Most Loaded Host First* VMPlacementPolicy.

B. Performance and Scalability

The first experiment is aimed at evaluating the scalability of the simulator. As the purpose of a simulator is to provide results both faster and on a larger scale than can be done with a real implementation, a reasonable execution time is vital. Six different experiments were performed with various numbers of hosts and VMs, and the time to run the simulation was measured. The first three experiments use a 4:1 VM-to-host ratio and scale up with each experiment. The last three experiments fix the number of hosts and increase only the number of VMs. Each simulation was run for 10 simulated days, with a basic Management Policy executing every 10 minutes to perform dynamic VM reallocation, as described in Section III-C. The experiments were performed on a typical desktop workstation, with an Intel Core i7 930 processor and 6GB of RAM.

Table II lists the results averaged over 5 repetitions. The *# Hosts* and *# VMs* columns indicate the total number of hosts and VMs present in each experiment. The *Time* column lists the actual time in seconds to run the simulation. As we can see, execution time scales well with the size of the simulation. Simulations involving 1000 hosts and 4000 VMs only require slightly more than 2 minutes to execute, while simulations as large as 10000 hosts and 40000 VMs can still be executed in approximately 1 hour. Furthermore, increasing the ratio of VMs to hosts does not appear to disproportionately increase the running time. There are a number of other factors that determine the execution time of the simulation, such as the frequency of simulation events, but this experiment adequately represents typical usage.

C. Dynamic VM Allocation

The second experiment is designed to demonstrate DCSim's usefulness in evaluating VM management techniques. The primary mechanisms for VM management in DCSim are the VMPlacementPolicy and ManagementPolicy. A basic Management Policy to perform dynamic VM reallocation using a greedy, First Fit heuristic algorithm has been implemented. Hosts that are not hosting any VMs are shut down to conserve power. Each experiment was run for 10 days, with 200 hosts and 400 VMs. An equal number of the two host types was created, as well as an equal number of each VM size. Each VM was attached to a random workload trace, such that each trace was used by an equal number of VMs. We compare the simulator under three allocation strategies.

Strategy 1 - Static Peak Allocation: Each VM is statically allocated enough resources to meet its peak level of resource demand. That is, it will never require any more resources than it is allocated at the start of the experiment. No dynamic re-allocation is performed. This represents a safe provisioning, ensuring that there is enough allocated resource to handle any possible workload level, while performing no dynamic reallocation.

Strategy 2 - Static Average Allocation: Each VM is statically allocated enough resources to meet its average level of resource demand, as calculated from the workload traces used. This is a conservative allocation, allocating only enough resource to handle the average case but not enough to handle peaks in resource demand. As with the Fixed Peak Allocation experiment, no dynamic reallocation is performed.

	Static Peak	Static Avg	Dynamic
# Migrations	0 [0]	0 [0]	18547 [737]
Avg. Active Hosts	58.2 [0.4]	24 [0.4]	36.4 [0.46]
Host Utilization	48.8% [0.3]	94.6% [1.4]	77% [0.5]
Kilowatt-hours	2804.1 [13.5]	1431.1 [12.3]	2009.9 [9.4]
SLA Violation	0% [0]	21.4% [0.34]	0.8% [0.05]

Table III: Dynamic Allocation Results

Strategy 3 - Dynamic Allocation: Each VM is initially provisioned its peak resource requirements, but in an attempt to reduce resource consumption while still meeting demand, VMs are dynamically reallocated using the VM management policy described above. The policy is executed every 10 minutes.

Results

The experiment was repeated 5 times for each policy. Each experiment is different due to the random offset in the workload traces as well as the random order in which initial VM placement is performed, though each policy was evaluated with the same 5 random configurations. The first day of each experiment was discarded to eliminate the influence of the initial VM placement and allow the system to stabilize before collecting metrics.

Table III lists a select set of metrics from the results of the experiments. Values are averaged over the 5 repetitions, with the standard deviation of the repetitions listed in square brackets. The Static Peak method statically allocated enough resources to handle the maximum load of each VM, and as such we can see that it had no SLA violation, but at the expense of the highest power consumption and lowest host utilization. The Static Average method experienced the highest SLA violation (21.4%), as an insufficient amount of resource was provisioned to meet demand when it exceeded the average. It did, however, have the lowest power consumption and highest host utilization. These represent two extreme cases of VM allocation, and clearly show the tradeoff between performance on one hand, and infrastructure requirements and power consumption on the other. The final experiment, Dynamic Allocation, employed dynamic VM reallocation in order to attempt to achieve the benefits of both Static Peak and Static Average allocation. It had a 0.8% SLA violation, which is more than with Static Peak, but far less than with Static Average. Host utilization and power consumption are also higher than Static Average but represent significant improvements over Static Peak. Overall, we can see that Dynamic Allocation offers many of the benefits of the smaller allocation of Static Average (though to a slightly lesser degree) while still maintaining an acceptable level of SLA violation, thus finding a good balance between the two fixed allocation methods.

D. VM Replication

The third and final experiment is also designed to demonstrate the usefulness and capabilities of DCSim, in this case in simulating the management of VM replication in a replicated and load balanced service tier. As resource demands increase due to increasing incoming workload, more resources may be required than a single VM can provide, even if it is being given all of its requested resources. In this case, a replica of the VM can be created and the incoming workload can be split between them, so that all incoming work can be completed. This is, of course, dependent on whether the application running in the VM can be replicated.

We simulate a set of single-tier web services, running on 100 HP ProLiant DL360G5 host servers (as defined earlier). The hosts were not oversubscribed, thus guaranteeing that each VM can use all of the resources it requires. All VMs are of size 3 as defined in Table I. The workload for each service is chosen randomly from the three HTTP traces only (no Google Cluster Data traces were used). The scale of each workload was chosen randomly so that the peak number of requests can be processed by either 2, 3, 4 or 5 replica VMs. Incoming work is load balanced such that each replica VM in a tier receives an equal amount of work.

A VM management policy was created to handle dynamic service tier replication by automatically scaling the number of VMs in the tier to match incoming workload. If the average utilization of each VM in the tier exceeds an *upper threshold*, the policy adds a new replica to the tier. If, on the other hand, removing a replica would result in the average utilization of the remaining replicas being still below a *lower threshold*, a replica is removed. To select which replica to remove, we select the replica residing on the host with the fewest number of VMs, with the aim to consolidate VMs on as few hosts as possible so that idle hosts may be suspended. No dynamic VM reallocation was performed. We compare the simulation under two policies.

Policy 1 - Static Peak: Each tier is statically allocated enough VM replicas to meet its peak level of resource demand. As defined by the scale of the workload discussed above, this can be either 2, 3, 4 or 5 VMs. No dynamic replication is performed.

Policy 2 - Dynamic Replication: Each tier is initially allocated a single VM. A VM management policy performing dynamic service tier replication is enabled, executing every 10 minutes to add or remove replicas in response to changes in workload. The upper threshold (triggering replications) was set to 85%, and the lower (triggering replica removal) was set to 70%.

Results

The experiment was repeated 5 times for each policy. We simulated 10 days, and discarded the first day to eliminate the influence of the initial VM placement and allow the system to stabilize before collecting metrics. Table IV shows the average results across all repetitions, with standard deviations in square brackets. As we can see, statically creating enough replica VMs to handle peak workload resulted in

	Static Peak	Dynamic Replication
# Replicas Added	0 [0]	18919 [878.6]
# Replicas Removed	0 [0]	18918 [877.2]
Avg. Active Hosts	75.6 [2.6]	46.5 [1.0]
Kilowatt-hours	3412.9 [119.4]	2253.4 [54.3]
SLA Violation	0% [0]	1.14% [0.04]

Table IV: Replication Results

relatively high power consumption and number of active hosts, but suffered no SLA violation. By dynamically adjusting the number of replicas in each service, we were able to significantly reduce power consumption and the number of active hosts, but experienced a 1.14% SLA violation. It did so by dynamically adding and removing replicas about 18918 times over the course of 9 days. Further improvement on these results could be obtained by combining this policy with a dynamic reallocation policy, and through careful tuning of the threshold values.

VI. Conclusions and Future Work

We have presented DCSim (Data Centre Simulator), an extensible simulation framework for simulating a data centre operating an Infrastructure as a Service cloud. DCSim allows researchers to quickly and easily develop and evaluate dynamic resource management techniques. It introduces key new features not found in other simulators, including a multi-tier application model which allows the simulation of interactions and dependencies between VMs, VM replication as a tool for handling increasing workload, and the ability to combine these features with a work conserving CPU scheduler. A number of abstract classes defining host resource management, the application model, and VM management policies, allow the simulator to be easily extended and customized for specific work in a range of key research topics in data centre management.

Through experimental results, we have shown that DCSim is scalable to large simulations (featuring 10000 hosts and 40000 VMs), allowing evaluation of techniques at a speed and scale not possible with a real implementation. We conducted a basic experiment comparing three different VM resource allocation and management methods, in which DCSim was able to provide useful metrics to draw conclusions on the characteristics and tradeoffs between the methods. We conducted a third experiment demonstrating dynamic service tier scaling using VM replication, where DCSim again was able to facilitate a quick and efficient evaluation of the technique. This demonstrates how DCSim can facilitate rapid development, evaluation and feedback on data centre management policies and algorithms.

There are a number of opportunities for future development on DCSim, which include fine-tuning of the simulator as well as the development of extensions to evaluate different aspects of data centre management. We plan to modify the simulator to organize hosts into clusters and racks in order to take such information into account when making management decisions. We plan to implement *memory overcommitting* and model the consequences of exceeding the size of physical memory. Host failure events could be made possible to test fault tolerant management algorithms. Finally, there are countless opportunities for extending the simulator, such as adding the ability to run distributed management algorithms, strategy trees [18], or thermal-aware algorithms [19].

References

[1] B. Stone and A. Vance, "Companies slowly join cloud-computing," New York Times, p. B1, Apr 19, 2010.

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

[3] "Amazon Elastic Compute Cloud," http://aws.amazon.com/ec2/, Aug 2012.

[4] D. Kliazovich, P. Bouvry, Y. Audzevich, and S. Khan, "Greencloud: A packet-level simulator of energy-aware cloud computing data centers," in *IEEE Global Telecommunications Conference (GLOBECOM)*, Dec 2010, pp. 1–5.

[5] "Ns-2," http://isi.edu/nsnam/ns/, Aug 2012.

[6] S. H. Lim, B. Sharma, G. Nam, E. K. Kim, and C. Das, "Mdcsim: A multi-tier data center simulation, platform," in *IEEE International Conference on Cluster Computing and Workshops (CLUSTER)*, Sep 2009, pp. 1–9.

[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 *International Green Computing Conference and Workshops (IGCC)*, Jul 2011, pp. 1–8.

[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," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[9] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, ser. CCGRID '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 826–831.

[10] "Kernel Basic Virtual Machine," http://www.linux-kvm.org/, Aug 2012.

[11] "Xen Hypervisor," http://xen.org/, Aug 2012.

[12] "VMWare," http://www.vmware.com/, Aug 2012.

[13] "Standard Performance Evaluation Corporation," http://www.spec.org/, Aug 2012.

[14] L. Grit, D. Irwin, A. Yumerefendi, and J. Chase, "Virtual machine hosting for networked clusters: Building the foundations for "autonomic" orchestration," in *International Workshop on Virtualization Technology in Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 7–.

[15] 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 and Computation: Practice and Experience*, 2011.

[16] "The Internet Traffic Archive," http://ita.ee.lbl.gov/, Aug 2012.

[17] "Google Cluster Data," http://code.google.com/p/googleclusterdata/, Aug 2012.

[18] B. Simmons and H. Lutfiyya, "Strategy-trees: A feedback based approach to policy management," in *Modelling Autonomic Communications Environments*, ser. Lecture Notes in Computer Science, S. van der Meer, M. Burgess, and S. Denazis, Eds. Springer Berlin / Heidelberg, 2008, vol. 5276, pp. 26–37.

[19] F. Norouzi and M. Bauer, "Compromise between energy consumption and qos," in *19th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Sept 2011.