Experimental Study of Virtual Machine Migration in Support of Reservation of Cluster Resources

Ming Zhao Renato J. Figueiredo
Advanced Computing and Information Systems Laboratory (ACIS)
Electrical and Computer Engineering, University of Florida
{ming, renato}@acis.ufl.edu

ABSTRACT

Virtual Machines are becoming increasingly valuable to resource consolidation and management, providing efficient and secure resource containers, along with desired application execution environments. This paper focuses on the VM-based resource reservation problem, that is, the reservations of CPU, memory and network resources for individual VM instances, as well as for VM clusters. In particular, it considers the scenario where one or several physical servers need to be vacated to start a cluster of VMs for dedicated execution of parallel jobs. VMs provide a primitive for transparently vacating workloads through migration; however, the process of migrating several VMs can be time-consuming and needs to be estimated. To achieve this goal, this paper seeks to provide a model that can characterize the VM migration process and predict its performance, based on a comprehensive experimental analysis. The results show that, given a certain VM's migration time, it is feasible to predict the time for a VM with other configurations, as well as the time for migrating a number of VMs. The paper also shows that migration of VMs in parallel results in shorter aggregate migration times, but with higher per-VM migration latencies. Experimental results also quantify the benefits of buffering the state of migrated VMs in main memory without committing to hard disks.

1. INTRODUCTION

With the rapid growth of computational power on compute servers, and the fast maturing of x86 virtualization technologies, Virtual Machines (VM) have become increasingly important to supporting efficient and flexible resource provisioning. Modern virtual machine technologies (e.g. [14][15][2]) allow a single physical server to be carved into multiple virtual resource containers, each delivering a powerful, secure, and isolated execution environment for applications. In addition to providing access to resources, such environments can be customized to encapsulate the entire software and hardware platform needed by the applications and support their seamless deployments.

The management of these VM-based resource containers, e.g. lifecycle management and resource allocation, can be conducted through the interfaces provided by the virtualization platforms. This allows the VMs to be scheduled as processes in typical operating systems, and QoS-aware schedulers, similar to those available in operating systems, can be employed to allow the VMs to time- and space-share resources, and in the meantime provide QoS guarantees for the applications running inside of the VMs.

This paper focuses on the VM-based resource reservation, that is, the reservations of CPU, memory and network resources for individual VM instances, as well as for VM clusters. The fundamental goal is to enable an application to request the creation of virtual machines and clusters based on high-level specifications of both the VMs' environments and its desired QoS. This scenario has been motivated by the need encountered by scientists in the brain-machine interface domain [5]. Their applications are time-sensitive during their execution, but need only be active during the execution of an experiment (e.g. a trial with an animal, or a training/testing run).

Allocating dedicated resources in this scenario can lead to resource inefficiencies; VMs here allow time-sharing of resources at a coarse granularity and can lead to better resource utilization. Hence, it is desirable to reserve cluster resources for creating a set of VMs to run these tasks. To implement such policy, all hosted VMs from the cluster to be reserved need to be vacating - through suspension, or if other resources are available, through migration. This preparation should be done in time to meet the reserved schedule, but cannot be too early and waste the resources that are useful to serve other tasks.

In order to make efficient resource reservation, this VM-based approach need take into account the overhead, which requires an accurate cost estimation for both the migration of the existing VMs, and the instantiation and configuration of the scheduled new VMs. In addition, the overhead on the applications running inside of the migrated VMs should also be considered. Previous work has shown that the VM creation's overhead can be small and accountable [8][13], which can be leveraged in this cost estimation. However, there is no extensive study on the cost associated with migration of multiple VMs with the goal of vacating a resource. Addressing to this problem, this paper seeks to provide a model that can characterize the VM migration process and predict its performance, based on a comprehensive experimental analysis.

A series of experiments were conducted to measure and model the different phases for migrating a number of running VMs from one physical host to another. The results show that, given a certain VM's migration time, it is feasible to predict the time for a VM with other configurations, as well as the time for migrating a number of VMs. The impact of a VM's migration on its application is also studied in this paper, which shows that it takes longer for the application to recover than the actual VM migration time. Finally, different migration strategies are compared and the results show that parallel migration is faster for migrating

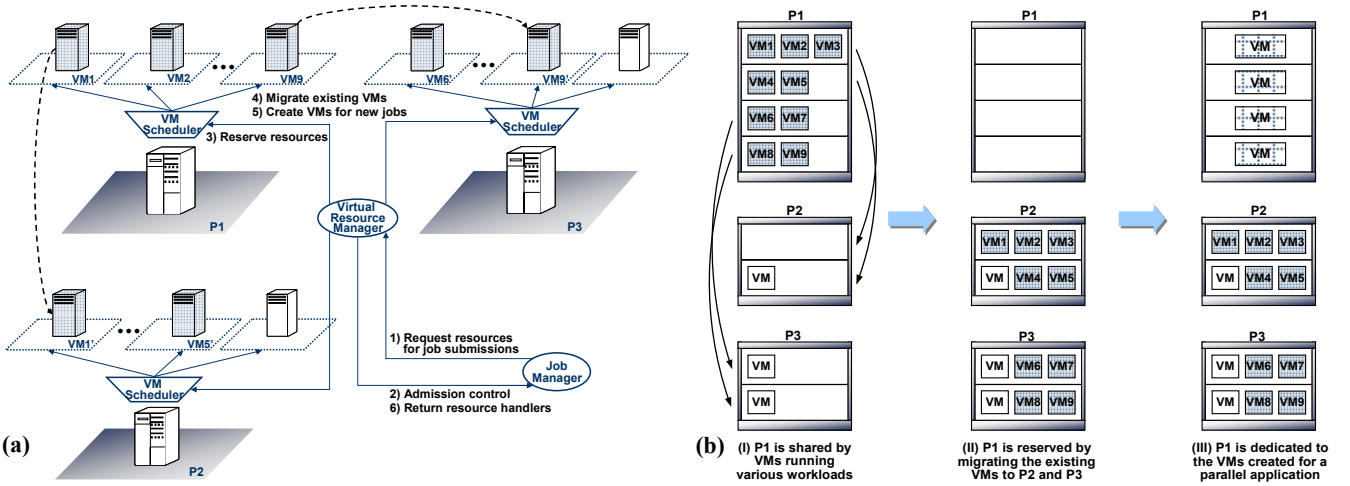

Figure 1: VM-based resource reservation. It consists of two levels of resource management which cooperate to serve resource reservation. As shown in (a), the virtual resource manager decides to vacate physical cluster P1 in order to start a set of new VMs to satisfy the resource request from the job manager. It coordinates with the VM schedulers to migrate the VMs (V1-V9) from physical cluster P1 to P2 and P3. After the new VMs are started, their resource handles are returned to the job manager for job submissions. The change of resource allocation on the clusters during this reservation process is also illustrated in (b).

multiple VMs, but it has more interference to the performance of the applications on the migrated VMs.

The rest of this paper is organized as follows: Section 2 describes the background of VM-based resource reservation; Section 3 presents the experimental analysis of VM migration; Section 4 discusses the related work, and Section 5 concludes the paper.

2. VM-BASED RESOURCE RESERVATION

Figure 1 illustrates the architecture for VM-based resource reservation. It consists of two levels of resource management, which cooperate to serve resource requests, received from, e.g. a job manager that schedules job submissions.

2.1 Virtual Resource Manager

The virtual resource manager provides a centralized management for the virtualized resources that are distributed across the physical hosts. It exposes an abstract interface to the resource clients and serves their resource requests. The clients do not need to know the details of the resource provisioning, and in fact, they can be even unaware of that the resources are virtualized. They only need to specify the types and quantities of resources that are necessary for the scheduled jobs, e.g. the amount of CPU cycles, memory space, storage capability and network bandwidth. To make an advance resource reservation, a time schedule can also be associated with the desired resource usage in the request. (Because a VM-based resource container incurs additional overhead from the virtualization, a resource controller that can correctly estimate the resource usage for a given job's VM is necessary for making the resource requests. However, this is not the focus of this paper, and previous work [17] can be leveraged to provide this functionality.)

Upon receiving such a resource request (1), the virtual resource manager first performs the admission control based on the current resource allocation and reservation state of the physical hosts (2). If there are resources available for the requested quantity and schedule, it then proceeds and interacts with the VM schedulers to make the resource reservation on the selected physical hosts (3). The VM schedulers then migrate the existing VMs as needed (4) and create new VMs for the scheduled jobs accordingly (5). After the reservation is completed, one or several resource handlers (e.g. IPs and accounts of the allocated VMs) are returned to the client (6), and they are valid for job submissions when the scheduled time arrives. On the other hand, a request is rejected if the virtual resource manager determines that the available resources are not sufficient to satisfy the request.

The virtual resource manger also supports the request of preparing a desired software environment on the reserved resources. Such an environment includes the operating system, applications, and libraries that are necessary for the job executions. For instance, a dedicated VMPlant service [8] can be leveraged by the virtual resource manager to provide this support. This service enables the automated VM creation and customization, using a graph-based model to define VM configuration actions, and providing instant VM creations based on cloning from a set of typical VM images.

2.2 Virtual Machine Scheduler

A VM scheduler is on every physical server to manage the VMs that are hosted on it. It runs side by side with the VM monitor, and leverages the interface provided by the VM software to control the configurations and lifecycles of the VMs. Such an interface can be the scripting API provided for VMware Server, the web service interface for VMware Infrastructure, and the command-line interface for managing Xen.

The VM schedulers provide a unified interface for resource reservation, which allows the virtual resource manager to make resource reservations without knowing their underlying control mechanisms that can be very different and specific to the VM software deployed on their physical hosts. The virtual resource manager only needs to specify the quantity and schedule of the necessary resources, and the VM schedulers are responsible to

carry out the resource reservation and VM creations using the mechanisms provided by the VM software.

2.3 VM Migration Based Resource Reservation

VM-based resource reservation needs to take into account the overhead associated with this approach, and an accurate cost estimation is important for the virtual resource manager to provide correct admission control and make efficient resource reservation. Specifically, the requested resources must be prepared in time to satisfy the requested schedule, but they should not be allocated too early and waste the resources that can potentially serve other tasks. The costs from VM-based resource reservation include both the overhead for migrating existing VMs and making the resources available for the new jobs, and the time needed to create and configure the desired environment with VMs. Previous work has shown that the later can be small and accountable [8][13], so this paper focuses on modeling the cost associated with the VM migrations.

In particular, we consider the problem of allowing a cluster resource to be reserved for a parallel application with real-time constraints. The motivation is drawn from brain-machine interface (BMI) experiments where a cluster is used to execute several computational models in parallel during a closed-loop experiment which involves data acquisition (from sensors in an implanted animal), processing, visualization and robot actuation [5]. The goal is to support parallel processing using dedicated resources when such an experiment takes place, while also allowing a cluster resource to be utilized by other workloads when such experiments are not taking place.

To support the above scenario, the virtual resource manager needs to vacate a cluster for the parallel application, and move all the existing VMs, which are running various other workloads, to other hosts (Figure 1(b)). It thus needs to make efficient reservations for resources on both the hosts being dedicated for the task, and the hosts where the VMs are migrated to. The migrations of these VMs can take a considerable amount of time, and may cause a certain amount of performance degradation on the jobs that are running on these migrated VMs. Hence, the virtual resource manager must consider these factors when it makes the reservation decision. In order to achieve this, a clear understanding of the VM migration process is necessary, and a model is also desirable for estimating the migration cost based on the configurations and running states of the migrated VMs. In the following section, an extensive experimental analysis is conducted towards these goals.

3. EXPERIMENTAL ANALYSIS

3.1 Setup

To help the decision of VM-based resource reservation, a series of experiments were conducted to model and analyze the process of migrating a number of running VMs from one host to another. The studies reported in this paper focus on the VMware Server (1.0.3) based VM monitor. The VMs are hosted on a cluster of physical servers. Each physical node has two dual-core 2.33GHz Xeon processors and 4GB of memory, runs Fedora Core 6 with kernel 2.6.22, and is connected with Gigabit Ethernet. Due to the limitation of the physical nodes, the VM memory size considered in these experiments is up to 1GB. However, the findings from the experiments are also applicable if more physical resources are available for VMs with larger memory sizes.

The VMs are installed with Ubuntu 7 with kernel 2.6.20. The VMs' virtual disks share the same read-only image, which is stored on a storage server and accessed through NFS (version 3 [3]). Changes to the virtual disks from the VM executions are stored in the form of redo log files. The running VMs' memory states are also mapped to files, and when they are suspended, these files capture their memory snapshots. In the absence of an efficient shared storage system in our setup, for performance reasons these disk redo logs and memory state files are stored on the local file system (EXT3 in the ordered mode) of the hosts.

The VM migration process considered in this paper entails of three phases, "suspend", "copy" and "resume". In the suspend phase, the VM is suspended on the origin host, and its memory is captured to the memory state file. In the copy phase, the VM's configuration, memory state and disk redo files are transferred to the destination host through FTP. In the resume phase, the VM restores its memory state from the snapshot and then resumes its execution. The default background memory restoration used by VMware is disabled so that an exact measurement of the resume phase can be obtained.

This migration strategy is not based on VMotion [18] or other migration mechanisms provided by VMware. It is analyzed since the primary goal is to vacate multiple VMs from a resource in a timely fashion rather than minimize the downtime per VM.

All the experiments were repeated for more than 50 runs, and their results are reported in the following subsections with both average values and standard deviations. Because the system time inside of VMs can be imprecise, the system time from a separate physical server was used for timekeeping during the experiments.

3.2 Migrating a Single VM

The first group of experiments studies the three migration phases for a single VM, and analyzes its migration time with different VM configurations.

3.2.1 Experiments with Different Memory Sizes

Since a VM's memory state file is often the major part of the data that need be transferred during the migration process, this experiment considers VMs with different memory sizes to investigate the impact of size on migration times. The experimental results (Figure 2) show that the time needed for the suspend and resume phases are relatively stable, and only increases slightly as the memory size increases, because more memory pages need be processed during these two phases. On the other hand, the copy time quickly grows and dominates the migration time for larger memory sizes.

In order to find out the relationship between the time needed for the copy phase and the size of the VM memory, regression methods are used to model it. Based on the data from this experiment, a polynomial function can best characterize this relationship, as illustrated by the diamond-shaped points and the solid line in Figure 3. The reason for a nonlinear model is that when the memory size is relatively small, the speed of the copy phase is limited by the network bandwidth (Gigabit/s); however,

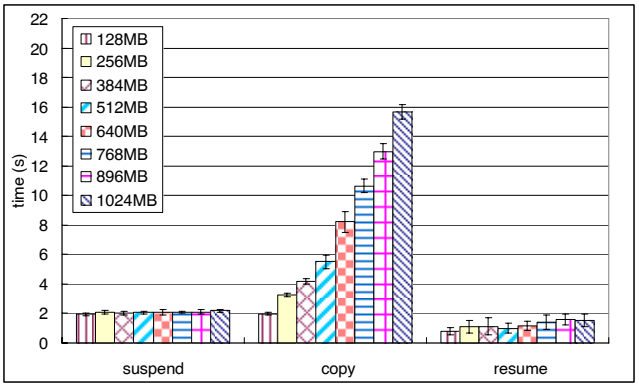

Figure 2: The time needed for the three phases of migrating a single VM with different memory sizes. A local disk on the destination host was used to store the migrated VM states.

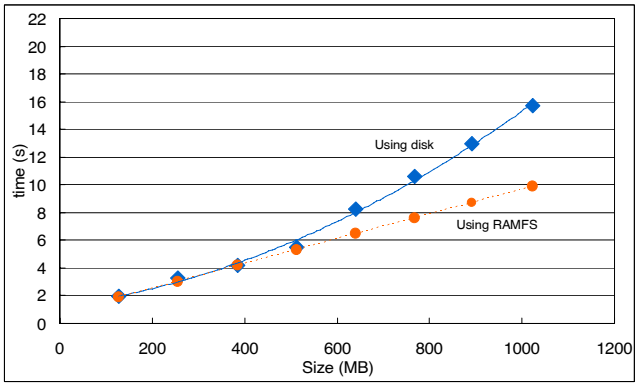

Figure 3: Using regression methods to model the relationship between the time needed for the copy phase and the size of the VM memory.

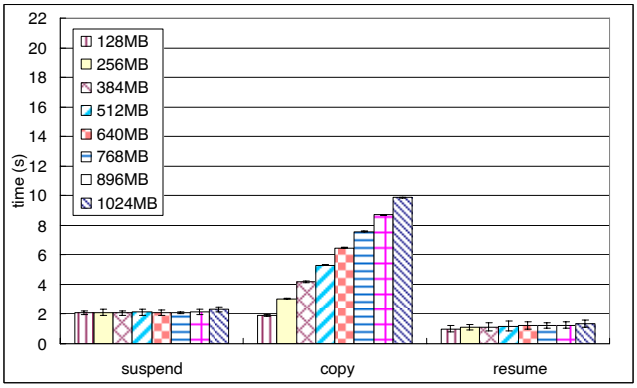

Figure 4: The time needed for the three phases of migrating a single VM with different memory sizes. A RAMFS on the destination host was used to store the migrated VM states.

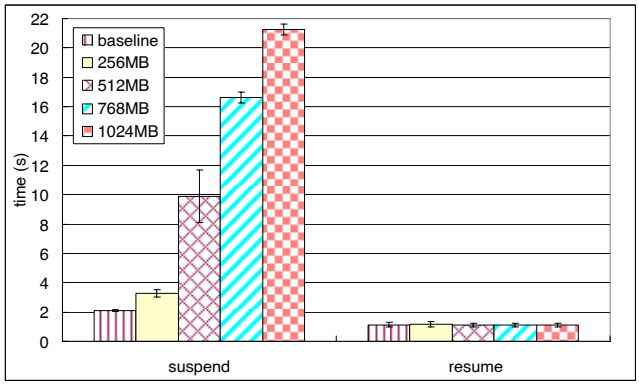

Figure 5: The time needed for the suspend and resume phases of migrating a VM with different amounts of continuously modified memory.

for greater memory sizes, a large amount of dirty pages are buffered in memory during the copy phase, and the kernel forces to flush these data in foreground, which then throttles the copy phase by the throughput of the disk (around 50MB/s). Because the kernel I/O scheduling policy decides when and how to flush the data, this model is dependent on the parameters used by the kernel.

To isolate the impact of the disk I/O on the migration process, we have also conducted an experiment using a RAM-based file system (RAMFS) on the destination host to store the state of the migrated VM. (RAMFS can also be set up on the source host to reduce disk I/Os in the suspend phase.) The results from this setup also represent the cases where other mechanisms, e.g. direct memory-to-memory copy, are available to avoid this problem. Since the VM's memory state is not backed by stable storage, a viable recovery scheme is necessary for the VM in case of a crash. Figure 4 shows the time needed for the migration phases when RAMFS is used, and the dashed line in Figure 3 models the copy phase (round-shaped points) using regression methods. It is evident that a linear function can very well characterize this relationship between the copy time and the VM memory size.

The above two models demonstrate that, given a VM's migration time for a particular memory size, it is feasible to predict the time for migrating a VM with other memory sizes. In addition, this analysis is also applicable if the size of other VM states, e.g. the disk redo files, need be considered.

3.2.2 Experiments with Memory-intensive Workloads

The above results also show that the time needed for both suspend and resume phases are small. The resume phase is typically very fast since after the copy phase is done, the VM's memory state is already buffered in memory (assuming the destination host has enough memory for the migrated VM), and thus the resuming does not require additional disk reads. The suspend time is also usually short, because the running VM's memory is frequently synchronized with its memory state file, and the suspend phase does not involve many disk writes either.

If a considerable amount of memory pages need be synchronized when the VM is suspended, this phase will take longer. To study the impact of this factor, a program which continuously touches a given amount of memory was intentionally started in a VM that was under migration (with a memory size of 1.1GB). Its resume and suspend phases, with different amounts of modified memory, is compared to the baseline performance when this program is not used (Figure 5). (The time for the copy phase is not affected by the workloads of the migrated VMs, since FTP need transfer the same amount data across network no mater whether the memory state files are sparse or not.) The results show that the resume phase is indeed fast and stable, but the suspending time increases nearly proportionally with respect to the size of the modified memory. Note that typical applications do not possess such a bad

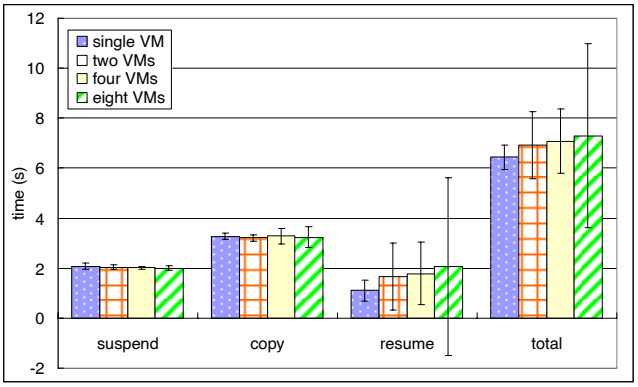

Figure 6: The per-VM migration time when different numbers of VMs, each with a 256MB memory, were migrated in sequence. A local disk on the destination host is used to store the migrated VM states.

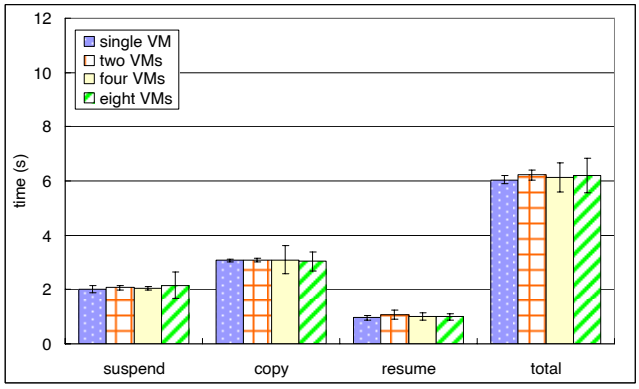

Figure 7: The per-VM migration time when different numbers of VMs, each with a 256MB memory, were migrated in sequence. A RAMFS on the destination host was used to store the migrated VM states.

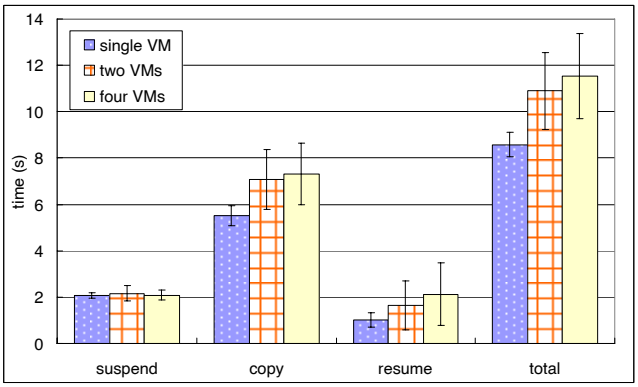

Figure 8: The per-VM migration time when different numbers of VMs, each with a 512MB memory, were migrated in sequence. A local disk on the destination host was used to store the migrated VM states.

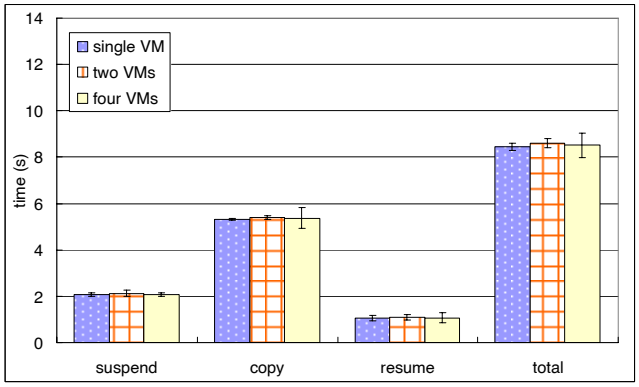

Figure 9: The per-VM migration time when different numbers of VMs, each with a 512MB memory, were migrated in sequence. A RAMFS on the destination host was used to store the migrated VM states.

behavior, and the suspend phase is generally fast as also confirmed by the following experiments. The results from using this "rogue" program give an upper bound on the resuming time based on this worst-case scenario.

3.3 Migrating a Sequence of VMs

The second group of experiments investigates the process of migrating a sequence of VMs, and study whether it is feasible to predict the total migration time based on the time for a single VM.

3.3.1 Experiments with Different Number of VMs

In this experiment, different numbers of VMs are migrated consecutively, each with a memory size of 256MB. Figure 6 plots the per-VM's migration time, when the local disk on the destination host is used to store the copied VM states. The results show that the suspend and copy phase are not affected by the number of VMs, but the resume phase becomes slower as more VMs are migrated, and its variance also increases significantly. This is also because of the aforementioned flushing of dirty data from copying a VM's state files. It not only throttles this VM's migration, but also interferes with the following VMs' migrations because of the uncompleted writes. This situation aggravates as more VMs are migrated in sequence. In the worst case, the entire CPU is in the I/O wait state, and this write "hog" blocks all the following migration processes for a considerable period of time.

Figure 7 shows the results from using the RAMFS on the destination host to store the copied VM states, which prove again that such a setup can effectively solve the above problem, and the migration time becomes very consistent regardless of the number of migrated VMs. Figure 8 and Figure 9 show the time for migrating a sequence of larger VMs, each with 512MB memory, using the local disk and the RAMFS, respectively. The results confirm that the former setup causes increase in the resume time and its variance, while the later one helps to make the migration process stable and predictable. Therefore, this RAMFS setup was used for all the following experiments.

Memory space on the physical host may be a concern for using RAMFS to store the VM states. However, a RAMFS' size is dynamically adjusted, growing or shrinking as needed by the data, and it does not necessarily consume extra memory, since the memory pages used by a VM's state files on the RAMFS can be shared with the VM's memory. Moreover, after the migration is completed, the VM's states can still be backed up on local disks when a snapshot need be taken on persistent storage.

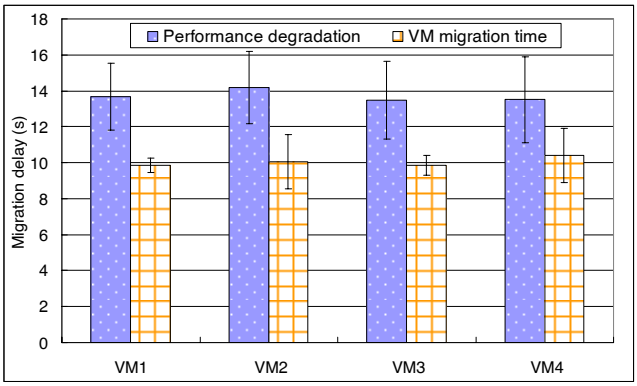

Figure 10: The migration time and performance degradation when four VMs were migrated in sequence, each with 512MB memory and a CPU-intensive benchmark running inside.

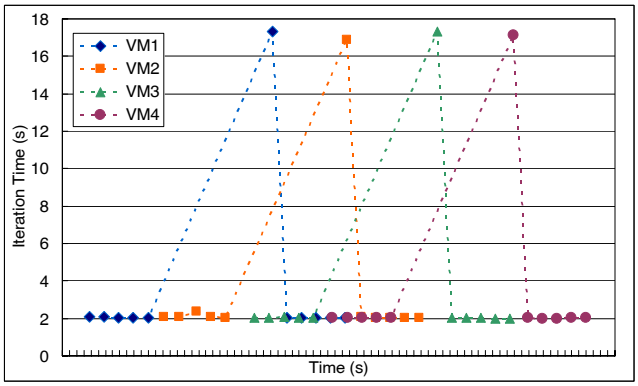

Figure 11: The performance of a CPU-intensive benchmark running inside of four VMs that were under migration.

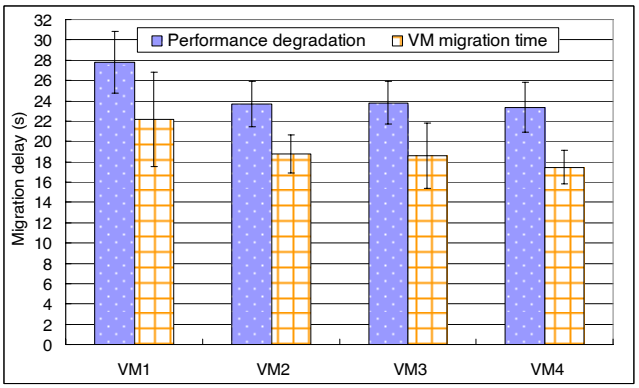

Figure 12: The migration time and performance degradation when four VMs were migrated in sequence, each with 512MB memory and a memory-intensive benchmark running inside.

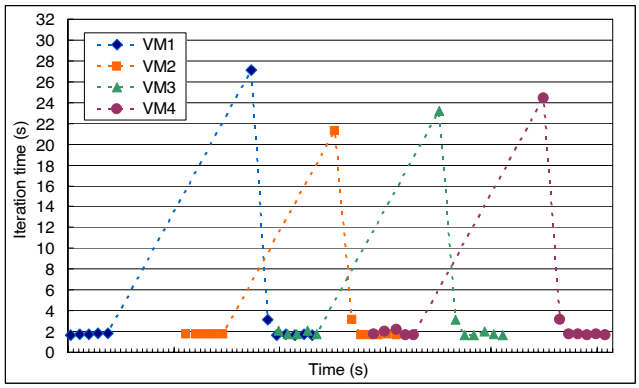

Figure 13: The performance of a memory-intensive benchmark running inside of four VMs that were under migration.

3.3.2 Experiments with Different Workloads

To further study the interference between the workloads running inside of the VMs and the migration process, several different types of workloads were used to load four VMs, each with 512MB of memory, and their sequential migrations are analyzed in this subsection. We have considered two representative cases of workloads in this study: CPU-intensive and memory-intensive. A comprehensive analysis using a larger set of benchmarks is subject of future work.

The first one is a CPU-intensive workload, which is adapted from the Freebench's Distray benchmark [19]. It runs iteratively, where each iteration takes exactly 2 seconds and consumes 100% of CPU when executed on the VMs. This benchmark was started on each of the four VMs, which thus fully utilized all the CPUs available on their physical host. Their migration time as well the benchmark's performance degradation is shown in Figure 10. The performance degradation is defined as the runtime of the iteration being affected by the migration, minus the regular iteration time. The results show that the performance degradation time is longer than the VM migration time by about 2 seconds, which means that it takes additional time for the benchmark to recover to its full performance after the its VM is migrated. Figure 11 illustrates one sample of the performance data collected from the benchmark during the migration process.

The second workload uses a memory-intensive program that is similar to the one used in Section 3.2.2. It runs iteratively, where during each iteration it modifies almost the entire VM's memory once, and then sleeps 1 second. The average iteration time is 1.75 second, with a standard deviation of 0.06 second. With this program running inside the VMs, their migration time and the program's performance degradation are plotted in Figure 12. Because two iterations of the program were affected by the migration, the performance degradation is the sum of these two iterations' time, minus twice of the regular iteration time.

The results show that the performance degradation experienced by the program is longer than the VM migration time by around 5 seconds. The migration has a greater impact on this program than the previous CPU-intensive workload, which infers that it is a more memory-intensive process. Figure 13 illustrates one sample of the performance data reported by the program during the migration process. It is noticeable that the first VM's migration takes more time than the other ones. This is because suspending the VMs for this memory-intensive program involves a considerable amount of disk writes, and the first VM's migration has an additional start-up overhead from initiating the I/Os.

The last workload considers typical web applications by using Apache (version 2.24) based web servers on the four VMs to serve HTTP requests. The HTTP clients were based on httperf [10], which issued requests with a constant rate (100 connections

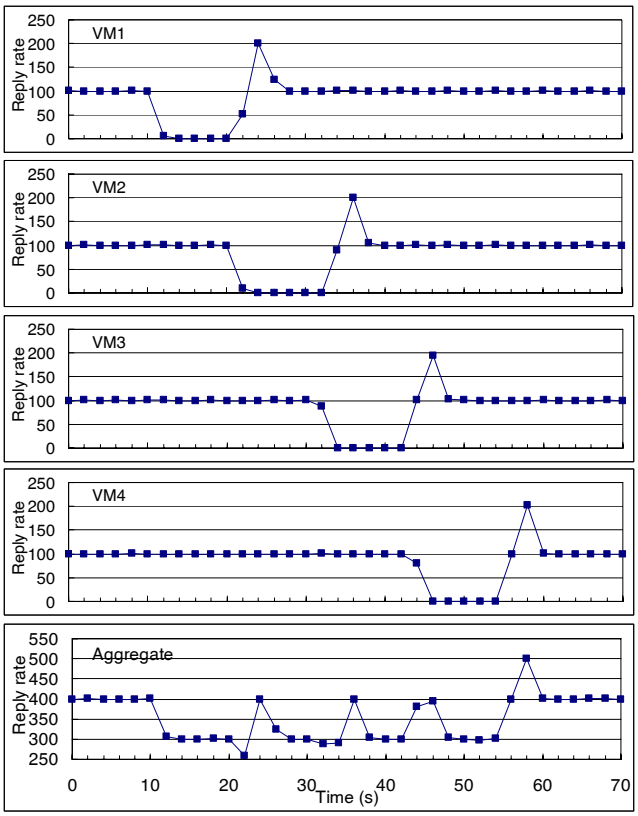

Figure 14: The throughput of web servers running on four VMs that were under migration.

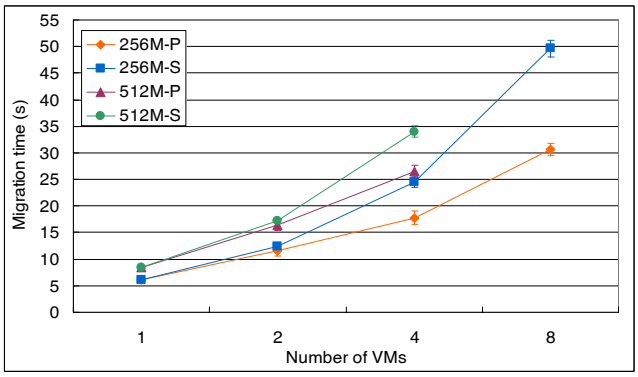

Figure 15: The comparison of total migration time for four VMs between parallel and sequential migration.

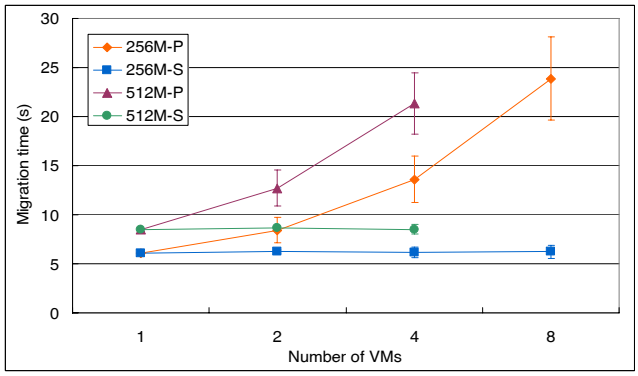

Figure 16: The comparison of per-VM migration time for four VMs between parallel and sequential migration.

per second), and were run separately on another four VMs, hosted on a different physical server. The average migration time for these VMs is around 9 seconds, and a sample of the web servers' throughputs, as well as the aggregate throughput, is plotted in Figure 14. The results show that the performance impact of the migration also stays a few seconds longer than the actual migration time.

3.4 Migrating Multiple VMs in Parallel

The last group of experiments considers migrating multiple VMs in parallel, in contrast to sequentially, and studies its benefits and costs. In these experiments, different numbers of VMs were migrated in parallel, each with the same memory size of 256MB or 512MB. Figure 15 shows the time for the entire process of migrating the considered VMs, compared to the time needed when they were migrated in sequential. The results show that parallel migration is faster, and the advantage becomes larger when more VMs are migrated together. For VMs with 256MB of memory, the speed up is 1.4 times for 4 VMs, and 1.6 times for 8 VMs. For VMs with 512MB memory size, the speedup is 1.3 times for 4 VMs, which is less than that of the smaller VMs. This is because the advantage from parallel migration is mostly from overlapping the suspend and resume phases of multiple VMs, since the copy phase is bounded by the available network bandwidth. For larger VMs, their migrations are dominated by the copy phase and thus cannot gain much from the parallel migration.

Figure 16 compares parallel migration to sequential migration from another perspective by looking at the per-VM migration time. Contrary to the above results, the parallel migration has a much worse per-VM migration time than the sequential one. For VMs with 256MB memory, the slowdown is 2.2 times for 4 VMs, and 3.8 times for 8 VMs. For VMs with 512MB memory size, the slowdown is 1.5 times for 2VMs and 2.5 times for 4 VMs. This means that parallel migration incurs more overhead for the applications running inside of the VMs. Therefore, there is a tradeoff between increasing the speed of migrating multiple VMs and reducing the impact on the performance of the VMs. These different migration strategies can be selected based on the optimization needs.

4. RELATED WORK

Substantial research has been done on using VMs for resource consolidation and management for various types of systems. The In-VIGO project [1] proposes to build virtual grid systems using VMs to share resources and provide customized execution environments, and leveraging the VMPlant service [8] to automate the VM creation and configuration. VMs are used in [6] to provide virtual workspaces with desired software environment and resource allocation, and [13] has studied the overhead from managing these VMs. The Virtuoso project also considers VMs for distributed computing, and is able to co-schedule batch and interactive jobs' VMs to satisfy the constraints on both responsiveness and compute rates [9]. The Shirako system [7]

uses VMs to provide on-demand leasing of network shared resources. In [17], virtualized data centers are realized using VM-based resource containers, and controllers are developed to estimate a VM's resource usage based on its workload demand.

In such VM-based systems, the migration of VMs is often considered as an important means of reallocating resources and improving performance. Particularly, in VIOLIN, a system built on VMs connected with virtual networks, migration is used to relocate the VMs for performance optimizations [11]. In [16], different strategies for VM migration are studied to eliminate performance hotspots from data centers. This paper considers resource management and VM migrations in the context of VM-based resource reservation, while its results can also be useful to estimate the migration cost for other management tasks on VM-based systems.

Because of the importance of VM migrations, the optimization of this process has also been studied in the related research. In [12], several techniques are presented to improve the migration of VMs, including using memory ballooning to make a VM's memory state more compressible, demand paging of VM disks, and content-based block sharing across disk states. Live migrations are also introduced for different VM technologies to provide zero downtime migration of VMs [4][18]. However, these techniques are not widely available, and the suspend-copy-resume scheme considered in the paper is still the common way of migrating VMs. In addition, live migrations often take longer time to finish and are not suited when timely migrations are needed. On the other hand, the methodology used in this paper can also be applied to study the migration when the above techniques can be leveraged.

5. CONCLUSIONS AND FUTURE WORK

VM migration is key to realizing VM-based resource reservation, and understanding its overhead is important to make efficient reservations. This paper seeks to model the migration process based on an extensive experimental study, and several interesting findings are revealed based on the results: An accurate estimation of the migration time for a number of VMs is possible given the measurement of a single VM's migration time; The performance degradation period caused by a VM's migration is relatively longer than the migration time; Different migration strategies can be selected for different optimizations, where parallel migration can deliver better speed when migrating multiple VMs, and sequential migration can reduce the performance overhead for the applications that are running on the migrated VMs.

The ongoing investigation is focused on generalizing this paper's results and evaluating the migration cost model across different hardware platforms, and different virtualization technologies. In the future work, the cost of another important type of migration mechanism, live migration, will also be modeled to help the resource reservation when it is available for use.

ACKNOWLEDGEMENT

Effort sponsored by NSF grant CNS-0540304. The authors are thankful to the anonymous reviewers for their useful comments. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

[1] S. Adabala, et al., "From Virtualized Resources to Virtual Computing Grids: The In-VIGO System", In Future Generation Computer Systems, Vol. 21, No. 6, June, 2005.

[2] P. Barham, et al., "Xen and the Art of Virtualization", ACM Symposium on Operating Systems Principles, October 2003.

[3] B. Callaghan, B. Pawlowski, P. Staubach, "NFS Version 3 Protocol Specification", RFC 1813, June 1995.

[4] C. Clark, et al., "Live migration of Virtual Machines", In USENIX NSDI, 2005.

[5] J. DiGiovanna, et al., "Towards Real-Time Distributed Signal Modeling for Brain Machine Interfaces", International Conference on Computational Science, 2007.

[6] K. Keahey, I. Foster, T. Freeman, X. Zhang, "Virtual Workspaces: Achieving Quality of Service and Quality of Life in the Grid", Scientific Progamming Journal, 2005.

[7] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, "Sharing Networked Resources with Brokered Leases", USENIX Technical Conference, 2006.

[8] I. Krsul, et al., "VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing", Supercomputing, 2004.

[9] B. Lin, P. Dinda, "VSched: Mixing Batch and Interactive Virtual Machines Using Periodic Real-time Scheduling", Supercomputing, 2005.

[10] D. Mosberger, T. Jin, "httperf: A Tool for Measuring Web Server Performance", First Workshop on Internet Server Performance, 1998.

[11] P. Ruth, J. Rhee, D. Xu, R. Kennell, S. Goasguen, "Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure", ICAC'06.

[12] C. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. Lam, M. Rosenblum, "Optimizing the Migration of Virtual Computers", Proceedings of the 5th Symposium on Operating Systems Design and Implementation, 2002.

[13] B. Sotomayor, K. Keahey, I. Foster, "Overhead Matters: A Model for Virtual Resource Management", VTDC, 2006.

[14] J. Sugerman, G. Venkitachalan, B-H. Lim, "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor", USENIX Annual Technical Conference, 2001.

[15] C. Waldspurger, "Memory resource management in VMware ESX server", SIGOPS Operating Systems Review, 2002.

[16] T. Wood, P. Shenoy, A. Venkataramani, M. Yousif, "Black-box and Gray-box Strategies for Virtual Machine Migration", NSDI, 2007.

[17] J. Xu, M. Zhao, J. Fortes, R. Carpenter, M. Yousif, "On the Use of Fuzzy Modeling in Virtualized Data Center Management", ICAC, 2007.

[18] M. Nelson, B.-H. Lim, G. Hutchins, "Fast Transparent Migration for Virtual Machines", USENIX Annual Technical Conference, 2005.

[19] P. Rundberg, F. Warg, "The FreeBench v1.0 Benchmark Suite", URL: http://www.freebench.org