Agility in Virtualized Utility Computing *

Hangwei Qian[1], Elliot Miller[2], Wei Zhang[2], Michael Rabinovich[1], Craig E. Wills[2]
[1]EE&CS, Case Western Reserve University, Cleveland, OH 44106
[2]CS, Worcester Polytechnic Institute, Worcester, MA 01609
[1]{hangwei.qian,michael.rabinovich}@cwru.edu, [2]{eamiller,weizhang,cew}@cs.wpi.edu

ABSTRACT

Virtual machines have emerged as an attractive approach for utility computing platforms because applications running on VMs are fault- and security- isolated from each other, yet can share physical machines. An important property of a virtualized utility computing platform is how quickly it can react to changing demand. We refer to the capability of a utility computing platform to quickly reassign resources as the *agility* of the platform. We are targeting hosting utility provider environments where the entire platform is under the control of a single administrative domain and application instances often form application-level clusters. In this work, we examine resource reassignment mechanisms in these environments from the agility perspective and outline a new mechanism that exploits properties of a virtualized utility computing platform.

This new mechanism employs *ghost* virtual machines (VMs), which participate in application clusters, but do not handle client requests until activated by the resource management system. We evaluate this, as well as other, mechanisms on a utility computing testbed. The results show that this ghost VM approach is superior to other approaches in its agility, and allows a new VM to be added to an existing application cluster in a few seconds with negligible overhead. This is a promising result as we develop resource management algorithms for a globally distributed utility computing platform.

Categories and Subject Descriptors

C.5 [**Computer System Implementation**]: Servers

General Terms

Experimentation, Measurement, Performance

Keywords

Virtualization, Utility Computing, Agility

1. INTRODUCTION

Web applications are often exposed to unpredictable and rapidly changing demand from the Internet users, making proper resource provisioning a challenging problem. Provisioning too little entails a risk of missing business; provisioning too much wastes resources. The concept of utility computing has gained popularity as an efficient and scalable way to deliver computing resources to applications in this environment. Utility computing promises significant benefits of scale to large hosting service providers. By dynamically reassigning resources among multiple applications, a utility computing platform can efficiently satisfy aggregate demand for these applications. Andrzejak et al. [1] found that by dynamically redistributing resources among applications, a hosting service provider can cut its resource requirements by half.

An important property of a utility computing platform is how quickly it can react to changing demand. Flash crowds have been shown to produce rapid surges in demand [3, 11], and the quicker the system can reassign resources the less slack it needs to provision for each application and the more efficient its operation will be. We refer to the capability of a utility computing platform to quickly reassign resources as the *agility* of the platform. In this paper we examine resource reassignment mechanisms from the agility perspective and outlines a new mechanism that promises to significantly improve agility at a negligible cost.

Our approach is based on using virtual machines, the technology increasingly employed in hosting platforms already, and was inspired by the observation that a virtual machine can be resumed from a suspended state quickly provided the suspended VM remains in memory. Unfortunately, application servers that implement a particular application usually operate in an application cluster; the application servers on a suspended machine will be considered inoperable by the cluster and, as we show later, will take a long time to reintegrate into the cluster upon reactivation.

Consequently, we propose to have a small pool of spare VMs on every physical machine, which remain active but detached from the Internet. Since they cannot be reached by the clients, these VMs will not service requests yet their

*This material is based upon work supported by the National Science Foundation under Grants CNS-0615190, CNS-0551603, and CNS-0615079. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

VTDC'07, November 12, 2007, Reno, NV, USA.

application servers will participate in their respective clusters, which are managed over the private network. We call these VMs *ghost* virtual machines because their existence is "invisible" to the content switch receiving and redirecting client requests. Ghost VMs only consume the CPU for application cluster management, and we show that this CPU consumption is negligible. We retain only a small enough number of ghost VMs on each physical machine to ensure that most of their memory footprint does not page out. Resource reassignment amounts only to the reconfiguration of the content switch, which takes on the order of single seconds as opposed to minutes in existing approaches.

The concept of ghost VMs introduces a hierarchy of states for a VM in a utility computing platform: active, ghost, suspended, and not booted. The resource management algorithm can now move VMs between these states on each machine, and in a global platform, between data centers. Because of the state hierarchy, the resource management problem becomes loosely analogous to cache management. Development of resource management algorithms in this setting is a subject of our ongoing work.

In the remainder of this paper we elaborate on the use of virtualization for utility computing platforms and possible techniques for improving the agility of these platforms. We move on to describe preliminary experiments and results to measure the effectiveness of these techniques. Finally we present related work along with a summary of our current status and plans for future work.

2. ENVIRONMENT

We consider the "utility provider" model for service-oriented computing, where multiple applications are hosted on a platform operated by a single administrative domain—the hosting service provider. Note that this model is different from the grid computing model, which manages resources across multiple organizations. We use the term Utility Computing for Internet Applications (UCIA) to describe this problem space of service-oriented computing platforms operated by a single administrative domain. UCIA platforms allow tighter cooperation between different components, potentially enabling more efficient mechanisms and policies than when multiple organizations are involved.

Figure 1 shows a high-level architecture typical of the platforms we are targeting. The platform consists of a number of data centers (only two are shown), with each data center having a server farm. Depending on the demand, an application can be allocated resources in multiple data centers and on multiple physical machines in each data center. DNS resolution is used to distribute user requests globally among data centers, and the load balancing switch in the selected data center performs local load balancing among application instances on the server farm. Optimizations and variations of this general setup are possible, but are peripheral to our current work.

UCIA platforms target applications that often follow a three-tier architecture: the Web tier, the application server tier (usually implementing the business logic in the J2EE environment) and the back-end database tier. The first two tiers are often co-located on the same physical machine, while the back-end tier is often located outside the UCIA platform's control at the customer premises. Different applications exhibit the bottleneck at different tiers. Our research focuses on the J2EE and the Web tiers. Other efforts have targeted the back-end tier (see, e.g., [6]).

Running Internet applications in a UCIA environment requires support from application servers for software and data replication, client session state management, fail-over, etc. Most modern application servers, including WebLogic [2], WebSphere [18], Oracle Application server [14], and JBoss [9], provide clustering functionality that implements the above tasks. This functionality allows all instances of a given application to be organized into an *application cluster* with coherent state. In this work, we assume that application instances operate in an application cluster with each hosted Web site having its own cluster.

3. VIRTUALIZATION FOR UTILITY COMPUTING

Traditional approaches to utility computing run multiple application servers on the same physical machine to better share resources or run these application servers on different physical machines to provide better isolation amongst these servers. Virtual machines have emerged as an attractive alternative to these approaches because resources within a virtual machine are fault and security isolated from each other, yet the resources of the physical machine can be shared amongst the virtual machines running on it.

The potential issue with using virtualization for application servers is the performance overhead when compared with using physical machines directly. Studies have considered this overhead and reported widely different results. The overhead is dependent on the application and the relative amounts of user- and system-level computing that it does.

For example, according to [16], a highly optimized VMware ESX server, where the VM monitor runs directly on a physical machine rather than on top of the host OS, showed an overhead of 29%. Figueiredo and Dinda reported much lower overhead, measured as a slow-down, for CPU-intensive scientific applications [5]. Their result showed the overhead in the single percentage points.

Because of recent advances in computer architectures, with emergence of multi-core servers and VMs, we measured this overhead in our testbed environment. In our experiment, we used VMware Server 1.0.1 as the virtualization technology and WebSphere V6 application server running the "Hit Count" sample application that comes with WebSphere distribution. Our experiments ran on a desktop with Intel dual-core 2.80GHz Pentium D CPU and 1G memory. To measure the virtualization overhead, we compared the throughput of the application running directly on the host OS with the throughput of the same application running in a VM. We further considered two VM configurations: one VM with two virtual CPUs and 800M memory, and two VMs with one virtual CPU and 400M memory each (in the latter case we measure the aggregate throughput across both VMs). Both host guest OSs were Linux 2.6.15.2054_FC5smp.

We used the httperf [13] workload generator with a rate option, sending a pre-defined number of HTTP requests per second to the server. For the configuration with two VMs, we ran two instances of httperf in parallel. To find the throughput for a given configuration, we gradually increased the request rate and used the results for the run with the maximum rate that had no errors. The results are shown in Table 1. We observe that our setup was able to load

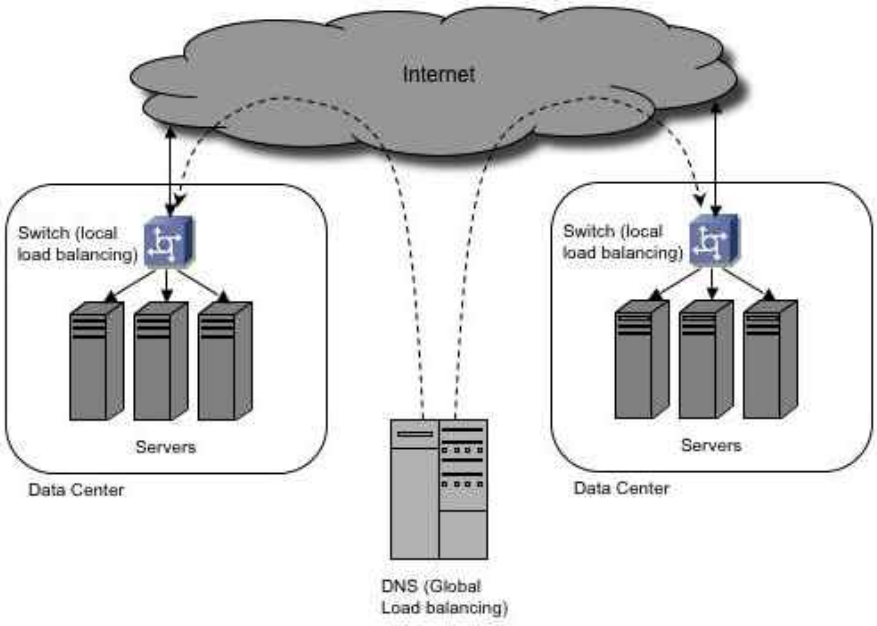

Figure 1: High-level architecture of a global hosting provider platform.

all configurations to 85-95% utilization, The VM configurations incurred an overhead of 35-40% in error-free sustained request throughput when compared to the throughput of the native OS configuration. This result is consistent with the result in [16], which reported a slightly lower overhead using the more efficient ESX technology.

Table 1: Throughput overhead of VMs

Configuration	CPU Utilization	Thruput (req/sec)
Native	90%	1700
1 VM, 2 vCPUs	85%	1050
2 VMs, 1 vCPU/VM	95%	1100

Although we observe that the virtualization overhead for Internet applications can be significant, the growing usage of this technology indicates the acceptance of this overhead as the cost for better utilization of physical resources while preserving application isolation[1]. Our approach suggests a new application of virtualization, which aims at improving the platform agility. Utilizing virtual machines for agility is in contrast to other VM-based work that has focused on fine-grained resource sharing [19, 10, 4].

4. AGILITY IN THE VIRTUALIZED UCIA PLATFORM

The UCIA platform should be able to reallocate resources among Web services quickly. Flash crowds surge fast [11, 3] and some Web servers melt down during an overload. Beside the obvious resource allocation method involving booting a new copy of a VM, we have considered a number of alternatives, all built upon virtualization. In addition, we propose a new approach to resource reassignment—the deployment of *ghost* VMs. Each of these alternatives along with the use of ghost VMs is described in the following.

4.1 Migration of VMs

By migrating a VM from an overloaded host to a relatively idle host, we can achieve better load balancing and utilize our resources more efficiently. There are several ways in which migration can be accomplished.

- Migration of a VM can be achieved by stopping a VM on one host and booting a pre-stored mirror VM on another host. In this case, the destination VM does not reflect the current state of the source VM.
- A VM from one host can be cloned to another host. This method requires that we first suspend the VM on the source host, then copy it across hosts, and finally resume it on the target host. The state of the source VM will be preserved but this method requires moving the VM across machines in the critical path.
- Recently, techniques have been developed for live migration of a VM, where the migration occurs without stopping the source VM, and the updates to the source VM state are forwarded to the target host during migration [4, 20]. This method dramatically reduces the downtime during migration but does not by itself improve the agility of resource reassignment. First, the target host will only be able to start performing its work after a delay that is similar to the cloning technique. Second, the source machine will not be relieved of the overload due to the operation of the source VM until the migration is complete.

[1]Improving VM isolation remains a subject of active research [7, 8], but it is already much higher than in alternative sharing technologies.

4.2 Redeployment of Application Servers On Demand

In this method, instead of migrating VMs across hosts, we pre-deploy VMs across the set of hosts and use them for different applications as dictated by the demand. In cases of overload, we determine which applications are targeted by large number of requests. We then search to find relatively idle hosts running VMs with applications that are not popular. At this point we stop application servers on those VMs with unpopular applications and remove them from their application clusters. Finally, we add new application servers on those VMs and join them to the corresponding application cluster.

4.3 Suspend/Resume VMs

This method is similar to the previous method in that we pre-deploy a number of VMs across all hosts, but each VM now always runs its assigned application. With some number of VMs left alive for serving client requests in the initial stage, all other VMs are suspended. We then suspend or resume various VMs on different hosts depending on the observed demand for their corresponding applications.

4.4 Active/Ghost VMs

All previous approaches share the property that newly active instances of application servers must (re)establish their cluster membership, which can take a long time. As a new approach to improve agility of resource allocation, we attempt to hide the delay in resource reassignment. This approach is illustrated in Figure 2. We pre-deploy VMs, each with its own application(s), on each host. In contrast to previous methods, all VMs are alive using this approach. However, the switch at each data center is configured so that it is not aware of all of these VMs and consequently not all are used to service client requests. We refer to VMs that are known by the switch and can handle client requests as *active* VMs and the others as *ghost* VMs. While hidden from the public Internet, the ghost VMs can still communicate with each other using either the hosts' second network card, available on most modern servers, or through the L2 functionality of the switch. All VMs (both active and ghosts) use this "private" network for managing application clusters. Thus, the application servers on ghost VMs fully participate in their application clusters and maintain both their membership in these clusters and the current application state.

Promoting a VM from the ghost to active state amounts only to the reconfiguration of a network switch, which takes on the order of single seconds as opposed to minutes in existing approaches. This approach forms the basis for agile resource reassignment. In the initial stage, there is an active list of VMs that are enough to serve the client requests. All the other VMs lie in the ghost list. After detecting an overloaded application, additional resources can be allocated to it by identifying a ghost VM on a host with this application and promoting it to the active state. Depending on the overall host utilization, an existing active VM with another application might be demoted to the ghost status.

Ghost VMs only consume CPU for application cluster management[2], and we show that this CPU consumption is negligible. At the same time, to ensure quick state promotion, we retain a small enough number of ghost VMs on each physical machine so that most of their memory footprint does not page out. In other words, while not consuming other resources, ghost VMs do consume memory. We believe this is an economical approach to utility computing when agility is a concern. Indeed, memory constitutes a relatively small portion of the server costs. A quick visit to dell.com shows that a PowerEdge 1950 server costs $4143, while doubling its memory from 4G to 8G costs only $500, or 12% of the cost of another server.

Because each host can only have a small number of ghosts, a larger pool of suspended VMs still must be kept on disk. If no appropriate ghost VM is found for an application in need of extra resources, the system will fall back to resuming a suspended VM from disk. Thus, our approach requires an intelligent algorithm for moving VMs between the levels of the state hierarchy. This algorithm is somewhat similar to distributed cache replacement algorithms, and is part of our ongoing work.

[2]Applications may perform significant background work even in the absence of user requests. We posit that these applications are not typical of the Internet applications, and are not targeted by our ghost approach.

5. PRELIMINARY INVESTIGATION

To better understand how well these alternatives work for resource reallocation, we constructed a test environment and performed preliminary experiments to evaluate each of the methods described in Section 4. Our testbed follows the architecture on Figure 1. It includes two "data centers", one at Case and the other at WPI. The WPI data center uses a Cisco Content Switch 11501, a VMware Server 1.0.1 as virtualization technology, and JBoss as the application server. The Case data center uses a Nortel Alteon 2208 Application Switch, a VMware Server 1.0.1 and 1.0.2 as the virtualization technology and WebSphere 6 Network Deployment as the application server. The experiments below concentrate on the Case data center. For these experiments, the Case site was configured with three identical machines as servers, each having Intel(R) 4-core 2.33 GHz CPU, 4G memory, 146G disk with 15K RPM and running Linux 2.6.17-10-generic SMP as host OS. We used VMware Server 1.0.1 on two of the machines and 1.0.2 on the third machine. Each VM is configured to have 2 virtual CPUs and 1G memory. All reported results are the average of at least three measurements.

Agility has two considerations—how quickly more resources can be assigned to an application and how quickly resources can be de-allocated. In most cases, much of the delay is due to the former, and we concentrate on the delay in assigning a new VM to an application in this section.

5.1 Migration of VMs

If we perform VM migration by stopping a VM on one host and starting its mirror VM on another host, three factors contribute to the total delay before the mirror VM can serve the client requests: time for starting the mirror VM, time for starting a cluster agent on this VM (the clustering mechanism of WebSphere, as well as WebLogic, requires a host to run a clustering agent before any of the application servers can participate in a cluster), and time for starting the application server on the mirror VM.

We found that the average startup time of a VM varies from 46s on an otherwise idle machine that may have some of the VM's memory pages in RAM already, to 55s on an idle machine with VM memory pages flushed from RAM prior

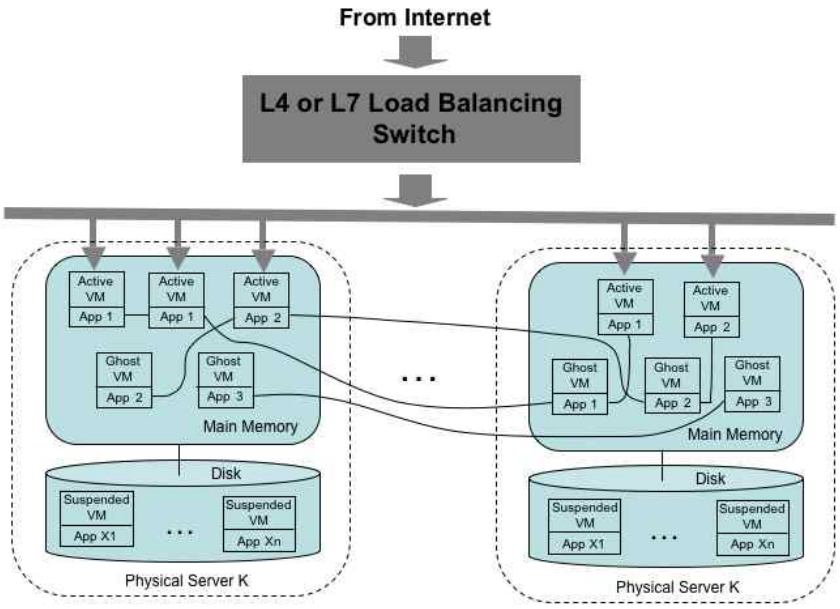

Figure 2: Agile resource management using ghost VMs. Thin lines connecting applications indicate active members of the application clusters.

to the startup (which we achieve by executing a memory-intensive application before the startup), to 64s on a machine with flushed VM memory and a moderately active (using 8% CPU) concurrent VM. We believe the last case represents a typical scenario in utility computing since physical machines are likely to be shared between VMs and the new VM is unlikely to retain its pages in memory from previous execution. We further found that starting the cluster agent can take around 20s. We measured the time to start an application server to be about 97s on average when done for the first time after a VM reboot. Thus, the total time it takes before the mirror VM can serve client requests to be on the order of 180s—not a good result for agility.

If we achieve VM migration by cloning, the VM must be resumed on the target host, which as we will see in Section 5.3 takes somewhat less time. However, before resuming the VM, cloning involves copying the state of the VM and its disk files to another host. For example, to clone a VM with 1G memory and 10G disk, we need to transfer 11G data across the hosts. The bandwidth between the hosts is 100Mbps in our environment. So the transfer time can be 15 minutes, making this method untenable for agility. To make the transfer time acceptable, the platform would have to use a 10Gbps network, which is economically feasible only on a LAN within one data center. Even then, the resume delays, as discussed in Section 5.3, are significant.

5.2 Redeployment of Application Servers On Demand

Unlike the migration method, this method does not include starting the cluster agent because there is only one agent per VM and the VM is not stopped here. The delay of this method mainly consists of three parts: time for stopping an old cluster member (application server) to free up a VM, time for creating a new cluster member (note that we did not need to create a cluster member in Section 5.1 because in that case the same application is associated with a VM and hence the cluster member persists across reboots), and time for starting the new cluster member. Using our experimental setup, we find that time for stopping a cluster member is about 95s on average, the time for creating a new cluster member is about 19s, and the time for starting an application as part of the cluster member is, as given in Section 5.1, about 97s on average. Note that these operations can take significantly less time when done repeatedly, due to caching effects. Because in a real platform these operations occur infrequently, only in the course of resource reassignment, we obtain multiple measurements by executing these operations each time for different clusters. In total, the delay in this method is on the order of 210s on average. Note that we cannot avoid stopping the old cluster member before starting the new one because this would violate the isolation requirement between the two application providers. Thus, again, this method is not good for agility.

5.3 Suspend/Resume VMs

Instead of migrating VMs or dynamically creating new application servers, this method maintains the "frozen" state of VMs and does not need to explicitly rejoin clusters or startup application servers: the application servers resume along with their VMs, and the clusters' health check functionality will detect the "recovered" members. In addition, a suspended VM consumes no CPU or memory resources. Measuring the time to suspend or resume a VM using our experimental setup we find that it takes about 6s to suspend a VM and 5s to resume one. The agility of this method seems good.

However, through further investigation, we find that if the memory of a suspended VM is swapped out, which we force by running a memory-intensive application after the VM is suspended, it can take much longer time to resume it. We measure the average resume delay of 10s (range 5-

21s) on an otherwise idle machine and 14s (range 7-31s) on a machine with one other moderately active competing VM. We attribute the ranges to the dependence on the state of the VM when it is suspended. Furthermore, this represents a lower bound on the actual delay because we only measure the time until the call to a resume API completes. However, this completion time does not mean that the memory required for the application to process a request has been swapped in. In fact, we observed that, the resumed VM had on average only 20-25% of its memory swapped in. Thus, the actual delay can be longer.

Another significant delay component in the VM resume case is due to the need for other cluster members to detect the return of the cluster member running on the resumed VM. Although a resumed VM can serve the client requests after the switch has detected it, the application server can not fulfill the responsibilities as a cluster member, such as client session replication and fail-over. In WebSphere, a new cluster member is admitted by the distributed voting of the existing members, which involves a 30s wait to amortize the voting for potential additional new members.

We conclude that the delay of the suspend/resume resource reassignment can be conservatively put at 44 seconds and in reality is likely to be much higher.

5.4 Active/Ghost VMs

In this method, not only are VM migration and dynamic new application server creation not needed, but there is also no cluster reconfiguration delay, since the ghost VMs are alive all of the time. However, two issues must be addressed here: overhead of the ghost VMs and the swap-in problem.

For the overhead of the ghost VM, we have focused on use of the CPU. According to our experiments, the CPU overhead of a ghost VM, both clustered and stand-alone, is 1.4%. Note that CPU utilization depends on the applications running on the VM. For example, applications that need to run on the background frequently may cost more CPU. In fact, a cache monitoring application, which is a sample application of WebSphere, has a CPU overhead of 3.3%. But even in this case, it is still acceptable.

The swap-in problem would arise if most of the memory of the ghost VM is swapped out. In this case, it can take a long time to swap in enough memory of the VM to work smoothly. Our experiments show that if active VMs run memory-consuming applications, as much as 98.5% of the memory of ghost VMs on the same host can be swapped out. We also estimated the upper bound on the swap-in time as follows. We first exhausted the memory of the host by running a competing memory-intensive application, so that the memory of the ghost VM is swapped out up to about 98%. We then ran a program allocating memory repeatedly in the ghost VM. Our result shows that to swap in all memory (1G) of the VM may take about 652s. For an application requiring half the memory (500M), swapping in still takes about 267s. We consider these results as upper bound for swap-in delay because memory requirements are application specific. Still it is clear that VMs with busy application servers may take a long time to swap in.

To address the swap-in problem we hypothesize that leaving some amount of memory, such as 1G, unused for the host OS, will avoid the problem of ghost VMs being swapped out and taking too long to swap in. For example, our host has 4G memory and we only deploy 3 VMs with 1G of memory each. If the host is devoted entirely to only VMs, the additional 1G can satisfy the memory demands well for the daemons on the host OS. On the other hand, the VM Monitor (VMM) makes sure that the memory usage of each VM is limited to 1G. Thus, regardless of the level of activity of these VMs, there is no need for the memory of any of them to be swapped out. Our experiments verify this expectation. Our experiment lasts about 5 hours and the memory of ghost VMs stays loaded in RAM. In the future, we expect to use VMware ESX, which can guarantee that each VM is provided with a minimum amount of RAM.

Based on these results, we extended our preliminary experiments. In this experiment, there are three identical physical machines. Each physical machine is installed with one VM belonging to the same cluster. VMs on physical machine 1 and 2 are active, while we have a ghost on physical machine 3. We send a large number of requests to the switch, which initially knows about only VMs on physical machine 1 and 2, and just forwards requests to them. When detecting overload on physical machines 1 and 2, we move the VM on physical machine 3 from the ghost list to the active list by configuring the switch. Our measurements show that it takes 7s for the switch to start forward requests to the VM on physical machine 3 for load balancing. This time includes 1s to apply and 6s to save the new configuration. As an alternative approach, we found we can reduce the reconfiguration time to 2s if we make the switch aware of the ghost VM, but limit the number of connections it can handle to one. Activating this ghost VM then simply requires increasing the number of connections the switch is allowed to make with the VM.

6. SUMMARY OF PRELIMINARY INVESTIGATION

Table 2 summarizes the average agility results we obtained for each of the methods in our preliminary investigation. As shown, the first two methods are not agile solutions as they do not allow for quick resource reallocation. The Suspend/Resume might be a better solution, but we currently have only a conservative agility estimate.

Table 2: Summary of agility results for time to reconfigure resource management (sec).

Method	Agility (sec)
Migration of VMs	~180
Redeployment of App Servers	~210
Suspend/Resume VMs	> 44
Active/Ghost VMs	2-7

Table 2 shows the active/ghost method as the most promising approach for attaining agility in our utility computing platform. Note that although our architecture requires extra memory, it is not the same as if we just purchased enough memory for all of the VMs. We only need extra memory for a few hot stand-by VMs (the ghosts), and the set of the ghost VMs changes over time. Thus, while we may have 100 applications at a 1G per application, we would only need to add a few gigabytes of memory, yet be able to largely hide the startup latency through an intelligent resource management algorithm. Analogous to cache management algorithms, this algorithm will use both global and local state information

to "cache" ghost VMs for only those applications with the highest need.

7. RELATED WORK

Previous work has explored using virtual machines for grids [5, 17, 12]. Unlike these studies, which focused on VM monitor implementation and/or fine-grained resource allocation, we investigate the implications of the interaction of application cluster and virtual machine technologies.

The use of virtualization has been investigated in other work as well. Among numerous efforts in this area, this configuration was discussed by Whitaker et al. [19] in their paper about the Denali project which discussed using virtual machines to 'push' untrusted web services into third party hosting infrastructure, demonstrating the security benefits of virtual machines. This configuration was also proposed by Figueiredo et al. [5] in their paper that discusses using virtual machines as an abstraction layer for grid computing, thus isolating resources, providing extra security and allowing legacy support. Another group that proposed this configuration was Clark et al. [4] in their discussion of the benefits of virtual machines in the ability to efficiently migrate them between different physical machines. VM migration is used to alleviate detected hotspots in [20]. Lastly, Gupta et al. [7] discussed this configuration when talking about SEDF-DC, a system that controls total CPU utilization across a number of virtual machines.

Another related strand of work addresses adaptive resource management control in a virtualized environment (see, eg. [15]). Ghost VMs introduce a hierarchy of VM states, which will require new algorithms on how resources are controlled and allocated in this environment.

8. SUMMARY AND FUTURE WORK

In this work we have examined an important problem in virtualized utility computing platforms—the need for quick reaction to changing demands. We refer to the reaction time of a platform as its agility. We are targeting hosting utility provider environments, where the entire platform is under the control of a single administrative domain and application instances often form application-level clusters.

In considering this problem, we have examined both traditional approaches for reallocation of resources in this domain as well as proposed a new method that exploits properties of virtualized utility computing platforms. Specifically, we use ghost VMs, which contain running application servers that are part of a cluster with active VMs, but are not configured to receive client requests from the front-end switch.

Preliminary results from testing this ghost/active approach against the other methods show that the time for a ghost VM to become an active VM within a cluster are on the order of a few seconds. This promising result leads us in a number of directions for future work.

1. A clear direction of work is to extend our evaluation in many ways. We are currently working to extend the scale of our test platform in terms of the number of data centers. We are also working to upgrade to VMware ESX, which runs natively on the hardware. We also need to look at more extensive client testing tools for this environment beyond load generators such as httperf, and at more representative test applications.

2. In addition to understanding the time to rapidly redeploy resources in a utility computing platform, we need to better understand the background resources consumed in providing the capability for rapid redeployment. Initial results indicate that CPU overhead is small for ghost VMs and we believe that we can use VM technology to control memory usage of these VMs. However, we do need a better understanding of these tradeoffs as we make decisions on how many ghost VMs can be deployed.

3. The concept of ghost VMs introduces a hierarchy of states for a VM in an utility computing platform: active, ghost, suspended, and not booted. This hierarchy affords more options for global and local resource management algorithms to use in deciding how to best satisfy the changing demands across a number of applications. We plan to explore these options.

4. We are building a full-function prototype of a distributed hosting utility provider, which will incorporate the mechanisms and algorithms discussed here, and be used to measure platform characteristics, such as agility, in an end-to-end manner.

9. REFERENCES

[1] A. Andrzejak, M. Arlitt, and J. Rolia. Bounding the resource savings of utility computing models. Technical Report HPL-2002-339, HP Labs, December 2002.

[2] BEA WebLogic Server. http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/weblogic/server/.

[3] Xuan Chen and John Heidemann. Experimental evaluation of an adaptive flash crowd protection system. Technical Report ISI-TR-2003-573, USC/ISI, July 2003.

[4] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpack, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *USENIX Symposium on Networked Systems Design and Implementation*, 2005.

[5] Renato J. O. Figueiredo, Peter A. Dinda, and José A. B. Fortes. A case for grid computing on virtual machine. In *ICDCS*, pages 550–559, 2003.

[6] T. Groothuyse, S. Sivasubramanian, and G. Pierre. GlobeTP: Template-based database replication for scalable web applications. In *Proceedings of the International World Wide Web Conference*, pages 301–310, Banff, Alberta Canada, May 2007.

[7] Diwaker Gupta, Ludmila Cherkasova, Rob Gardner, and Amin Vahdat. Enforcing performance isolation across virtual machines in Xen. In *Proceedings of the ACM/IFIP/USENIX 7th International Middleware Conference*, November 2006.

[8] R. Iyer, L. Zhao, F. Guo, S. Makineni, D. Newell Y. Solihin, R. Illikkal, L. Hsu, and S. Reinhardt. QoS policy and architecture for cache/memory in CMP platforms. In *SIGMETRICS*, pages 25–36, 2007.

[9] JBoss application server. http://www.jboss.org/products/jbossas.

[10] Xuxian Jiang and Dongyan Xu. Soda: A service-on-demand architecture for application service

hosting utility platforms. In *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, 2003.

[11] Jaeyeon Jung, Balachander Krishnamurthy, and Michael Rabinovich. Flash crowds and denial of service attacks: characterization and implications for cdns and web sites. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 293–304, 2002.

[12] Ivan Krsul, Arijit Ganguly, Jian Zhang, Jose A. B. Fortes, and Renato J. Figueiredo. Vmplants: Providing and managing virtual machine execution environments for grid computing. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, 2004.

[13] David Mosberger and Tai Jin. httperf – a tool for measuring web server performance. In *Workshop on Internet Server Performance*, Madison, Wisconsin USA, June 1998.

[14] Oracle application server. http://www.oracle.com/appserver/index.html.

[15] Pradeep Padala, Kang Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kenneth Salem. Adaptive control of virtualized resources in utility computing environments. In *Proceedings of the EuroSys 2007 Conference*, Lisbon, Portugal, March 2007.

[16] Performance characteristics of virtualized systems with the vmware esx server and sizing methodology for consolidating servers. IBM White Paper. ibm.com/websphere/developer/zones/hipods, April 2007.

[17] Ananth I. Sundararaj and Peter A. Dinda. Towards virtual networks for virtual machine grid computing. In *Virtual Machine Research and Technology Symposium*, pages 177–190, 2004.

[18] Websphere application server. http://www-306.ibm.com/software/webservers/appserv/was/.

[19] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. Scale and performance in the denali isolation kernel. In *OSDI'02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 195–209, 2002.

[20] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Black-box and gray-box strategies for virtual machine migration. In *USENIX Symposium on Networked Systems Design and Implementation*, Cambridge, MA, USA, April 2007.