Performance Modeling of Virtual Machine Live Migration

Yangyang Wu Ming Zhao
School of Computing and Information Sciences
Florida International University
Miami, FL, USA
{ywu009, mzhao}@fiu.edu

***Abstract*—System virtualization is becoming pervasive and it is enabling important new computing diagrams such as cloud computing. Live virtual machine (VM) migration is a unique capability of system virtualization which allows applications to be transparently moved across physical machines with a consistent state captured by their VMs. Although live VM migration is generally fast, it is a resource-intensive operation and can impact the application performance and resource usage of the migrating VM as well as other concurrent VMs. However, existing studies on live migration performance are often based on the assumption that there are sufficient resources on the source and destination hosts, which is often not the case for highly consolidated systems. As the scale of virtualized systems such as clouds continue to grow, the use of live migration becomes increasingly more important for managing performance and reliability in such systems. Therefore, it is key to understand the performance of live VM migration under different levels of resource availability. This paper addresses this need by creating performance models for live migration which can be used to predict a VM's migration time given its application's behavior and the resources available to the migration. A series of experiments were conducted on Xen to profile the time for migrating a DomU VM running different resource-intensive applications while Dom0 is allocated different CPU shares for processing the migration. Regression methods are then used to create the performance model based on the profiling data. The results show that the VM's migration time is indeed substantially impacted by Dom0's CPU allocation whereas the performance model can accurately capture this relationship with the coefficient of determination generally higher than 90%.**

I. INTRODUCTION

System virtual machines (VMs [1][2][3]) are widely used in modern computer systems, from individual personal computers to large enterprise data centers. System virtualization provides a powerful abstraction for both application and resource provisioning, upon which shared physical resources can be flexibly allocated to VM-hosted applications and at the same time applications can be conveniently deployed within VM-based enclosures. The emergence of VMs has brought tremendous impact to the computing field, catalyzing important new paradigms such as cloud computing. Cloud systems are unique in their unprecedented elasticity, i.e., the ability to dynamically grow and shrink resources available to the hosted applications on demand. Such elasticity is generally enabled by virtualization and is the driving factor behind the success of the emerging commercial and academic cloud computing offerings [4][5][6].

VM migration is one of the important capabilities of system virtualization, which allows applications to be transparently migrated along with their execution environments across physical machines. Live migration further allows the VM to be migrated almost without any interrupt to its application's execution. VM migration is an important means for managing applications and resources in a large virtualized system. It enables resource usage to be dynamically balanced in the entire virtualized system across physical host boundaries, and it also allows applications to be dynamically relocated to improve performance and reliability. As the scale of virtualized systems such as clouds continue to grow, the use of VM migration, particularly live VM migration, becomes increasingly more important and frequent for optimizing application executions and resource usages in the systems.

VM migration can be resource intensive on its own and specifically it can consume substantial CPU cycles and network bandwidths. Hence, one VM's migration would compete for resources with other VMs' application executions as well as their migrations. Meanwhile, the resources available to perform a VM migration would also affect the performance of the migration and consequently the performance of the VM's application. However, existing studies on live migration performance are often based on the assumption that there are sufficient resources on the source and destination hosts [7][8][9], which is often not the case for highly consolidated systems. Therefore, it is important to understand the performance of VM migration under different levels of resource availability. With such knowledge, the management software in a virtualized system can take it into consideration when allocating resources and migrating VMs across the system according to the application and system optimization objectives.

This paper addresses the above need by creating performance models for VM live migration which can be used to predict the migration time given the application behavior in the migrating VM and the resource available to the migration. A series of experiments were conducted on Xen-based VM environment [2] in order to create such a model through profiling and modeling. Specifically, the migration time of a VM running a CPU-, memory-, disk-I/O-, or network-I/O-intensive benchmarks [10][11][12] is measured with different amount of CPU allocated to the Xen's Dom0 which processes the migration. These data are used to train the performance model using regression methods. The results confirm that the VM's live migration time indeed varies substantially as the amount of resource given to Dom0 changes. The results further demonstrate that the generated models can achieve good accuracy for predicting the migration times with the coefficients of determination generally higher than 90%.

The rest of the paper is organized as follows: Section II introduces the background and related work; Section III describes the methodology of our approach; Section IV presents the experimental results; Section V offers additional discussions; and Section VI concludes this paper.

II. BACKGROUND AND RELATED WORK

System VMs are becoming pervasively used in today's computer systems as they provide a powerful abstraction for application and resource provisioning. With such VMs, shared physical machine resources, including CPU, memory, and I/O devices, can be flexibly allocated to applications hosted on the VMs, whereas the applications can also be conveniently deployed along with VM provided encapsulation. System virtualization is enabled by the layer of software called Virtual Machine Monitor (VMM, a.k.a., hypervisor), which is responsible of multiplexing physical resources among the VMs. Full-virtualized VMs [1][3] present the same hardware interface to guest operating systems (OSes) as the physical machines and support unmodified OSes. Paravirtualized VMs [2] present a modified hardware interface modified for reducing virtualization overhead but they require the guests OSes to be modified as well.

Virtualization is the key technology that enables the emerging cloud computing paradigm [4][5][6], because it allows resources to be allocated to different applications on demand and hides the complexity of resource sharing from cloud users. VMs are generally employed in different types of cloud systems as containers for hosting application execution environments and provisioning resources. For example, in Infrastructure-as-a-Service (IaaS) clouds [4], VMs are directly exposed to users to deliver a full computer infrastructure over Internet; In Platform-as-a-Service (PaaS) clouds [6], VMs are also used by the clouds internally to manage resources across the application execution platforms delivered to users.

VM migration is a unique capability of system virtualization which allows an application to be transparently moved from one physical host to another and to continue its execution after migration without any loss of progress. It is generally done by transferring the application along with its VM's entire system state, including the state in CPU, memory, and sometimes disk too, from the source host to the destination host. VM migration is an important means for managing applications and resources in large-scale virtualized data centers and cloud systems. It enables resource usage to be dynamically balanced in the entire virtualized system across physical host boundaries, and it also allows applications to be dynamically relocated to hosts that can provide faster or more reliable executions.

There are mainly two types of migration strategies supported by modern VMMs. The first strategy, *suspend-copy-resume* based migration [13][14], migrates a VM based on the VMM's capability of suspending and resuming a VM. By suspending the VM, its entire state is dumped to persistent storage on the source host, which can be then copied and used on the destination host to resume the VM's execution. This strategy is easy to implement, but it can introduce significant overhead to the application in the VM, because its execution is completely paused during the entire migration. The other strategy is *live* migration [7][15][16], which allows a VM to almost continuously run during the migration and the application in the VM to perceive no or only small interrupt to its execution.

Xen live VM migration typically entails two phases, *pre-copy* and *stop-and-co*py. In the *pre-copy* phase, the VM is still running on source host while its memory pages are copied to the destination host. Every page that is modified after its previous transfer needs to be copied again to the destination host. In the *stop-and-copy* phase, the VM is stopped on the source host and all the remaining unsynchronized pages of the VM are copied at once to the destination host. After this phase, the VM is activated on the destination host and continues its execution.

The performance of VM migration is often measured by two time-related metrics. The first one is *down time* which is the duration when the VM is completely stopped and its application's service is entirely unavailable. This time corresponds to the length of the stop-and-copy phase in a live migration. Down time is a critical metric because it captures how much interrupt the users or applications of the VM perceive of the migration. Nonetheless, for an application with a small writable working set, the migration down time is typically short because most of the VM's memory state is transferred during the pre-copy phase without interrupting the application.

The other metric for measuring migration performance is *migration time*, which is the total time for all the involved migration phases. Migration time is also critical to both the virtualized system and applications. First, it decides how quickly the VM can be relocated to meet the system management objectives such as performance and reliability targets. Second, during the migration, the applications on both the source and destination hosts are impacted by the migration and their performance may degrade substantially [7][15]. From the perspective of the applications, the VM migration time is also their *performance degradation time*. Therefore, in this paper, we focus on the migration performance metric of total migration time and study its relationship with the migration's resource availability.

Live VM migration can consume substantial resources such as CPU cycles and I/O bandwidths on both the source and destination hosts, because it involves iterative copying of a VM's memory contents across network. Consequently, resources available to the migration can have a significant impact on the total migration time [7][15]. The migration's I/O bandwidth usage is dependent on the amount of state that needs to be transferred across the network, and the impact of network bandwidth allocation on the migration time is well studied in the literature [7]. Therefore, this paper considers mainly the CPU usage of VM migration and the modeling of migration performance under different CPU allocations on the source and destination hosts.

Our previous work [8] studied how to predict the migration time for *suspend-copy-resume* based migration of

VMs with different configurations, assuming that there are sufficient resources for performing the migration. This paper considers the modeling of more advanced live VM migration and it addresses the problem of predicting the migration time under different resource allocations for the migration. Related work also studied the performance of migration time given the available network bandwidth and the VM's page dirty rate [7][9]. However, as shown in our results, the amount of CPU resource available to the migration can also substantially affect the migration time. Therefore, this paper studies this important factor and proposes to use migration performance models to capture its impact on migration performance.

III. METHODOLOGY

In this section, we describe the methodology used in this study. Our fundamental goal is to build a performance model for live VM migration which can accurately estimate the migration time based on the resource allocation and guide the resource management decisions for a virtualized data center or cloud system. Towards this goal, this paper models the relationship between resource allocation and migration time by profiling the migration of VMs running different types of highly resource-intensive benchmarks. Such performance models can then be used to predict the migration times or at least their upper bounds for VMs that host the real-world, typically less intensive applications. Specifically, Xen-based system virtualization and live VM migration are studied in this paper. Xen is a widely used x86 VMM which can support VMs with strong resource isolation and performance guarantee [2]. In a typical Xen environment, *Dom0* is the privileged VM that has direct access to physical devices and manages the other non-privileged VMs, called *DomUs*, on the same physical host.

During a live VM migration, the Dom0s on both the source and destination hosts coordinate the transfer of the VM's memory and other system related information. Therefore, CPU usages of Dom0s on the two hosts have an impact on how fast the live migration can be finished. However, Xen does not provide a direct way to monitor and control how much CPU is spent in Dom0 for a VM's migration. In order to perform the proposed performance modeling for VM live migration, we control a migration's CPU usage on a host by leveraging Xen's ability to assign a specific amount of CPU to the entire Dom0. By controlling the amount of CPU given to Dom0, we can effectively limit the amount of CPU available to a VM's migration. The assumption is that the Dom0's CPU allocation is mainly used to process the target DomU VM's activities so that it can reflect the level of resource available to this VM only. This assumption is valid in our experiments as our profiling is done in a carefully controlled environment where the entire physical host is dedicated the target DomU VM.

The performance models created through this approach can be used to guide the resource allocation decisions in a virtualized system in two different ways. First, the models can be used to predict a specific VM's migration time given the VM's application behavior and the resources available to the migration. Without an exact control on the amount of resource used by Dom0 for a specific migration, we can still use the performance model to derive a lower bound of the migration time and understand its impact on the VM's application's performance and reliability. Second, the models can be also used to predict the necessary resource allocation for migrating a VM in order to meet its migration time target. Without an exact control on the resources used by Dom0 to process the concurrent migrations and other DomU activities, we can still use the models produce a lower bound on its total resource demand in order to meet all the migration time targets.

In our future work, we will implement the necessary mechanisms in the Xen VMM for directly and precisely measuring and controlling the resources consumed by a specific VM's migration. The performance modeling methodology taken by this paper would still apply with such direct mechanisms whereas the results of this paper would also provide key insights into the effectiveness of these mechanisms.

We implemented our profiling experiments with a VM running four types of resource-intensive benchmarks during the VM's live migration. The resources considered in this paper include CPU, memory, disk I/O, and network I/O. Specifically, in the experiments, we move a DomU VM from a source host to a destination host using Xen's live migration functionality. The CPU usage of Dom0 on the hosts is controlled by setting the CPU cap parameter of Xen's credit CPU scheduler [17]. Experiments included in this paper are based on the following configurations. Both the source and destination hosts have two 6-core 2.4GHZ Opteron CPUs and 32GB of memory. Xen 3.2.1 is installed on both servers to provide VM environments. The migrated VM has 1GB of memory and 4GB of disk using EXT3 file system, unless otherwise noted. The VM runs Ubuntu Hardy with the Linux version 2.6.24. To facilitate live migration, the VM image is stored on a network file system server accessible to both hosts.

IV. EXPERIMENTAL ANALYSIS

A. Migration of CPU-intensive VM

If the live VM migration is indeed a CPU intensive operation, then a VM's migration time would increase as the CPU available to the migration decreases. In this group of experiments, we study the impact of Dom0's CPU allocation on a DomU VM's migration time. The migrated VM runs a CPU-intensive benchmark [10] which does mainly intensive calculations without consuming much of other types of resources. We control its intensity by setting a different CPU cap on the DomU during the migration.

Figure 1 shows that as the CPU allocated to Dom0 on the source host increases from 10% to 50%, the migration time drops dramatically, from about 100 seconds to about 20 seconds. However, after Dom0's CPU allocation exceeds 50%, the migration time stays at the same level of about 17 seconds. By monitoring Dom0's actual CPU usage during the whole migration, we find out that when Dom0 is assigned more than 50% of CPU it only consumes

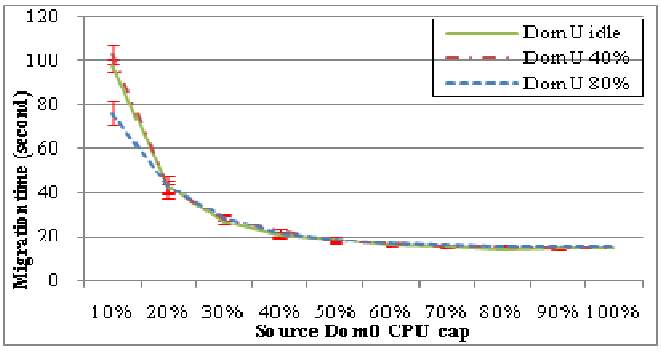

Figure 1 Migration time for a CPU-intensive VM with different CPU allocations to Dom0 on the source host

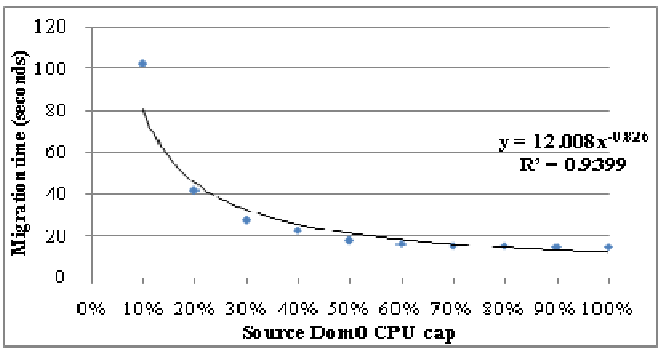

Figure 2 Model for the migration time for a 40% CPU-intensive VM with different CPU allocation to the source host's Dom0

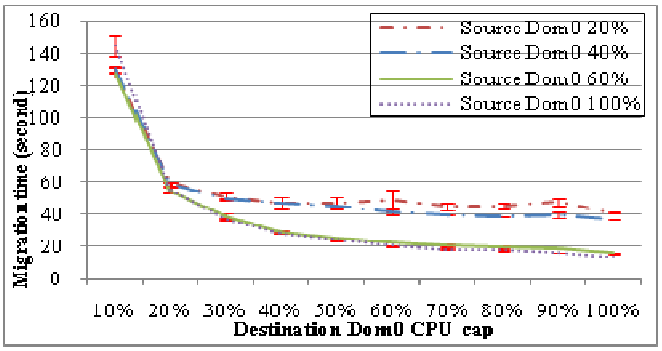

Figure 3 Migration time for a CPU-intensive DomU with different CPU allocations to Dom0s on the source and destination hosts

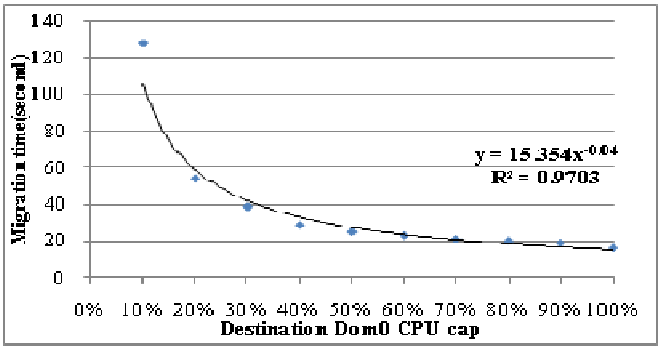

Figure 4 Model for the migration time with 60% CPU to the source host's Dom0 and different allocation to the destination host's Dom0

at most 50%. Therefore, additional CPU allocation does not help further speed up the live migration.

The results also show that the migration time is almost identical when the DomU's CPU intensity changes from idle (without running the benchmark), 40%, to 80%. This observation can be explained by two factors. First, the CPU usages by Dom0 and DomU are well isolated without much interference. Second, there is abundant CPU for both Dom0 and DomU to consume on this particular physical host which has 12 2.4GHz CPU cores.

Based on these data we build a performance model for the migration of such a CPU-intensive VM using statistical modeling. Specifically, we use power regression to generate the model that best fits these data. Figure 2 visualizes the modeling results when the VM runs a 40% CPU intensive workload, which shows that the produced model can well fit the experiment data. We also use the coefficients of determination, R^2, a metric often used to measure the accuracy of modeling. The value of R^2 ranges from 0 to 1, and the closer it is to 1 the more accurate the model is in capturing the input data. Specifically in this experiment, the value of R^2 for modeling the migration of a VM with 0%, 40%, and 80% CPU-intensity is 94.1%, 93.99%, and 95.81% respectively. These results further quantitatively prove that the model can accurately capture the relationship between Dom0's CPU allocation and DomU's migration time.

The above experiments were done when we controlled the CPU allocation to Dom0 on the source host. Moreover, the destination host's Dom0 CPU allocation can also impact the VM's migration time. To study this impact, we conducted another experiment by also setting different caps on the destination host's Dom0. In this experiment, the migrating DomU VM is always given 100% of CPU.

From the results in Figure 3 we can see that, no matter how much cap is set on the source host's Dom0, the migration time always drops when the destination host's Dom0 is given more CPU resource. The drop is significant when this cap is less than 50%. After that, the decrease on migration time is not significant as the CPU allocation further increases, which is also due to the fact that the destination host's Dom0 does not need more than 50% of CPU during the migration.

The results also show that as we control the CPU allocation to the destination host's Dom0, the CPU allocated to the source host's Dom0 also affects the DomU VM's migration time, particularly when the source host Dom0's cap is less than 50%. Therefore, in order to achieve a desired migration time, both the source and destination hosts' CPU resources need to be carefully managed, as they may be both under contention and can both affect the migration performance. However, for simplicity, in the rest of this paper, we only study the impact of the source host Dom0's CPU allocation to the live migration time.

Figure 4 shows the modeling results for varying CPU cap on the destination host's Dom0 while the source host's Dom0 has a CPU cap of 60%. The coefficient of determination for this modeling is 97.03%, which means that most of the data can be captured by this model. In the other models created when the source host's Dom0 is given

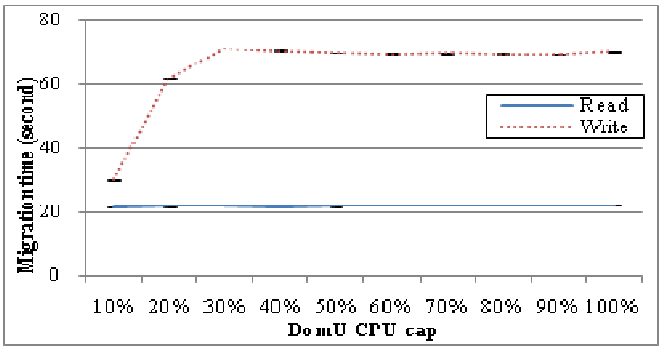

Figure 5 Migration time for a memory-intensive VM with different CPU allocations to DomU on the source host

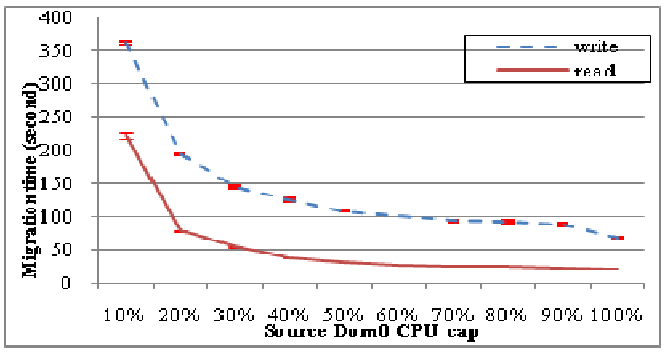

Figure 6 Migration time for a memory-intensive VM with different CPU allocations to Dom0 on the source host

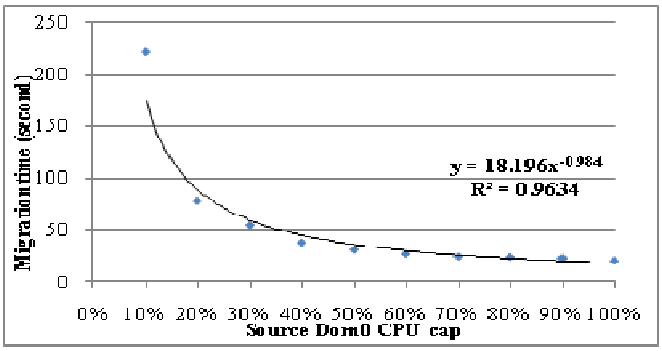

Figure 7 Model for the migration time for a memory read intensive VM with different CPU allocation to the source host's Dom0

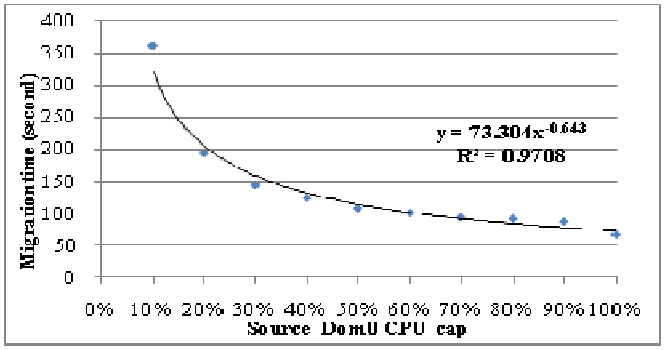

Figure 8 Model for the migration time for a memory write intensive VM with different CPU allocation to the source host's Dom0

20%, 40%, and 100% of CPU, the coefficients of determination are 75%, 85.53%, and 97.72%, respectively.

B. Migration of Memory-intensive VM

In this subsection, we study the performance model for migrating VMs running memory-intensive applications. Specifically, we consider two types of memory operations, read and write. We created two synthetic benchmarks which continuously read and write, respectively, 1GB of data from memory. The DomU VM used in this group of experiments is configured with 2GB of memory. Both the memory read and write benchmarks would consume substantial memory bandwidth, whereas the migration also needs to transfer the VM's memory state between the source and destination hosts. In addition, the write benchmark would incur substantial additional work on the VMM and Dom0, as the memory dirtied by it during the migration needs to be iteratively copied to the destination.

First we want to study the impact of an application's memory operation intensity on the VM's migration time. To control the intensity of the memory benchmarks, we set different CPU caps to the DomU VM, because the more CPU cycles the VM can use the more memory operations its application can issue. We then migration this VM without any CPU cap on the Dom0s. The results in Figure 5 show that when DomU is running an application that is memory read intensive, the migration time almost stays the same at about 20 seconds. There is no obvious increase on migration time when the memory read intensity increases. From this experiment, we do not see the impact of a VM's memory bandwidth usage on its migration time. This observation can be at least partly explained by the fact that the amount of state that the migration needs to transfer is bounded the VM's memory size as the memory read benchmark does not modify the VM's memory pages at all.

The results from migrating a VM running the memory write benchmark are more interesting and show a quite different pattern. When the CPU cap on the DomU VM is set less than 30%, the migration time increases quickly as its cap goes up. However, after the CPU cap is set more than 30%, the migration time stays almost the same, regardless of the further increase of CPU cap on the DomU VM. These results can be explained by how Xen transfers a VM's memory state during the live migration. During the pre-copy phase, Xen will copy and transfer the memory state in several rounds. In each round, only the memory that has been dirtied after the previous round will be transferred again to the destination host. Normally when there is not many dirty memory pages left, Xen will move on to the stop-and-copy phase. However, in some extreme cases, like the one with the memory write benchmark, the memory is dirtied frequently, so transferring the dirtied memory pages iteratively will substantially slow down the migration process. When Xen recognizes this problem in the migration, it would immediately enter the stop-and-copy phase and transfer all the dirtied memory pages in a single round, in order to save the migration time. This is what happens in our experiment with the memory write application when the VM's CPU cap is set at 30% or more. Because the VM's memory pages are updated so frequently,

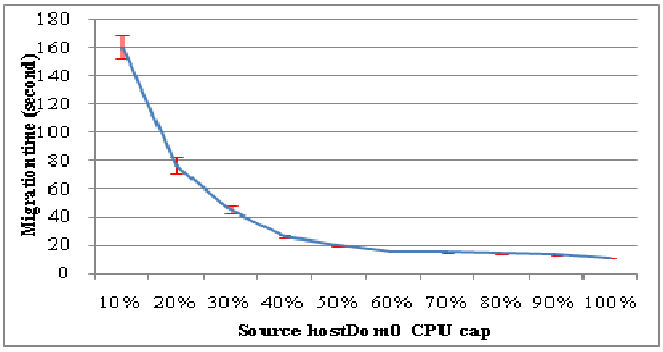

Figure 9 Migration time for a disk I/O-intensive VM with different CPU allocations to Dom0 on the source host

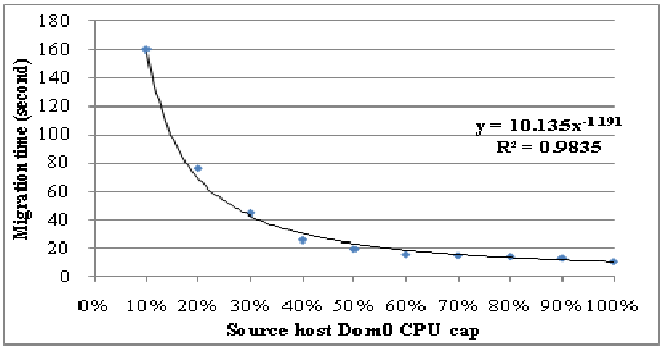

Figure 10 Model for the migration time for a disk I/O-intensive VM with different CPU allocations to Dom0 on the source host

it triggers Xen's immediate stop-and-copy and as a result the VM's migration time stays the same.

In the next two experiments, we study the impact of Dom0's CPU availability to the DomU VM's migration time by migrating the VM running the memory read or write benchmark with a 100% CPU cap on DomU and different caps on Dom0. Figure 6 shows that when DomU is running either the memory read or write benchmark, the migration time drops quickly as more CPU cycles are available for Dom0 to process the migration. In the meantime, running the memory write intensive benchmark on the VM always makes its migration much slower than running the memory read benchmark.

Figure 7 and Figure 8 show the modeling results for the migration time of the VM running the memory read and write benchmark respectively. The coefficients of determination (R^2) are 96.34% and 97.08% which are both close to 1, showing good accuracy of these models.

C. *Migration of Disk I/O-intensive VM*

Because a DomU VM's disk accesses have to be processed by Dom0 which has the actual access to the physical disk devices, a disk I/O intensive VM would involve substantial overhead in Dom0 when it is migrated. In this subsection, we study the impact of Dom0's CPU allocation to the migration time for a VM running a disk I/O intensive application. Specifically, we run the fio benchmark [11] on the DomU VM to sequentially read an 8GB file. The file cannot be cached in the DomU's 1GB memory and we also drop the cache on Dom0, so the benchmark's execution will create intensive I/Os to the physical disk. In this experiment, the VM's disk size is increased to 20GB in order to store the file. The sequential read pattern is chosen to run the benchmark because it can generate intensive and consistent IOPS (I/O Operations Per Second) compared to sequential write or random read or write patterns. So it is useful for creating a performance model that can predict the upper bound migration time for a VM running disk I/O intensive applications.

Figure 9 shows the relationship between the Dom0's CPU cap and the DomU VM's migration time. As more CPU cycles become available for Dom0 to handle the migration, the migration time reduces accordingly. However, after the cap goes more than 50%, there is no substantial change on migration time anymore, again due to the fact that Dom0 does not use more than 50% of CPU for the migration. We also use power regression to build a performance model for such a VM running the disk I/O-intensive application. The modeling results are illustrated in Figure 10. With the coefficient of determination (R^2) equal to 98.35%, most of the data points from experiments can fit in this model, meaning that if it can produce good accuracy when used to predict the VM's live migration time.

D. *Migration of Network I/O-intensive VM*

Similarly to the above disk I/O intensive VM, a DomU VM running a network I/O intensive application would also incur additional overhead in Dom0 as the DomU's network I/Os have to be handled by Dom0. In addition, transferring a VM's state between different physical hosts requires substantial network bandwidth, so the available network bandwidth would affect the VM's migration time. This latter impact is well studied in the related work [2]. So this group of experiments focuses on the impact of CPU allocation to Dom0 on the migration time for a DomU VM running a network I/O intensive application.

We use TTCP [12] as the benchmark for generating continuous network I/Os between a sender and a receiver using UDP. We use a third physical machine to run another instance of TTCP which transfers a 512MB file to or from the TTCP running on the migration VM. The file is preloaded into the sender's so no disk I/Os are involved in the experiments. During the two conducted experiments, the migrating VM is set up to be either a sender or receiver.

Being a sender or receiver during the live migration does not make much a difference on the VM's migration time, as shown in Figure 11 and Figure 12. When we vary the cap set on Dom0 to control its CPU allocation, the migration time decreases as the cap goes up. When the cap is set more than 60%, although there is some decrease on migration time, it is not substantial. Noticed that when Dom0 has 100% access to its CPU resource, it consumes only about 60% of the CPU most of the time, so given more than 60% of CPU to Dom0 would not further help reduce the migration time.

Given these experimental data, we created models for the two cases, where the migration VM acts as a sender or a receiver running a network I/O intensive application. The modeling results are shown in Figure 13 and Figure 14. The coefficients of determination (R^2) of the models are both

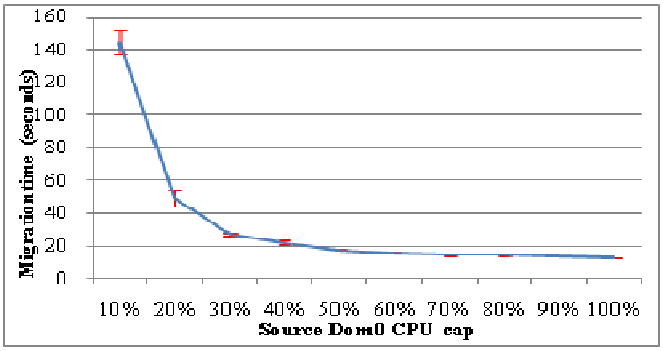

Figure 11 Migration time for a network-send-intensive VM with different CPU allocations to Dom0 on the source host

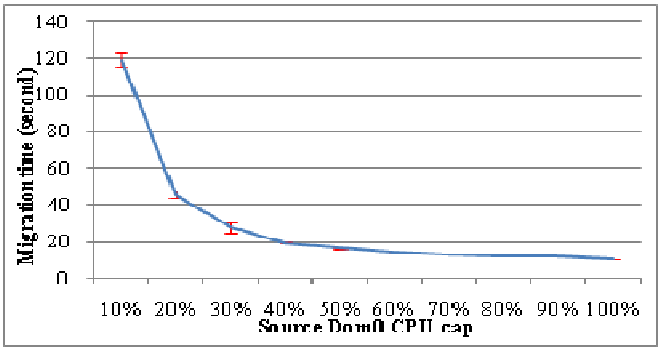

Figure 12 Migration time for a network-receive-intensive VM with different CPU allocations to Dom0 on the source host

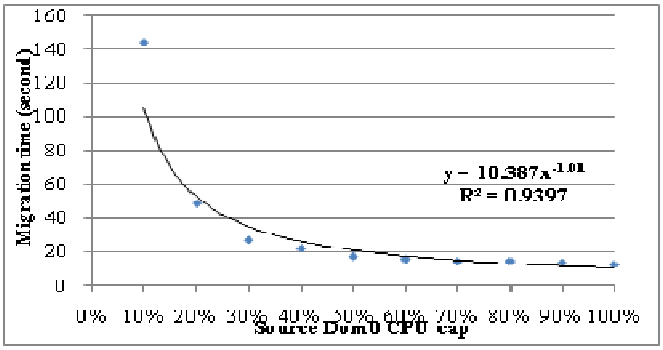

Figure 13 Model for the migration time for network-send-intensive VM with different CPU allocations to Dom0 on the source host

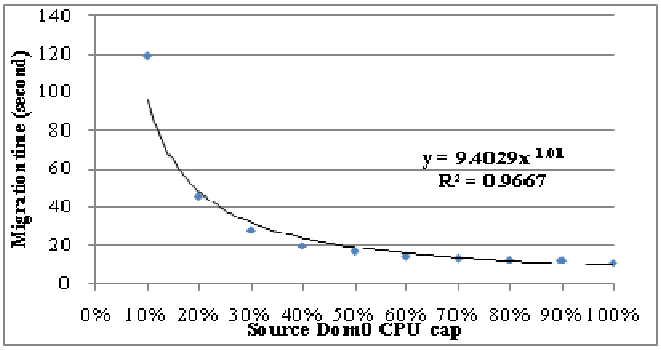

Figure 14 Model for the migration time for network-receive-intensive VM with different CPU allocations to Dom0 on the source host

Table 1 Model Prediction Error

VM	Prediction Error (s)	
	Average	Median
CPU-intensive	1.58	1.74
Memory read intensive	7.63	3.33
Memory write intensive	9.25	6.79
Disk I/O intensive	4.01	2.97
Network send intensive	3.53	3.16
Network receive intensive	4.94	2.43

close to 1, again indicating that most of the experimental data fit in the produced models.

From the modeling results in the above subsections, we can also see that as a VM runs different types of resource-intensive benchmark, its migration time is different even with the same amount of resource allocated to the migration. In particular, when the benchmark makes intensive use of memory write, disk I/O, or network I/O, its VM's migration time is much longer than when it is only CPU-intensive. Therefore, when applying these models to predict a migration's performance and resource demand, we also need to understand the resource usage of the application on the VM, which can be monitored through the typical profiling tools available in guest OSes and VMMs.

E. Model Prediction

Finally, we evaluate the prediction accuracy of the migration performance models created using our approach by using separate datasets for training the migration performance models and evaluate the prediction accuracy of these models. For each experiment, 5 training data points are evenly picked from the entire CPU allocation range and used to create the model, which is then used to predict the migration times for the other CPU allocation values in the test data. The predictions are compared to the measured migration times to compute the prediction errors. For the experiments of migrating CPU-intensive VMs, only the one with 80% CPU-intensive VM is evaluated here.

The results are summarized in Table 1 and they show that the difference between the predicted and measured migration times is typically within a few seconds. The prediction accuracy is always good for migrating a CPU intensive VM. The prediction error is higher when the CPU allocation to Dom0 is low for migrating a memory-, disk-, or network-intensive VM. This result demonstrates that when Dom0 is under heavier contention from the DomU activities, its behavior becomes less predictable.

V. DISCUSSION

As demonstrated in the proceeding results, the resource intensity of a migrating VM has a substantial impact on its migration time. Therefore, an assumption made by our performance modeling approach is that the VM's behavior stays stable during the period of the migration. Given the fact that live migration usually finishes within seconds for VMs hosting real-world applications, we believe that this is a reasonable assumption. The other assumption is that the platforms where the migration performance models are

trained and where they are used for prediction are the same. Because the differences in software (VMM) and hardware (CPU, memory, disk, and network) can affect a VM's migration, cross-platform performance prediction is not trivial and will be considered in our future investigation.

For live VM migration, both the availability of CPU cycles and the network bandwidth are major factors that can affect migration time. Although in this paper, we focus on only the impact of CPU availability, we have found out that, through other experiments, the use of CPU and network bandwidth are highly correlated during a migration. For example, when we fix only the CPU cap to Dom0, the network transmission rate is also relatively stable and it changes as the CPU caps varies. This observation implies that from the performance models created using only the CPU cap as the input, we can infer the appropriate network bandwidth cap and obtain the same migration performance. Therefore, it is unnecessary to take both CPU availability and network bandwidth availability to build performance models for live VM migration. This conclusion is also confirmed experimentally.

Although this paper does not study the performance of migrating multiple VMs in parallel, its methodology of performance modeling can also be applied for that purpose. Moreover, performance models created from non-parallel migrations can be also used to provide an estimate for a parallel migration. On one hand, the prediction made for individual VM migrations using such models can provide a lower bound for when they are migrated in parallel. (Due to interference among concurrent migrations, they are likely to be slower than when they are migrated sequentially.) On the other hand, given the desired migration times for all the concurrently migrated VMs, we can also use these models to determine a lower bound for the total required resource, which is the sum of the individual migrations' resource demand. We will conduct a more thorough study on the performance of parallel VM migrations in our future work.

VI. CONCLUSION

System virtualization is a powerful platform for provisioning applications and resources in the emerging computer systems such as utility data centers and cloud systems. Live VM migration is an important tool for managing such systems in various critical aspects such as performance and reliability. Understanding the role that the resource availability plays on the performance of live migration can help us make better decisions on when to migrate a VM and how to allocate the necessary resources. This paper is an effort towards this goal by creating a performance model that can be used to predict migration time and guide resource management decisions.

Our approach is to profile the migration time for VMs running different representative benchmarks and then build the performance model using statistical methods such as regression. Specifically, we did a series of experiments by migrating a Xen-based VM running CPU, memory, or I/O-intensive applications and allocating different amount of CPU share to the migration. The results demonstrate that the amount of resource available to live migration indeed has a substantial impact on the migration time. Nonetheless, the models created using this approach can precisely capture this relationship and can be effectively used to predict a VM's live migration time.

ACKNOWLEDGEMENT

This research is partly sponsored by the National Science Foundation under grant CCF-0938045 and the Department of Homeland Security under grant 2010-ST-062-000039. The authors are also thankful to the anonymous reviewers for their constructive 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] Carl A. Waldspurger, "Memory Resource Management in VMware ESX Server," Proceedings of the 5th Symposium on Operating Systems Design and Implementation, December 2002.

[2] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield, "Xen and the Art of Virtualization," Proceedings of the 19th ACM Symposium on Operating Systems Principles, October 19-22, 2003.

[3] Kernel Based Virtual Machine, URL: http://www.linux-kvm.org/page/Main_Page.

[4] Amazon Elastic Compute Cloud (Amazon EC2), URL: http:// aws.amazon.com/ec2/.

[5] Windows Azure Platform, URL: http:// www.microsoft.com/ windowsazure/.

[6] Google App Engine, URL: http://code.google.com/appengine/.

[7] Christopher Clark, Keir Fraser, and H. Steven, "Live Migration of Virtual Machines," 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation, 2005.

[8] Ming Zhao and Renato J. Figueiredo, "Fast Transparent Migration for Virtual Machines," 2nd International Workshop on Virtualization Technology in Distributed Computing, 2007.

[9] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper, "Predicting the Performance of Virtual Machine Migration," 18th International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems, Aug. 2010.

[10] Isolation Benchmark Suite, URL: http://web2.clarkson.edu/class/ s644/isolation/.

[11] FIO Benchmark Tool, URL: http://freshmeat.net/projects/fio/.

[12] TTCP Benchmark Tool, URL: http://www.pcausa.com/Utilities/ pcattcp.htm.

[13] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M.Rosenblum, "Optimizing the Migration of Virtual Computers," In Proc. of the 5th Symposium on Operating Systems Design and Implementation (OSDI-02), December 2002.

[14] M. Kozuch and M. Satyanarayanan, "Internet Suspend/Resume," In Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications, 2002.

[15] M Nelson, B. H. Lim, and G. Hutchins, "Fast Transparent Migration for Virtual Machines," USENIX Annual Technical Conf., 2005.

[16] Uri Lublin and Anthony Liguori, "KVM Live Migration," KVM Forum, 2007.

[17] Credit-Based CPU Scheduler, URL: http://wiki.xensource.com/ xenwiki/CreditScheduler.