A Look Into Virtual Machine Malleability

By

Jonathan Goldszmidt

A Thesis Submitted to the Graduate
Faculty of Rensselaer Polytechnic Institute
in Partial Fulfillment of the
Requirements for the degree of

MASTER OF SCIENCE

Major Subject: COMPUTER SCIENCE

Approved by the
Examining Committee:

_____
Carlos Varela
Thesis Adviser

_____
Christopher Carothers, Member

_____
Stacy Patterson, Member

Rensselaer Polytechnic Institute
Troy, New York

April 2014
(For Graduation May 2014)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I would like to thank my advisor, Professor Carlos A. Varela, for his encouragement and support on my research. His challenges and guidance helped me significantly improve my thesis. Many thanks to my colleagues at IBM for their support, encouragement and flexibility. I also would like to thank my family, friends and girlfriend for their moral support throughout my journey at Rensselaer Polytechnic Institute.

ABSTRACT

Many have leveraged virtual machine cloning, migration and other features of virtualization to improve and guarantee high availability and performance of their applications. Virtual machine *malleability* is the ability to *split* and *merge* virtual machines based on demand and performance, to improve virtualization functionality. The splitting function will take a virtual machine with a workload of N independent processes and split the independent processes into at most N virtual machines. Similarly, the merging function will take at most N previously-split virtual machines and merge their independent processes into a single virtual machine. There has been previous research in malleability at the application layer. However, most of the benefits of application layer malleability can also be realized at the virtual machine layer. These benefits include dynamic workload reconfigurability while affording better transparency. The benefits provided by virtual machine malleability can be put into 3 categories: scalability and elasticity, energy improvements and process scheduling. Traditionally, the approach of application scaling at the application layer involves writing an application with malleability in mind. Using our method of scaling, applications do not need to be written with malleability in mind to be able to scale, as long as the application workloads are independent or they can communicate over a network. The approach of energy optimization typically involves consolidating virtual machines onto fewer hosts and putting idle hosts in a low power or off state. Although this method works quite well, certain virtual machine configurations cannot be consolidated without significantly impacting performance. Using virtual machine malleability, processes running on a virtual machine that is on an overloaded physical machine can be split across multiple virtual machines, each with a smaller resource footprint. Each of these split machines can then be consolidated without causing the physical hosts to become overloaded. Such flexibility in virtual machine scheduling enables better energy consumption. Our experiments with virtual machine malleability show that application scalability is possible with negligible performance degradation, that it is possible to dynamically reconfigure workloads to achieve set resource utilization, which in turn improves energy utilization. We evaluate scalability and elasticity by splitting and merging virtual machines running many instances of the same CPU intensive process. Our evaluation of splitting and merging demonstrates linear performance changes. Finally, we discuss some future work which includes the design of middleware for managing virtual machine malleability.

1. Introduction

1.1 Motivations

Over the past decade a huge push toward energy efficiency and cost reduction has driven server virtualization. Virtual machines have revolutionized the way workloads can be handled. In the past, hardware and software limited what could and could not be run on a physical machine. As hardware improved, applications were migrated from older machines onto newer machines, which offered greater resources. Unfortunately, most applications running on these machines never fully utilized all of the available resources. Although these machines only utilized a fraction of the available resources, they still consumed a large amount of power. Virtualization made it possible to consolidate many of these systems onto fewer physical machines. This virtualization migration had significantly reduced cost by eliminating excessive hardware, reducing electricity waste, reducing IT costs and overall datacenter costs [1] [2] [3] [4].

The current market share of virtual server workloads is two-thirds worldwide as of June 2013 and it is still expected to grow [5]. However, not all virtual machine configurations are conducive to all of the cost reductions and benefits of virtualization. For example, having too many compute intensive virtual machines on a host causes performance degradation, which results in a decrease in the compute/watt ratio [6] [7].

Another reason to virtualize is scalability. Virtual machines can be reconfigured relatively quickly to scale with application demand [6] [8]. Features related to scalability include cloning, migration and deployment. Cloning makes an exact replica of a virtual machine, migration moves the virtual machine from one physical host to another physical host and deployment allows virtual machines to be created based on a template.

This thesis presents two functions that provide hypervisors with additional functionality which improve scalability, improve power utilization and manipulate process scheduling algorithms. These functions are virtual machine splitting and virtual machine merging. Like most hypervisor functionality, these functions are based on and leverage existing functionality.

1.2 Structure of Thesis

The structure of this thesis is as follows: In Chapter 2, we will discuss the types of virtual machine splitting, the strengths and weaknesses related to each type and some changes to improve the overall performance of this operation. In Chapter 3, we will introduce virtual machine merging, its limitations, and some workarounds. In Chapter 4, we will compare virtual machine malleability to application malleability. In Chapter 5, we will dive into application scaling, the energy optimization problem and scheduling manipulation. In Chapter 6, we will discuss the middleware for enabling virtual machine splitting and merging. In Chapter 7, we will explore related work. Chapter 8 concludes this thesis and discusses future work.

2. Virtual Machine Splitting

Virtual machine splitting is defined as the process of taking one virtual machine and splitting it into two or more virtual machines. Each of these split machines contain a subset of the original machine's user processes.

2.1 Virtual Machine Splitting

Virtual machine splitting is based on the cloning and migration features offered by virtual machine hosts and virtual machine monitors respectively. The most basic form of virtual machine splitting is stateless splitting as contents of memory are discarded before cloning, so that the children virtual machines do not start at their parent's state. Stateful virtual machine splitting is a more advanced form as it preserves the parent's memory, so the children virtual machines resume from where the parent left off. To draw a parallel to biology, if a cell was considered to be a virtual machine, its DNA would represent the disk of the virtual machine. When a parent cell splits into two, its entire DNA is duplicated before the split is complete; this is equivalent to duplicating a virtual machines disk before splitting. By creating additional middleware consisting of a malleability manager and a process running on the virtual machine, it is possible to split running processes between virtual machines and reconfigure networking. The middleware is covered in Chapter 6.

2.1.1 Stateless Virtual Machine Splitting

Stateless virtual machine splitting follows the same process as the cloning feature provided by the virtual machine host. Cloning in its most basic form makes a copy of the virtual machine files, such as the virtual machine configuration and the virtual machine disks. Then it registers the cloned virtual machine on the host. Stateless virtual machine splitting provides the ability to reconfigure the virtual machine before it is registered on the host. This is particularly useful in cases where a virtual machine is running multiple stateless applications such as RESTful services, APIs or daemons that have too many or too few resources allocated to each.

2.1.2 Stateful Virtual Machine Splitting

Stateful virtual machine splitting is almost identical to stateless virtual machine splitting, except it provides a copy of the parent's memory for each child. This is particularly useful when multiple compute intensive, long running processes are living on a virtual machine and the

system is not configured to handle live resource allocation. In this case, it is possible to pseudo-allocate more resources to each process by creating a stateful split and terminating the redundant, long running processes (including their dependencies), so that at most one copy of each long running process exists in one of the child virtual machines.

2.1.3 Implementation of Virtual Machine Splitting

Virtual machine splitting begins when the malleability manager is invoked. It powers off or suspends the running virtual machine. During the suspend process the host writes the contents of the virtual machine's memory to disk. The malleability manager then reads the virtual machine configuration file to determine which virtual disks it needs to copy. It then creates the destination directory and copies the virtual disks into it. In the case of a stateless virtual machine split, the virtual machine configuration file can be modified to better reflect the needs of each split virtual machine. The stateful virtual machine configuration cannot be edited as stateful virtual machines need to restore their entire state before hardware modifications can be made. In the case of stateful virtual machines, the parent's state is then copied to each of the split virtual machines. Once all of these tasks have completed, the virtual machine is registered on the destination host and powered on or resumed. The malleability manager establishes connections to the malleability process running on each of the virtual machine splits and requests a list of running user processes. It then terminates a subset of these processes on each virtual machine, so that only one copy of each process remains.

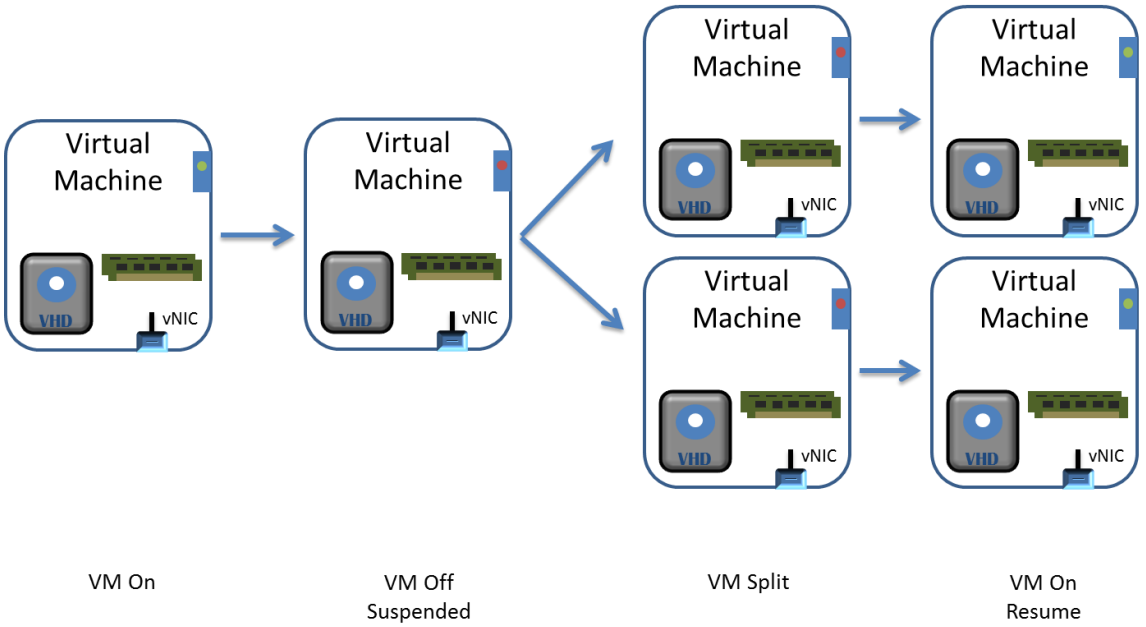

Figure 2.1: Virtual Machine Splitting

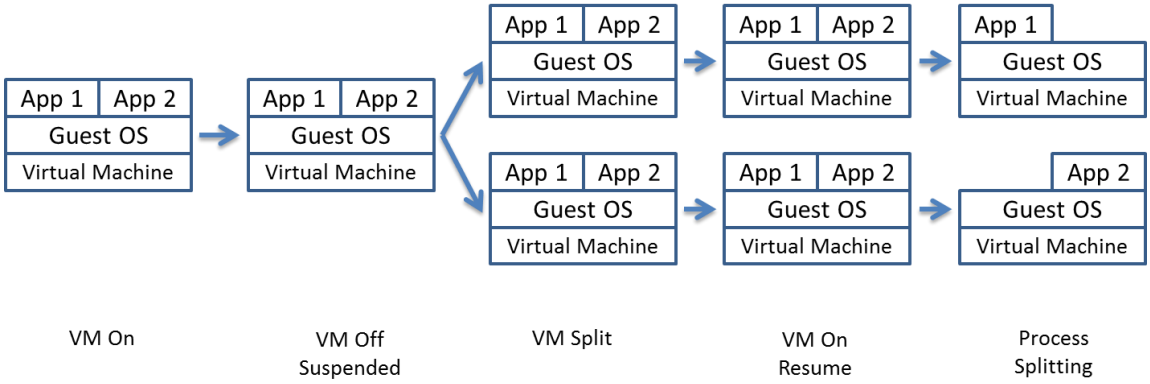

Figure 2.2: Virtual Machine Splitting at the Application Layer

2.1.4 Experimental Setup

In the experiment of virtual machine splitting, we use a server (CPU: Intel i7 3930k overclocked to 4.0 GHz, Memory: 64 GB, OS: Windows 8.1 Pro) running 4 instances of VMware Player. First instance (OS: Windows Server 2008 R2, 8 GB memory, 1 vcpu, 1 dedicated HDD using pass-through, 1 virtual hard drives). Second instance (OS: Ubuntu, 4 GB memory, 1 vcpu, 1 virtual drive). Third and fourth instances (OS: ESXi 5.5, 16 GB memory, 6 vpu, 1 virtual drive,

both connected to the iSCSI devices, both have SSH enabled). All of these virtual machines are connected to the same network.

The windows server is configured to act as an iSCSI adapter for its physical hard disk. To enable ESXi to do migration, both the source and destination host need to be connected to the same iSCSI disk. Setting up an iSCSI was the most cost effective method of satisfying this requirement.

The Ubuntu virtual machine is configured as the malleability manager. It interacts with both ESXi hosts and performs the tasks required by virtual machine splitting. The malleability manager tasks are defined in a series PHP scripts. These scripts leverage SSH to interact with each of the ESXi hosts. The public key of the user running the malleability manager is stored on each ESXi host.

During the experiment, no user applications were being run on the Windows 8.1 host, ESXi hosts and Windows Server 2008 R2 virtual machine. The Ubuntu virtual machine ran the scripts required to split a virtual machine and its processes.

2.1.5 Benchmarking Virtual Machine Splitting Process

Virtual machine splitting is complicated to benchmark. It consists of 3 components: The first is shutdown or suspend, the second is copy and modify and the third is power on or resume. A large part of virtual machine splitting involves significant disk I/O, and performance is significantly affected by hard disk utilization, specification, fragmentation, source and destination. Other factors that affect the amount of disk I/O include memory utilization, size of virtual machine files and size of virtual machine state. Factors that affect writing to and reading from the disk include network utilization and configuration. All of these factors can significantly impact the performance of virtual machine splitting. As reading and writing to memory is very fast compared to reading and writing to disk, its impact is treated as negligible. Table 2.1 contains a method to effectively calculate the cost of virtual machine splitting based on its disk I/O driven nature.

Table 2.1: Virtual Machine Splitting Cost Model

Virtual Machine Splitting Cost Model
V_{DS} = Virtual Disk Size **V_{MS} = Virtual Memory Size** **V_{FS} = Virtual Machine File Size** **N_S = Number of Virtual Machine Splits** **S = Is Stateful (0 or 1)** **P = Performance**
$P(V_{DS},V_{MS},V_{FS},N_S,S) = 2*N_S*(V_{DS} + V_{FS} + S*V_{MS})$

As virtual machine files are relatively small compared to virtual machine disk size and virtual machine memory size, we can break down the equation in Table 2.1 to account for the cost of copying the virtual machine disk and the cost of copying the virtual machine state as seen in the equations on Table 2.2.

Table 2.2: Virtual Machine Splitting Cost Dissected

Virtual Machine Splitting Cost Dissected
$P_{DISK}(V_{DS},N_S) = 2 * N_S * V_{DS}$
$P_{STATE}(V_{DS},V_{MS},N_S) = 2 * N_S * V_{MSs}$

The equations from Table 2.2 are graphed in Figure 2.3 and Figure 2.4 to show the significance that virtual machine disk size, memory size and number of splits adds to the overall cost of virtual machine splitting.

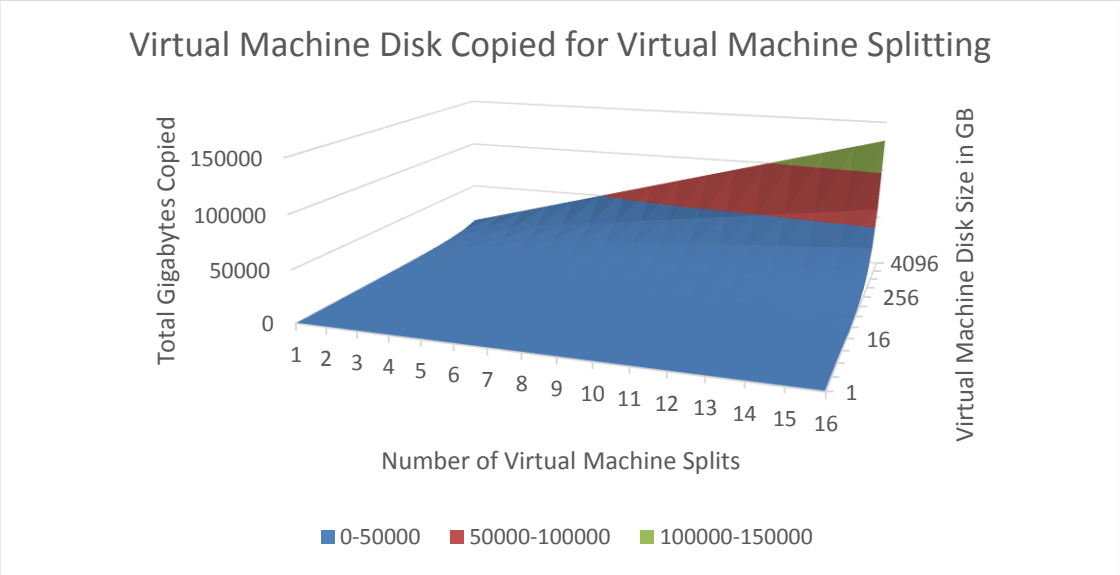

Figure 2.3: Virtual Machine Splitting Disk Copying Cost

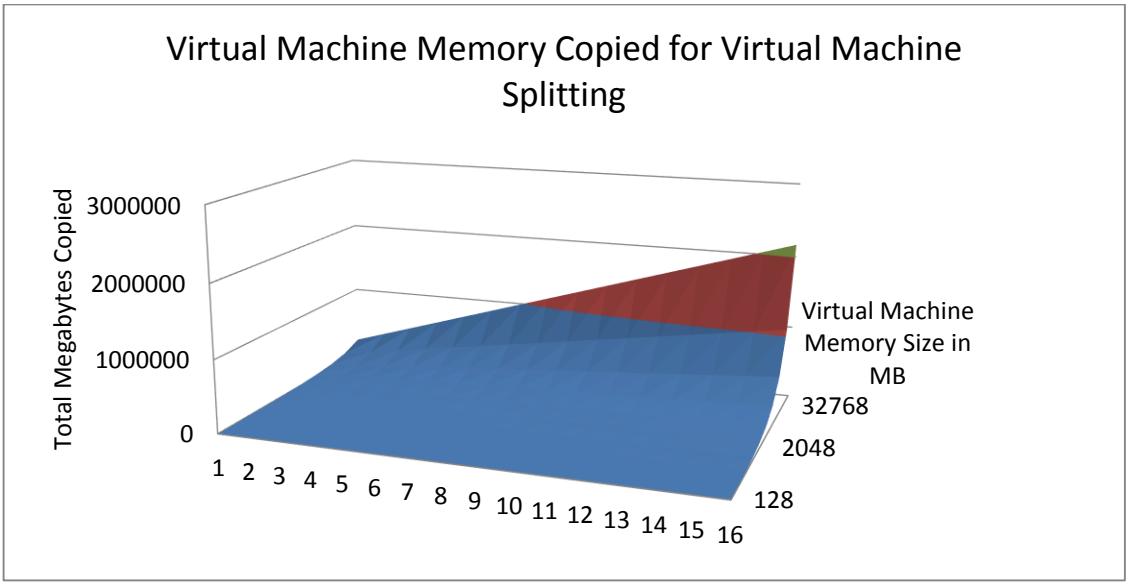

Figure 2.4: Virtual Machine Splitting Memory Copying Cost

2.1.6 Cost of Splitting a Virtual Machine

As we can see in Figure 2.3, the cost of a stateless virtual machine split is most affected by the size of the virtual machine's disk. The time to split a virtual machine increases linearly as the size of its virtual disk and the number of splits increase. In Figure 2.4 we see the additional

cost is caused by statefully splitting a virtual machine. Similarly to the first result, the time to split a virtual machine increases linearly as the size of the memory and the number of splits increase. Although linear time increase is significantly better than exponential time increase, it makes it hard to justify virtual machine splitting. Furthermore, for statefully splitting virtual machines with significantly larger virtual disks than memory, the cost of maintaining state is negligible compared to copying the disk. Also for systems with large or mostly empty virtual disks, the splitting process is significantly impacted.

2.1.7 Similarities to Virtual Machine Cloning

Cloning a virtual machine faces many of the same challenges as splitting a virtual machine. It is very costly as it requires making an exact duplicate of all virtual machine related files. Although most of these files are generally small, the virtual hard disks are large relative to other virtual machine files. Most of the cloning process is spent copying a virtual hard disk. Other factors that affect cloning include having the same source datastore and destination datastore or having both datastores residing on the same physical disk. Another significant downside to cloning is creating unnecessary redundancy; once a clone completes, there are two identical copies of every bit of data, the original virtual disk and the copied virtual disk.

2.1.8 Improvements to Virtual Machine Splitting

Although it is still faster to split or clone a virtual machine than to setup a new virtual machine, the performance hit of copying a virtual machine disk is too great. To solve this performance hit, we will modify the virtual machine splitting process to use delta disks. A delta disk is a virtual hard disk that stores all file system changes since its creation. When created, a delta disk keeps a reference to its parent disk. The parent disk is set to be read-only. When a virtual machine reads from a delta disk, it acts like a normal virtual disk. However, if a page requested by the virtual machine is missing from the delta disk, it is retrieved from the parent disk. When a virtual machine writes to the delta disk, the delta disk stores the pages like a normal virtual disk. As delta disks do not contain their parent's data, they begin very small. Since the parent disk is set to read-only, multiple delta disks can reference and utilize the same parent disk without causing corruption.

2.1.9 Virtual Machine Splitting using Delta Disks

The process of virtual machine splitting using delta disks is identical to regular virtual machine splitting, except after powering off or suspending a virtual machine, a delta disk is created and instead of copying the virtual disk, the delta disk is copied. When a delta disk is created its total size is a fraction of a megabyte. The virtual machine associated with this delta disk has its configuration file modified, so that it points to the delta disk instead of its original hard disk. Since we will be copying delta disks to potentially different datastores and directories, we need to implement a feature to read and modify the delta disk's reference to its parent disk. As the reference to a parent disk is statically assigned, it will no longer be possible to perform a datastore migration of the original virtual machine, without powering off all of the children virtual machines and updating their delta disk references. The children virtual machines can still be migrated to any datastore and any host, as long as the host has access to the datastore with the parent virtual disk. With this modified version of virtual machine splitting, less than 1 MB of data needs to be copied for stateless virtual machines. This makes it possible to perform all of the copy functions of stateless virtual machine splitting in less than one second.

2.1.10 Memory-Bound Splitting Limitations

Stateful virtual machine splitting requires maintaining a copy of a virtual machine's memory state for each of the children virtual machines. Virtual machine splitting currently implements this by having the hypervisor write the contents of a virtual machine's memory to disk and then copying it to all of the children virtual machines. When a child virtual machine is registered on a host, the hypervisor sees that a saved state is associated with it. The hypervisor then enables the ability to resume the virtual machine, which loads the saved state into memory.

2.2 Virtual Machine Splitting with Live Memory Copying

Most hypervisors have the ability to copy the contents of a virtual machine's memory when it migrates from one host to another host without powering off the machine. We will refer to this function as live memory copying. As seen in Table 2.3 writing the state to disk and copying state are very time intensive operations compared to live memory copying. Therefore, it is necessary to leverage this feature to make stateful virtual machine splitting more effective.

Table 2.3: Stateful Virtual Machine Splitting Time For Single Split

RAM	Suspend (s)	Copy State (s)	Live State Copying (s)
2 GB	108	630	5
4 GB	218	1260	8
8 GB	461	2520	11
16 GB	690	5040	17

The need for live state copying becomes more apparent when more than one virtual machine split occurs. The linearly increasing cost of splitting a virtual machine with 2 GB of RAM seen in Table 2.4 results from having to copy the state from disk multiple times. The live state copying, on the other hand, has a constant completion time because it utilizes multicast to send a copy of the entire state to multiple destination hosts simultaneously.

Table 2.4: Cost of Virtual Machine Splitting as Number of Splits Increases

VM Splits	Suspend (s)	Copy State (s)	Live State Copying (s)
1	108	630	5
2	108	1260	5
4	108	2520	5
8	108	5040	5

2.3 Virtual Machine Splitting Granularity

When splitting a virtual machine, there are two levels of granularity to consider. At the lowest level is resource granularity. At the highest level is workload reconfiguration granularity.

2.3.1 Resource Granularity

Resource Granularity is used to describe how many virtual resources are provided to each child virtual machine when a virtual machine split occurs. This type of granularity is restricted by the type of virtual machine splitting and the resource that is being split. For instance, when doing a stateless virtual machine split, each child can have fewer resources allocated to it than its parent virtual machine. However, for stateful virtual machine splits, each child must have the same number of resources as its parent virtual machine. The earlier property can be exploited to improve physical resource utilization; the later property can be used to scale up applications over multiple physical hosts.

Table 2.5: Supported Resource Granularity for Stateful and Stateless Splits

Resource Granularity		
	Stateful Split	Stateless Split
Fewer Resources	×	✓
Equal Resources	✓	✓
More Resources	×	✓

2.3.2 Workload Reconfiguration Granularity

Workload Reconfiguration Granularity is used to describe at what level processes running on a virtual machine can be split. This type of granularity depends on the type of applications being run on the virtual machine. A virtual machine with tightly coupled processes that cannot communicate over the network cannot be split. However processes that are not coupled or can communicate over the network can be split into multiple virtual machines. To make process splitting easier, process groups can be defined to ensure that applications do not break when the virtual machine they are running on is split.

3. Virtual Machine Merging

As virtual machine workload can vary overtime, virtual machine managers have the ability to consolidate virtual machines onto fewer hosts. Although this is the ideal behavior for virtual machines with specific workloads, it is not ideal when there are many splits of the same virtual machine. To solve consolidation of nearly identical machines, it is necessary to implement a method to merge virtual machines back together.

3.1 Divergence and Redundancy

Once a virtual machine has been split using the method mentioned in Chapter 2, its children virtual machines' state and disk diverge. Different sets of processes are running on each child and different writes have been done to the delta disks. Being able to extract the appropriate differences and consolidating them back into a single virtual machine is very difficult.

3.2 Workarounds for Disk Divergence and Redundancy

The most basic way of preventing disk divergence and redundancy is by creating a partition between the operating system disk and process disks. When a merge occurs, only one copy of the operating system disk must be maintained. As each virtual machine will contain the same process disks, only the one running the process associated with the disk must be maintained.

3.2.1 Process Specific Virtual Hard Disks

If every process group on a virtual machine was given its own virtual hard disk for storing data pertaining to it, merging virtual machine data would be relatively easy. For each merge, the hard disk relevant to the running processes would be disconnected from the virtual machine and connected to the merged virtual machine. Then the process will be restarted on the merged virtual machine. The other virtual machine and its files will be deleted. This type of merging works if the processes on the two virtual machines to be merged do not rely on the same virtual disk.

3.2.2 Storage Area Network Disks

To eliminate the need to merge virtual disks onto a single virtual machine, storage area network (SAN) disks can be used. These SAN disks utilizing iSCSI or similar technologies can be mounted and unmounted from virtual machines to act like locally attached disks. By using SAN disks with iSCSI, applications running on any virtual machine will be able to access all of the data once the disk is mounted. During a merge, these SAN disks will be unmounted from the source virtual machines and mounted onto the destination virtual machine. The processes on the source virtual machines will be started on the destination virtual machine. Then source virtual machines and their files will be deleted.

3.3 Resource Merging

As processes on virtual machines can be split onto multiple virtual machines running on fewer virtual resources, it is obvious that the reverse process must also exist. Currently we are able to merge virtual hard disks or SAN disks onto a single virtual machine; to make merging more complete, we need to be able to merge virtual machine resources. We can leverage Hot-Add Memory and Hot-Plug CPU to merge these virtual machines resources.

3.3.1 Hot-Add Memory and Hot-Plug CPU

Hot-Add Memory and Hot-Plug CPU are features that enable adding and removing hardware from a physical machine [9] [10]. These features have also been implemented on virtual machine hypervisors allowing administrators to allocate additional resources to a virtual machine on demand. Hot-Add Memory and Hot-Plug CPU are not supported by all hypervisors and guest operating systems. Some guest operating systems support adding resources but not removing them. When a virtual machine merge has completed, the source virtual machine is powered off, then Hot-Add Memory and Hot-Plug CPU are used to allocate more resources to the merged virtual machine. Hot-Add Memory and Hot-Plug CPU can be used before or after resuming the processes that were running on the source virtual machine. However, it is not recommended to use Hot-Add Memory and Hot-Plug CPU after resuming the processes if the merged virtual machine is running at capacity.

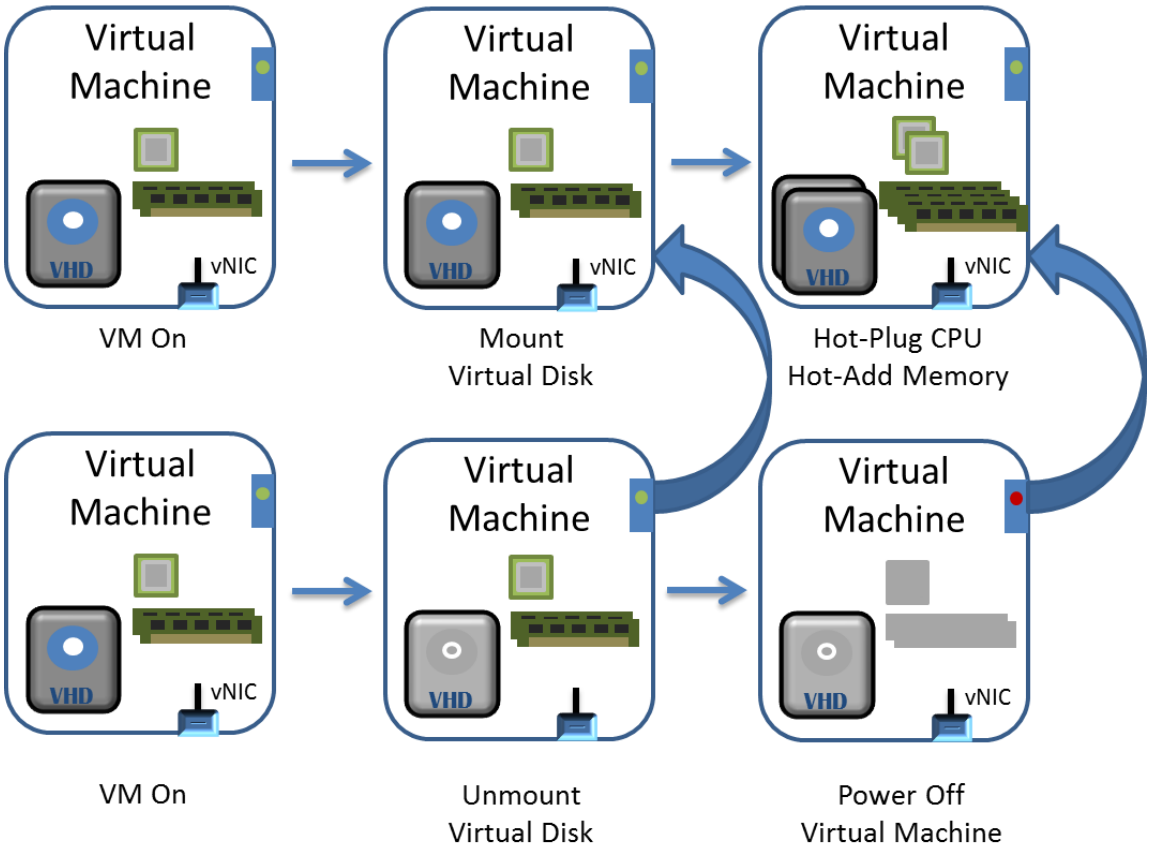

Figure 3.1: Virtual Machine Merging

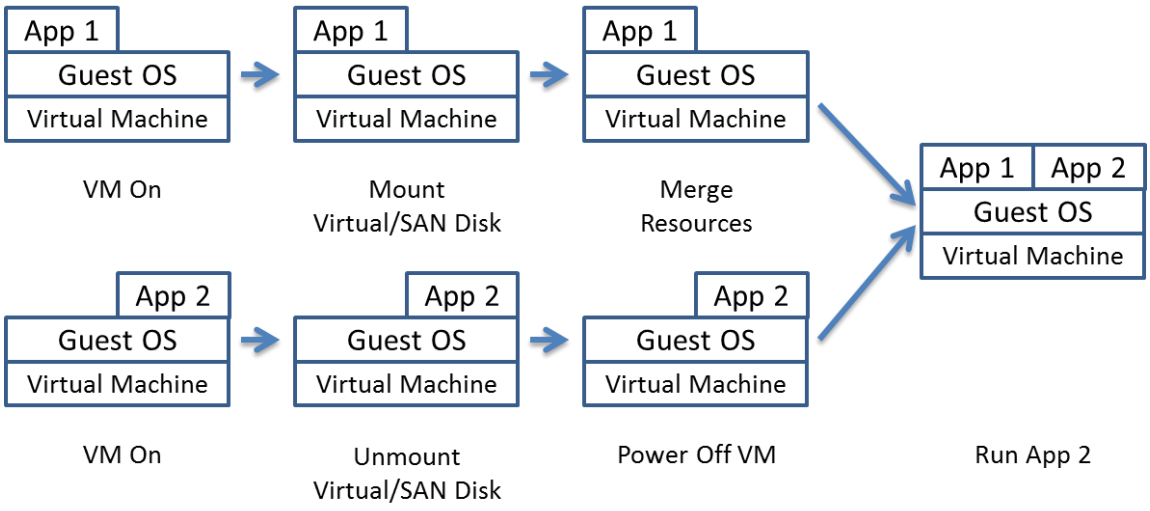

Figure 3.2: Virtual Machine Merging at the Application Layer

3.4 Benchmarking Experiments

Similarly to virtual machine splitting, it is complicated to benchmark virtual machine merging. Performance of virtual machine merging will be defined by the time required to complete a merge operation.

3.4.1 Process Specific Virtual Disks

In this experiment, a virtual disk will be unmounted from the source virtual machine, removed from the source virtual machine, added to the destination virtual machine and mounted to the same mount point. Figure 3.3 shows that the time required to unmount virtual hard disks increases as the number of virtual disks increase. Figure 3.4 shows that the time required to remove a virtual hard disk from a virtual machine increases as the number of virtual hard disks increase. Figure 3.5 shows that the time to add a virtual hard disk increases as the number of virtual hard disks increase. Figure 3.6 shows that the time required to mount virtual disks increases as the number of virtual hard disks increases. These results indicate that performance will be significantly impacted by the adding and removing virtual disks more than by mounting and unmounting their partitions.

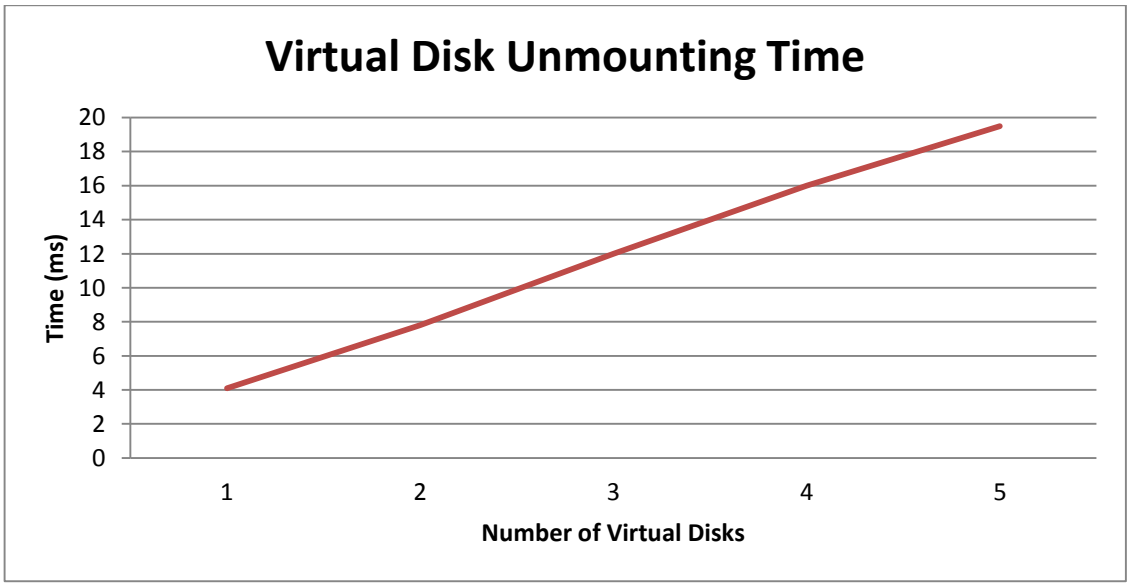

Figure 3.3: Virtual Disk Unmounting Time

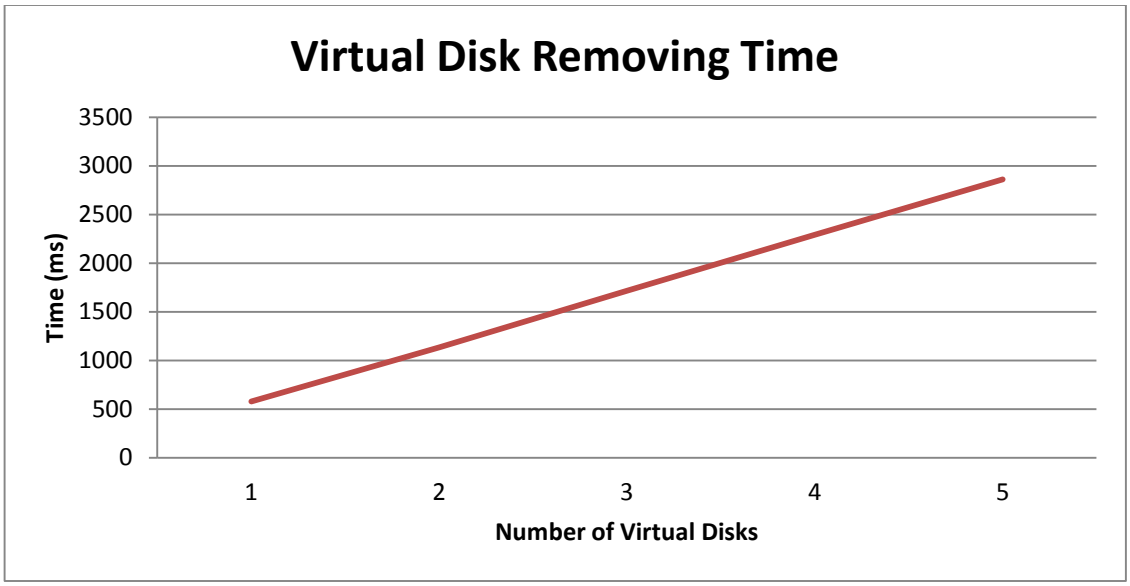

Figure 3.4: Virtual Disk Removing Time

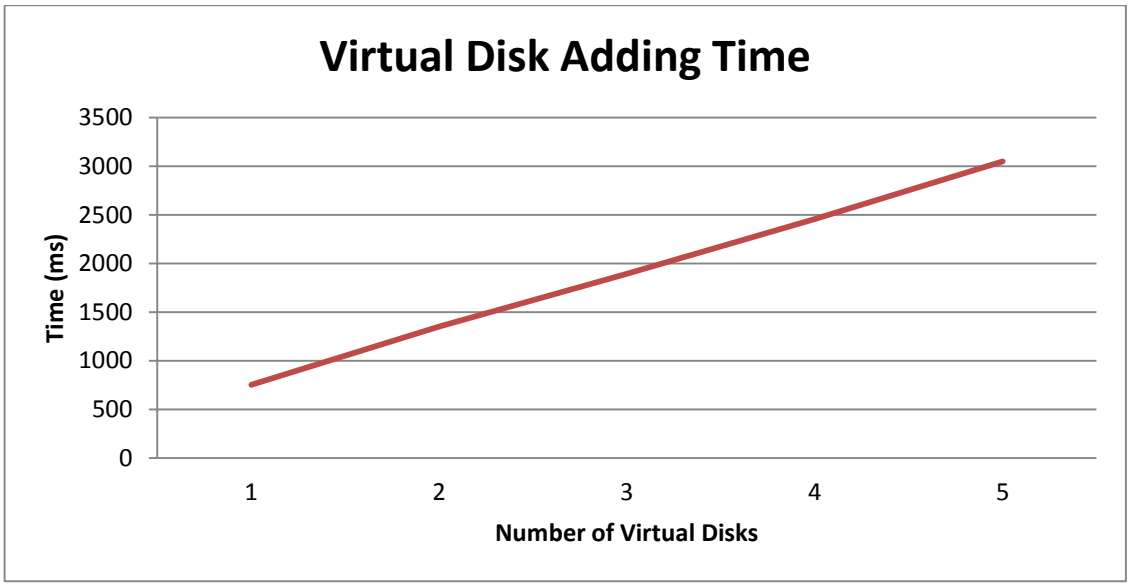

Figure 3.5: Virtual Disk Adding Time

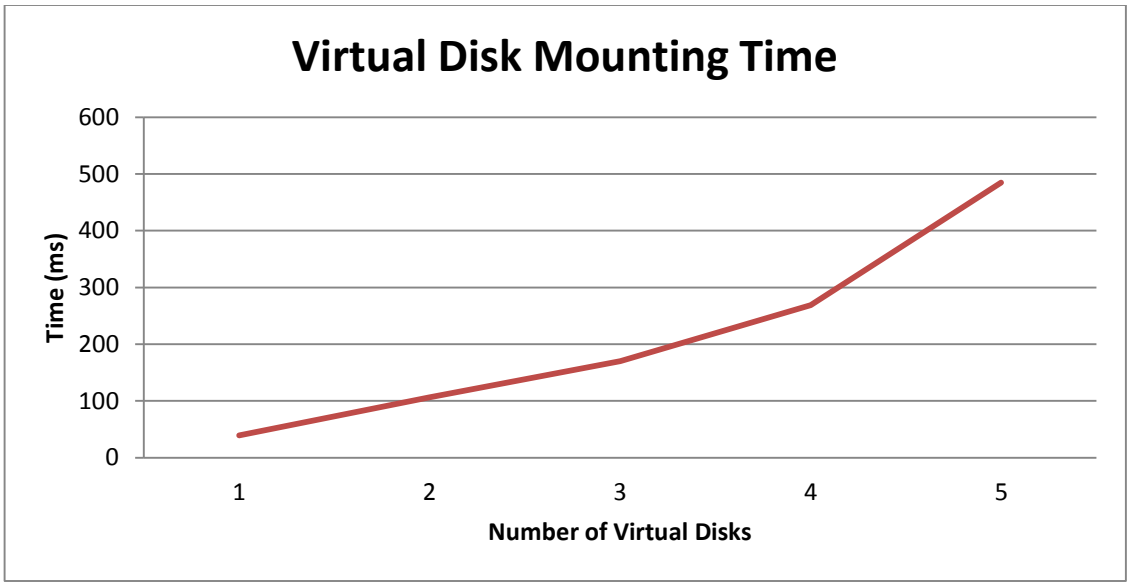

Figure 3.6: Virtual Disk Mounting Time

3.4.2 Process Specific SAN Disks

In this experiment, a SAN disk will be unmounted from the source virtual machine and mounted to the same mount point on the destination virtual machine. Figure 3.7 shows that the time required to unmount SAN disks increases as the number of SAN disks increase. Figure 3.8 shows that the time required to mount SAN disks increases as the number of SAN disks increases.

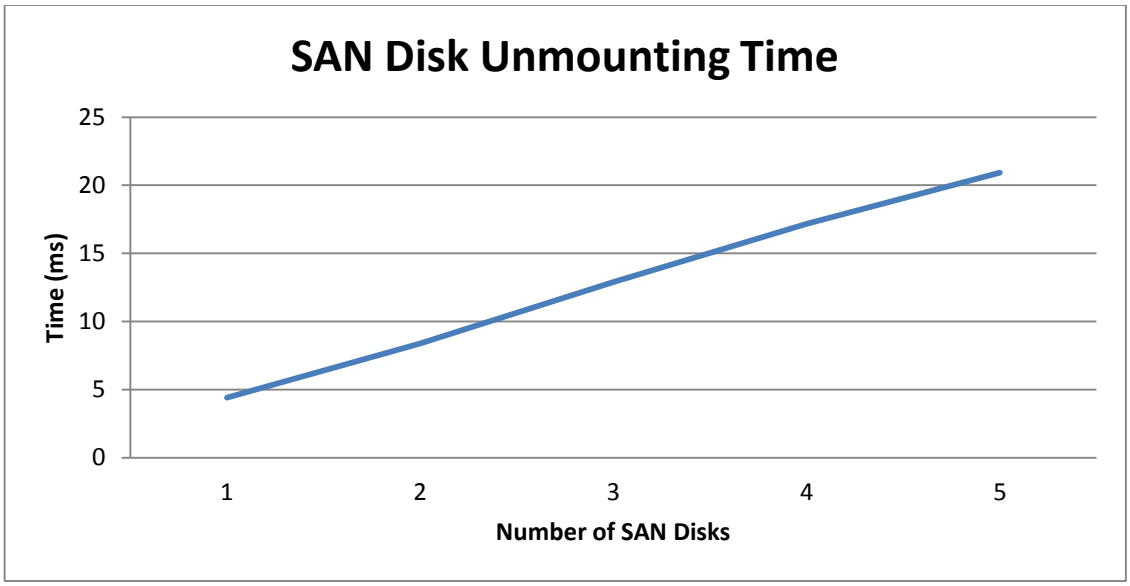

Figure 3.7: SAN Disk Unmounting Time

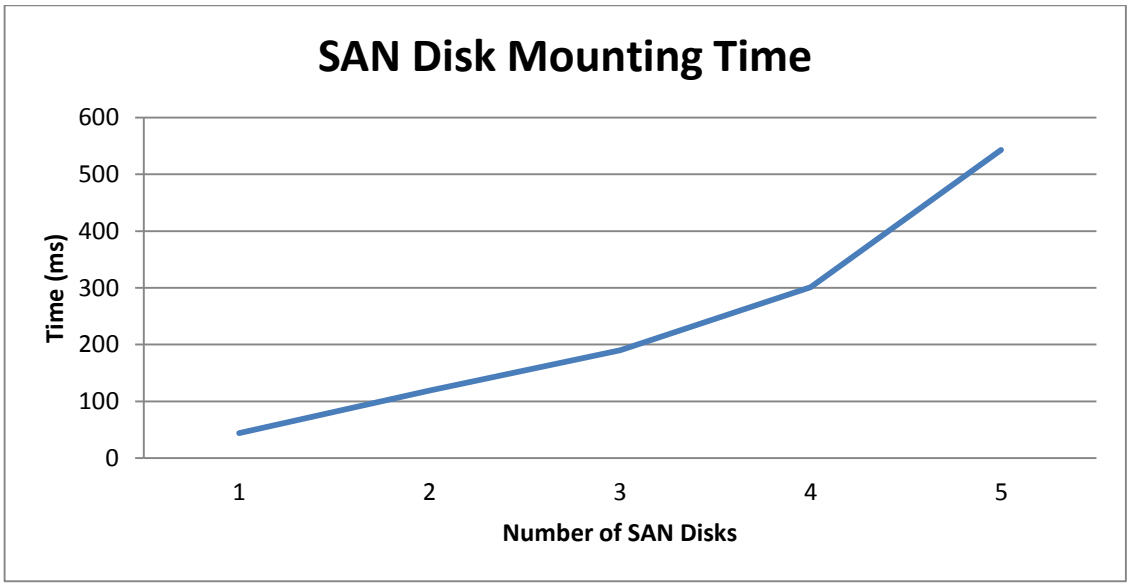

Figure 3.8: San Disk Mounting Time

3.4.3 Hot-Add Memory and Hot-Plug CPU

In this experiment, memory and vCPUs will be added to a virtual machine. Different quantities of vCPUs and memory will be added to a virtual machine starting with 1 vCPU and 2

GB of memory. Figure 3.9 shows that the hot-plug CPU operation runs at a constant time of approximately 2.588 seconds, regardless of the number of CPUs added. Figure 3.10 shows that the hot-add memory operation runs at a constant time of approximately 3.110 seconds, regardless of the amount of memory added.

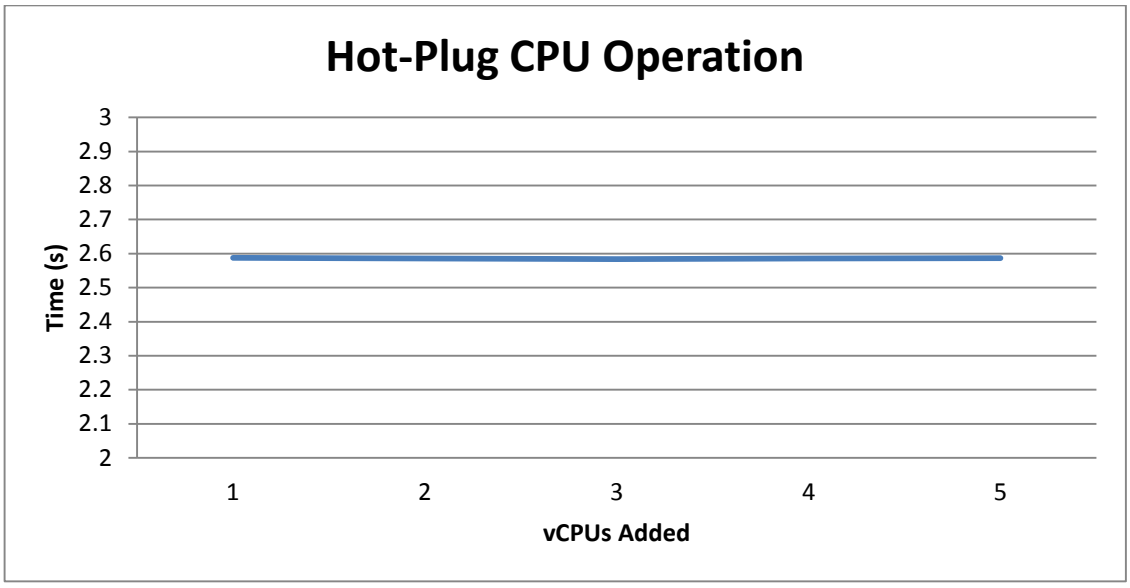

Figure 3.9: Hot-Plug CPU Operation

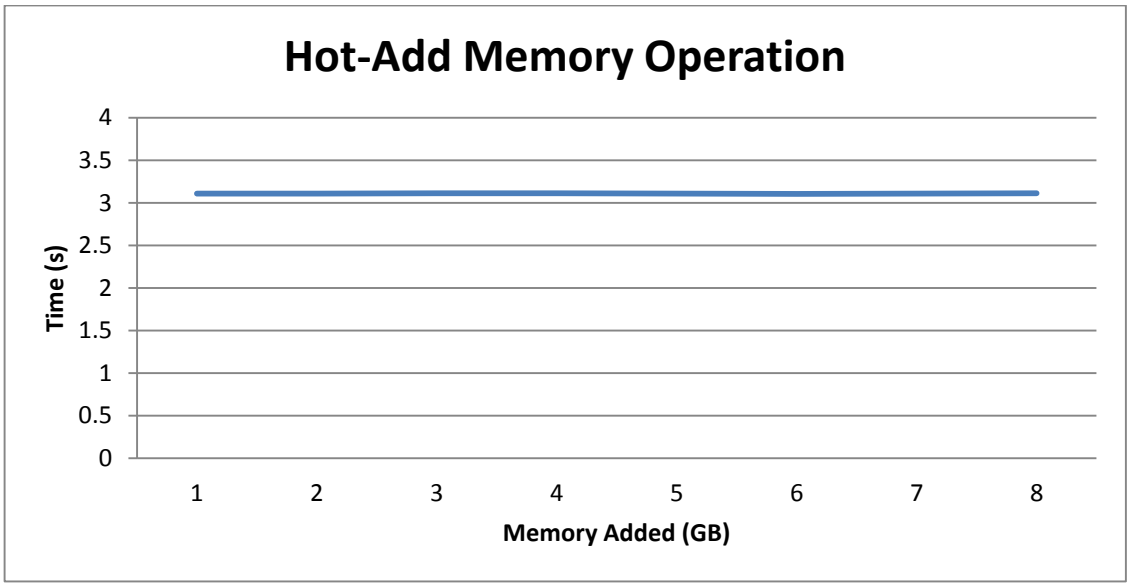

Figure 3.10: Hot-Add Memory Operation

3.5 Faults in Merging

Virtual machine merging is not the opposite function of virtual machine splitting. When a virtual machine is split, its state can be maintained. When children virtual machines are merged, the state from the children virtual machines cannot be merged back together. Therefore, the method of virtual machine merging, defined in this chapter, can only be used statelessly.

4. Comparison to other Malleability Mechanisms

4.1 Background

Traditionally, to scale an application, its environment and its state need to be reproduced on its destination. The environment, consists of all resources the application has access to and the state is the virtual memory assigned to the application. To scale an application, first it is suspended, its state is sent to the destination virtual machine and its environment is rebuilt, finally the application is resumed. This process is very similar to that of virtual machine migration [11] [12] [13].

Malleability enables an application to modify its running processes. By leveraging this malleability functionality it is possible to scale applications to attain good performance at the lowest possible cost [14]. Maghraoui, Desell, Szymanski and Varela [15] introduce a form of malleability which modifies parallel processes granularity using split and merge functions. Their implementation enables applications to adapt to the fluctuating nature of resource availability in shared environments. This form of malleability requires middleware to monitor and modify the processes as application demand changes.

Osman, Subhraveti, Su and Nieh [16] introduce a method of enabling application migration between similar host operating systems. They implemented a virtualization layer on top of a host operating system which provides access to system resources, they called this Zap. Then they created pods, processes groupings which reside on top of Zap. When a pod is migrated from one Zap host to another, Zap remaps the resources to the Pod on the new Zap host, making the migration transparent to the running applications. Their work on Zap enables any application to become part of a malleable process group without requiring any changes to an existing application. The main pitfall of this approach is that workload reconfiguration granularity of virtual machine malleability is limited to predefined pods. In order to achieve high workload reconfiguration granularity, every pod will only contain one process or multiple processes that are dependent on each other.

4.2 Problem

While an application is running, the performance can fluctuate depending on demand. If the demand of an application is very high, the performance of the application decreases.

Similarly when the demand of an application is very low, the performance of the application increases. Application malleability enables applications to dynamically scale onto more resources as demand increases to maintain acceptable performance.

There exist two types of applications, those written with malleability in mind (Type I) and those that are not (Type II). For the earlier type, applications can migrate their processes to other systems directly, and complexities such as communication are handled by the malleability layer. For the latter type, applications are run in a virtual environment, which hides the migration process from the applications.

4.2.1 Type I Malleability

As Type I applications are written with malleability in mind, they are able to achieve most, if not all of the benefits of application malleability. These benefits include dynamic workload reconfiguration and transparency [17]. Dynamic workload reconfiguration is the ability to control the size of the workload running on any particular machine. Transparency is the ability for an application in a workload to scale directed by an external agent. Desell, Maghraoui and Varela [18] demonstrate Type I malleability for applications written in the SALSA programming language [19]. Figure 4.1 is based on work from [15] and [18] and Figure 4.2 is based on work from [20].

4.2.2 Type II Malleability

As Type II applications are not written with malleability in mind, therefore they do not inherently gain all of the benefits of application malleability. For example and application running on Zap in a Pod has limits on its granularity [16]. A Pod is a process group, which cannot be sub-divided, therefore limiting workload granularity. Similarly, a Pod which migrates from one Zap host to another Zap host recreates its virtual environment, so the application can continue running. However, the application is unaware that it has been migrated to a different host. Unlike Type I malleability that transparently handles dependencies, Type II malleability requires application dependencies to exist in the same Pod, otherwise the dependency will not be available for the application. A lack of workload granularity prevents dynamic workload reconfiguration, which can be a huge problem. If an application running in a Pod has very high demand, the only operation that can be performed is to migrate the Pod to a different Zap host. If this Zap host does not have sufficient resources to fulfill demand, then performance will

be impacted. Likewise lack of transparency prevents applications from being able to scale without having built in malleability and scalability.

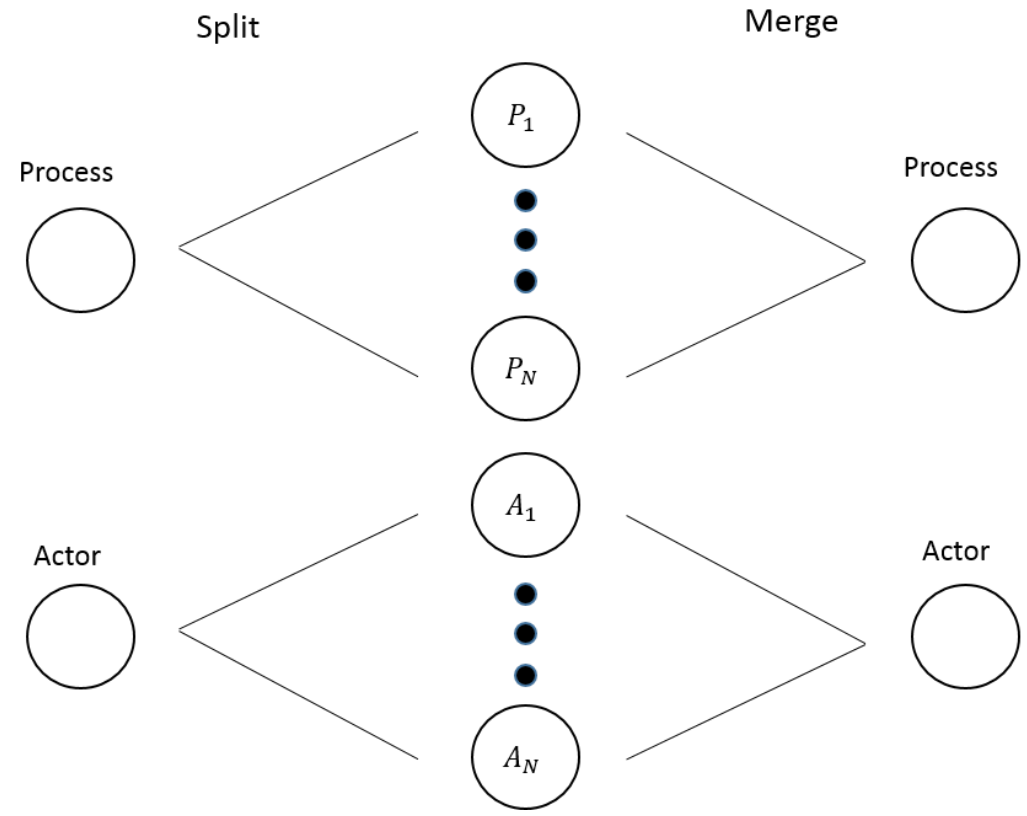

Figure 4.1: Application Malleability

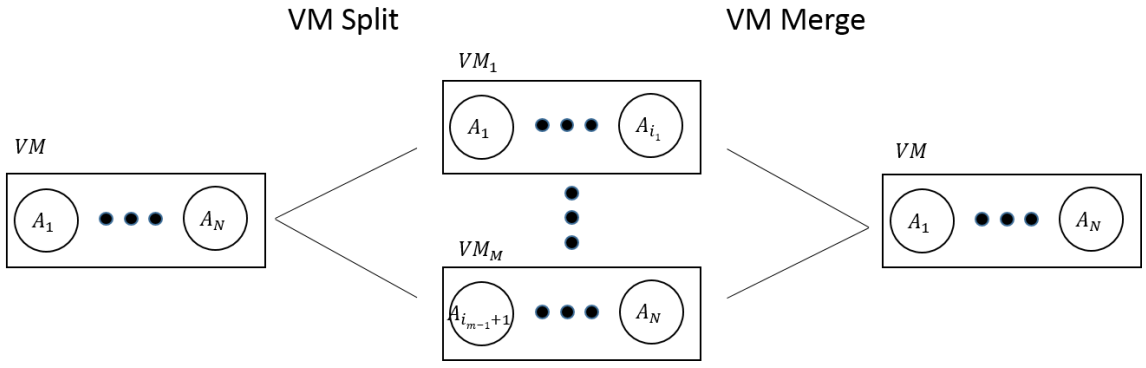

Figure 4.2: Virtual Machine Malleability using Application-level Migration

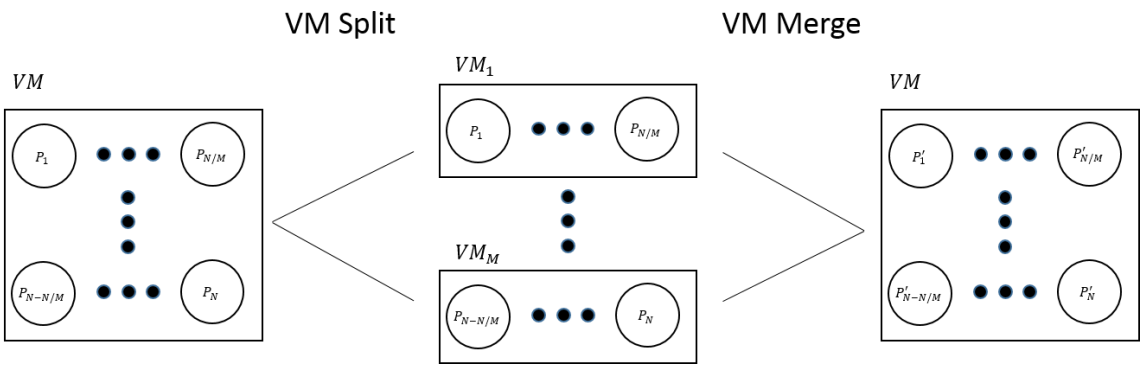

Figure 4.3: Virtual Machine Malleability for Workloads with Independent Process Group

4.3 Virtual Machine Malleability

By running virtual machine splitting and merging, we have created a solution that consists of a combination of Type I malleability and Type II malleability benefits. The benefits we have attained include dynamic workload reconfiguration and transparency. Transparency being the ability to not have to write the application with malleability in mind.

4.3.1 Dynamic Workload Reconfiguration

Dynamic workload reconfigurability is defined as the ability to increase or decrease the number of independent processes running on a virtual machine. This means that when a virtual machine is split, it is possible to split the workloads of a virtual machine between any number of children virtual machines, as long as no split separates process groups. Similarly for merging, any number of processes can be consolidated back into a single virtual machine.

4.3.2 Transparency

Transparency is defined as the ability for an application in a workload to scale directed by an external agent. This means that when an application is split, its components are unaware of the split.

4.3.3 Rewriting Applications

When a virtual machine split or merge occurs, applications can be split up or consolidated without having to be written with malleability in mind. This is incredibly important as many legacy applications do not implement any level of malleability.

5. Applications of Virtual Machine Splitting and Merging

Using the implemented and conceptual ideas mentioned in Chapters 2 and 3, it is possible to use virtual machine splitting and merging for multiple applications. Splitting can be used to scale up applications, optimize throughput and performance and increase running priority to a subset of processes. Merging can be used to scale down applications and combine virtual resources into fewer virtual machines, improving on energy and resource utilization.

5.1 Application Scaling Up

Cloud computing has opened doors allowing anyone with a budget to run distributed applications across hundreds and thousands of physical resources. Depending on budget and time constraints, distributed applications can be scaled significantly to run an application faster. Cloud computing has also made it economically feasible to run such jobs, as one does not need to purchase hundreds or thousands of physical machines to reduce distributed application run time.

Once an application is written and tested on a small set of machines, it can be scaled up to use more cloud resources using virtual machine splitting. Consider an instance of an application that consists of a series of 10 processes. Each of these proceses have a runtime of 1 unit. Assuming that a single virtual machine could process 1 unit per hour, it would take 10 hours for this virtual machine to run the entire application. Using the virtual machine splitting implementation described in 2.2 it is possible to split this virtual machine into 10 children virtual machines with equal resources and $1/10^{th}$ of the workload, all being distributed across 10 physical hosts. As each of these children virtual machines will receive 1 process with a runtime of 1 unit and as these children virtual machines have the same virtual resources as their parent, they too can process at a rate of 1 unit per hour. Therefore, by using virtual machine splitting, it is possible to process the entire workload in 1 hour, by splitting it amongst 10 children virtual machines each doing $1/10^{th}$ of the workload. Since cost is the same, this feature of cloud computing is called *cost associativity* [21].

5.1.1 Measuring the Cost of Virtual Machine Splitting

Each time a virtual machine splits, a certain amount of virtual machine data needs to be copied. This affects the speed of virtual machine splitting significantly. The two significant

variables that impact the splitting process include the size of the virtual machine files to be copied and the number of copies that are made. As we can see in Figure 5.1 the cost of application scaling using virtual machine splitting with delta disks is very high. Both copying files and creating the delta disk are constant and take approximately 4 seconds each. However, suspending a virtual machine and copying the state from disk from the parent to its children significantly increases as size of memory increases.

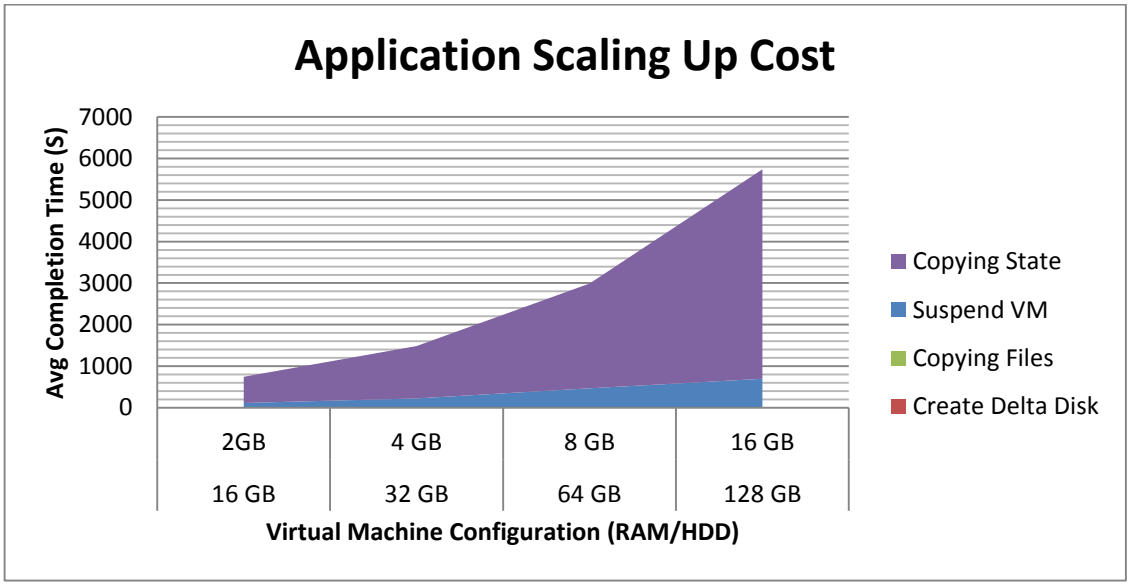

Figure 5.1: Application Scaling Up Cost Utilizing Disk Based State Copying

5.1.2 Measuring the Cost of Virtual Machine Splitting with Live Memory Copy

Similarly to application scaling using virtual machine splitting, the normal process of splitting occurs, except instead of suspending the virtual machine and copying its state, the live memory copy version makes a copy directly to the destination host of each child virtual machine. To emulate this, we will utilize virtual machine live migration to create an upper bound on the process of live memory copy time. Virtual machine migration uses unicast to send a copy of the virtual machine state to the destination host. For the purpose of application scaling using virtual machine splitting, it is recommended that multicast be used, as multiple hosts will be receiving identical copies of the virtual machine state.

As we can see in Figure 5.2, virtual machine splitting using live state migration is less time intensive than the regular state migration version. Live migration of state took 5 seconds for a single virtual machine split with 2 GB of RAM and took 17 seconds for a single virtual machine split with 16 GB of RAM. If implemented with multicast instead of unicast, the live state migration time will be an upper bound, as the data is sent once to all hosts. Furthermore, if copying files and creating delta disks were considered to be one process that took 8 seconds and live state migration were a separate process, then virtual machine splitting can be done in parallel as the two processes utilize different components. The former utilizes disk whereas the latter utilizes memory. Therefore, the total virtual machine splitting time will be bound by whichever process has a longer completion time.

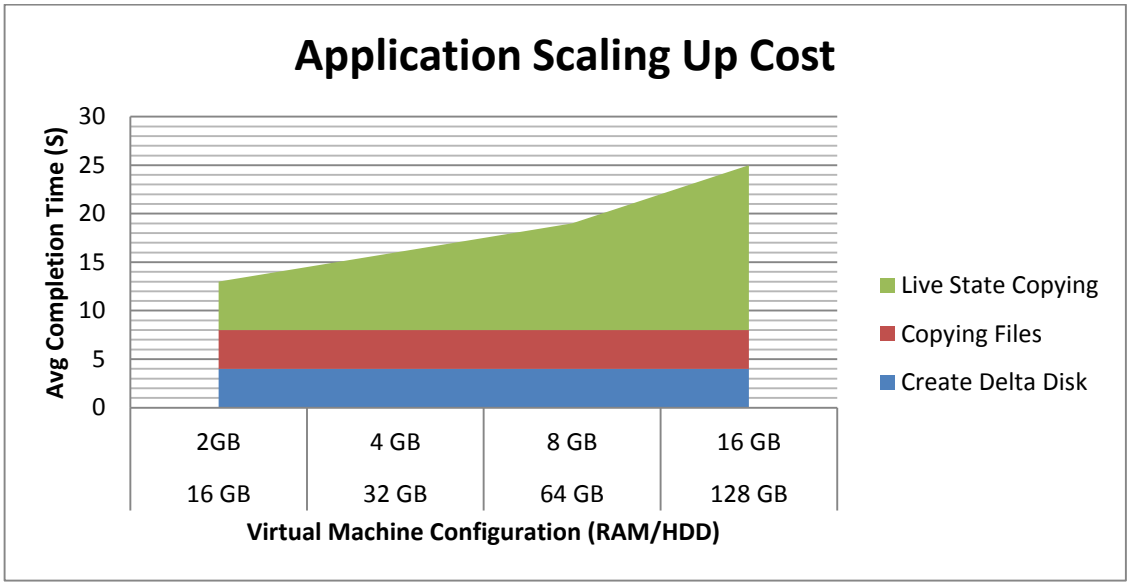

Figure 5.2: Application Scaling Up Cost Utilizing Live State Copying

5.1.3 Measuring Performance of Application Scaling Up

Each time an application is scaled up using virtual machine splitting, its performance should increase linearly to the total number of virtual machines. To ensure that this is the case, we will test workloads of 10, 20, 30 and 40 embarrassingly parallel processes. We chose embarrassingly parallel processes because they do not interact with each other. Otherwise, factors like networking configuration would impact the performance as the application is scaled

up. These processes will be run on 1 virtual machine and scaled up to run on 2, 5, 10, 20, 30 and 40 virtual machines. When scaling up, the processes are divided equally between all available virtual machines. In the case that there are more virtual machines than processes, some virtual machines will not run any processes, nor will they be included in the performance calculation.

As we can see in Figure 5.3, completion time of a particular workload deceases as the number of virtual machine splits increases. Once the total number of virtual machine splits is greater than or equal to the number of processes, the completion time becomes constant. Furthermore, the completion time of one virtual machine running N processes is equivalent to the sum of completion times of N virtual machines running 1 process. In other words, the completion time of N embarrassingly parallel processes split amongst M virtual machines can be calculated by taking the completion time of 1 virtual machine running N processes and dividing it by M.

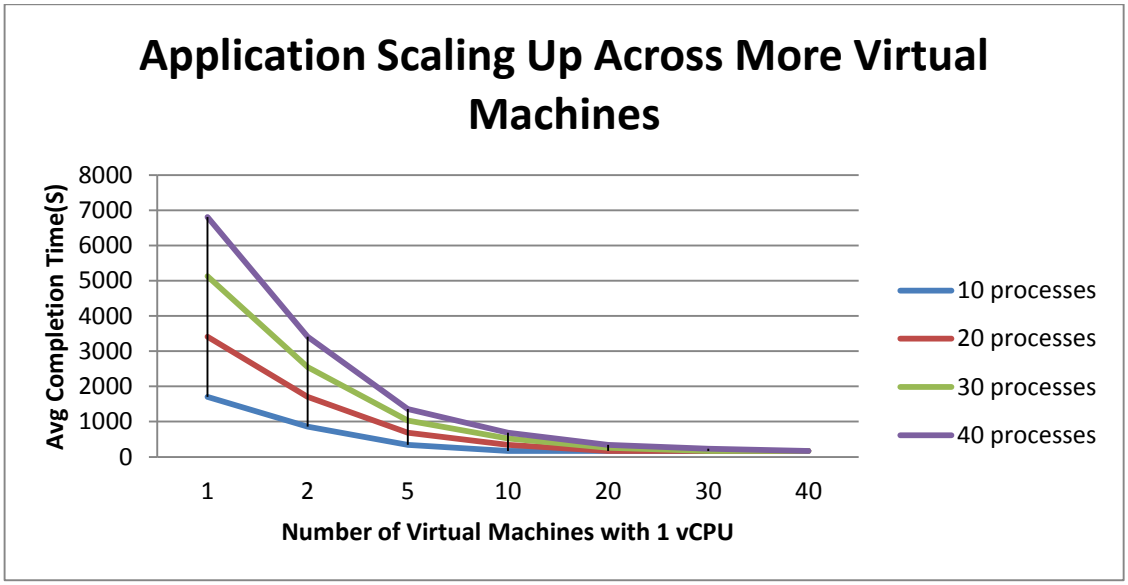

Figure 5.3: Application Performance When Scaling Up across More Virtual Machines on Hosts with 1:1 vCPU to Physical CPU Ratio

5.1.4 Leveraging Virtual Machine Splitting using SALSA

SALSA is an actor-oriented programming language. Actors in SALSA are portable so that they can be migrated from a theater running on one virtual machine to a theater running on

another virtual machine. This makes SALSA a very good programming language to develop distributed applications. Utilizing virtual machine splitting, it is possible to configure a virtual machine with a theater and split it hundreds of times, until every physical host has M virtual machines, where M is equal to the total number of physical CPUs in a host. Once these children virtual machines are running, SALSA can begin migrating actors onto these newly available resources.

5.1.5 Performance of Distributed Heat Application

As we can see in Figure 5.4, as the number of virtual machines increases, the overall completion time of the distributed heat application decreases. There is a slight overhead as doubling the number of virtual machines does not yield a 100% increase in performance. This slight overhead is to be expected as any non-embarrassingly parallel application relies on a network to send data between each of its processes. In Figure 5.5 we can see that as the number of virtual machines increases, the rate of execution increases approximately linearly.

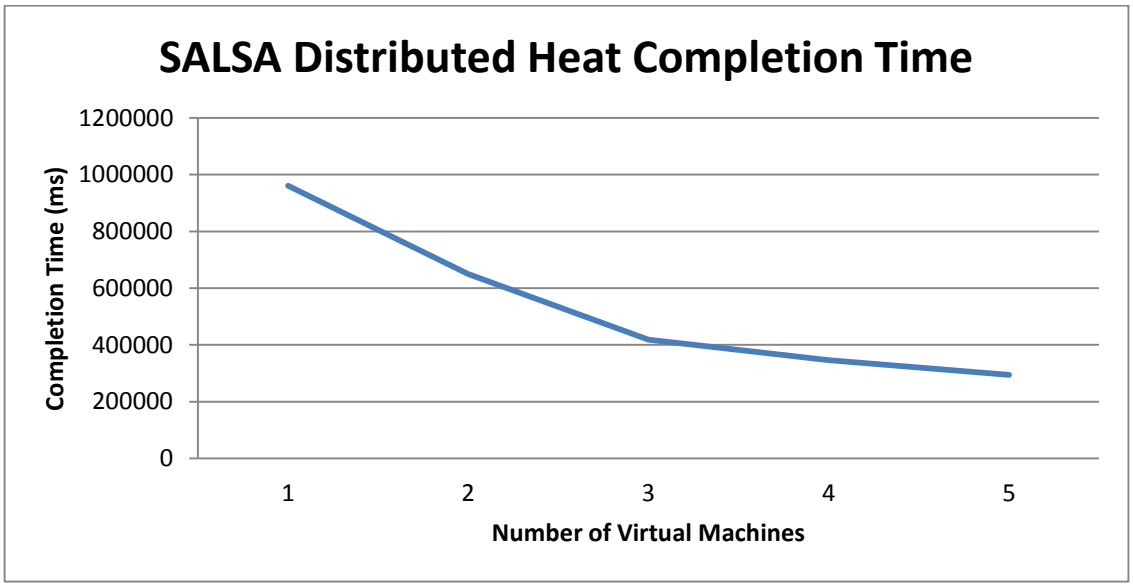

Figure 5.4: SALSA Distributed Heat Completion Time

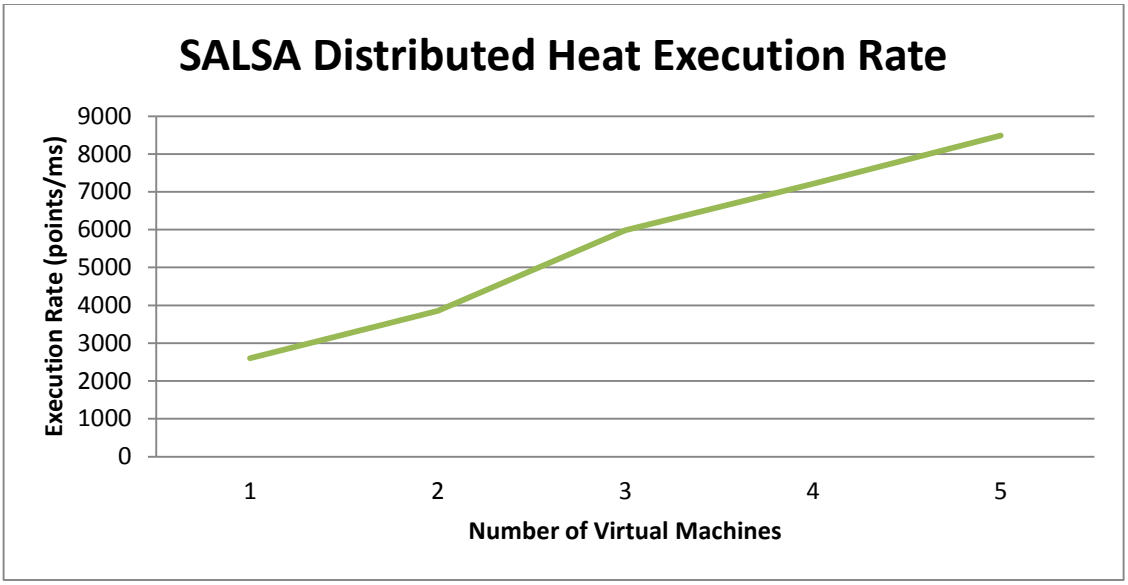

Figure 5.5: SALSA Distributed Heat Execution Rate

5.2 Application Scaling Down

Web servers and database servers have significantly increased in popularity since the dot-com boom in the late 1990s. Typically when a web application is built the webserver and database server reside on the same system. Although this is acceptable for small applications, there is always the possibility that the application might experience responsiveness issues during its peak usage hours. To solve this issue, application scaling using virtual machine splitting can be used. However, during the non-peak hours, this application might be allocated too many physical resources making it very costly to maintain. Therefore, this application needs to be able to scale down during its idle hours. To that end virtual machine merging can be used.

5.2.1 Virtual Machine Merging with Virtual Hard Disks

To merge a virtual machine with stateless processes and dedicated virtual hard disks is relatively simple. First the processes running on the source virtual machine are terminated and their respective virtual hard disks are unmounted. Each of the virtual hard disks relevant to processes are removed from the source virtual machine and added to the destination virtual machine using hot-swap disk, the ability to add and remove disks while a machine is running. The guest operating system on the destination virtual machine then rescans its SCSI bus for new devices. Finally, the virtual hard disks are mounted to the same mount points as the source

virtual machine, thus allowing the applications to be started on the destination and reload and serve any data from their respective disks.

5.2.2 Virtual Machine Merging with Network Disks

To merge a virtual machine with stateless processes and network disks is very similar to merging with virtual hard disks. The main difference is that the process is shorter as network disks only need to be unmounted from the source virtual machine and mounted to the destination virtual machine. Once mounted, the processes can be started on the destination virtual machine.

5.2.3 Measuring the Cost of Virtual Machine Merging

Each time virtual machines merge, a few operations may occur. Including Hot-Plug CPU, Hot-Add memory, unmounting and mounting virtual disks or SAN disks and removing and adding virtual disks. As discussed in Section 3.4, Hot-Add memory and Hot-Plug CPU are constant time operations. Unmounting and mounting virtual disks or SAN disk and removing and adding virtual disks are linear time operations. As we can see in Figure 5.6, the cost of application scaling using virtual machine splitting is most affected by the adding and removing virtual disk operations, which grow linearly as the number of virtual disks increases.

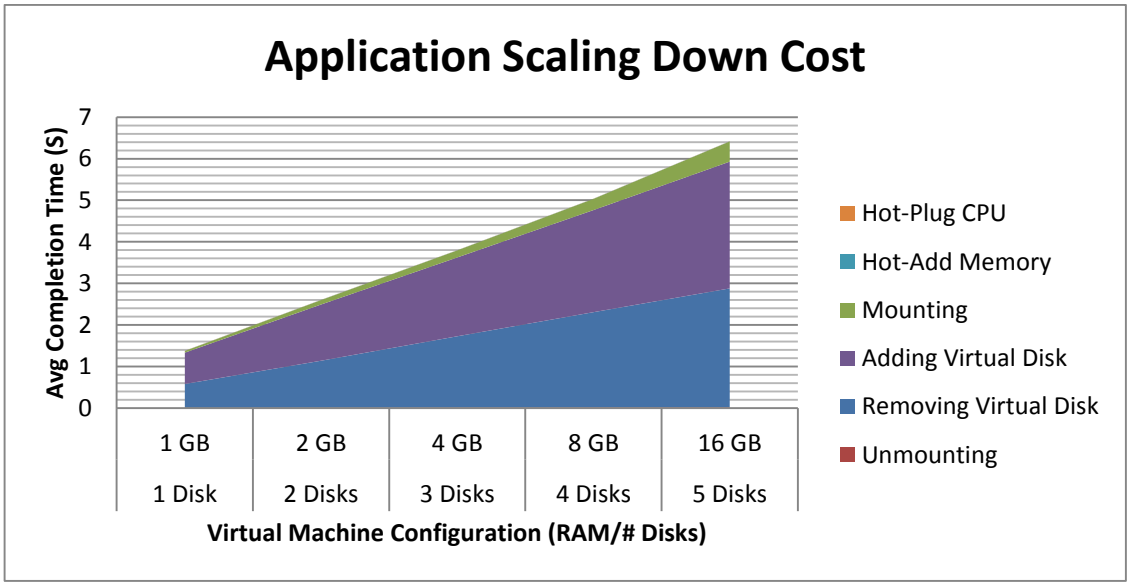

Figure 5.6: Application Scaling Down Cost

5.2.4 Measuring Performance of Application Scaling Down

Each time an application is scaled down using virtual machine merging, its performance should decrease linearly to the total number of virtual machines assuming a 1:1 vCPU to physical CPU ratio. To ensure that this is the case, we will test workloads of 10, 20, 30 and 40 embarrassingly parallel processes for the same reasons as in Section 5.1.3. These processes will be run on 40 virtual machines and scaled down to run on 30, 20, 10, 5, 2 and 1 virtual machines. When scaling down, the processes are divided equally between all available virtual machines. In the case that there are more virtual machines than processes, some virtual machines will not run any processes, nor will they be included in the performance calculation. As we can see in Figure 5.7, completion time of a particular workload increases as the number of virtual machine merges increases. Similarly to Section 5.1.3, the completion time of one virtual machine running N processes is equivalent to the sum of completion times of N virtual machine running 1 process.

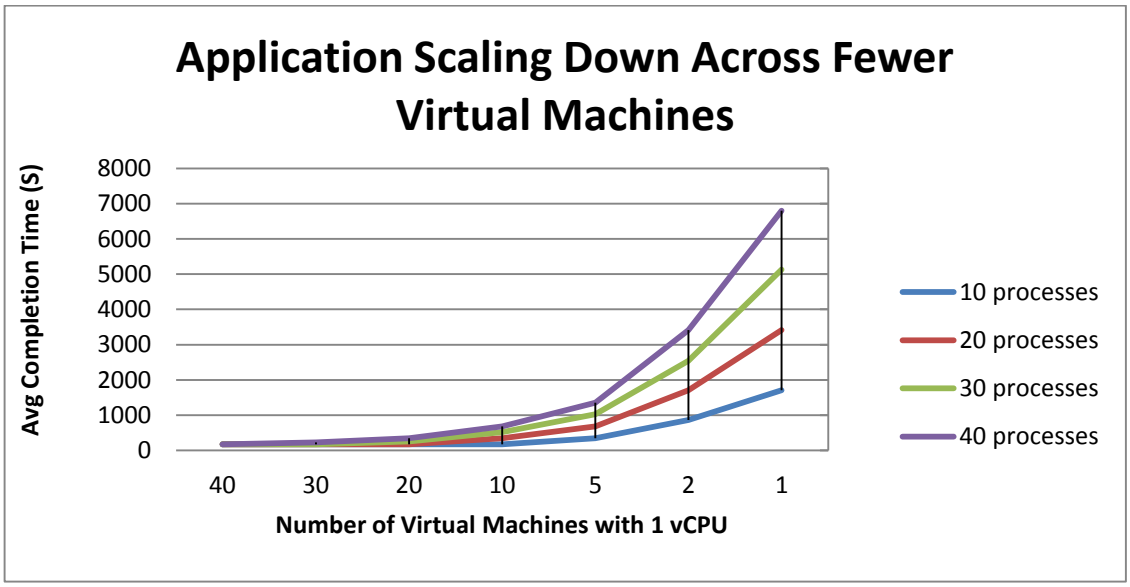

Figure 5.7: Application Performance When Scaling Down across Fewer Virtual Machines on Hosts with 1:1 vCPU to Physical CPU Ratio

5.3 Energy Optimization Problem

One of the main benefits of virtualization is to reduce energy consumption. This has been accomplished by consolidating virtual machines and turning off idle physical machines [22]. Just like any optimization problem, being too far on either end of the resource usage spectrum causes energy waste. On one end there is having too many idle resources; every idle resource still has a minimum power requirement even if it is idle. On the other end is over utilized resources; when resources are over utilized they begin to thrash and performance takes a hit. A lot of research has gone into energy optimization and has determined that there is an ideal utilization percentage that achieves optimal performance per watt [23].

5.3.1 Bin-Packing

The energy optimization problem has been related to a bin-packing problem. One of the approaches used to solve this bin-packing problem is to put virtual machines with large workloads on hosts before putting virtual machines with smaller workloads [24]. Although this algorithm can typically solve the bin-packing problem, it fails to work properly for certain configurations of virtual machines. For example, if there exist 3 physical hosts and 4 virtual machines with equal workloads, the ideal workload utilization on a host is 80% and a single virtual machine running on a host utilizes 60% of the host's resources, it would be impossible to get these 4 virtual machines to fit and attain a host utilization of 80%. One of the hosts would be over-utilized, having allocated 20% more virtual resources than physical resources, and the other two hosts would be under-utilized, each having 60% utilization.

5.3.2 Bin-Packing Improved

To solve the utilization issue, it is necessary to split the workload of one virtual machine into 3 smaller virtual machines. Assuming that the original workload can be equally split, each of these virtual machines will receive exactly 1/3rd of the original workload and resources. Now that one of the virtual machines was split into 3 virtual machines, there are a total of 6 virtual machines. The original 3 virtual machines still have 60% utilization on a host, and the 3 split virtual machines each have 20% utilization on a host. Using the bin-packing algorithm, defined in Section 5.3.1, each of the full virtual machines will be assigned to one of the hosts, and then each of the split virtual machines will be assigned to a host. As there will now be 2 virtual

machines on each host: one full virtual machine and one split virtual machine. Therefore, the host will be at its optimal utilization of 80%.

5.3.3 Verifying Bin-Packing

Although it is possible to split a virtual machine to divide its workload and resources, it does not necessarily mean that the performance of the virtual machine is maintained. To examine this, a series of experiments are needed to determine whether or not performance is maintained for a variety of host configurations.

5.3.3.1 Experimental Setup

In this experiment, the same hardware and software configuration from Section 2.1.4 will be used. To accurately measure performance, we will run workloads of multiple sizes on one parent virtual machine that will be running alone on a host. We will then split the parent virtual machine into two children virtual machines, each with half of the parent's resources and workload. Then we will turn off the parent virtual machine. As we do not want to measure the time it takes to split the virtual machines as part of the performance of the machine, the workloads for the children virtual machines will begin once the split has been completed. A heat application was created to provide a benchmark for measuring performance in virtual machines.

5.3.3.2 Oversubscription v. Undersubscription

When running many virtual machines on few hosts, it is difficult to get a perfect balance between demand and availability of resources. For that reason, there needs to be an account for splitting performance degradation: oversubscription and undersubscription. In an oversubscription scenario, more virtual resources are allocated and used by virtual machines than available physical resources. As such, the host needs to aggressively handle resource allocations in order to give every virtual machine some time to utilize the physical hardware. In the case of undersubscription, there are fewer virtual resources allocated and used by virtual machines than available physical resources. In this case, the host does not need to aggressively handle resources; it can be generous with the amount of time a virtual machine has to utilize the physical hardware.

5.3.3.3 Oversubscription Experiment

For the oversubscription experiment, we will compare the performance of a virtual machine that is allocated 100% of the host's resources to two virtual machines each with 100% of the host's resources. We will run our heat application in multiples of two, to make it easier to divide the workload into half. The purpose of this experiment is to examine if the host produces significant overhead when context switching between two virtual machines as compared to one virtual machine.

5.3.3.4 Undersubscription Experiment

For the undersubscription experiment, we will compare the performance for a virtual machine allocated 2 vCPUs and 4 GB of memory to 2 virtual machines each allocated 1 vCPU and 2 GB of memory. We will run our heat application in multiples of two, to make it easier to divide the workload into half. The purpose of this experiment is to examine if the host does not produce context switching overhead, as there are more resources available than needed.

5.3.3.5 Results Oversubscription Experiment

In the oversubscription experiment, the performance of having one overprovisioned virtual machine, which is running processes with or without another overprovisioned idle virtual machine, yielded approximately the same process completion times. The performance yield was also approximately the same when comparing these results to the aggregate results of running two virtual machines with half the workload. However, when 6 processes were running, the completion performance was significantly impacted, relative to the single virtual machine that was running 6 processes. This is a significant finding as it may indicate that certain virtual machine configurations on overprovisioned systems yield poor performance.

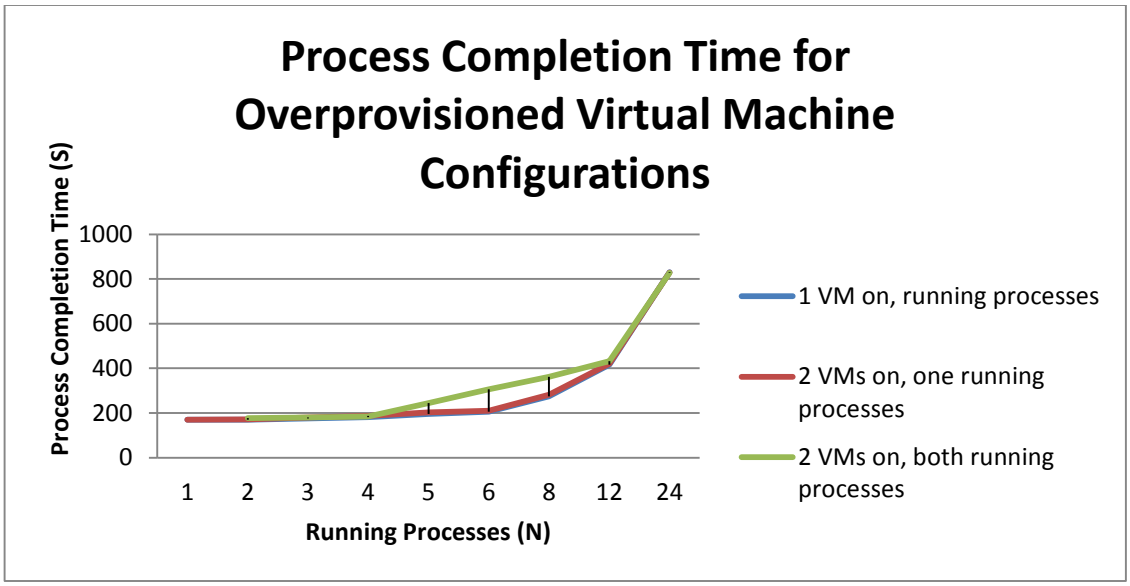

Figure 5.8: Process Completion Time for Overprovisioned Virtual Machines

5.3.3.6 Results Undersubscription Experiment

In the undersubscription experiment, the performance of having one virtual machine with 2 vCPUs and 4 GB of memory was approximately equivalent to having two virtual machines, each with 1 vCPU and 2 GB of memory and half the workload. This experiment was conducted for workload sizes ranging from 2 – 8 processes. Figure 5.9 represents the average completion time for each process. The results for the completion time for the split virtual machines are combined and averaged, so that if each machine ran 2 processes, the completion times of all 4 processes are averaged and compared to the single virtual machine that ran 4 processes. This finding is significant, as it shows that it is possible to split virtual machines and have an insignificant impact on the performance of the running processes.

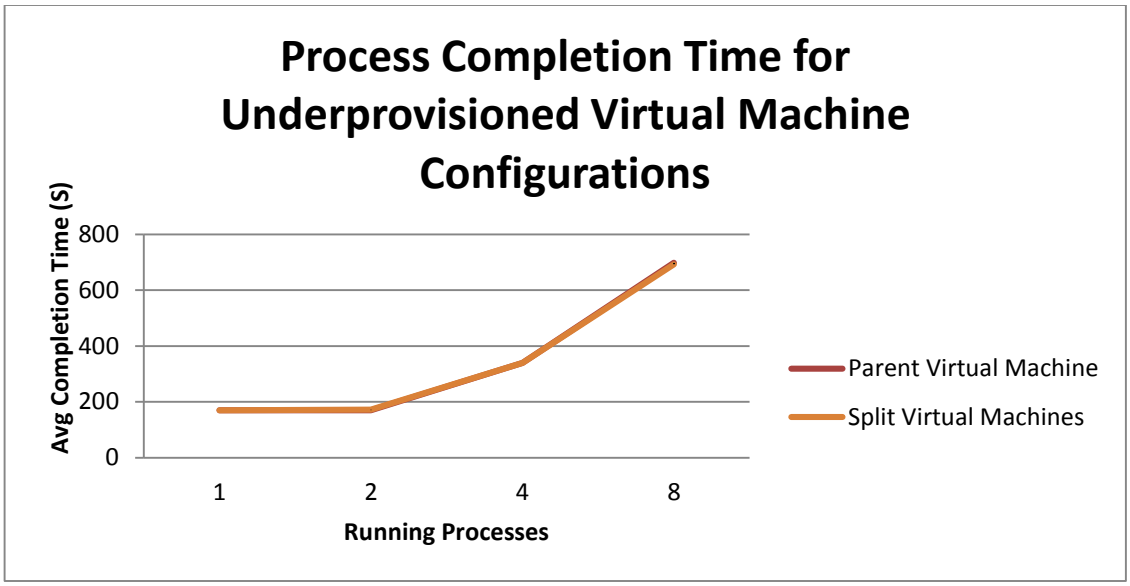

Figure 5.9: Process Completion Time for Underprovisioned Virtual Machines

5.3.3.7 Bin-Packing Conclusions

As seen in the results above, when equally splitting both a virtual machine's processes and resources, the virtual machine's performance is not affected. This means that virtual machine splitting can be used to divide virtual machines with large workloads to a more appropriate fit on physical hosts. Therefore it enables the bin-packing algorithm to attain optimal resource utilization on hosts, which leads to energy consumption improvement and cost savings.

5.4 Scheduling Manipulation through Granularity Refinement

Every operating system has a system in place for scheduling processes. The algorithms vary, yet all of them attempt to prevent CPU starvation. CPU starvation occurs when a process does not get to execute on a processor even though the process has been indefinitely ready to execute, in the waiting queue. Fair scheduling algorithms ensure that both long running and short running processes get a chance to execute on the processor. Although every process gets a chance to run, the completion times of a process can be significantly longer on a system with many compute intensive processes compared to a system with fewer compute intensive processes. Using virtual machine splitting, it is possible to maintain the same level of performance while decreasing process completion time.

5.4.1 Virtual Machine Splitting to Prioritize Process Execution

As we have seen in this chapter, splitting a virtual machine into smaller systems does not significantly affect the performance of the virtual machine. We can split virtual machines to produce unevenly balanced workloads and resource allocation to increase the amount of time a process gets to execute on the CPU.

5.4.2 Unequal Workload Splitting Experiment

In this experiment, we will compare the completion time of an odd number of heat applications on a single virtual machine with 2 vCPUs and 4 GB of memory to 2 equally resource split virtual machines with 1 vCPU each and 2 GB of memory, with an unequal workload split. The purpose of this experiment is to show that if a running virtual machine is split and its workload is split unequally, the system with the smaller workload should yield lower process completion times.

5.4.3 Unequal Resource Splitting Experiment

In this experiment, we will compare the completion time of an even number of heat applications on a single virtual machine with 4 vCPUs and 8 GB of memory to 2 unequally resource split virtual machines, the sum of both virtual machine resources will be equivalent to 4 vCPUs and 8 GB of memory. Each of the virtual machines will receive half of the parent's workload. The purpose of this experiment is to show that if a running virtual machine has its resources split unequally and its workload split equally, then the child with more resources should yield lower process completion times.

5.4.4 Results of Unequal Workload Splitting Experiment

As we can see in Figure 5.10, unequally splitting the workload makes it possible to have processes complete faster than they would have if they had run on the original virtual machine. However, this comes at a cost at significantly increasing the running time for processes running on the system with the larger workload. When averaging the total completion time amongst all processes, this average is greater than that of the parent virtual machine.

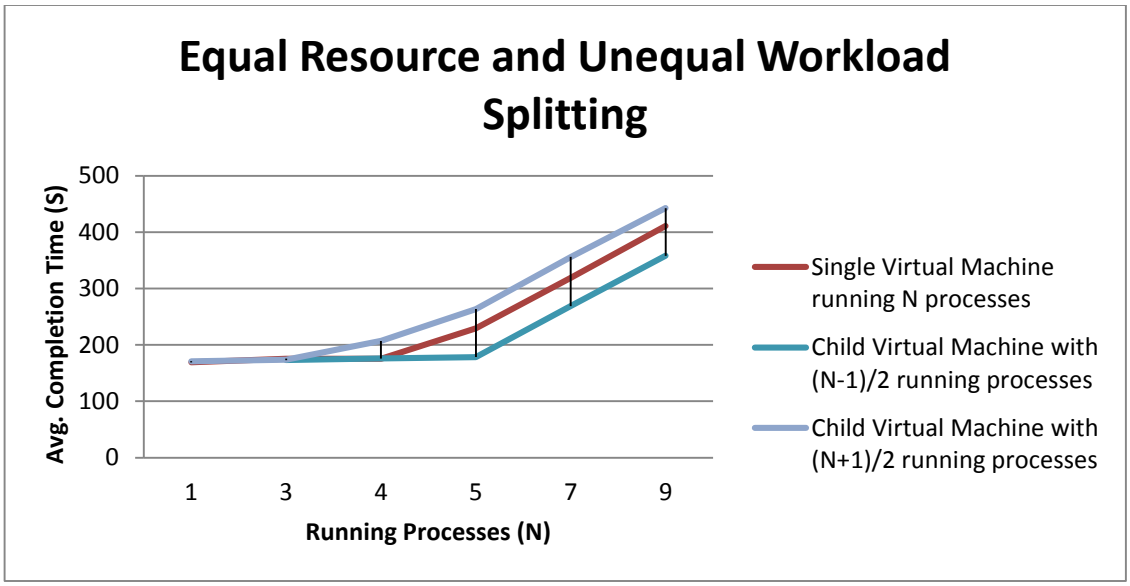

Figure 5.10: Equal Resource, Unequal Workload Splitting

5.4.5 Results of Unequal Resource Splitting Experiment

As we can see in Figure 5.11, unequally splitting resources makes it possible to have processes complete faster than they would have if they had run on the original virtual machine. This is very similar to the result for unequal workload splitting as the processes running on the system with fewer resources now have a significantly higher completion time. Another similarity is that the average completion time of all processes is greater than that of the parent virtual machine.

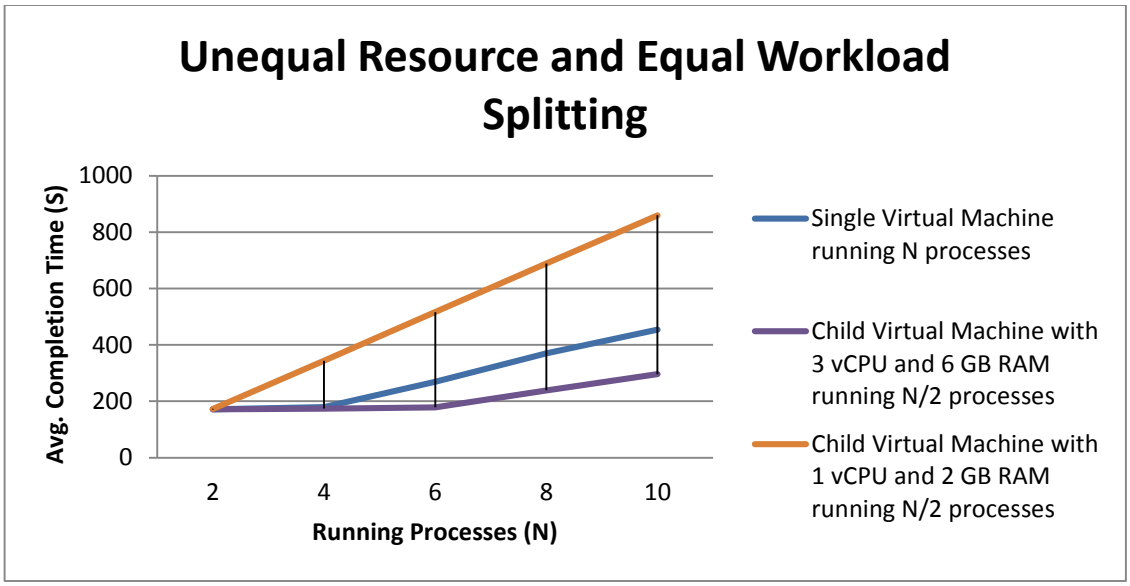

Figure 5.11: Unequal Resource, Equal Workload Splitting

5.4.6 Scheduling Manipulation Improvements

Although the same compute resources exist on the host, the reason for the higher average completion times is that when the virtual machine with the smaller workload or greater resources completes, it does not release its idle resources to the other virtual machine that is still working on its processes. Although this is the expected result of using virtual machines, it is not necessarily the intended result when using virtual machine splitting for scheduling manipulation, as a significant loss of performance is undesirable for most applications.

5.4.7 Resource Merging

Assuming that it is possible to determine when all processes in a workload complete on a virtual machine, it is possible to free idle resources and reallocate them without having to turn off the virtual machine that has running processes as seen in Figure 5.12. The most effective method is powering off the virtual machine that has completed its workload. Once the virtual machine is off, we can leverage hot-plug CPU and hot-add memory to add additional virtual resources to the running virtual machine. If this entire process could be done instantaneously the average completion times of the split virtual machine with fewer resources or more processes should be identical to that of its parent. As we can see in Figure 5.13 and Figure 5.14

it is possible to achieve equal completion times to that of the parent using resource merging as soon as one virtual machine completes its entire workload.

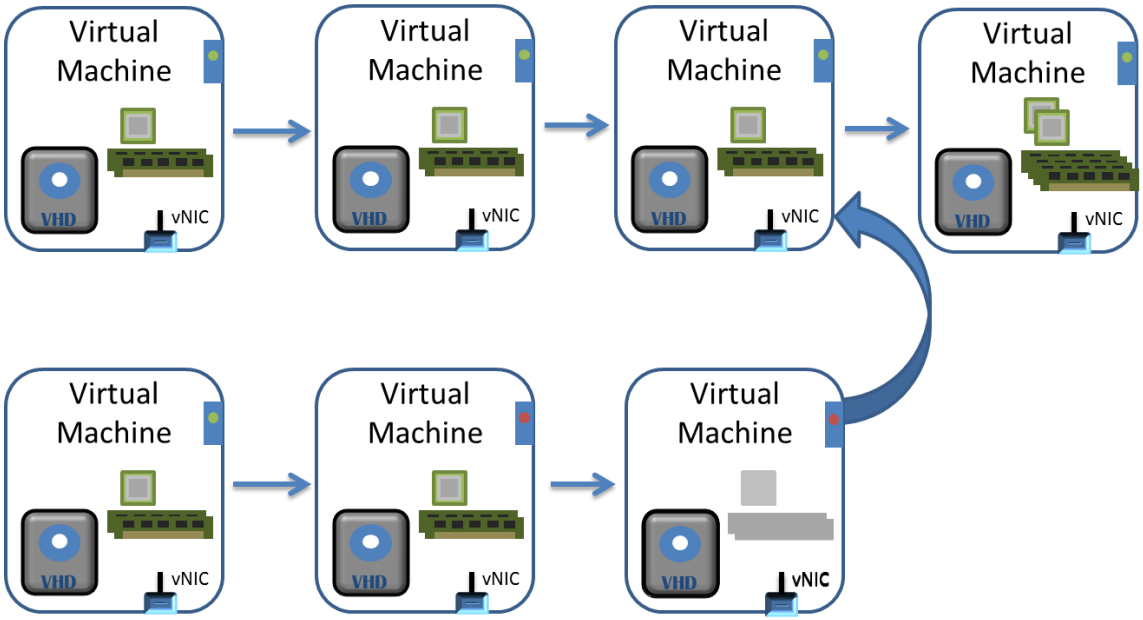

Figure 5.12: Virtual Machine Resource Merging Process

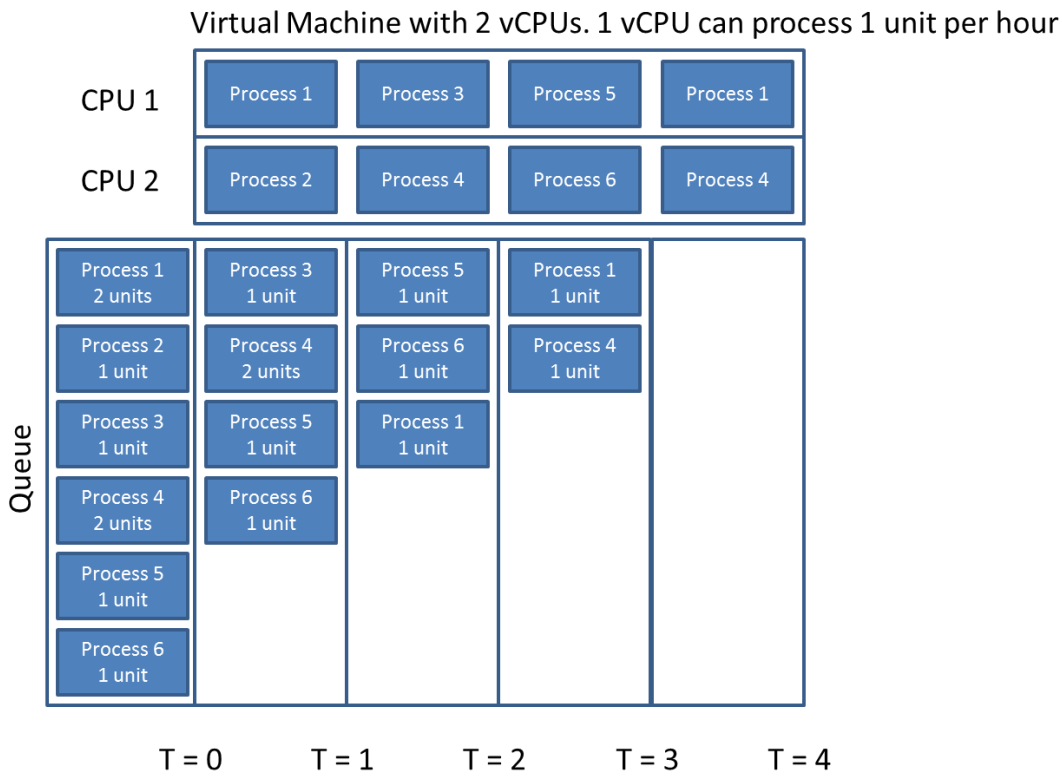

Figure 5.13: Single Virtual Machine with 2 vCPUs Processing Workload of 8 Units

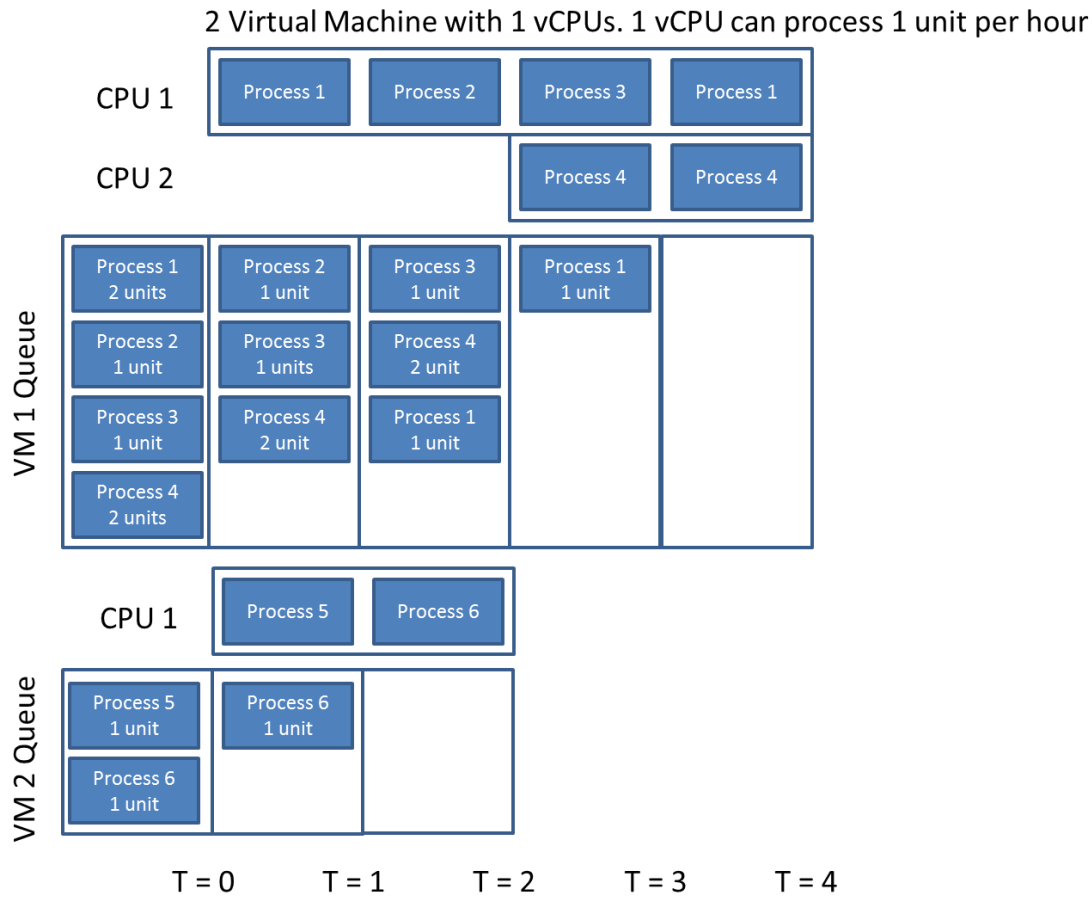

Figure 5.14: Two Virtual Machines with 1 vCPU each Processing Unevenly Split Workload of 8 Units with Resource Merging

5.4.8 Unequal Workload Splitting with Merging Experiment

In this experiment, we will test resource merging on an uneven workload split. The heat application will be used as a method of benchmarking performance. For a baseline, one virtual machine with 4 vCPUs and 8 GB of memory will be used. Average completion times will be recorded for 1 – 9 processes. After the baseline, two virtual machines each with 2 vCPUs and 4 GB of memory will be used to test uneven workload splitting. Once a virtual machine completes its workload, its resources will be reallocated to the other virtual machine.

5.4.9 Unequal Resource Splitting with Merging Experiment

In this experiment, we will test resource merging on an uneven resource split. The heat application will be used as a method of benchmarking performance. For a baseline, one virtual

machine with 4 vCPUs and 8 GB of memory will be used. Average completion times will be recorded for 2 – 10 processes. After the baseline, two virtual machines, one with 3 vCPUs and 6 GB of memory and the other with 1 vCPU and 2 GB of memory will be used to test even workload splitting. Once a virtual machine completes its workload, its resources will be reallocated to the other virtual machine.

5.4.10 Results of Unequal Workload Splitting with Resource Merging Experiment

As we can see in Figure 5.15, unequally splitting the workload with merging makes it possible to reduce the completion time of the system with the larger workload to the performance of its parent. At the same time, it does not negatively impact the virtual machine running the smaller workload. Furthermore, as seen in Figure 5.16, when averaging the total completion time amongst all processes, the combined completion time of the two split virtual machines is slightly lower than that of its parent and the non-merging virtual machines.

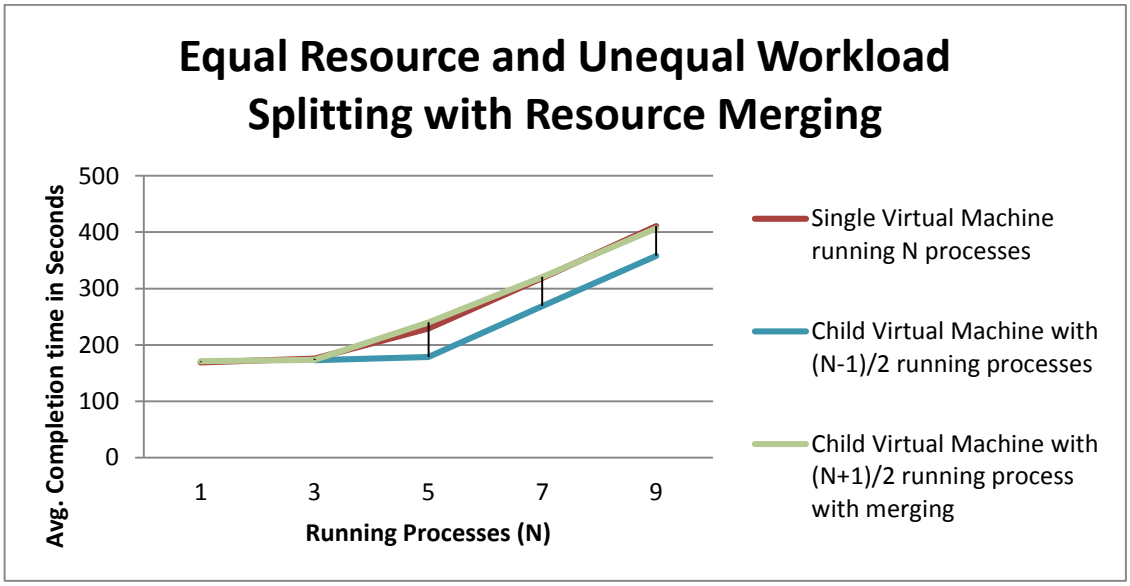

Figure 5.15: Equal Resource, Unequal Workload Splitting with Resource Merging

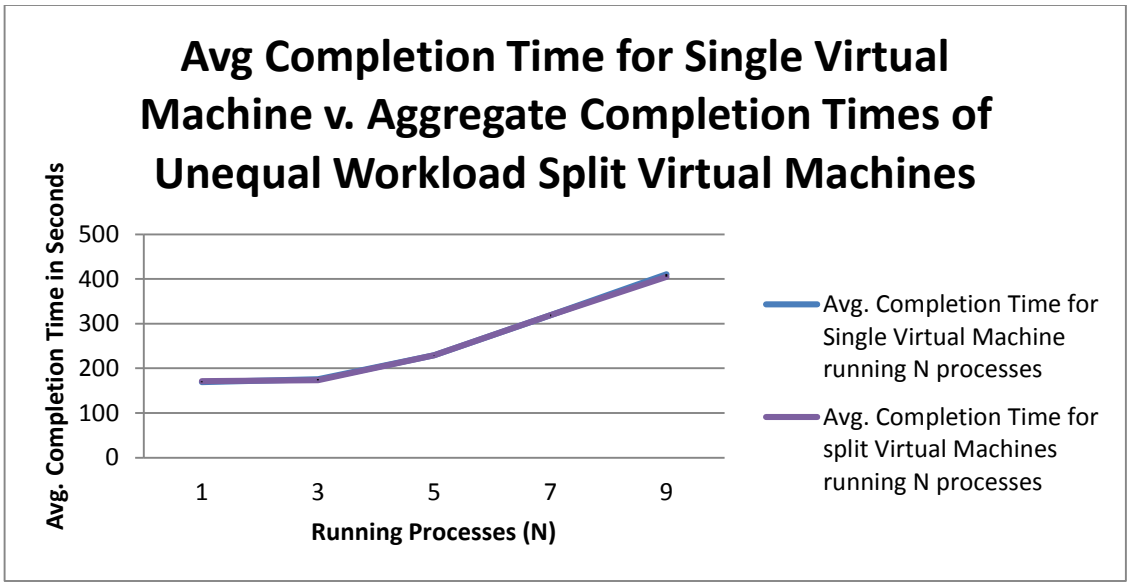

Figure 5.16: Average Completion Times of Equal Resource, Unequal Workload Split Virtual Machines

5.4.11 Results of Unequal Resource Splitting with Resource Merging Experiment

As we can see in Figure 5.17, unequally splitting the resources with merging makes it possible to reduce the completion time of the system with fewer resources to the performance of its parent. At the same time, it does not negatively impact the virtual machine running the more resources. Furthermore, as seen in Figure 5.18, when averaging the total completion time amongst all processes, the combined completion time of the two split virtual machines is significantly lower than that of its parent and the non-merging virtual machines.

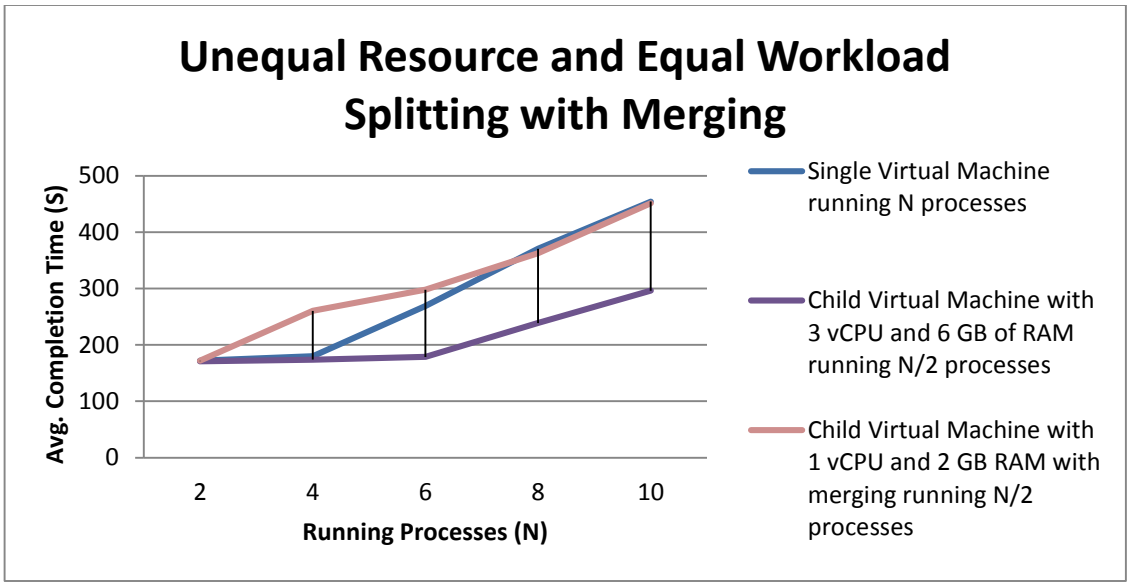

Figure 5.17: Unequal Resource, Equal Workload Splitting with Resource Merging

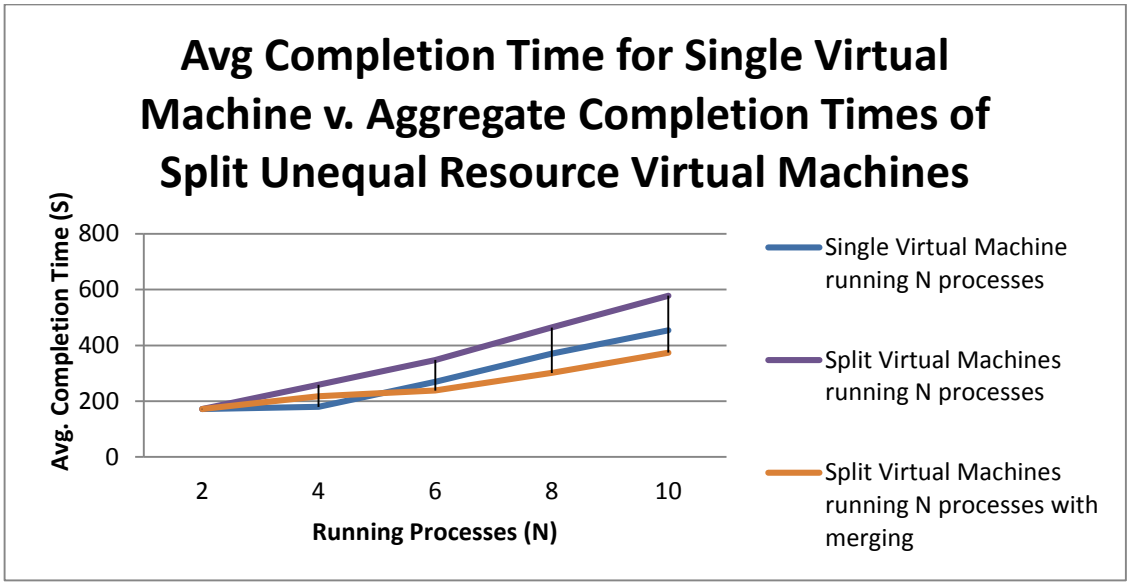

Figure 5.18: Average Completion Times of Unequal Resource, Equal Workload Split Virtual Machines

6. Conceptual Middleware for Virtual Machine Malleability

Virtual machine splitting is a very complicated process. Although it can be done manually, it is very tedious and the smallest mistake can render the split useless. To make the process easier and more streamlined, middleware should be implemented.

6.1 Malleability Manager

At the highest level of virtual machine malleability middleware is the malleability manager. When invoked, it interacts with the hypervisor, the operating system and the network. The malleability manager is the application that knows how to read a virtual machine's configuration files and determine what files to copy, what hypervisor functions to invoke and it is aware of all possible destination hypervisors and datastores. Once all of the corresponding virtual machine files have been modified, copied and registered it is able to interact with the guest operating system to handle process splitting and it can modify networking configurations to support networking based applications.

6.2 Process Splitting

When the malleability manager invokes process splitting it first requests a list of all running user processes. In its most basic form it terminates n-1/n processes, where n is the number of virtual machine splits, so that 1/n processes are running on each of the virtual machines. It can also be configured to be aware of process groups and other rules to help it better split processes running on a virtual machine. A process group is a logical grouping of processes that cannot be split from each other due to inability to communicate over the network or tight coupling. For example, if a system with a web server, database server and mail server were to be split, and the web server is tightly coupled with its database, the web server and database server would be put into the same process group. Then when the process termination phase of process splitting begins, the web server and database server will be terminated on one virtual machine and the mail server would be terminated on the other. Another example would include splitting processes based on user. All processes of a user can be grouped together in a process group, so when process termination occurs, the entire process group of each user will reside on one child virtual machine.

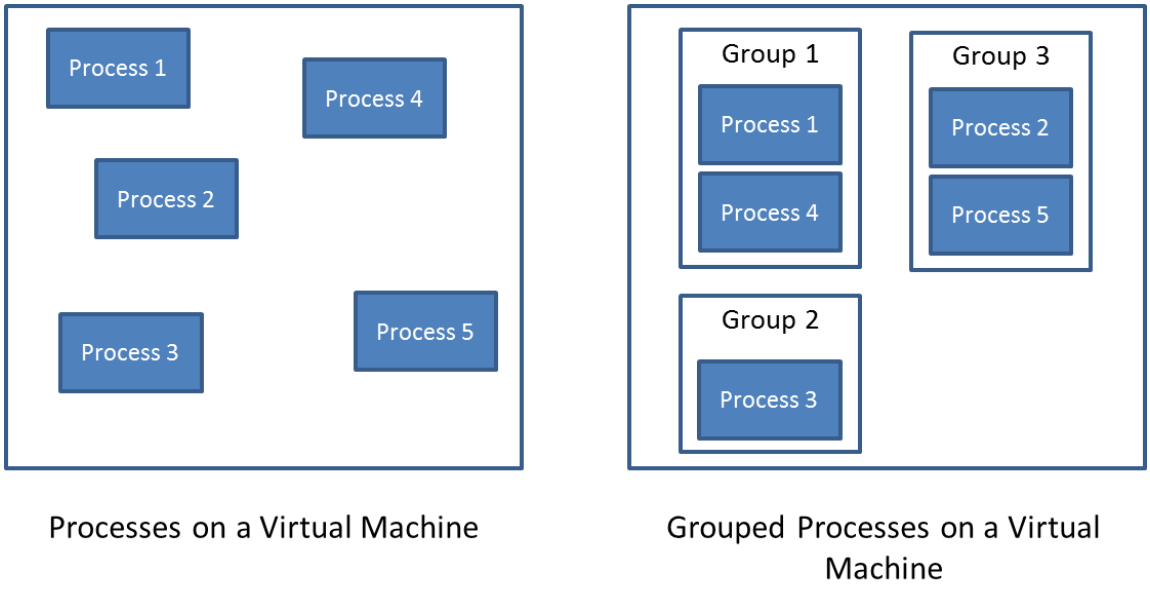

Figure 6.1: Comparison of Ungrouped Processes to Grouped Processes

6.3 Process Merging

Process merging is significantly harder than process splitting. Process merging is not implementable on a stateful level, as it is a very difficult process to selectively copy memory from one virtual machine to another virtual machine. To satisfy process merging, we depend on virtual disk storage allocated to each process or network storage disk and stateless applications. The malleability manager is able to determine which disks on a virtual machine are being used by processes, so when a merge occurs, it unmounts the virtual disk or the network storage disk and mounts it to the merged virtual machine. It then starts the stateless processes on the merged virtual machine, so it can begin working on requests and turns off the virtual machines that have been merged.

6.4 Hot-Plug CPU and Hot-Add Memory

Hot-Plug CPU and Hot-Add Memory are two technologies that can be leveraged to provide resource merging. Hot-Plug CPU allows vCPUs to be added to running virtual machines, if the guest operating system supports it. Similarly Hot-Add Memory allows virtual memory to be added to running virtual machines if the guest operating system supports it. The malleability manager is able to utilize Hot-Plug CPU and Hot-Add Memory to reallocate resources from two

or more virtual machines that are merging into a single virtual machine. When the malleability manager is invoked to do a resource merge, it turns off the specified virtual machines and adds their resources to the destination virtual machine. Resource merging can be used in conjunction with process merging, so that once all the processes have moved onto the merged virtual machine, the resources they have been using will be moved to the merged virtual machine.

6.5 Networking

Many applications require some form of access to a network. This access can be limited to a local area network, so that processes can communicate to each other on a local area network, or it can be connected to the internet, so that users all over the world can interact with the application. There are two options for supporting network access for virtual machine splitting and merging. The first utilizes DNS servers. Each application is assigned a domain name. The record for the application name points to the server the process is running on. When a virtual machine is split, the record is updated to point to the virtual machine the process was split onto. When virtual machines are merged, the record is updated to point to the virtual machine the process was merged onto. Using the DNS server method, it is always possible to connect to an application using its domain name. The main issue with this method is that connections to the process are lost whenever a split occurs. The second method is using a proxy. Unlike the DNS server method, where clients and applications directly connect to each other, in the proxy method all clients and applications connect to the proxy and the proxy connects to each application. When a virtual machine split occurs, the client or application does not lose its connection to the proxy, however the proxy loses its connection to the application. Once the split is complete, the proxy reestablished communication to the application and directs client and application traffic through it. The same process occurs for virtual machine merging. The disadvantage of the proxy method is that it is not transparent.

6.6 Malleability Profile

Virtual machine splitting and merging is useful to improve performance, manipulate scheduling and improve energy utilization. However there also exist some limitations to what splitting and merging can do.

6.6.1 Obvious Cases Against Using Splitting and Merging

Process groups or single processes running on a virtual machine cannot be split; therefore virtual machine splitting will not offer any performance improvements. Stateful merging cannot occur as virtual machines have divergence with memory and disk, however disks dedicated to a single process or process group can be moved from one virtual machine to another and get started on the destination virtual machine.

6.6.2 Splitting Profile

Virtual machine splitting cost is relatively light as it is either bounded by disk I/O when copying files or network when copying state. If the remaining runtime of a set of processes is greater than the splitting cost and greater than the longest runtime of any subset of split processes then it is worth splitting a virtual machine. Another case is when there are more available physical hosts than virtual machines. In this case, it is better to split the virtual machine to increase overall host utilization. Another case is if any particular host is over utilized and there exists capacity to handle the workload across multiple underutilized hosts, then splitting a virtual machine increase performance.

Table 6.1: Virtual Machine Splitting Profile

Virtual Machine Splitting Profile	
Number of Splits	N
Cost of Splitting Disk	D
Cost of Splitting State	S
Remaining runtime for all processes on single virtual machine	$\sum_{1}^{i} P_i$
Cost of splitting	$N * D + S$
Maximum runtime for any split processes	M
If true, split virtual machine. **Else, do not split.**	$\sum_{1}^{i} P_i > N * D + S + M$

6.6.3 Merging Profile

Merging virtual machines can be used to reduce overhead caused by having many idle split virtual machines. Although the overhead of splitting a virtual machine is low, and the impact to performance is negligible, when many idle virtual machines reside on a single host, the sum of their idle resource utilization can become significant. By utilizing merging, the idle processes can be consolidated onto a single virtual machine, and idle resource utilization can be scaled down.

Table 6.2: Virtual Machine Merging Profile

Virtual Machine Merging Profile	
Number of Idle Split Virtual Machines	N
Resource Utilization of Idle Virtual Machine	R_i
Resource Utilization of Merged Virtual Machine	R_m
If true, merge virtual machine. **Else, do not merge.**	$R_m < N * R_i$

6.6.4 Energy Saving Profile

Every physical host has some optimal resource utilization to achieve the best performance per watt. Depending on the resources available to the physical host, this utilization can vary. Virtual machine merging and splitting can be used to achieve better resource utilization. In order to utilize merging and splitting, the optimal resource utilization of a host and the resource utilization of a virtual machine must be known. A split should occur if the resource utilization of one or more hosts is greater than the optimal resource utilization. If there exists a host, that has the capacity to take one or more virtual machines, then the virtual machines will be migrated from the over utilized host to the underutilized host. However in the case, that no host has sufficient capacity to take one of the virtual machines from the over utilized host, then the virtual machine will be split until each of its splits can fit onto the hosts with available capacity. In the case that every host is over utilized, then virtual machine splitting will not occur. In the case that many hosts are underutilized, virtual machine merging will be

used to consolidate many split virtual machines into a single virtual machine. There larger virtual machines will be migrated onto as few hosts as possible and the hosts with no virtual machines will be shutdown.

7. Related Work

7.1 Process Malleability

Osman, Subhraveti, Su and Nieh [16] introduce a method of enabling application migration between similar host operating systems. They implemented a virtualization layer on top of a host operating system which provides access to system resources, they called this Zap. Then they created pods, processes groupings which reside on top of Zap. When a pod is migrated from one Zap host to another, Zap remaps the resources to the Pod on the new Zap host, making the migration transparent to the running applications. Their work on Zap enables any application to become part of a malleable process group without requiring any changes to an existing application.

Maghraoui, Desell, Szymanski and Varela [15] introduce a method of modifying parallel process granularity using split and merge functions. Their implementation enables iterative applications to adapt to the fluctuating nature of resource availability in shared environments. To support this type of malleability, applications need to specify how to change process granularity and middleware needs to be aware of resource availability and trigger changes to the application.

7.2 Preserving State

Hirofuchi, Nakada, Itoh and Sekiguchi [25] use a method of virtual machine migration to preserve a virtual machine's state and move it to a host with more available resources to improve performance. Their method copies the virtual machine state from the source host to the destination host by sending the minimal state, starting the destination virtual machine, suspending the source virtual machine and then pushing the remaining state from the source to the destination. This form of migration is known as post-copy, as the state from the source virtual machine is pushed to the destination virtual machine after the destination is started.

Milojičić, Goudlis, Paindaveine, Wheeler and Zhou [26] propose a method of state migration, which does not suspend a virtual machine, until the majority of its state has been copied. In the first phase the state is copied from the source host to the destination host, this process iterates multiple times over the virtual machines state to copy all of the dirty pages. In the second phase, the virtual machine is stopped and the remaining dirty pages are copied

over. This form of migration is known as pre-copy, as the destination virtual machine is only started once most of the state from the source virtual machine has been copied.

Kozuch and Satyanarayanan [27] introduce a method of suspending and resuming a system, so that it can be moved from one machine to another machine. Their method utilizes a distributed file system that many machines have access to. When a system is suspended, the machine it is running on writes the system's state to the distributed file system. To resume the system, any machine with access to the distributed file system, can read the state and resume the system's execution.

7.3 Energy Efficiency

Beloglazov and Buyya [22] propose a method of consolidating virtual machines onto fewer hosts to save power. Their method relies on live migration and turning off idle hosts, while maintaining the Quality of Service required by Service Level Agreements (SLA). As most cloud computing offers rely on SLA, it is important to consider maintaining performance when consolidating workloads.

Srikantaiah, Kansal, Zhao [23] introduce the tradeoff between resource utilization, energy utilization and performance. They propose that a modified bin-packing problem can model the optimal resource utilization of a host, to achieve the highest performance per energy.

7.4 Network Malleability

Celesti, Villari and Puliafito [28] propose a cloud naming framework to enable applications and resources to interact using URIs. Every application and resource receives a name based on type and location by using name space federation. Applications can then use resolution to connect to each other and other resources.

Yang, Qin, Zhang, Zhou, Wanga [29] propose a service for finding resources based on a combination of P2P technologies and DNS. Their service allows legacy URI resolution through a URI server to connect applications to resources residing on a logical P2P network.

Varela and Agha [19] describe a universal naming and locator convention for SALSA actors. Every actor is assigned an UAN, which is independent of the actor's current location on the internet. An UAL, identifies the actor's current location on the internet. When the actor migrates to a new theater, the actor's run-time system, the naming server is updated with a <UAN, UAL> pair representing the actor and its current theater.

Meng, Pappas and Zhang [30] propose a method of optimizing virtual machine scalability based on network patterns. Their method introduces migrating virtual machines that frequently interact over the network, to hosts that are closer in network proximity to achieve higher performance.

7.5 Virtualization Scalability

Jamal, Qadeer, Mahmood, Waheed and Ding [31] examined virtual machine scalability on multi-core systems using single type workloads to identify bottlenecks. They showed that processes running in virtual machines perform equally as well as threaded processes, when CPU count is equal. They also showed that inter-VM communication is severely limited by LAN and WAN bandwidth.

Wang and Varela [6] examined the impact of virtual machine configurations on CPU intensive workloads. They noticed that loosely coupled workloads spread across the same number of physical resources, had approximately the same performance regardless of virtual machine configuration. However for tightly coupled workloads, the optimal performance was achieved when there were twice as many vCPUs as physical CPUs and that doubling the memory gave at most a 15% performance improvement.

8. Discussion

8.1 Contributions

Most recent malleability research has been focused on application malleability. This thesis discusses virtual machine malleability. In particular, we explore the split and merge functions of malleability and show that it is possible to implement both with minimal modifications to the hypervisor. Conceptually, this is useful as the features leveraged by the malleability functions are available on most modern hypervisors. Therefore it should be possible to implement virtual machine malleability on any modern hypervisor. Although this type of malleability cannot be leveraged by strongly coupled and dependent workloads, it does provide malleability features such as dynamic workload reconfiguration which are typically only seen in application layer malleability, while affording better transparency.

8.2 Conclusion

New methods of malleability are important as each provides applications with a broader type of scalability and elasticity. In this thesis we discuss virtual machine malleability using the methods defined as split and merge. We show that splitting and merging virtual machines enables applications to scale without having any form of built in malleability. We explore the strengths and weaknesses of each of these methods. We observe linear performance change as applications are scaled. We also observe linear and constant costs for using each of these methods. We discover that these methods provide benefits from both Type I and Type II malleability, including dynamic workload reconfiguration, transparency and supporting applications without built-in malleability. We also present applications of using the splitting and merging functions, including scaling, energy optimization and scheduling manipulation.

For scaling applications, we have proposed a method of splitting virtual machines statefully using both delta disks and live memory copy. Based on our models, it can be seen that stateful virtual machine splitting is bound by the amount of time to copy files N times or bound by transferring virtual machine state over the network. We have also discussed the problem with virtual machine state and disk divergence, which makes virtual machine merging a complicated problem. We proposed a solution, which dedicated virtual disks or network disks to a process, so that when a merge occurs, all process related disks are merged onto a single virtual machine.

For the energy optimization problem, we have proposed a method of splitting a virtual machine's resources and processes between multiple virtual machines. As seen in our experiments, there is no degradation of performance when both resources and processes are equally split. This result means that it is possible to divide virtual machines equally and split their children across multiple physical hosts; as each child is smaller in terms of resource utilization it is easier to satisfy the energy optimization bin-packing problem.

For scheduling manipulation, we have shown that using virtual machine splitting can be used to give processes a higher priority. Our experiments have shown that virtual machine splitting can reduce a process's completion time, when there are more compute intensive processes running than vCPUs dedicated to the virtual machine. We also proposed using virtual machine merging as a method of correcting the significantly higher completion times of the lower priority processes. By doing such, we were able to reduce the completion time to approximately that of the single virtual machine running all of the processes. This in turn reduced the average process completion time for all of the processes.

These results are promising as they show that virtual machine malleability can be leveraged for many different applications and provides similar functionality to application malleability without having to rewrite applications to be malleable.

8.3 Future Work

The malleability manager discussed in Chapter 6, has only been partially implemented. It is not recommended to do virtual machine malleability by hand as it is a very tedious and intricate process. For those reasons, the remaining parts of the malleability manager need to be implemented, these parts include Hot-Plug CPU, Hot-Add memory, networking and malleability profiles.

As my research has been conducted using ESXi, it has not been possible to implement stateful splitting using the live memory copy component of virtual machine migration. Stateful splitting might be a feature limited to open source hypervisors such as Xen, unless VMware add an API call to leverage their live memory copy function.

Performance degradation in regards to virtual machines running on delta disks should be examined. There is a possibility that if a virtual machine is split hundreds of times that the

parent virtual disk, which all of the delta disks reference will thrash. If this is the case, it might make sense to create M copies of the parent disk, so that at most $1/M^{th}$ of the virtual disks reference the same parent disk.

9. Literature Cited

[1] VMware. (2006). *Reducing Server Total Cost of Ownership with VMware Virtualization Software.* [Online]. Available: http://www.vmware.com/pdf/TCO.pdf. [Accessed: March 3, 2014].

[2] B. P. Tholeti. (2011). *Hypervisors, virtualization, and the cloud: Learn about hypervisors, system virtualization, and how it works in a cloud environment.* [Online]. Available: http://www.ibm.com/developerworks/cloud/library/cl-hypervisorcompare/. [Accessed: March 3, 2014].

[3] L. M. Vaquero et al., "A Break in the Clouds: Towards a Cloud Definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50-55, Dec. 2008.

[4] A. Greenberg et al., "The Cost of a Cloud: Research Problems in Data Center Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68-73, Dec. 2008.

[5] T. J. Bittman et al. (2013). *Magic Quadrant for x86 Server Virtualization Infrastructure.* [Online]. Available: http://www.gartner.com/technology/reprints.do?id=1-1GRGRRU&ct=130702&st=sb. [Accessed: February 11, 2014].

[6] Q. Wang and C. Varela, "Impact of Cloud Computing Virtualization Strategies on Workloads' Performance," in *Proc. of the 2011 4th IEEE Int. Conf. on Utility and Cloud Computing*, Melbourne, Australia, 2011, pp. 130-137.

[7] A. Verma et al., "pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems," in *Proc. of the 9th ACM/IFIP/USENIX Int. Conf. on Middleware*, Leuven, Belgium, 2008, pp. 243-264.

[8] K. Hess. (2010). *10 New Reasons to Virtualize Your Infrastructure.* [Online]. Available: http://www.serverwatch.com/trends/article.php/12128_3910526_2/10--New-Reasons-to-Virtualize-Your-Infrastructure.htm. [Accessed: March 3, 2014].

[9] M. Kammerer. (2008). *CPU hotplug.* [Online]. Available: https://www.ibm.com/developerworks/linux/linux390/perf/tuning_cpuhotplug.html. [Accessed: March 3, 2014].

[10] J. J. Johnson et al., "Hot-upgrade/hot-add memory," U.S. Patent 6,854,070, Feb. 8, 2005.

[11] C. Clark et al., "Live Migration of Virtual Machines," in *Proc. of the 2nd conf. on Symp. on Networked Systems Design & Implementation*, Berkeley, CA, 2005, pp. 273-286.

[12] J. G. Hansen and E. Jul, "Self-migration of operating systems," in *Proc. of the 11th Workshop on ACM SIGOPS European Workshop*, Leuven, Belgium, 2004.

[13] C. P. Sapuntzakis et al., "Optimizing the Migration of Virtual Computers," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 377-390, Dec. 2002.

[14] P. Wang et al., "Impact of Virtual Machine Granularity on Cloud Computing Workloads Performance," in *11th IEEE/ACM Int. Conf. on Grid Computing*, Brussels, Belgium, 2010, pp. 393-400.

[15] K. E. Maghraoui et al., "Dynamic Malleability in Iterative MPI Applications," in *Proc. of 7th IEEE Int. Symp. on Cluster Computing and the Grid*, Rio de Janeiro, Brazil, 2007, pp. 591-598.

[16] S. Osman et al., "The Design and Implementation of Zap: A System for Migrating Computing Environments," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 361-376, Dec. 2002.

[17] K. E. Maghraoui et al., "Malleable Iterative MPI Applications," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 3, pp. 393-413, Mar. 2009.

[18] T. Desell et al., "Malleable components for scalable high performance computing," in *Proc. of the HPDC'15 Workshop on HPC Grid programming Environments and Components*, Paris, France, 2006, pp. 37-44.

[19] C. Varela and G. Agha, "Programming Dynamically Reconfigurable Open Systems with SALSA," *SIGPLAN Not.*, vol. 36, no. 12, pp. 20-34, Dec. 2001.

[20] Q. Wang, "Middleware for Autonomous Reconfiguration of Virtual Machines," M.S. thesis, Dept. Comp. Sci., Rensselaer Poly. Inst., Troy, NY, 2011.

[21] M. Armbrust et al., "A View of Cloud Computing," *Commun. ACM*, vol. 53, no. 4, pp. 50-58, Apr. 2010.

[22] A. Beloglazov and R. Buyya, "Energy Efficient Allocation of Virtual Machines in Cloud Data Centers," in *Proc. of the 2010 10th IEEE/ACM Int. Conf. on Cluster, Cloud and Grid Computing*, Melbourne, Austrailia, 2010, pp. 577-578.

[23] S. Srikantaiah et al., "Energy Aware Consolidation for Cloud Computing," in *Proc. of the 2008 Conf. on Power Aware Computing and Systems*, San Diego, CA, 2008, pp. 10-10.

[24] B. Li et al., "EnaCloud: An Energy-saving Application Live Placement Approach for Cloud Computing Environments," in *IEEE Int. Conf. on Cloud Computing*, Bangalore, India, 2009, pp. 17-24.

[25] T. Hirofuchi et al., "Reactive Consolidation of Virtual Machines Enabled by Postcopy Live Migration," in *Proc. of the 5th Int. Workshop on Virtualization Technologies in Distributed Computing*, San Jose, CA, 2011, pp. 11-18.

[26] D. S. Milojičić et al., "Process migration," *ACM Comput. Surv.*, vol. 32, no. 3, pp. 241-299, Sept. 2000.

[27] M. Kozuch and M. Satyanarayanan, "Internet Suspend/Resume," in *Proc. of the 4th IEEE Workshop on Mobile Computing Systems and Applications*, Callicoon, NY, 2002, pp. 40-46.

[28] A. Celesti et al., "Design of a Cloud Naming Framework," in *Proc. of the 7th ACM Int. Conf. on Computing Frontiers*, Bertinoro, Italy, 2010, p. 2.

[29] D. Yang et al., "URNS: A new name service for uniform network resource location," in *2006 IET Int. Conf. on Wireless, Mobile and Multimedia Networks*, Hangzhou, China, 2006, pp. 1-4.

[30] X. Meng et al., "Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement," in *Proc. of the 29th Conf. on Information Communications*, San Diego, CA, 2010, pp. 1154-1162.

[31] M. H. Jamal et al., "Virtual Machine Scalability on Multi-Core Processors Based Servers for Cloud Computing Workloads," in *IEEE Int. Conf. on Networking, Architecture, and Storage*, Hunan, China, 2009, pp. 90-97.