Hybrid resource provisioning for clouds

Mahfuzur Rahman and Peter Graham
Dept. of Computer Science, University of Manitoba, Winnipeg, MB, Canada R3T 2N2
E-mail: {mahfuz, pgraham}@cs.umanitoba.ca

Abstract. Flexible resource provisioning, the assignment of virtual machines (VMs) to physical machine, is a key requirement for cloud computing. To achieve “provisioning elasticity”, the cloud needs to manage its available resources on demand. A-priori, static, VM provisioning introduces no runtime overhead but fails to deal with unanticipated changes in resource demands. Dynamic provisioning addresses this problem but introduces runtime overhead. To reduce VM management overhead so more useful work can be done and to also avoid sub-optimal provisioning we propose a hybrid approach that combines static and dynamic provisioning. The idea is to adapt a good initial static placement of VMs in response to evolving load characteristics, using live migration, as long as the overhead of doing so is low and the effectiveness is high. When this is no longer so, we trigger a revised static placement. (Thus, we are essentially applying local multi-objective optimization to tune a global optimization with reduced overhead.) This approach requires a complicated migration decision algorithm based on current and predicted future workloads, power consumptions and memory usage in the host machines as well as network burst characteristics for the various possible VM multiplexings (combinations of VMs on a host). A further challenge is to identify those characteristics of the dynamic provisioning that should trigger static re-provisioning.

1. Introduction

The growth in popularity of clouds for providing IT services continues at a rapid pace. Cloud-based approaches are attractive because enterprises do not need to purchase, maintain and support their own computing infrastructure which often represents a significant and sometime hard to predict operational cost. Further, the large scale of cloud infrastructure (tens of thousands of cores and beyond) allows cloud providers to optimize the efficiency of the systems, via the use of virtual machines (VMs) to permit effective sharing, so they can offer services more cost-effectively than end users could. This makes the use of clouds price-attractive as well.

End users and cloud providers typically enter into contracts, quantified as service level agreements (SLAs), to describe the expected requirements, in terms of computing, etc. capacity, required for each user application. Cloud providers use this information to determine the amount of infrastructure they need to purchase and later allocate to users when their applications run as well as the cost to the users. This allocation is done by “provisioning” algorithms that pack virtual machines running end-user applications into physical cloud infrastructure (compute nodes) that the cloud provides.

For applications that have predictable and generally unchanging demands/workloads, it is possible to use “static provisioning” effectively. In this approach VMs are carefully packed into clouds as they are brought online and continue to execute effectively for potentially very long periods of time on the selected nodes. Using static partitioning, the cost of making the placement
decision is incurred once, before the user’s application is running, and therefore may safely incur greater overhead to achieve a better result without negatively impacting the end users. In cases where demand by applications may change or vary, “dynamic provisioning” techniques have been suggested whereby VMs may be migrated on-the-fly to new compute nodes within the cloud. This allows the system to adapt to changing conditions but at the cost of runtime overhead and potential execution delays that may negatively affect end users. To minimize this overhead, restrictions on the complexity of provisioning algorithms are necessary.

To effectively address scenarios with changing load (e.g. web servers hosting hot content) we propose a hybrid provisioning approach that starts with a good static placement that can be tuned by dynamic re-provisioning as long as doing so is effective and not excessively costly. At that point, we propose to incur the greater cost of some form of static re-provisioning of potentially all running VMs based on the newly available demand information to return the system to a state of efficient sharing and execution. This can be done concurrently with the ongoing execution of VMs in the cloud as long as there are available idle resources – a likely situation or one that can be planned for.

To implement and assess hybrid provisioning strategies, it is necessary to have both a testbed environment for verifying the correctness and small-scale performance of developed algorithms as well as a simulation system to assess larger-scale characteristics. In this paper, we describe our extensions to the Eucalyptus open source cloud system to support hybrid provisioning, we discuss our hybrid algorithm framework and then briefly discuss our planned evaluation strategy using CloudSim based on the framework and results from our modified version of Eucalyptus.

The rest of this paper is organized as follows. Section 2 briefly reviews work related to this paper. Section 3 states the problem to be solved and Section 4 describes the proposed solution. Our prototype, framework and planned assessment strategy are discussed in Section 5. Finally, Section 6 presents our conclusions and our planned directions for future work.

2. Related Work

Cloud computing [1–3] is based on large-scale computing infrastructure and introduces an alternative way to efficiently deliver IT services. Clouds can offer hardware resources (Infrastructure-as-a-Service/IaaS), software platforms (Platform-as-a-Service/PaaS) or hosted applications (Software-as-a-Service/SaaS) on an on-demand basis. Clouds consist of a huge number of LAN-interconnected physical machines supporting carefully assigned virtual machines that ultimately run the end-user applications that are accessed remotely by clients.

2.1. Provisioning in Clouds

In clouds, application developers, who are sometimes referred to as SaaS (Software as a Service) providers or cloud clients, pay to allocate as much computational resources; memory space, disk storage, network bandwidth, etc. as they need to host their applications based on their current and/or expected load. Depending on SaaS providers’ requirements, cloud infrastructure providers allocate the necessary resources, in the form of Virtual Machines (VMs). Efficient resource utilization is a key requirement in clouds from the provider’s perspective and VMs are used in cloud computing to enable efficient resource provisioning as well as enhanced security. Cloud-based data centers normally provide a huge amount of hardware resources (CPU, memory, network capacity, etc.) in the form of a large number of compute nodes networked together to form a massive, efficiently sharable computing resource. These physical resources in the cloud are then made available to different VMs to run applications. VM provisioning helps ensure efficient application execution, load balancing and proactive failure handling, and thus can increase the reliability and efficiency of the whole cloud system and the applications running in it. VMs may be assigned to the physical machines in a cloud using either static or dynamic VM provisioning,
In static VM provisioning, the required hardware resources are selected as the VMs are requested based on agreed-to Service Level Agreements (SLAs) but in the case of dynamic VM provisioning, the resources used can be changed dynamically to handle unexpected fluctuations in workloads. Simple static algorithms track available resources and allocate VMs to physical machines using techniques designed to achieve certain goals. Techniques such as Round Robin (where VMs are evenly spread over the physical machines to achieve load balance and, presumably, good performance) and Greedy (where the goal is to pack VMs onto the fewest physical machines to achieve low power consumption since unused machines may be powered down) are examples of such algorithms. Of course, for both static and dynamic provisioning, “sizing” of the required resources must be done to determine the appropriate hardware resources needed. Meng et al. [4] introduced joint VM sizing which can aggregate the total capacity requirements of multiple VMs. Joint VM sizing is useful when multiple VMs are hosted in the same physical host and the unused resources, identified by previous resource usage patterns and/or future usage prediction, of one VM can be allocated to other VM(s). Total capacity measurement is used to select the appropriate VMs to host in a particular physical machine. In static VM provisioning, the resources that are allocated to a VM may not be fully utilized and in that case VM multiplexing helps to improve utilization of the hardware resources. As part of their work, Meng et al. also developed a workload forecasting model to determine compatible VMs that can be consolidated on a single physical machine without violating any of the VM’s SLAs. Fito et al. [5] developed related criteria for cloud-based SLAs and a model to monitor SLA satisfaction in clouds.

In dynamic VM provisioning, additional required resources can be dynamically allocated for a VM when its resource requirement exceeds the currently allocated resources. Also resources can be de-allocated when a VM doesn’t need them. Kim et al. [6] proposed a prediction-based dynamic VM provisioning model based on the previous resource usage patterns of a VM. They introduced pattern based provisioning (where additional resources are provisioned depending on resource usage patterns) to avoid the negative affects of threshold based provisioning (where additional resources are provisioned only when a certain threshold of resource use has already been crossed) which sometimes leads to frequent and inappropriate provisioning requests. Rolia et al. [7] also used historical resource use information to predict future resource requirements for applications. Though previous use information may not always provide accurate prediction, especially for web-based interactive applications and in the case of shared virtualized infrastructure where resource availability also depends on the requirements of co-hosted applications and their priorities, this approach can decrease the number of active hosts/nodes thereby decreasing cloud resource requirements and power consumption.

2.2. Virtual Machine Migration
Sometimes the resource requirements of a VM may exceed the capacity of the current host. In that case, the VM needs to be migrated to another, more suitable host. (This is how dynamic VM provisioning is effected.) VMs can also be migrated off physical machines when the current host requires maintenance or to reduce power consumption by consolidating VMs into a smaller number of hosts so the unused hosts can be shut down. Clark et al. [8] described “live” migration of a VM across physical hosts where execution carries on either in the source or in the destination host during the transfer of memory contents. At a certain point, VM migration needs to stop execution in the VM at the source host and to start execution at the destination. The relatively short time between these events is when no progress can be made. In VM migration, any such downtime is important to minimize since it may degrade the system’s ability to meet the relevant VM SLAs. In VM migration, CPU and network overheads may also degrade application performance. In spite of this, VM migration is very useful to deal with runtime overload situations, especially in the case of sustained overload [9]. Hines et al. [10]
compare pre-copy and post-copy based live migration of VMs. In pre-copy based live migration, the CPU state transfer takes place after the memory transfer phase and in post-copy based live migration, the CPU state transfer takes place before the memory transfer phase.

2.3. Assessing Cloud Systems

Assessment of cloud systems (including VM provisioning techniques) is complicated by the scale of cloud infrastructure. Few researchers have access to large-scale cloud facilities or data from them that they can use to do assessments. Small scale prototype systems can and should be used to demonstrate algorithm correctness but such systems cannot be used to assess scale-related effects. To address this shortcoming, some form of simulation is therefore also required. Most cloud systems software is commercial and hence proprietary and unavailable to academic researchers. An exception is the Eucalyptus [11] open source cloud system which provides a basic but largely complete cloud testbed environment supporting cloud infrastructure consisting of clusters containing compute nodes running a variety of operating systems under a number of different virtual machine management packages. A number of simulation systems have also been developed for assessing cloud-based systems. Perhaps the most common and widely used of these is the Cloudsim [12, 13] simulator which supports the simulation of both the system and behaviour of cloud components including VMs, resource provisioning policies, etc. Of importance to the work described in this paper, it also provides custom interfaces for implementing new provisioning techniques. An issue related to the simulation of clouds is the generation of cloud workloads to use with simulation systems. As many cloud applications are web-based, the use of tools such as Olio [14] and the Faban web load generator [14, 15] to create reasonably realistic synthetic workloads is common.

3. The Problem

A resource provisioning “bottleneck” occurs when a cloud cannot provide the needed resources to applications when those applications need them (e.g. in response to changing load conditions). This means that SLAs may not be met in the face of increased load. Static VM provisioning, the most common current strategy, introduces no runtime VM management overhead but fails to address such resource bottleneck problems. On the other hand, dynamic VM provisioning can address or, possibly, even proactively avoid resource provisioning bottlenecks, but it introduces additional VM management overhead due to the need to predict future application workloads, and then allocate VMs to appropriate physical machines at runtime. Ideally, a provisioning solution would be able to offer the adaptability of dynamic techniques but with less VM management overhead than is currently the case using dynamic provisioning. To accomplish this, a different approach is required.

4. The Proposed Solution

The challenge of dynamic provisioning is to ensure that the benefit of any migration significantly outweighs the overhead of doing so. We propose to combine the best features of both static and dynamic VM provisioning via a hybrid approach where dynamic tuning carefully refines a good initial static placement and, if necessary, may eventually trigger a revised static placement. At least two challenges must be met in any such hybrid provisioning technique. First, it must be possible to determine when the migration of a VM should take place to avoid resource provisioning bottlenecks and to provide efficient and cost effective use of physical machines in the cloud. Second, it must be possible to determine whether or not a proposed migration will provide ongoing performance benefit. Meeting these challenges will require an accurate prediction model that will depend on both the current and predicted future workload of the current host, current and predicted future workload of other hosts, power consumption (if the selected destination...
host needs to be powered on), memory usage patterns, network burst characteristics, various multiplexing possibilities with other VMs, etc.

By combining static and dynamic provisioning we expect to create a system where a good, inexpensive static provisioning can be cost-effectively “tuned”, when necessary, by a careful and hence lower overhead dynamic migration-based technique thereby providing adaptation to unanticipated changes in cloud workload. Further, we expect to be able to identify characteristics of the dynamic provisioning done that can be used to trigger static re-provisioning to limit the need for future dynamic provisioning and associated overhead (such static re-provisioning can be done, up to the point of migrating VMs, entirely in parallel with ongoing operations in the cloud) thereby limiting the impact on VM performance and SLA conformance. Finally, we plan to develop a technique for selecting the order of VM migration to achieve a selected re-provisioning that will minimize disruption to computations taking place in the cloud.

5. Our Prototype and Framework
To assess the effectiveness of hybrid algorithms for resource provisioning in clouds we have to use a mixture of prototype implementation and simulation. This paper addresses the development of a suitable testbed prototype based on an extension of the Eucalyptus open source cloud system that is used to ensure algorithm correctness. We begin by describing the structure of Eucalyptus and then the changes that were necessary to make to it so it could serve as a platform for testing hybrid provisioning algorithms.

5.1. Eucalyptus
Eucalyptus implements IaaS (Infrastructure as a Service) level cloud computing functionality which is very similar to Amazon’s EC2 [16] and can be easily extended due to its simple organization and modular design. There are three major high level components in Eucalyptus: the Node Controller (NC), the Cluster Controller (CC) and the Cloud Controller (CLC) as shown in Figure 1.

![Figure 1. Different Levels in Eucalyptus (adapted from [11])](image)

Node Controllers (NCs) are installed on each physical node/host machine that is part of the Eucalyptus cloud infrastructure. Each NC makes the physical resources (i.e. computing core(s), memory, storage etc.) of the associated node available to the Eucalyptus cloud. When any new node is attached to a cloud, the Cloud Controller (CLC) can then allocate and manage its physical resources in the form of VM instances via the NC of that particular node. Nodes, having the same type of hypervisor, are arranged under a cluster where each cluster contains
a Cluster Controller (CC). The single CLC communicates with NCs through each particular cluster’s CC. Each NC can query, control, allocate and/or manage the host operating system, the hypervisor of the node, VM instance(s) on the node, etc. upon request from the CLC.

Cluster Controllers (CCs) act as intermediaries between the CLC and the NC(s) registered in the corresponding cluster. Each CC gathers VM instance information from its associated nodes. Each CC also handles requests for new VM instance creation (from the CLC) by determining a node that has enough free resources to satisfy the new VM instance’s resource requirements (i.e. it does the VM provisioning). Additionally, each CC passes-thru VM instance termination or reboot requests from CLC to the relevant node’s NC. The interfaces that maintain the communication between CC and NC is described using WSDL/SOAP.

The Cloud Controller (CLC) is the management entry point and major decision making component in Eucalyptus. The CLC is accessed using a number of web-services which can be categorized into: a resource management service, a data service and an interface service. Resource management services interact with CC(s) to allocate and de-allocate physical resources and to create and control VM instances. The data services create, integrate, modify and contain information related to users, systems and data storage. This information normally includes key-pairs, security groups, network definitions, management and monitoring information of VM instances, etc. The interface services are provided to allow communication with internal Eucalyptus objects using standard WS-security policies and to initiate high-level system management tasks.

At present there are three static VM provisioning techniques implemented (by the CCs) in Eucalyptus: the Round Robin, Greedy and PowerSave algorithms. The Round Robin algorithm allocates new VMs to the host machines in a circular fashion distributing the load across the available nodes. The Round Robin algorithm maintains a list of host machines and also a counter for identifying the next available host machine in that list. The counter is incremented modulo the number of nodes after each new VM allocation to point to the next host machine in the list where the next VM allocation will take place. The Round Robin algorithm can prevent starvation as it considers each host machine equally and it is very good for balancing load (i.e. equal distribution of total workload among the host machines in the cloud). The Greedy algorithm maintains a sequence of host machines with available capacity and when a new VM needs to be allocated, the Greedy algorithm uses that sequence to find possible VM allocation locations. The Greedy algorithm allocates the new VM to the first host machine meeting the needs of the new VM thereby packing VMs onto as few nodes as possible. The Greedy algorithm provides locally optimal allocation but often fails to achieve the best overall optimal solution as it always considers the first possible solution without considering other possible allocation locations. Though the Greedy algorithm cannot balance the total VM workload among the host machines, it is faster because of less allocation workload. The PowerSave algorithm enhances the Greedy algorithm by putting host machines to sleep when they are not running any VM and reawakens a host machine when it is needed for allocating new VM(s). By shutting down the unused host machines, the PowerSave algorithm can reduce overall power consumptions but it decreases the speed of allocation due to the need to reawaken a node if no capacity exists on other nodes.

5.2. Implemented Changes to Eucalyptus

In Eucalyptus, the CLC contains ‘euca2ools’ which is used to query and display information about currently running VM instances in the cloud. The information produced provides VM instances’ ID along with their IP addresses and the relevant cluster’s name. This information can be used to access any particular VM instance, through its IP address, to terminate or to reboot the VM instance. But the query result doesn’t include information related to the node where the VM instance is currently running. The query result also doesn’t provide any
resource usage information for VM instances. Both of these pieces of information are required to support a cloud-wide hybrid provisioning approach. We have therefore enhanced Eucalyptus so that a query result now also provides the corresponding nodes’ performance information, considering available and used resources, along with their hosted VM instances’ performance information, also considering allocated and used resources. We have also modified the data structures maintained in the CLC, CCs, and NCs and also the message passing parameters among them to permit collection and storage of the required performance information. This performance information provides the basis upon which we can identify over-utilized and under-utilized nodes and VM instances in the cloud that may need to be migrated as part of a hybrid provisioning technique.

![Diagram](image)

**Figure 2.** Modified Communications (blue arrows) and Control (brown arrows) in our Modified Eucalyptus (compare to Figure 1).

Currently in Eucalyptus the VM provisioning sub-module occurs in each CC and provides different provisioning algorithms (such as Round Robin, Greedy, etc.) to identify a suitable node to place each new VM instance on as it is created. This obviously does not permit provisioning across the cloud as a whole and also prevents the possibility of migration across clusters in the cloud (due to lack of overall placement information) thereby limiting choices in re-provisioning. We have therefore moved the VM provisioning sub-module to the CLC level instead of at the CC level. This rearrangement allows us to allocate new VM instances or to migrate existing VM instances to a node across clusters in the cloud. In the CLC, we have also introduced a new communication interface permitting migration decisions to be delivered to the affected NCs. To do this, each CC was modified to pass-thru VM instance migration requests from the CLC to the relevant node’s NC and we have made each NC capable of directing its hypervisor to perform live migration of a VM instance. (Refer to the control communication paths on the left side of Figure 2). We have also enhanced the NCs to communicate with their underlying hypervisors to query and collect VM instances’ performance information and we have modified the CCs so that when they gather VM instance information from their associated nodes they also collect the additional resource usage information of the nodes and the VM instances running on them. Additionally, each CC summarizes the information collected and forwards the information up to the CLC, as shown in the data communication paths on the right side of Figure 2. Finally, we have made it so each NC periodically refreshes and updates the CCs about the resource use by their nodes and VM instances.

All of these changes have been implemented in Eucalyptus and deployed on a small scale cloud testbed system running the modified Eucalyptus.
5.3. Framework for Hybrid Provisioning Algorithms

Given the pre-requisite ability to collect performance information on VM instances and nodes, the first component of a hybrid provisioning scheme, assuming an existing static placement mechanism (as is available in Eucalyptus), is a mechanism to decide when and which VMs should be migrated. As discussed earlier, this must include a complex heuristic assessing the costs and benefits of a VM migration which includes many factors. A key challenge to implementing such a heuristic is that migration decisions must not be made based on transient changes in system load and hence performance. Reacting too quickly to changes in load may result in unnecessary and ultimately non-beneficial VM migrations. An appropriate heuristic must be able to ensure that the benefits of migration will continue to be effective over time. Ultimately, the inability to meet this criteria suggests that a static re-balancing (i.e. re-running a static provisioning algorithm for all existing VMs) may be necessary. We now sketch our initial framework for supporting such heuristic algorithms. Our framework for migrating a single VM based on a threshold being exceeded over some period of history is shown in Algorithm 1. Naturally, use of this framework provides for local but not global optimization in re-provisioning.

To implement the framework, each CC collects the performance information from each of its registered nodes and the VMs running in them using getNodeResourceUsageHistories() [line 1, Algorithm 1]. The collected information is forwarded to the CLC by each CC via getClusterResourceUsageHistories() [line 2]. After aggregating and sorting the performance information from the CCs according to load/utilization [lines 3–4], the CLC iterates through the usage records corresponding to over-utilized nodes [lines 5-6]. For each such node it generates a set of potential underUtilizedNodes that can support the originally specified (i.e. SLA) requirements (r.resourceRequirements) via a call to findUnderUtilizedNodes() [line 7]. The nodes generated are used [line 8] together with the history of actual resource usage (r.resourceUsageHistory) to create a list of selectedNodes that could be good migration targets. Information in r.resourceUsageHistory permits assessment of past variation in use from the expected use (r.resourceRequirements) which is used in predicting future use. The list of potential migration targets is ordered by the expected benefit of migration to that node. The function benefitAnalysis that produces the list considers such additional factors as the predicted longer-term network bandwidth and latency available, the cost of memory state migration, file migration and network re-configuration as well as the benefits of power savings etc. Migration is only done if the benefit of the migration to the selectedNode exceeds another threshold (thresholdBenefit) [lines 10-11].

In our framework, resource requirements, resource usage and threshold information are all compound representations encompassing such factors as CPU core, memory, storage, network, etc. need, use and limits, respectively. While the pseudo-code in Algorithm 1, presents operations on these values as simple comparisons, they are not. In different situations, the operations required may differ in behaviour to ensure appropriate outcomes. Generally, comparisons are done component-wise (e.g. CPU need vs. CPU available, memory need vs. memory available, etc.). Simple absolute comparisons of components, however, may be inappropriate in some cases so a “fuzzy” form of comparison can be done where the degree of flexibility may vary between component values during comparisons. (E.g. The need for CPU capacity may be harder than for expected available network capacity.) Similarly, the thresholds referred to in the algorithm (thresholdResourceUsage and thresholdBenefit) appear to be absolute and arbitrary. Initially these values will be absolute but must be carefully chosen experimentally. In the future, we will make the setting of these thresholds dynamic based on sampled system behaviour to adapt to changing global conditions in the cloud system and thereby improve the overall performance benefit of the migrations selected.
Hybrid Algorithm Framework: Threshold-Driven Dynamic Provisioning (Single VM)

1: clusterHistories ← getNodeResourceUsageHistories( )
2: cloudHistory ← getClusterResourceUsageHistories(clusterHistories)
3. usageRecords ← extractRecords(cloudHistory)
4. usageRecords ← sortRecordsByUsage(usageRecords)
5. foreach record r in usageRecords do
6. if r.resourceUsage ≥ thresholdResourceUsage then
7. underUtilizedNodes ← findUnderUtilizedNodes( r.resourceRequirements )
8. selectedNodes ← benefitAnalysis(underUtilizedNodes, r.resourceUsageHistory)
9. selectedNode ← first element in selectedNodes
10. if selectedNode.benefit ≥ thresholdBenefit then
11. performMigration( r.InstanceId, selectedNode)
12. end if
13. else
14. exit // no remaining records exceed threshold
15. end if
16.end for

Algorithm 1.

5.4. Status and Planned Evaluation Strategy
At this point we have made all the changes to Eucalyptus and tested their functionality. We are refining the details of the framework described and should be implementing and testing the first of a family of algorithms shortly. Once we have working algorithms, it will be necessary to evaluate their effectiveness against static provisioning alone. Intuitively, as the workload variation increases over time, the effectiveness of rigid static provisioning should decrease. At this point, application of dynamic re-provisioning, as described, should improve effectiveness. If the changes in workload are gradual dynamic re-provisioning alone should remain effective for some time. However, if the changes are more significant and/or as changes compound over time we expect the overhead of dynamic re-provisioning to grow to the point where it represents a significant drain on the cloud system’s capacity. It is at this point that we will trigger a static repartitioning (i.e. the use of a static algorithm that will look at a large-scale re-provisioning for all running VMs). While the cost of such a re-partitioning will be high (and will, necessarily, have to be done in a phased-approach to avoid disruption of cloud services) the end result should limit the need for ongoing migrations and their associated overhead. This leads directly to the question of “How will we assess the effectiveness of our hybrid provisioning algorithms?”

Ultimately, any evaluation must focus on how much the proposed integration of static and dynamic VM provisioning improves overall resource utilization in a cloud while maintaining SLA commitments. We will conduct experiments to measure capacity savings with and without using hybrid techniques. We will first test each of our implemented algorithms in our small scale modified Eucalyptus cloud testbed environment to assess correctness and accuracy, in terms of selecting the appropriate VMs to migrate, and to collect actual performance data (e.g. live migration costs). We will then do further, scale-related experiments in a simulated cloud environment based on Cloudsim.
We will use Cloudsim [12, 13] to create a much larger simulated cloud environment for our assessment of scale related effects. From our initial evaluation it seems that the Cloudsim simulator will support the straightforward addition of the necessary hybrid provisioning policies and VM migration behaviour. As we do not currently have access to cloud system trace data we plan to follow common practice and use Olio [14] along with the Faban load generator [14, 15] to create reasonably realistic synthetic workloads to drive our simulations. Olio allows the creation of sample Web 2.0 applications to evaluate the performance of web based cloud applications. Faban can be used along with Olio to create a Markov-chain based workload for the VMs (this approach was previously used in [17]). We will also continue looking for suitable trace data from real world cloud implementations which would help to provide validation of our simulation experiments, if obtained.

Using our simulated cloud environment, we will assess the effectiveness and duration of migration in a large-scale, realistic cloud scenario based on data collected from the prototype implementation. We will also analyze trends in the effectiveness of migration over time to help determine when and under what conditions, static re-provisioning may be necessary. Our performance metric will be the number of VM migrations eliminated using our approach. Fewer VM migrations reduces overhead and thus improves overall system performance. We will need to assess the total number of VM migrations within a time period for different workloads, considering frequency and extent of change in VM workloads. For migrating VMs, our proposed algorithm will select appropriate VM(s) to migrate to capable alternative physical machine(s) considering total downtime, overall migration time, maximum number of hosts it will allow to shutdown, etc. We will design our simulation experiments to collect information on total downtime, migration time and power saving with and without hybrid provisioning.

6. Conclusions and Future Work
In this paper we have motivated the need for a hybrid (static/dynamic) VM provisioning approach in clouds, discussed the changes we have made to the Eucalyptus open source cloud system to support hybrid provisioning, and described our initial framework for developing hybrid provisioning heuristics as well as our planned evaluation strategy. In the short-term, we will be implementing our first threshold-driven dynamic provisioning algorithms, verifying their correctness using our testbed system and assessing their basic performance characteristics (e.g. migration overhead) for later use in simulation experiments. We will also be implementing more advanced static provisioning algorithms in our modified version of Eucalyptus to ensure that fair comparisons are possible. We will then explore techniques for making the setting of the framework thresholds dynamic based on global cloud conditions. We then plan to explore two additional extensions to enhance over performance of the framework and algorithms implemented using it: (i) dynamically adjusting the historical periods over which resource use is evaluated, and (ii) working towards global rather than local optimization of migration decisions by using a pre-processing step to identify similarly mis-behaving VMs that should be considered for migration as a group. Finally, we will explore the characteristics of our dynamic migration algorithms that should be used to trigger to re-static provisioning, implement that process and assess its cost and effectiveness.

Acknowledgments
This research was supported in part by the Natural Sciences and Engineering Research Council of Canada under grant number RGPIN 194227-01 and also by the University of Manitoba Graduate Fellowship (UMGF) program.
References

[1] Armbrust M, Fox A, Griffith R, Joseph A D, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I and Zaharia M 2010 Commun. ACM 53 50–58 ISSN 0001-0782 URL http://doi.acm.org/10.1145/1721654.1721672

[2] Rimal B, Choi E and Lumb I 2009 INC, IMS and IDC, 2009. NCM ’09. Fifth International Joint Conference on pp 44–51

[3] Zhang S, Zhang S, Chen X and Huo X 2010 Future Networks, 2010. ICFN ’10. Second International Conference on pp 93 –97

[4] Meng X, Isci C, Kephart J, Zhang L, Bouillet E and Pendarakis D 2010 Proceeding of the 7th International Conference on Autonomic Computing ICAC ’10 (New York, NY, USA: ACM) pp 11–20 ISBN 978-1-4503-0074-2 URL http://doi.acm.org/10.1145/1809049.1809052

[5] Fito J O, Presa I G and Guitart J 2010 Parallel, Distributed, and Network-Based Processing, Euromicro Conference on 0 111–118 ISSN 1066-6192

[6] Kim H, Kim W and Kim Y 2010 Grid and Distributed Computing, Control and Automation (Communications in Computer and Information Science vol 121) ed Kim T h, Yau S S, Gervasi O, Kang B H, Stoica A and Dominik (Springer Berlin Heidelberg) pp 84–94 ISBN 978-3-642-17625-8

[7] Rolia J, Cherkasova L, Arlitt M and Andrzejak A 2005 Proceedings of the 5th International Workshop on Software and Performance WOSP ’05 (New York, NY, USA: ACM) pp 229–237 ISBN 1-59593-087-6

[8] Clark C, Fraser K and Steven H 2005 URL http://www.usenix.org/events/nadi05/tech/clark.html

[9] Amza C, Chanda A, Cox A, Ehikety S, Gil R, Rajamani K, Zwaenepoel W, Cecchet E and Marguerite J 2002 Workload Characterization, 2002. WWC-5. 2002 IEEE International Workshop on pp 3 – 13

[10] Hines M R and Gopalan K 2009 Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments VEE ’09 (New York, NY, USA: ACM) pp 51–60 ISBN 978-1-60558-375-4 URL http://doi.acm.org/10.1145/1508293.1508301

[11] Inc E S Learn about cloud computing http://open.eucalyptus.com/learn/what-is-eucalyptus accessed on April 25, 2011

[12] Calheiros R N, Ranjan R, Rose C A F D and Buyya R 2009 CoRR abs/0903.2525

[13] Shi Y, Jiang X and Ye K 2011 Cluster Computing (CLUSTER), 2011 IEEE International Conference on pp 595 –599

[14] Foundation A S Olio http://incubator.apache.org/olio/ accessed on April 25, 2011

[15] Bahga A and Madisetti V K 2011 Journal of Software Engineering and Applications 4 396 – 410

[16] EC2 Amazon elastic cloud computing http://aws.amazon.com/ec2/ accessed on March 15, 2011

[17] Voorsluys W, Broberg J, Venugopal S and Buyya R 2009 Proceedings of the 1st International Conference on Cloud Computing CloudCom ’09 (Berlin, Heidelberg: Springer-Verlag) pp 254–265 ISBN 978-3-642-10664-4