Performance Implications of Virtualization

Abstract

Virtualization is a component of cloud computing. Virtualization transforms traditional inflexible, complex infrastructure of individual servers, storage, and network hardware into a flexible virtual resource pool and increases IT agility, flexibility, and scalability while creating significant cost savings. Additional benefits of virtualization include, greater work mobility, increased performance and availability of resources, and automated operations. Many virtualization solutions have been implemented. There are plenty of cloud providers using different virtualization solutions to provide virtual machines (VMs) and containers, respectively. Various virtualization solutions have different performance overheads due to their various implementations of virtualization and supported features. A cloud user should understand performance overheads of different virtualization solutions and the impact on the performance caused by different virtualization features, so that it can choose appropriate virtualization solution, for the services to avoid degrading their quality of services (QoSs).

In this research, we investigate the impacts of different virtualization technologies such as, container-based, and hypervisor-based virtualization as well as various virtualization features such as, over-allocation of resources, live migration, scalability, and distributed resource scheduling on the performance of various applications for instance, Cassandra NoSQL database, and a large telecommunication application. According to our results, hypervisor-based virtualization has many advantages and is more mature compare to the recently introduced container-based virtualization. However, impacts of the hypervisor-based virtualization on the performance of the applications is much higher than the container-based virtualization as well as the non-virtualized solution. The findings of this research should be of benefit to the ones who provide planning, designing, and implementing of the IT infrastructure.

Acknowledgment

This research work was founded by the Swedish Foundation for Knowledge and Development, KK-Stiftelsen, in Stockholm. Part of this research is result of the collaboration between Blekinge Institute of Technology, Ericsson Company and Compuverde Company located in Karlskrona, Sweden.

I would like to thank my Supervisor, Professor Lars Lundberg for his invaluable guidance and support during my studies, and for giving me this opportunity to work in a real-world industrial project. I would also like to thank my secondary supervisors David Erman and Dragos Ilie for their invaluable feedback and suggestions. In addition, I would like to thank my colleagues both in Ericsson and Compuverde for their invaluable support. I would also like to thank all my colleagues, the library staff, and the supporting departments at BTH.

Finally, I am also very grateful to my family and friends for supporting, encouraging, and motivating me.

Preface

This thesis is based on the work presented in the following eight papers. The papers II, III, IV, V, and VI are published in peer-reviewed conference proceedings. Paper I is published in a journal and Paper VII is been submitted to a conference and is currently under peer-reviewing. Paper VIII is been submitted to a Journal as well.

The included papers have been modified to fit this format, but the content is unchanged.

Paper I

S. Shirinbab, L. Lundberg, D. Erman, ”Performance Evaluation of Distributed Storage Systems for Cloud Computing”, published in International Journal of Computer Applications (IJCA), 2013.

Paper II

S. Shirinbab, L. Lundberg, D. Ilie, ”Performance Comparison of KVM, VMware, and XenServer using a Large Telecommunication Application”, Proceedings of the Fifth International Conference on Cloud Computing, GRIDs, and Virtualization, pp. 25-29, 2014.

Paper III

S. Shirinbab, L. Lundberg, “Performance Implications of Over-Allocation of Virtual CPUs”, published in International Symposium on Networks, Computers and Communications (ISNCC), pp.1-6, 2015.

Paper IV

S. Shirinbab, L. Lundberg, J. Håkansson, ”Comparing Automatic Load Balancing using VMware DRS with a Human Expert”, published in IEEE International Conference on Cloud Engineering Workshop (IC2EW), pp. 1-8, 2016.

Paper V

S. Shirinbab, L. Lundberg, “Performance Implications of Resource Over-Allocation During the Live Migration”, published in IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 1-6, 2016.

Paper VI

S. Shirinbab, L. Lundberg, E. Casalicchio, ”Performance Evaluation of Container and Virtual Machine Running Cassandra Workload”, published in 3rd International Conference on Cloud Computing Technologies and Applications (CloudTech), 2017.

Paper VII

S. Shirinbab, L. Lundberg, E. Casalicchio, ”Performance Comparison between Horizontal Scaling of Hypervisor and Container Based Virtualization using Cassandra NoSQL Database”, submitted to a conference, pp. 1-6, 2018.

Paper VIII

S. Shirinbab, L. Lundberg, “Scheduling Tasks with Hard Deadlines in Cloud-Based Virtualized Software Systems”, will be submitted to a Journal.

There are other papers that are not included in this thesis but are related to this research:

Paper IX

E. Casalicchio, L. Lundberg, S. Shirinbab, ”Optimal Adaptation for Apache Cassandra”, published in Self Orgnizing Self Managing Clouds (SoSeMC) workshop at 13th IEE International Conference on Autonomic Computing, pp. 1-6, 2016.

Paper X

E. Casalicchio, L. Lundberg, S. Shirinbab, ”An Energy-Aware Adaptation Model for Big Data Platforms”, (poster) published in Autonomic Computing, pp. 1-2, 2016.

Paper XI

S. Shirinbab, L. Lundberg, “Real-time Scheduling in Cloud-based Virtualized Software Systems”, Proceedings of the second Nordic Symposium on Cloud Computing and Internet Technologies, pp. 54-58, 2013.

Paper XII

S. Shirinbab, L. Lundberg, E. Casalicchio, ”Performance Evaluation of Containers and Virtual Machines when Running Cassandra Workload Concurrently”, submitted to a Journal.

Paper XIII

E. Casalicchio, L. Lundberg, S. Shirinbab, ”Energy-aware auto-scaling algorithms for Cassandra virtual data centers”, published in Cluster Computing journal, vol. 20, issue 3, pp. 2065-2082, 2017.

Table of Contents

1 Introduction

In the past few years, the IT industry's focus on virtualization technology has increased considerably. Virtualization is often confused with Cloud Computing, virtualization is the fundamental technology that enables Cloud Computing. Essentially, in this study our focus would be more on virtualization technology rather than the Cloud. Virtualization implementation for organizations requires an in-depth analysis of the organization's specific needs and requirements. However, a more important step before fully implementing a system or plan is to consider pros and cons of the technology. In this study, we consider different types of virtualization for instance container based, hypervisor based, and storage virtualization. We evaluate virtualization solutions provided by different vendors and compare the impacts of different solutions on the performance in terms of CPU, and disk utilization as well as response time and latency using real workloads for example a large real-time telecommunication application and the Cassandra NoSQL database. In addition, we investigate some of the key features in virtualization for instance live migration, over-allocation, horizontal and vertical scaling provided by various vendors such as VMware, KVM, Xen and Docker. The result of this research provides guidance for organizations to choose the right virtualization solution to fit their organization's needs.

1.1 Background

The concept of virtualization has been around for long time, it has been developed in the late 1960s and early 1970s by International Business Machines (IBM). Virtualization is commonly defined as a software that separates physical infrastructure to create multiple dedicated resources.

There are different types of virtualization such as, application virtualization, desktop virtualization, hardware virtualization, network virtualization, and storage virtualization.

1.1.1 Application virtualization

Application virtualization is a process where applications get virtualized and are delivered from a server to the end user's device, such as laptops, smartphones, and tablets. The user can then access and use the applications from virtually anywhere. Any user actions are transmitted back to the hosting server. This type of virtualization is particularly popular for businesses that require the use of their applications on the go. Application virtualization vendors and their products include Microsoft App-V, Citrix XenApp, VMware Horizon Apps.

1.1.2 Desktop virtualization

Similar to application virtualization mentioned above, desktop virtualization separates the desktop environment from the physical client device that is used to access it. The major advantages of desktop virtualization are that users are able to access all their personal files and applications from any location and on any computer. It also lowers the cost of licensing for installing software on desktops and maintenance and patch management is very simple. Virtual desktop infrastructure (VDI) is a type of desktop virtualization.

1.1.3 Hardware virtualization

Hardware virtualization also known as hardware-assisted virtualization or server virtualization is the abstraction of computing resources from the software that uses those resources. There are different virtualization techniques, such as full virtualization, emulation virtualization, paravirtualization, and operating system level virtualization. All these techniques vary in the virtualization solutions and the level of abstraction while having the same goal.

1.1.3.1 Emulation virtualization

In traditional physical computing environments (see Figure 1.1), the operating system is directly installed on hardware devices and has direct access to the underlying computer hardware and components including the processor, memory, storage and so on. This caused limitations for allocation of CPU and memory resources also required service down during hardware upgrades on each server. However, emulation virtualization installs a hypervisor or virtual machine monitor (VMM), which creates an abstraction layer between the operating system and the underlying hardware. This approach is known as “bare-metal/native hypervisor” (see Figure 1.1). The alternative to a bare-metal approach involves installing a host operating system first and then installing a hypervisor (see Figure 1.1). This approach is often referred to as “hosted hypervisor”. Examples of a bare-metal/native hypervisors are Oracle VM, Microsoft Hyper-V, VMware ESX and Xen. A well-known example of a hosted hypervisor is Oracle VM VirtualBox, others includes VMware Server and Workstation, Microsoft Virtual PC, KVM, QEMU and Parallels.

1.1.3.2 Operating system level virtualization

In operating system level virtualization, the operating system is altered so that it operates like several different, individual systems. The virtualized environment accepts commands from different users running different applications on the same machine. The virtualized operating system separately handles the users and their

requests. The most popular operating system level virtualization method is called containerization or container-based virtualization. Container-based virtualization approach is to deploy and run distributed applications without launching an entire virtual machine for each application (see Figure 1.1). Instead, multiple isolated systems, called containers, are run on a single control host and access a single kernel. Because containers share the same operating system kernel as the host, containers can be more efficient than virtual machines, which require separate operating system instances. Some popular implementations are Docker, Linux containers (LXC), and OpenVZ.

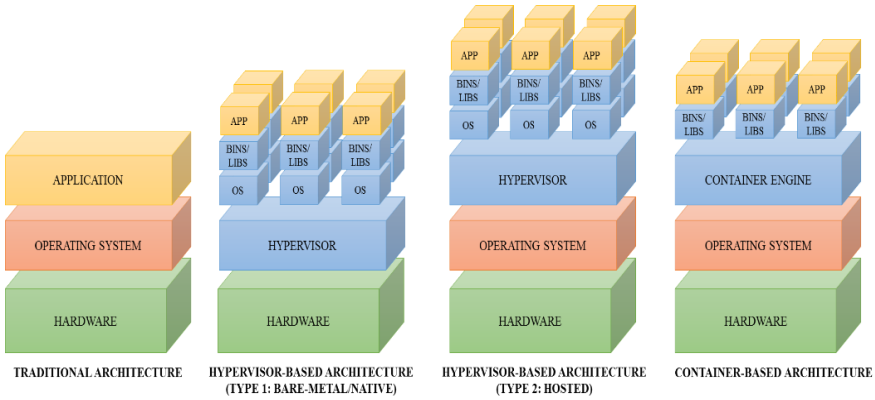


Figure 1.1. Traditional and Virtual Architecture

1.1.4 Network virtualization

Network virtualization is defined by the ability to create a logical software-based view of the underlying hardware and software networking resources (switches, routers, etc.). As a result, network virtualization can better integrate with and support increasingly virtual environments. Some of the well-known network virtualization vendors are Arista, Nicira, and Cisco.

1.1.5 Storage virtualization

Storage virtualization is the grouping the physical storage from multiple network storage devices into what appears to be a single storage device that is managed from a central console. The management of storage such as performing the tasks of backup, archiving, and recovery is becoming more difficult and time consuming. Storage virtualization helps to address this problem by hiding the actual complexity of storage area network (SAN). Storage administrators can implement storage virtualization with software applications or by using hardware and software hybrid appliances. Some of the benefits of storage virtualization include automated management, expansion of storage capacity, reduced time in

manual supervision, easy updates, and reduced downtime. Vendors of software defined storage includes Compuverde, RedHat's Ceph and Gluster storage, and VMware's vSAN.

1.2 Advantages and Disadvantages of Virtualization

Making the commitment to switching over to a virtualized IT environment can cause a fair amount of uncertainty among business owners. It is important to know the advantages and disadvantages of doing so before making the leap. To help with this, below is an overview of the key advantages and disadvantages of going to a virtual environment.

1.2.1 Advantages of virtualization

There are many good reasons for companies and organizations to invest in virtualization today. Below is an overview of the key benefits of virtualization.

1.2.1.1 Resource optimization

Many servers typically run for most of the day at low levels of utilization. With features like thin provisioning, memory transparent page sharing and Dynamic Resource Scheduler load balancing for CPU and Memory, the administrators are better able to fully utilize the hardware resources. By virtualizing the hardware and allocating parts of it based on the real needs of users and applications, the available computing power, storage space and network bandwidth can be used much more effectively.

1.2.1.2 Consolidation

Reduce datacenter costs by reducing the physical infrastructure. It is common practice to dedicate individual computers to a single application. If several applications only use a small amount of processing power, the administrator can consolidate several computers into one server running multiple virtual environments. For organizations that own hundreds or thousands of servers, consolidation can dramatically reduce the need for floor space, HVAC, A/C power, and co-location resources. This means the cost of ownership is reduced significantly, since less physical servers and floor and rack space are required, which in turn leads to less heat and power consumption, and ultimately a smaller carbon footprint.

1.2.1.3 Increased availability

Virtualization brings new opportunities to data center administration, allowing guaranteed uptime of servers and applications; speedy disaster recovery if large scale failures do occur. Instant deployment of new virtual machines or even aggregated pools of virtual machines via template images. Elasticity, that is, resource provisioning when and where required instead of keeping the entire data center in an always-on state. Reconfiguration of running computing environments without impacting the users. Server virtualization provides a way to implement redundancy without purchasing additional hardware. Redundancy, in the sense of running the same application on multiple servers, is a safety measure: if for any reason a server fails, another server running the same application takes over, thereby minimizing the interruption in service. This kind of redundancy works in two ways when applied to virtual machines, if one virtual system fails, another virtual system takes over or by running the redundant virtual machines on separate physical hardware you can also provide better protection against physical hardware failure.

1.2.1.4 Operational flexibility

Migration refers to moving a server environment from one place to another. With most virtualization solutions it is possible to move a virtual machine from one physical machine in the environment to another. With physical servers this was originally possible only if both physical machines ran on the same hardware, operating system and processor. In the virtual world, a server can be migrated between physical hosts with entirely different hardware configurations. Migration is typically used to improve reliability and availability: in case of hardware failure the guest system can be moved to a healthy server with limited downtime, if any. It is also useful if a virtual machine needs to scale beyond the physical capabilities of the current host and must be relocated to physical hardware with better performance.

1.2.2 Disadvantages of virtualization

The disadvantages of virtualization are mostly those that would come with any technology transition. With careful planning and expert implementation, all of these drawbacks can be overcome.

1.2.2.1 Upfront costs

For the providers of a virtualization environment, the implementation costs can be quite high for the servers and software licenses. Hardware and software are required at some point and that means devices must either be developed, manufactured, or purchased for implementation. This obstacle can also be more

readily navigated by working with a Managed IT Services provider, who can offset this cost with monthly leasing or purchase plans.

1.2.2.2 Servers and applications compatibility issues

Virtualization still has limitations, for example not every application or server is going to work within an environment of virtualization. That means an individual or corporation may require a hybrid system to function properly and since not every vendor supports virtualization and some may stop supporting it after initially starting it, there is always a level of uncertainty when fully implementing this type of system.

1.2.2.3 Data security risks

Because data is crucial to the success of a business, it is targeted frequently. The average cost of a data security breach in 2017, according to a report published by the Ponemon Institute, was $3.62 million. While a universal security model in a virtualized environment makes it easier, generally, to manage security, the virtual server might be on the same physical server as another company and that could open to risks.

1.2.2.4 Scalability issues

Server sprawl is one of the unintended consequences of virtualization. Although administrators can grow a business or opportunity quickly because of virtualization, they may not be able to become as large as they would like. They may also be required to be larger than they want to be when first starting out. Because many entities share the same resources, growth creates lag within a virtualization network. One large presence can take resources away from several smaller businesses and there would be nothing anyone could do about it.

1.2.2.5 Bleed over issues

Bleed over occurs when the contents of one virtual server affect other virtual servers. Bleed over issues are possible issues to be aware of when subscribing to a virtualized server.

1.3 Thesis Outline and Structure

Chapter 2 presents the Related Work, the Aim & Scope together with the Research Questions and the Methodology. In Chapter 3, the contributions are presented, and the results are discussed and concluded in Section 3.2 and 3.3

respectively. Finally, the proposed Future Work is presented in Section 3.3 and the publications are then presented in Chapters 4–11.

2 Approach

This chapter presents the Related Work for the publications in this thesis. Followed by Aim & Scope, Research Questions, and Research Methodology.

2.1 Related Work

Today, storage virtualization is set to play an important role in modern cloud computing infrastructures. In [1], the authors described the evolution of storage systems and presented a set of challenges driving research and development efforts in storage systems. Among the difficulties that storage administrators can encounter when managing storage for virtual environments are lowering the cost of storage, improving the performance, and increasing flexibility. Many types of distributed storage systems have emerged to solve these problems, such as commercial services, Google Drive [10], Dropbox [11], Compuverde [17] and Windows Azure Storage [12], open source system, HDFS [13], Ceph [14], GlusterFS [6], IOStack [9], OpenStack Swift [2] and Rackspace Cloud Files [15]. Storage can be divided into different types, such as traditional file and block storage technologies, object-based storage, software-defined storage (SDS) [4], and the latest data-defined storage [16].

There are few studies on implementing different techniques to improve the performance of the storage for instance, in [3], the authors implemented a bandwidth differentiation technique for OpenStack Swift that can control each data stream and guarantees a high utilization of the device. In [5], the authors proposed a solution to enable multilevel data fault tolerance on a single disk to reduce storage system energy consumption. In [8], the authors proposed a self-learning scheduler for OpenStack Cinder. In [18], the authors proposed an approach to separate the distributed storage's journal and data partitions and others to different storage pools in order to speed up the I/O operations.

Many studies exist that either directly or indirectly conducted a performance analysis or comparison between different storage solutions. For instance, in [7], the authors performed availability and sensitivity analysis on OpenStack Swift. In [19], the authors presented details on how different parameter and hardware configurations affect the performance of the OpenStack Swift. In [20], the authors reported performance results of Ceph in terms of latency and throughput during insert, read, update, and delete operations. In [21], the authors compared three storage solutions, Lustre, GlusterFS, and Hadoop. According to their results Hadoop performed better than Lustre and GlusterFS. In [22], the authors compared

performance of four different storage solutions, Amazon S3, Amazon Glacier, Windows Azure Blob, and Rackspace Cloud Files. According to their results all services show weaknesses related to some workload and there was no clear winner. In [23], the authors compared the performance of Google Drive, Dropbox, and OneDrive. According to their results, OneDrive has a good responsiveness to display learning object, Dropbox has a good performance to upload and to download the learning object, and Google Drive delivers good performance for manipulating the learning object. In [24], the authors compared features of different storage types as provided by Amazon Web Services, Windows Azure, and Google AppEngine. The focus of our work is to provide an extensive survey on differences between various storage types as well as a performance comparison between different storage systems (e.g., structured, unstructured, and software-defined storage) in terms of IOPS and response time as well as the throughput and recovery time during read, write, delete operations.

When it comes to the hardware virtualization, the de facto solution is to employ the hypervisor-based and the container-based technologies. In the hosted hypervisor-based virtualization the access is only provided to the physical hardware, and each virtual machine needs a complete implementation of a guest operating system including the binaries and libraries necessary for applications [35]. As a result, the guest operating system will automatically compete on resources with the applications running on the virtual machine, and essentially decreases the quality of service (QoS) from the application's perspective.

In recent years, there have been several efforts to compare performance of different hypervisors. For instance, in [25], the authors compared Vmware Server, Xen Server and OpenVZ, in terms of network utilization, SMP performance, file system performance, and MPI scalability. According to their results OpenVZ provides the best overall performance. In [28], the authors compared the performance of Xen and KVM. According to their results Xen outperformed KVM on a kernel compile test and KVM outperformed Xen on I/O-intensive tests. In [30], the authors compared performance of Hyper-V, KVM, vSphere, and Xen. According to their results there is no perfect hypervisor, and that different workloads may be best suited for different hypervisors. In [31], the authors compared Xen and KVM performance using Hadoop MapReduce. According to their results, KVM was better for disk reading. Xen was better when there was a combination of disk reading and writing with CPU intensive computations. In [32], VMware compared performance of Xen and VMware ESX Server with non-virtualized. According to their results, VMware ESX Server is far better equipped to meet the demands of an enterprise datacenter than the Xen hypervisor. There are some similarities between these studies and ours. However, none of these studies considered real-time telecommunication applications or NoSQL Cassandra database. In our work, we also considered various virtualization features, such as,

over-allocation of resources, VMware's distributed resource scheduler (DRS) and live migration.

To mitigate the performance overhead of hypervisor-based virtualization, researchers and practitioners recently started promoting container-based virtualization [35]. This technology is eventually evolved into virtualization mechanisms like Linux Vserver, OpenVZ and Linux Containers (LXC) [36], [37]. Unlike hypervisors, containers would be more resource efficient by excluding the execution of hypervisor and guest operating system, and more time efficient by avoiding booting (and shutting down) a whole operating system [38], [39]. Nevertheless, it has been identified that the cascading layers of container images come with inherent complexity and performance penalty [40]. In other words, the container-based virtualization technology could also negatively impact the corresponding quality of service (QoS) due to its performance overhead. Although the performance advantages of containers were investigated in several pioneer studies [25], [26], [36], [41], the container-based virtualization solution did not gain significant popularity until the recent underlying improvements in the Linux kernel, and especially until the emergence of Docker (Starting from an open-source project in early 2013) [42]. To examine the performance of Docker containers, a molecular modeling simulation software [43] and a PostgreSQL database-based Joomla application [44] have been used to benchmark the Docker environment against the VM environment. In [27], the authors compared energy efficiency of four hypervisors, KVM, Xen, VMware, and Hyper-V, as well as Docker container. According to their results, container virtualization which is light-weight in terms of system implementation and maintenance, is not more power efficient than hypervisors. In [29], the authors compared the performance of KVM, Docker and LXC with non-virtualized. According to their results the level of overhead introduced by containers can be considered almost negligible. The closest work to ours are [34], [45], and the IBM research report [33] on the performance comparison of VM and Linux containers. However, we recognized that these studies are incomplete (e.g., they did not consider non-CPU features, or did not finish the container's network evaluation). In addition, our work denies the IBM report's finding that "containers and VMs impose almost no overhead on CPU and memory usage" Furthermore, in addition to the average performance overhead of virtualization technologies, we are more concerned with their impact on the applications performance during horizontal or vertical scaling. There are also performance studies on deploying containers inside VMs (e.g., [46], [47]), however we did not include this virtualization scenario in this study.

There has been a significant amount of effort dedicated to optimizing the performance of virtual environments [48], [49], [50], [54]. However, there is comparatively lesser work in real-time scheduling of tasks with hard deadlines in virtualized environments [55]. In [51], the authors studied a constant bandwidth server (CBS) on top of Earliest Deadline First (EDF) for scheduling real-time tasks

with hard deadlines on virtual machines. The results show that virtual machine technology and scheduling algorithm can affect the real-time application performance. In [52], the authors developed a Compositional Scheduling Architecture (CSA) that is built on the Xen virtualization platform. The architecture allows timing isolation among virtual machines and supports timing guarantees for real-time tasks running on each virtual machine. In [53], the authors addressed the problem of scheduling hard real-time tasks with arbitrary deadlines on multiprocessors. In [56], the authors proposed a combination of the two scheduling algorithms (one which assigns priorities to tasks in a monotonic relation to their request rates and the other one was the dynamic deadline driven scheduling algorithm) which appears to provide most of the benefits of the deadline driven scheduling algorithm, and can be readily implemented in existing computers. However, none of these works considered how the overhead for switching from one virtual machine to another affects the schedulability of task set running in the virtual machine. In our method we considered virtual machines with one virtual core. However, it is easily extendable to virtual machines with multiple cores as long as each real-time task is allocated to one of the (virtual) cores. So, in that case the analysis needs to be repeated for each of the virtual cores to make sure that all real-time tasks on each core meet their deadlines.

2.2 Aim and Scope

The main focus of this thesis is how to optimize various platform virtualization software and shared storage in order to improve the performance of the real-time applications. This thesis aims to provide useful insights to system designers, as well as data center operators for real-time workload placement and virtual machine scheduling. Currently, a comparison of performance impacts of different hypervisors and containers on real-time applications is lacking. Therefore, the intention of this thesis is to extensively evaluate performance of different server virtualization solutions and virtualization features for instance, load balancing, resource scheduling, live migration, and scalability using a large real-time telecommunication application and NoSQL Cassandra database workload. In addition, performance evaluation and comparison of different storage virtualization solutions as well as real-time scheduling of tasks with hard deadlines are included in this study.

2.3 Research Questions

The main questions we explore in this thesis is: How to choose from various hardware and storage virtualization solutions in order to improve the performance of the real-time applications and at the same time maximize the utilization of resources. While investigating this question other challenges have risen. First, virtualization's performance overhead must be considered specially in case of

over-allocation of resources and the impact of the overhead on the performance of the real-time applications. Second, maturity of different technologies in terms of what virtualization features they offer for instance, load balancing, live migration, horizontal and vertical scaling as well as which has the lowest negative impact on the QoS of the real-time applications. The main research question has been approached using the following eight sub-research questions covered in this thesis. Research questions 2, 3, 4, 5, 6, and 7 investigating different hardware virtualization solutions and features. Optimization of storage virtualization solutions is Research question 1. Research question 8 is again optimizing scheduling algorithms for virtualization platforms.

2.3.1 Research question 1

What kind of shared storage do we need in terms of hardware and software to minimize performance bottlenecks?

Storage for virtualization has changed data centers in many ways. Many companies moved most of their data from local disk resources to some form of shared storage. Therefore, it is of interest to evaluate performance of different storage solutions for virtualization. We investigate this research question in Paper I, Section 4.

2.3.2 Research question 2

Which platform virtualization software should we use to get the best performance of the real-time applications?

There are number of vendors that provide hardware virtualization solutions. Every solution has its own pros and cons. Therefore, when organizations consider any of those options, they need to be aware of the tradeoff they are making between performance and virtualization flexibility. In Paper II (Section 5) we addressed this research question.

2.3.3 Research question 3

How the resource allocation parameters in the platform virtualization software should be configured to provide high utilization of the hardware resources, without affecting the performance of the real-time applications?

In a virtualized environment, the physical hardware is partitioned into virtual components. Resources like memory and CPU must be precisely allocated. Any deviations will result in either under-utilization of resources or allocating too many resources that can negatively impact the performance of the real-time applications.

Therefore, in paper III (Section 6) we investigate different resource allocation scenarios to find the most efficient and accurate scenario.

2.3.4 Research question 4

How well VMware's load balancer (DRS) performs compared to a human expert in terms of efficiently utilizing resources without affecting the performance of the real-time applications?

Load balancing is a key component of highly-available infrastructures. It used to improve the performance and reliability of applications by distributing the workload across multiple servers. It is of interest to observe how a load balancer work in general and investigate its impact on the performance of real-time telecommunication applications. This research question is addressed in Paper IV (Section 7).

2.3.5 Research question 5

How does resource over-allocation affects the performance of the real-time applications during live migration, in terms of CPU and disk utilization as well as the application response time?

Live migration and resource over-allocation technologies can contribute to efficient resource management in a cloud datacenter. However, live migration will inevitably entail downtime for the virtual machine involved. Even if the downtime is relatively short, its effect can be serious for real-time applications which are sensitive to response time degradations. So, the challenge is to over-allocate resources in a way to improve the performance of the real-time applications during live migration. We investigate this challenge in Paper V, Section 8.

2.3.6 Research question 6

Which hardware virtualization solution performs better, container-based, or hypervisor-based, in terms of how much performance overheads they produce and how this overhead affects the performance of NoSQL database applications?

Recently, virtual machines and containers received a widespread attention and many IT organizations overlooking how the two compare and contrast. Since containers are well-known for being light-weight and producing less performance overhead than virtual machines, it is of interest to compare the two technologies and their impact on the NoSQL database applications which are adopted increasingly by many organizations as a more cloud-friendly solution to their big data problems. In Paper VI (Section 9) we investigate this research question.

2.3.7 Research question 7

How different hardware virtualization solutions, container-based and hypervisor-based perform during horizontal scaling and how do they impact the performance of the NoSQL database applications?

One of the biggest advantages of virtualization in cloud is the scalability feature. There are two different approaches to accomplish scaling, vertical and horizontal. Vertical scaling is more costly compare to horizontal scaling and is limited by the fact that the virtual machines/containers can only get as big as the size of the server. However, horizontal scaling offers the ability to scale wider to deal with the traffic. The challenge with horizontal scaling is to calculate how many virtual machines/containers can be placed on the same server without negatively impacting the performance of the NoSQL database applications. In addition, it is of interest to understand how well containers and virtual machines scale, in terms of performance overhead produced by an extra instance added on the same server. This research question is addressed in Paper VII (Section 10).

2.3.8 Research question 8

How can we schedule a virtual machine containing a number of real-time applications with hard deadlines on the hypervisor level, in a way that all the tasks inside the virtual machine meet their deadlines?

This research question can be broken down into the following sub-questions:

2.3.8.1 Research question 8.1

How can we schedule a hypervisor that contains number of virtual machines in a way that, all the real-time applications with hard deadlines that are running inside these virtual machines meet their deadlines?

In virtualized environments performance of a virtual machine can be negatively affected by co-resident virtual machines which can result in missing deadlines for applications with hard deadlines. It is of interest to understand how scheduling is done in hypervisors and it can be optimized to avoid such negative impacts. We proposed an approach to solve this problem in Paper VIII, Section 11.

2.3.8.2 Research question 8.2

How the scheduling algorithm will be affected by the overhead that will be produced by switching from one virtual machine to another?

In real systems the hypervisor scheduler should consider a performance overhead while switching between virtual machines in order to not negatively affecting the applications with hard deadlines. In Paper VIII (Section 11), we proposed an approach to calculate for the performance overhead while scheduling virtual machines on hypervisor level.

2.4 Research Methodology

2.4.1 Experimental Study

We used comparative studies to address research questions 1, 2, 6 and 7. For research question 1 we did an experiment with three different distributed storage solutions (Compuverde vs. Gluster vs Openstack's Swift). Then we gathered the measurement data, we performed an analysis of the data and presented the results. For research question 2, we considered various hypervisors, VMware, KVM, and XenServer. We collected data related to the performance in terms of CPU and disk utilization as well as response time of a real-time telecommunication application. In addition, we tested the three hypervisors' live migration techniques. Here we also measured downtime and total migration time of different hypervisors and compared them with each other. For research question 6, we compared a hypervisor-based (VMware) and a container-based (Docker) virtualization solutions. We collected data related to the performance in terms of CPU and disk utilization as well as latency of the Cassandra NoSQL database application under write, read and mixed workloads. For research question 7, we focused on comparing horizontal scalability of the two solutions and presented our observations as well as analysis of the performance data under different scenarios.

For research question 3 we used VMware hypervisor and tested various scenarios for resource allocation. Here we gathered data from performance measurements, CPU and disk utilization as well as response time of a real-time telecommunication application. After we gathered all data, we conducted an analysis of the data and presented our observations.

For research question 4, we compared performance of VMware's automatic load balancer distributed resource scheduler (DRS) technology versus human expert manual decisions under various scenarios. We collected data related to the performance in terms of CPU utilization both before and after balancing the load as well as the number of application requests failure. We analyzed the collected data and presented our observations.

For research question 5, we used platform virtualization software (VMware) and tested various scenarios for resource over-allocation in combination with live migration using a real-time telecommunication application. Here we gathered data

from performance measurements, CPU utilization, downtime, and total migration time of the application. After we gathered all data, we analyzed the data and present our observations.

2.4.2 Theoretical and Simulation Study

For research question 8, we used simulation and defined a prediction model. We proposed a mathematical equation that helped us when scheduling hypervisors and we implemented the equation in a simulation application, using the Java programming language. We did a number of experiments in the simulated environment to verify that our technique can be applied in various scenarios. In addition, we proposed a prediction model where we included performance boundaries and compared these bounds with our simulation studies.

3 Results

3.1 Contributions

3.1.1 Contributions in Paper I

The main focus in Paper I (Section 4) is to evaluate four large distributed storage systems. Two of these use Distributed Hash Tables (DHTs) in order to keep track of how data is distributed, and two systems use multicasting to access the stored data. We measured the read/write/delete performance, as well as the recovery time when a storage node goes down. The evaluations are done on the same hardware, consisting of 24 storage nodes and a total storage capacity of 768 TB of data. These evaluations show that the multicast approach outperforms the DHT approach.

3.1.2 Contributions in Paper II

Paper II (Section 5) presents the results from a performance comparison of three well-known virtualization hypervisors: KVM, VMware and XenServer. In this paper, we measured performance in terms of CPU utilization, disk utilization and response time of a large industrial real-time application. The application runs inside a virtual machine (VM) controlled by the KVM, VMware and XenServer hypervisors, respectively. Furthermore, we compared the three hypervisors based on downtime and total migration time during live migration. The results show that the Xen hypervisor results in higher CPU utilization and thus also lower maximum performance compared to VMware and KVM. However, VMware causes more write operations to disk than KVM and Xen, and Xen causes less downtime than KVM and VMware during live migration. This means that no single hypervisor has the best performance for all aspects considered here.

3.1.3 Contributions in Paper III

In Paper III (Section 6), we presented that overall system performance is sensitive to appropriate allocation of resources (e.g., CPU), and there are other factors such as context-switch overhead and waiting in a queue that are affecting the performance. We observed that over-allocation of (virtual) CPU resources to virtual machines when there are many numbers of them running on one host in a virtualized environment is a big challenge. Thus, it is important to identify performance bottlenecks and avoid them when running in virtualized environment. The results of this study will help virtualized environment service providers to decide how much resources should be allocated for better performance as well as how much resource would be required for a given load.

3.1.4 Contributions in Paper IV

In Paper IV (Section 7), we evaluated VMware's Distributed Resource Scheduler (DRS) in a number of realistic scenarios using multiple instances of a large industrial telecommunication application. We also measured the performance on the hosts before and after the migration in terms of CPU utilization and compared automatic DRS migrations with manual human expert migrations. According to our results, DRS with the most aggressive threshold gave us the best results. It could balance the load in 40% of cases while in other cases it could not balance the load properly. DRS did completely unnecessary migrations back and forth in some cases. The results of this study should help IT organization to better understand how DRS works in general as well as how to configure migration thresholds in their environments to prevent DRS from additional vMotion activities.

3.1.5 Contributions in Paper V

In Paper V (Section 8), we conducted an experiment using a large telecommunication application that runs inside virtual machines, here we varied the number of vCPU resources allocated to these virtual machines in order to find the best choice which at the same time reduces the risk of under-allocating resources after the migration and increases the performance during the live migration. We used VMware's vMotion to migrate virtual machines while they are running. The results of this study should help virtualized environment service providers to decide how much resources should be allocated for better performance during live migration as well as how much resource would be required for a given load.

3.1.6 Contributions in Paper VI

In Paper VI (Section 9), we presented an extensive performance comparison between VMware and Docker container, while running Apache Cassandra as workload. Apache Cassandra is a leading NoSQL distributed database when it comes to Big Data platforms. As baseline for comparisons we used the Cassandra's performance when running on a physical infrastructure (non-virtualized). Our study shows that Docker had lower overhead compared to the VMware when running Cassandra. In fact, the Cassandra's performance on the Dockerized infrastructure was as good as on the Non-Virtualized.

3.1.7 Contributions in Paper VII

In Paper VII (Section 10), we compared performance differences caused by scaling of the different hardware virtualization technologies in terms of CPU utilization, latency, and the number of transactions per second. The workload is Apache Cassandra, which is a leading NoSQL distributed database for Big Data platforms. Our results show that running multiple instances of the Cassandra database concurrently, affected the performance of read and write operations differently; for both VMware and Docker, the maximum number of read operations was reduced when we ran several instances concurrently, whereas the maximum number of write operations increased when we ran instances concurrently.

3.1.8 Contributions in Paper VIII

In Paper VIII (Section 11), we described a technique for calculating a period and an execution time for a VM containing a real-time application with hard deadlines. This result makes it possible to apply existing real-time scheduling theory when scheduling VMs on the hypervisor level, thus making it possible to guarantee that the real-time tasks in a VM meet their deadlines. If overhead for switching from one VM to another is ignored, it turns out that (infinitely) short VM periods minimize the utilization that each VM needs to guarantee that all real-time tasks in that VM will meet their deadlines. Having infinitely short VM periods is clearly not realistic, and in order to provide more useful results we have considered a fixed overhead at the beginning of each execution of a VM. Considering this overhead, a set of real-time tasks, the speed of each processor core, and a certain processor utilization of the VM containing the real-time tasks, we present a simulation study and some performance bounds that make it possible to determine if it is possible to schedule the real-time tasks in the VM, and in that case for which periods of the VM that this is possible.

3.2 Discussion

In terms of storage virtualization, we compared two unstructured storage systems for Cloud Computing (Compuvede Unstructured and Openstack's Swift) and two structured storage systems for Cloud Computing (Compuverde Structured and Gluster). Compuverde uses multicasting and Openstack's Swift and Gluster use Distributed Hash Tables (DHTs). The architectural advantage of DHTs compared to multicasting is that they do not need to broadcast requests; the hash table gives them the address of the nodes that store of the requested data and it avoids communication overhead. However, the disadvantage with DHTs is that they need to run a hash function to obtain the address of the data, which introduces processing overhead. This means that the architectural decision, whether to use DHTs or multicasting will introduce different kinds of overhead: processing overhead for DHTs and communication overhead for multicasting. We believe that the main reason for the higher performance of Compuverede compared to Gluster and Openstack's Swift is that Compuverde uses multicast instead of DHTs. The communication overhead introduced by multicasting does not affect the performance as negatively as the processing overhead introduced by DHTs. The recovery tests show that Compuverde recovers from a storage node failure much faster than OpenStack's Swift and Gluster. One additional reason for Gluster to perform slower than Compuverde Structure could be that Gluster involves proxy servers in self-healing while Compuverde uses the many-to-many replication pattern and only involves storage nodes in self-healing. Another reason could be that Compuverde has built its own recovery protocol from scratch, whereas OpenStack's Swift and Gluster base their protocols on existing applications (e.g., rsync). Moreover, the processor utilization for Gluster never exceeds 50%, even for high loads. This indicates that there are internal performance bottlenecks in Gluster, which probably contributes to the relatively long time for self-healing.

Further in the thesis we investigate impacts of different hardware virtualization solutions on performance of real-time applications. The results of the performance tests indicate that KVM and VMware performed better in terms of CPU utilization compared to Xen. While, in terms of disk utilization, KVM and Xen had similar performance while VMware had the highest disk utilization. In terms of response time of the real-time application, Xen had the longest response times compared to KVM and VMware. We believe that the method that a hypervisor uses to accurately emulate the physical hardware to prevent guests from accessing it except under carefully controlled circumstances is a key factor in its performance. For instance, Type 1 hypervisors (KVM and Xen) avoid the overhead that a Type 2 hypervisor (VMware ESXi) incurs when it requests access to physical resources from the host OS. However, other factors also play an important role in a hypervisor's performance. For example, VMware ESXi generally requires more time to create and start a server than KVM and Xen. In addition, we believe that the type of workload is a very important factor as well and different virtualization

solutions may behave differently if the type of workload/application changes. The work presented in this thesis is limited to a test of only one real-time application, therefore our recommendation for IT organizations is to conduct a similar test using their application instead before they decide which type of hypervisor to use.

In the thesis, we tested live migration technologies of different hypervisors (KVM, VMware ESXi and Xen). The results indicate that Xen's live migration technology, XenMotion, performed better than VMware's vMotion and KVM live migration technology in terms of downtime and total migration time. further, we investigate different ways of using the physical CPU resources and how it can affect system performance. In virtualized environments the number of virtual CPUs (vCPUs) does not need to match the number of CPUs (or CPU cores) on the physical server. So, if we sum up the number of vCPUs in all virtual machines on a physical server, we could end up in three situations: the total number of vCPUs exceeds the number of physical CPU cores (over-allocation), the total number of vCPUs is the same as the number of physical CPU cores (balanced allocation), or the number of vCPUs is smaller than the number of physical CPU cores (under-allocation). Resource allocation is very important especially during live migration of virtual machines. Most of the time the source host will become under-allocated and the destination host will become over-allocated. It is clear that under-allocation will result in sub-optimal resource utilization since some physical CPU cores will not be used. Therefore, we conbined the live migration of VMware ESXi vMotion and over-allocation of CPU resources in order to figure out how different allocation of vCPUs to each virtual machine will affect the performance during live migration. Also, investigated how other virtual machines in the background are affected during the live migration of one virtual machine from the source host to the destination host. We hope that these studies can be helpful in optimizing existing live migration mechanisms.

VM migrations can be used for balancing the utilization of server host resources in order to avoid having heavily loaded hosts while lightly loaded servers are available. Load balancing helps to maximize resource by optimizing the mapping of VMs to hosts. We considered five different test scenarios, and three different loads to test VMware distributed resource scheduler (DRS) under its three different levels of aggressiveness. This means that we looked at in total 5*3*3 = 45 cases. In 23 of these cases DRS did nothing. In 11 cases it did (more or less) the same decision as a human expert. And in 7 cases DRS balanced the load to some extent, but in 4 cases DRS suffered from the "ping-pong" effect and did completely unnecessary migrations back and forth. We observed that when the migration threshold was set to level 1 (conservative) or level 3 (moderate/default) DRS did not migrate any virtual machine in most of the cases. However, in some cases, DRS started to migrate virtual machines even when the migration threshold was set to level 1 (conservative). One reason could be that VMware designers considered similar test scenarios when they designed the DRS algorithm (evacuating a physical machine for maintenance is a very common scenario); however, for some

more complex test scenarios VMware's DRS was unreliable, according to our results. Overall if we compare the system performance after DRS migration we can observe that we obtained better results - more close to human expert migrations - with the aggressive threshold (level 5). In 15 cases out of 45 cases where we used level 5, we got no migrations in two cases, good (human expert quality) migrations in 7 cases, reasonably good migrations in three cases, and the undesirable "ping-pong" effect in three cases. So even if the migrations are better in general using level 5, the risk of suffering from the "ping-pong" effect is also considerably higher compared to the other levels of aggressiveness. One very important factor here is to understand how well the application can handle the migrations and how often the migration should happen. For some real-time applications having too many migrations in a short time may cause request failures and disturbance in the application performance. Therefore, it is important for administrators of big data centers to investigate different migration thresholds and always select the one that results in an overall improvement in resource utilization and load balancing while at the same time does not negatively impacting the application performance.

When it comes to deciding between container-based and hypervisor-based virtualization, we believe that it is important to look at the "scope" of the work, as others have suggested it as well. For instance, if one would like to run multiple instances of the same application can benefit from using container-based virtualization, however, if one is interested in running multiple applications that require different operating systems, hypervisor-based virtualization is recommended. Even though container solution is showing very low overhead and system resource consumption, it suffers from securing stored data which is crucial for database protection. Comparing containers architecture with virtual machines, containers cannot be secure candidate for databases because all containers share the same kernel and are therefore less isolated than virtual machines. A bug in the kernel affects every container and results in data loss. On the other hand, hypervisor-based virtualization is a mature and secure technology. Virtual machines are able to partition and distribute resources viably in the hypervisor without relying on kernel support or separate hardware. We investigate in the thesis, performance of NoSQL Cassandra database on both containers and virtual machines. According to our results, hypervisor-based virtualization suffers from noticeable overhead which effects the performance of the databases. Since both containers and virtual machines have their set of benefits and drawbacks, an alternative solution could be to combine the two technologies to get the benefits of both security of the virtual machine with the execution speed of containers.

Scalability is also the major discussion point in designing the infrastructure. Today's applications need t scale based on the demand. Containers and virtual machines have their own advantages and limitation when it comes to scalability. There is always a difficult decision to decide which technology to choose considering requirements of different applications. In this study, we tested both technologies in terms of their scalability and their impact on the performance of a NoSQL database. We observed that running multiple instances of the Cassandra database concurrently, affected the performance of read and write operations

differently; for both VMware and Docker, the maximum number of read operations was reduced when we ran several instances concurrently, whereas the maximum number of write operations increased when we ran instances concurrently. In general, according to our results, running Cassandra inside multiple clusters of VMware virtual machines was showing less performance in terms of maximum number of transactions per second compared to the Docker containers.

Further in the thesis, we investigate hypervisor scheduling of virtual machines considering real-time applications with hard deadlines. Based on an existing real-time application and the processor speed of the physical hardware, we calculated a period and an execution time such that the existing real-time application meets all deadlines when it is executed in a virtual machine. We show that if overhead for switching from one virtual machine to another is ignored, it turns out that (infinitely) short virtual machine periods minimizes the utilization that each virtual machine needs to guarantee that all real-time tasks in that virtual machine meet their deadlines. Having infinitely short virtual machine periods is clearly not realistic, and in order to provide more useful results we consider a fixed overhead at the beginning of each execution of a virtual machine. Considering this overhead, a set of real-time tasks, the speed of each processor core, and a certain processor utilization of the virtual machine containing the real-time tasks, we presented a simulation study and some performance bounds that made it possible to determine if it is possible to schedule the real-time tasks in the virtual machine, and in that case for which periods of the virtual machine that this is possible. We based our calculations on the case when we used static priorities, and thus RMS, in the original real-time applications. Our approach can easily be generalized to cases when other scheduling policies, such as EDF, are used in the original real-time applications. The work presented in this thesis is limited to cover only processor, however with some modification this approach can be used for multiprocessor scenarios as well.

3.3 Conclusion and Future Work

This thesis investigates how different virtualization solutions affect the performance of various applications. First, we evaluated different storage virtualization solutions and compared their I/O performance as well as their recovery time (when one node goes down). According to our study, Compuverde outperforms the Gluster and Openstack solutions in terms of read/write/delete process as well as recovery test. In the future studies, it would be interesting to compare Compuverde with other solutions such as Hadoop, Lustre, EMC Atmos, etc.

Second, we measured the performance of three different hardware virtualization solutions in terms of CPU utilization, disk utilization and response time using a large real-time industrial application. In addition, we measured the performance of

these three hypervisors in terms of downtime and total migration time during the live migration and compared their results. According to the results, both VMware and KVM perform better in terms of application response time and CPU utilization. Xen's performance was below that of the two other virtualization systems tested. However, Xen's live migration technology, XenMotion, performed better than VMware's vMotion and KVM live migration technology in terms of downtime. For this experiment we used only two servers and maximum four virtual machines (two virtual machines on each server), it would be interesting to scale up the environment and run the same sort of tests and performance measurements when there are number of servers and many number of virtual machines. Also, it would be very interesting to test the live migration in that environment to see how scalable these three different hypervisors are and how powerful their live migration technique has been designed.

Further, we quantified the cost of having different amounts of over-allocation. Providers of virtualized service can use this quantification in order to do a balanced trade-off between the flexibility offered by over-allocation and the performance penalty. Our results indicate that it is in many cases wise to use a moderate level of over-allocation (not exceeding a factor of two) which gives some flexibility at a very modest performance cost. Our measurements also show that the write latency decreases with the number of VMs sharing the physical server (seen indirectly as by considering that the amount of writes to disk increases with the number of VMs sharing the same physical server - due to write throttling in the application). Over-allocation increases the write latency (seen indirectly as by considering that the amount of writes to disk increases with over-allocation - due to write throttling in the application). There was no clear connection between the application level response time with neither the number of VMs nor with the degree of over-allocation. To investigate over-allocation more, we measured the performance in terms of CPU utilization, migration down time and total migration time of a large telecommunication application during the live migration. According to our results, over-allocation has a small effect on the CPU utilization of the low loaded VMs, while it highly effects the downtime and total migration time. However, once we have reached a certain amount of over-allocation, then having more over-allocation does not have noticeable effect even on the downtime and the total migration time. Also, we show that live migration of a heavy loaded VM when the amount of over-allocation is medium or massive increases the risk of getting request failures especially for large real-time applications.

In the thesis, we compared VMware's DRS migrations versus human expert migrations using various realistic test scenarios. We show that there is still considerable room for improvement of VMware's state-of-the-art DRS load balancing systems. In particular, load balancing needs to be more robust in the sense that completely unnecessary migrations should be avoided.

To answer research question 6, we tried to address the problem of which solution is better for distributed databases such as Cassandra, Non-Virtualized,

Virtualized (VMware) or Docker? The overall result showed that the biggest issue with running the Cassandra-Virtualized, is the significant resource and operational overheads of the virtualization layer which affects the performance of the application too. According to the results, Cassandra-Dockerized consumed fewer resources and operational overheads compared to the Cassandra-Virtualized. In addition, the performance of Cassandra-Dockerized was as good as Cassandra-Non-Virtualized. Since both containers and virtual machines have their set of benefits and drawbacks, an alternative solution could be to combine the two technologies. In the future, it would be of interest to investigate the alternative solution by running containers inside virtual machines running Cassandra workload. In this way, IT organization can get the benefits of both security of the virtual machine with the execution speed of containers. Further in the thesis, we compared the performance of running multiple clusters of the NoSQL Cassandra database inside Docker containers and VMware virtual machines. We measured the performance in terms of CPU utilization, Latency mean and the maximum number of Transactions Per Second (TPS). According to our results, running Cassandra inside multiple clusters of VMware virtual machines was showing less performance in terms of maximum number of transactions per second compared to the Docker containers. The overall performance difference was around 20% lower during the mixed workload, around 16% lower during the write-only workload and around 29% lower during read-only workload. As it has been discussed before in general the read-only workload is showing less performance than the write-only workload, and the impact of the different types of workloads on the performance in terms of CPU utilization is higher on virtual machines than containers. However, considering the scalability aspects of the virtual machines and the containers, according to our results, containers scale better without losing too much performance while virtual machines overhead is very high, and it has a negative impact on the performance of the application. This might differ depending on the application and the type of workload as we have seen during our experiments. Therefore, cloud providers need to investigate this issue while deploying both virtual machines and containers across data centers also at larger scale.

To answer research question 8, we proposed a new algorithm for calculating a period and an execution time for a virtual machine containing a real-time application with hard deadlines. We also presented a simulation study and some performance bounds that made it possible to determine if it is possible to schedule the real-time tasks in the virtual machine, and in that case for which periods of the virtual machine that this is possible. This result makes it possible guarantee that the real-time tasks in a virtual machine meet their deadlines. If overhead for switching from one virtual machine to another is ignored, it turns out that (infinitely) short virtual machine periods minimize the utilization that each virtual machine needs to guarantee that all real-time tasks in that virtual machine will meet their deadlines. Having infinitely short virtual machine periods is clearly not realistic, and in order to provide more useful results we have also considered a fixed overhead at the beginning of each execution of a virtual machine. In our study we considered non-preemptive scheduling; however, it would be interesting to modify this algorithm

in order to make it work as a pre-emptive scheduling algorithm. In the future, it would be interesting to execute this algorithm on in a real environment and see how it will perform.

3.4 References

[1] R. J. T. Morris, and B. J. Truskowski, “The Evolution of Storage Systems”, IBM Systems Journal, vol.24, issue.2, pp. 205-217, 2003.

[2] openStack, “OpenStack”, docs.openstack.org.

[3] R. Nou, A. Miranda, M Siquier, T. Cortes, ”Improving OpenStack Swift Interactions with I/O Stack to Enable Software Defined Storage”, published in SC2 conference, pp. 63-70, 2017.

[4] G. Kandiraju, H. Franke, M. Williams, M. Steinder, and S. Black, “Software Defined Infrustructure”, IBM journal of Research and Development, vol.58, issue.2, pp. 2:1-2:13, 2014.

[5] Sh. Chen et al., “Utilizing Multi-Level Data Fault Tolerance to Converse Energy on Software-Defined Storage”, published in SmartCloud conference, pp. 1-6, 2017.

[6] GlusterFS. Glsterfs @ONLINE, https://www.gluster.org/, 2018.

[7] M. D. Mauro, M. Longo, F. Postiglione, G. Carullo, M. Tambasco, “Software defined storage: Availability Modeling and Sensitivity Analysis”, published in SPECTS conference, pp. 1-7, 2017.

[8] B. Ravandi, and I. Papapanagiotou, “A Self-Learning Scheduling in Cloud Software Defined Block Storage”, published in CLOUD conference, pp. 415-422,2017.

[9] R. Gracia-Tinedo et al., “IOStack: Software-Defined Object Storage”, published in IEEE Internet Computing Journal, vol. 20, issue. 3, pp. 10-18, 2016.

[10] Ch. Chu et al., “Security Concerns in popular Cloud Storage Services”, published in IEEE Pervasive Computing, vol. 12, issue. 4, pp. 50-57, 2013.

[11] R. Gracia-Tinedo et al., “Actively Measuring Personal Cloud Storage”, published in CLOUD conference, pp. 301-308, 2013.

[12] B. Calder et al., “Windows Azure Storage: A Highly Available Cloud Stoage Service with Storage Consistency”, published in SOSP conference, pp.143-157, 2011.

[13] K. Shvachko, H. Kuang, S. Radia, R. Chansler, ”The Hadoop Distributed File System”, published in MSST conference, pp. 1-10, 2010.

[14] S. A. Weil, S. A. Brandit, . L. Miller, D. D. E. Long, and C. Matzahn, “Ceph: A Scalable, High-Performance Distributed File System”, published in OSDI conference, pp. 1-14, 2006.

[15] E. Bocchi, M. Mellia, S. Sarni, ”Cloud Storage Service Benchmarking: MEthodologies and Experimentations”, pp. 395-400, 2014.

[16] Tarmin, Tarmin@ONLINE, https://www.tarmin.com/, 2018.

[17] Compuverde, Compuverde@ONLINE, http://compuverde.com/, 2018.

[18] M. Karim, L. Tuan, W. Ming-Tat, and H. Ong, “Improving Performance of Databse Appliances on Distributed Object Storage”, published in ICCCRI conference, pp. 45-52, 2015.

[19] L. Li, D. Li, Z. Su, L. Jin, and G. Huang, ”Performance Analysis and Framework Optimization of Open Source Cloud Storage System”, published in China Communications journal, vol. 13, issue. 6, pp. 110-122, 2016.

[20] J. Lee, C. Song, and K. Kang, ”Benchmarking Large-Scale Object Storage Servers”, published in COMPSAC conference, pp. 594-595, 2016.

[21] W. Chen and C. Liu, “Performance Comparison on the Heterogeneous File System in Cloud Storage System”, published in CIT conferenc, pp. 694-701, 2017.

[22] E. Bocchi, M. Mellia, and S. Sarni, ”Cloud Sorage Service Benchmarking: Methodologies and Experimentations”, published in CloudNet conference, pp. 395-400, 2014.

[23] R. Ferdiana, “The Comparison of Consumer Cloud Storage for a Storage Extension on the E-Learning”, published in InAES conference, pp. 1-5, 2016.

[24] H. Dewan, and R. C. Hansdah, “A Survey of Cloud Storage Facilities”, published in SERVICES conference, pp. 224-231, 2011.

[25] J. Walters, V. Chaudhary, M. Cha, S. Guercio, and S. Gallo, “A Comparison of Virtualization Technologies for HPC”, published in AINA conference, pp. 861-868, 2008.

[26] J. Walters, et al., “GPU Passthrough Perfromance: A Comparison of KVM, Xen, VMware ESXi, and LXC for CUDA and OpenCL Applications”, published in CLOUD conference, pp. 636-643, 2014.

[27] C. Jiang et al., “Energy Efficiency Comparison of Hypervisors”, published in IGSC conference, pp. 1-8, 2016.

[28] T. Deshane et al., “Quantitative Comparison of Xen and KVM”, published in Xen Summit, pp. 1-3, 2008.

[29] R. Morabito, J. Kjallman, and M. Komu, “Hypervisors vs. LightWeight Virtualization: A Performance Comparison”, published in IC2E, pp. 386-393, 2015.

[30] J. Hwang, S. Zeng, F. Wu, and T. Wood, “A Component-Based Performance Comparison of Four Hypervisors”, published in IFIP conference, pp. 269-276, 2013.

[31] J. Li et al., “Performance Overhead Among Three Hypervisors: An Experimental Study using Hadoop Benchmarks”, published in BigData congress, pp. 9-16, 2013.

[32] VMware, “A Performance Comparison of Hypervisors” @ONLINE, https://www.vmware.com/pdf/hypervisor_performance.pdf, 2018.

[33] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An Updated Performance Comparison of Virtual Machines and Linux Containers”, published in ISPASS conference, pp. 171-172, 2015.

[34] Z. Li, M. Kihi, Q. Lu, and J. Andersson, ”Performance Overhead Comparison between Hypervisor and Container Based Virtualization”, published in AINA conference, pp. 955-962, 2017.

[35] D. Bernstein, “Containers and Cloud: From LXC to Docker to Kubernetes”, published in Cloud Computing journal, vol. 1, issue. 3, pp. 81-84, 2014.

[36] M. G. Xavier, M. V. Neves, and c. Rose, “A Performance Comparison of Container-Based Virtualization for MapReduce Clusters”, published in PDP conference, pp. 299-306, 2014.

[37] C. Pahl, “Containerization and the PaaS Cloud”, published in Cloud Computing journal, vol. 2, issue. 3, pp.24-31, 2014.

[38] D. Merkel, “Docker: Lightweight Linux Containers For Consistent Development And Deployment”, published in Linux journal, vol. 239, pp. 76-91, 2014.

[39] C.Anderson, “Docker”, published in Software journal, vol. 32, issue. 3, pp.102-105, 2015.

[40] T. Banerjee, “Understanding The Key Differences Between LXC And Docker”, @ONLINE, https://archives.flockport.com/lxc-vs-docker/, 2018.

[41] J. Che, C. Shi, Y. Yu, and W. Lin, “A Synthetical Performance Evaluation Of Openvz, Xen And KVM”, published in APSCC conference, pp. 587-594. 2010.

[42] D. Strauss, “Containers - Not Virtual Machines - Are The Future Cloud”, published in Linux journal, vol. 228, pp. 118-123, 2013.

[43] T. Adufu, J. Choi, and Y. Kim, “Is Container-Based Technology A Winner For High Performance Scientific Applications?”, published in APNOMS conference, pp. 507-510, 2015.

[44] A. M. Joy, “Performance Comparison Between Linux Containers And Virtual Machines”, published in ICACEA conference, pp. 507-510, 2015.

[45] K.-T. Seo, H.-S. Hwang, I.-Y. Moon, O.-Y. Kwon, and B.-J. Kim, “Performance Comparison Analysis Of Linux Container And Virtual Machine For Building Cloud”, published in Advanced Science and Technology Letters journal, vol. 66, pp. 105-111, 2014.

[46] R. Dua, A. R. Raja, and D. Kakadia, “Virtualization Vs Containerization To Support PaaS”, published in IC2E conference, pp. 610-614, 2014.

[47] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, “Efficient Virtual Machine Sizing For Hosting Containers As A Service”, published in SERVICES conference, pp. 31-38, 2015.

[48] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel, “Diagnosing Performance Overheads In The Xen Virtual Machine Environment”, published in VEE conference , pp. 13-23, 2005.

[49] P. Apparao, R. Iyer, X. Zhang, D. Newell, and T. Adelmeyer, “Characterization & Analysis Of A Server Consolidation Benchmark”, published in VEE conference, pp. 21-30, 2008.

[50] A. Menon, A. L. Cox, and W. Zwaenepoel, “Optimizing Network Virtualization In Xen”, published in ATEC conference, pp. 1-14,2006.

[51] T. Cucinotta, G. Anastasi, and L. Abeni, “Respecting Temporal Constraints In Virtualised Services”, published in COMPSAC conference, pp. 73-78, 2009.

[52] J. Lee, S. Xi, S. Chen, L. T. Phan, C. Gill, I. Lee, C. Lu, and O. Sokolsky, “Realizing Compositional Scheduling Through Virtualization”, published in RTAS conference, pp. 13-22, 2012.

[53] S. Ramamurthy, “Scheduling Periodic Hard Real-Time Tasks with Arbitrary Deadlines on Multiprocessors”, published in RTSS conference, pp. 1-10, 2002.

[54] A. Magalhaes, M. Rela, and J. Silva, ”Deadlines in Real-Time Systems”, technical report, pp. 1-20, 1993.

[55] L. Briand, and D. Roy, “Meeting Deadlines in Hard Real-Time Systems”, book published by IEEE Computer Society Press Los Alamitos, ISBN. 0818674067, 1997.

[56] C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment”, published in JACM journal, vol. 20, issue. 1, pp. 46-61, 1973.

4 Performance Evaluation of Distributed Storage Systems for Cloud Computing

Abstract

The possibility to migrate a virtual server from one physical computer in a cloud to another physical computer in the same cloud is important in order to obtain a balanced load. In order to facilitate live migration of virtual servers, one needs to provide large shared storage systems that are accessible for all the physical servers that are used in the cloud. Distributed storage systems offer reliable and cost-effective storage of large amounts of data and such storage systems will be used in future Cloud Computing. We have evaluated four large distributed storage systems. Two of these use Distributed Hash Tables (DHTs) in order to keep track of how data is distributed, and two systems use multicasting to access the stored data. We measure the read/write/delete performance, as well as the recovery time when a storage node goes down. The evaluations are done on the same hardware, consisting of 24 storage nodes and a total storage capacity of 768 TB of data. These evaluations show that the multicast approach outperforms the DHT approach.

4.1 Introduction

The possibility to migrate a virtual server from one physical computer in a cloud to another physical computer in the same cloud is important in order to obtain a balanced load. In order to facilitate live migration of virtual servers, one needs to provide large shared storage systems that are accessible for all the physical servers that are used in the cloud. This is an important reason why the demand for storage capacity has increased rapidly during the last years.

One problem with traditional disk drives is that data losses are common due to hardware errors. A solution to this is Redundant Array of Independent Disks (RAID) storage. RAID storage systems can automatically manage faulty disks without losing data, and scale by attaching new disk drives. However, the scalability of RAID is too limited for large cloud systems; this limitation is the main reason for using distributed storage systems.

Distributed storage systems should be capable of sustaining rapidly growing storage demands, avoid loss of data in case of hardware failure, and they should provide efficient distribution of the stored content [2]. Two examples of distributed storage systems are OpenStack's Swift and Gluster. We have evaluated the performance of three distributed storage systems: Compuverde, OpenStack's

Swift, and Gluster. Openstack's Swift and Gluster are both open-source distributed storage systems that are available for downloading and testing.

Some distributed storage systems use Distributed Hash Tables (DHTs) for mapping data to physical servers. In the DHT approach file names and addresses are run through a hashing function in order to indentify the nodes that have the requested data. Two examples of systems that use DHTs are Gluster and OpenStack's Swift [22]. An alternative approach to using DHTs is to use multicasting where data requests are sent to multiple storage nodes and the nodes that have the requested data answer. Compuverde uses the multicast approach. The architectural advantage of DHTs compared to multicasting is that we do not need to broadcast requests; the hash table gives us the address of the nodes that store the requested data and we avoid communication overhead. However, the obvious disadvantage with DHTs is that we need to run a hash function to obtain the address of the data, which introduces processing overhead. This means that the architectural decision, whether to use DHTs or multicasting will introduce different kinds of overhead: processing overhead for DHTs and communication overhead for multicasting. Using DHTs or multicasting is a key architectural decision in distributed storage systems for Cloud Computing and this performance evaluation will give important insights regarding the performance implications of this decision.

4.2 Background

In distributed storage systems, the most common interfaces are Web Service APIs (Application Programming Interface) like Internet Small Computer System Interface (iSCSI) [25]; REpresentational State Transfer (REST)-based [23, 24] and Simple Object Access Protocol (SOAP)-based [27]. REST is a HTTP-based architectural style to build networked applications that allows access to stored objects by an Object Identifier (OID), i.e., no file or directory structures are supported [4]. We will refer to object-based storage systems as *unstructured storage systems*.

There are other access methods like Network File System (NFS) and Common Internet File System (CIFS) which are used for accessing storage on a private network or LAN and Web-based Distributed Authoring and Versioning (WebDAV) which is based on HTTP. These APIs are file-based (variable-size) and use a path to identify the data; we denote these as *structured storage systems*. The architecture of structured storage systems is similar to Network Attached Storage (NAS) which provide file system functionality, i.e., structured storage systems support variable file and directory structures [3, 21].

The most well-known distributed storage systems are Amplistor [5, 30], Caringo's CAStor[6, 31], Ceph [7], Cleversafe, Compuverde , EMC Atmos [8],

Gluster [9], Google File System [10], Hadoop [11, 20], Lustre [12], OpenStack's Swift [19], Panasas [13], Scality and Sheepdog. Some of the distributed file systems could be used by other applications, i.e., BigTable is a distributed storage for structured data and it uses GFS to store log and data files [36].

As shown in Table 4.1 AmpliStor, CAStor, Ceph, Cleversafe and Scality are unstructured distributed storage systems. Amplistor is designed to work with HTTP/REST. Just as in Amplistor, CAStor's Simple Content Storage Protocol (SCSP) is based on HTTP using a RESTful architecture [34]. Ceph provides an S3-compatible REST interface that allows applications to work with Amazon's S3 service. Cleversafe provides an iSCSI device interface, which enables users to transparently store and retrieve files as if they were using a local hard drive.

EMC Atmos is a structured distributed storage system that provides CIFS and NFS interfaces, as well as web standard interfaces such as SOAP and REST. Other distributed file systems such as Google File System, Hadoop Distributed File System (HDFS), Lustre and Panasas provide a standard POSIX API. Sheepdog is the only distributed storage system which is based on Linux QEMU/KVM and is used for virtual machines.

Some of the distributed file systems are also used for computing purposes, e.g., the Hadoop Distributed File System (HDFS) which distributes storage and computation across many servers. HDFS stores file system metadata and application data separately and users can reference files and directories by paths in the namespace (a HTTP browser can be used to browse the files of an HDFS instance) [35]. Lustre is an object-based file system used mainly for computing purposes. The Lustre architecture is designed for High Performance Computing (HPC). Panasas is also used for computing purposes and similar to Lustre, it is designed for HPC.

Scality uses a ring storage system which is based on a Distributed Hashing Mechanism with transactional support and failover capability for each storage node. The Sheepdog architecture is fully symmetric and there is no central node such as a meta-data server (Sheepdog uses the Corosync cluster engine [32] to avoid metadata servers). Sheepdog provides an object (variable-sized) storage and assigned a global unique id to each object. In Sheepdog's object storage, target nodes calculated based on consistent hashing algorithm which is a schema that provides hash table functionality and each object is replicated to 3 nodes to avoid data loss [33].

The remaining distributed storage systems in Table 4.1 are Compuverde, Gluster and OpenStack's Swift. We have ported these three systems to the same hardware platform (see Section 4.3), thus making it possible to compare their

Table 4.1 : Overview of different distributed storage systems

	INTERFACE				SOLUTION		REPLICATION		METADATA	
	Unstructured		Structured							
	Web Service APIs (REST, SOAP)	*Block-based APIs (iSCSI)*	*File-based APIs (CIFS, NFS)*	*Other APIs (WebDAV, FTP, Proprietary API)*	DHT	Multicast	Copying	Striping	Centralized	Distributed
AmpliStor	X	-	-	-	-	-	-	X		X
Caringo's CAStor	X	-	X	-	-	X	X	-	X	-
Ceph	X	-	-	-	X	-	-	X	-	X
Cleversafe	-	X	-	-	-	-	-	X	X	-
Compuverde	X	-	X	X	-	X	X	-	-	X
EMC Atmos	X	-	X	-	X	-	-	X	-	X
Gluster	-	-	X	X	X	-	X	-	-	-
Google File System (GFS)	X	-	X	-	-	-	-	X	X	-
Hadoop	-	-	X	-	X	-	-	X	X	-
Lustre	-	-	X	-	X	-	-	X	X	-
OpenStack's Swift	X	-	-	-	X	-	X	-	-	X
Panasas	-	-	X	-	-	-	-	X	-	X
Scality	X	-	-	-	X	-	X	-	-	X
SheepDog	-	X	-	-	X	-	X	-	-	X

performance (see sections 4.4 and 4.5). In subsections 4.2.1, 4.2.2, and 4.2.3, we discuss these three systems in detail.

Distributed storage systems use either multicasting or Distributed Hash Tables (DHTs). Data redundancy is obtained by either using multiple copies of the stored files or by so called striping using Reed-Solomon coding [1]. When using striping the files are split into stripes and a configurable number of extra stripes with redundancy information are generated. The stripes (in case of Striping) and file copies (in case of Copying) are distributed to the storage nodes in the system.

4.2.1 Compuverde

Compuverde has no separate metadata. The system uses its own proprietary caching mechanism (SSD Caching that employs Write-back policy) [26] in the storage nodes. The solution uses multicasting, and supports geographical dispersion, heartbeat monitoring, versioning, self-healing and self-configuring. Compuverde supports a flat 128 bit addresses space (for unstructured storage) and NFS/CIFS (for structured storage). The system supports both Linux and Windows. Compuverde's storage solution consists of two parts: The first part is unstructured and it contains all storage nodes (clusters). The other part is the structured part of the storage solution. This part contains gateways (this corresponds to what OpenStack calls proxy servers) to communicate with storage nodes. The communication is based on TCP unicast and UDP multicast messages. Structured data storage is achieved by storing information about the structure in envelopes. An envelope is an unstructured file that is stored on the storage nodes and contains information about other envelopes and other files.

The storage cluster provides mechanisms for maintaining scalability and availability of the structured data by replicating the envelopes a (configurable) number of times within the cluster as well as providing access to them by the use of IP-multicast technology.

The communication between the structured and the unstructured layers starts with an IP-multicast of a key from the gateway; this key identifies the requested envelope. All nodes that have the requested envelope reply with information about the envelope and what other nodes contain the requested envelope, with the current execution load on the storage node. The gateway collects this information and waits until it has received answers from more than 50% of the listed storage nodes that contains the identifier before it makes a decision on which one to select for retrieval of the file.

4.2.2 Gluster

Gluster is a structured distributed storage system. Storage servers in Gluster support both NFS and CIFS. Gluster does not provide a client side cache in the default configuration [28]. Gluster only provides redundancy at the server level, not at the individual disk level. For data availability and integrity reasons Gluster recommends RAID 6 or RAID 5 for general use cases. For high-performance computing applications, RAID 10 is recommended. Distribution over mirrors (RAID 10) is one common way to implement Gluster. In this scenario, each storage server is replicated to another storage server using synchronous writes. In this strategy, failure of a single storage server is transparent, and read operations are spread across both members of the mirror.

Gluster uses the Elastic Hash Algorithm (EHA). EHA determines where the data are stored and is a key to the ability to function without metadata. A pathname/filename is run through the hashing algorithm. After that, the file is placed on the selected storage. When accessing the file, the Gluster file system uses load balancing to access replicated instances. Gluster offers automatic self-healing [9, 14].

4.2.3 OpenStack's Swift

OpenStack's Swift is an unstructured distributed storage system. A number of "zones" are organized in a logical ring which represents a mapping between the names of entities stored on disk and their physical location. Swift is configurable in terms of how many copies (called "replicas") that are stored, as well as how many zones that are used. The system tries to balance the writing of objects to storage servers so that the write and read load is distributed. The mapping of objects to zones is done using a hash function. Swift does not do any caching of actual object data but Swift-proxys can work with a cache (Memcached[1]) to reduce authentication, container, and account calls [29]. In Swift, there are separate rings for accounts, containers, and objects. When other components need to perform any operation on an object, container, or account, they interact with the appropriate ring to determine its location in the cluster. OpenStack's Swift's rings are responsible for determining which devices to use in failure scenarios [15, 16, 17, 18, 19].

OpenStack's Swift divides the storage space into partitions. In our case, 18 bits of the GUID are used to decide on which partition a certain file should be stored, i.e., there are 2^{18} = 262 144 partitions. These partitions are divided into 6 zones. Zone 0 is mapped to storage nodes 0 to 3, zone 1 is mapped storage nodes 4 to 7, and zone 5 is mapped to storage nodes 20 to 23. Storage nodes 0 to 7 are

[1] Memcached is a distributed memory object caching system

handled by one switch, nodes 8 to 15 by one switch and nodes 16 to 23 by one switch (see Figure 4.1). There are 24*16 = 384 disks in the system and the 262 144 partitions are spread out with 682 or 683 partitions on each disk (262144/384 = 682.666…). If a file is stored on partition X, the two extra copies of the file (there are three copies of each file) are stored on partitions (X + 87 381) mod 262 144, and (X + 2 * 87 381) mod 262 144 (262 144 / 3 = 87 381.333…).

4.3 Experimental Setup

4.3.1 Test Configurations

Four different storage system configurations have been evaluated:

1. Compuverde Unstructured
2. Compuverde Structured
3. OpenStack's Swift (an unstructured storage system)
4. Gluster (a structured storage system)

The measurements use two load generating clients (see Figure 4.1). We use the same load for each configuration; the only part that has been changed is the interface. The clients work synchronously and report the result to the master controlling the clients (see Figure 4.1), which is responsible for monitoring the throughput.

In the configurations 1 and 2, Compuverde 0.9 has been installed on CentOS 6.2. In the configuration 3, version 1.4.3 of the OpenStack's Swift (release name: Diablo) has been installed on Linux Ubuntu 10.04 and in the configuration 4, Gluster 3.2.5 has been installed on CentOS 6.2.

The same hardware is used in each configuration. The storage system consists of 24 storage nodes, each containing sixteen 2 TB disks, i.e., a total of 32 TB for each node and 768 TB storage for all 24 nodes. With the exception of configuration 1 (Compuverde Unstructured), all accesses to the storage system are routed through four proxy (gateway) servers. In configuration 1 the clients communicate directly with the storage system.

Each proxy server has an Intel Quad processor, 16 GB RAM, and two 10 Gbit network cards. Each storage node has an Intel Atom D525 processor, 4 GB RAM, and a 1 Gbit network card. All storage nodes and proxy servers run the Linux operating system. There are four switches that are used to transmit data from four proxy servers and two load generating clients to the 24 storage nodes. The central

switch is a Dell 8024F and the other three switches are Dell 7048Rs. Four proxy servers are connected to the central switch via four 20 Gbit fibers. Two load generating clients are connected to a central switch via two 10 Gbit fibers and the central switch is connected to the other three switches via three 40 Gbit fibers.

The four test configurations will now be described.

4.3.1.1 Compuverde Unstructured

In this configuration three copies of each file are created. The proxy servers are not used, and the load generating clients communicate directly with the storage nodes.

4.3.1.2 Compuverde Structured

In this case two copies of each file are created. The reason for this is that this case will be compared with Gluster, and Gluster only supports two copies of each file. The two load generating clients communicate with two proxy servers each. The communication protocol between the load generating clients and the proxy servers is NFS/CIFS.

4.3.1.3 OpenStack's Swift

OpenStack's Swift has three copies of each file, and the two load generating clients communicate with two proxy servers each.

4.3.1.4 Gluster

Gluster dedicates a volume to the lock file. In Gluster the storage nodes are arranged in pairs to obtain fault tolerance. This means that there are only two copies of each file. The communication protocol between the load generating clients and the proxy servers is NFS/CIFS.

4.3.2 Test Cases

Two kinds of tests are considered in this study: performance tests and recovery tests.

4.3.2.1 Performance Tests

In these test cases the read, write and delete performance are measured:

There are four test cases for each test configuration:

1. We measure write performance. In these tests, a number of clients (implemented as full speed threads, i.e., as threads that issue write requests in a tight loop without any delay and with only minimal processing done between each request) running on two servers (see Figure 4.1) create files of size 0 KB, 10 KB, 100 KB, 1 MB and 10 MB, respectively. Writing 0 KB corresponds to creating a file and will be reported separately. We vary the number of clients using the steps 2, 4, 8, 16, 32, 64, 128, and 256 clients. A write operation is a combination of Open, Write and Close. We measure MB/s and operations/s.
2. We measure read performance. In these tests, a number of clients (implemented as full speed threads) running on two servers (see Figure 4.1) read files of size 0 KB, 10 KB, 100 KB, 1 MB and 10 MB, respectively. Reading 0 KB corresponds to opening a file and will be reported separately. We vary the number of clients using the steps 2, 4, 8, 16, 32, 64, 128, and 256 clients. A read operation is a combination of Open, Read and Close. We measure MB/s and operations/s.
3. We measure delete performance. In these tests, a number of clients (implemented as full speed threads) running on two servers (see Figure 4.1) delete files of size 10 KB, 100 KB, 1 MB and 10 MB, respectively. We vary the number of clients using the steps 2, 4, 8, 16, 32, 64, 128, and 256 clients. We measure operations/s.
4. For the structured storage case, we use the SPECsfs2008 performance evaluation tool. The tool can be configured to issue a number of I/O Operations per Second (IOPS), and it then measures the actual achieved throughput in terms of IOPS and the average response time.

The performance tests for small file sizes (0 KB and 10 KB) have been done by writing/reading/deleting 1,000,000 files to/from the storage nodes, but for larger file sizes (100 KB, 1 MB and 10 MB) the test has been continued by writing/reading/deleting files (between 50,000 and 100,000 files) until the results become stable.

Gluster and OpenStack's Swift do not use caching. In order to get fair results, the test has been done for Compuverde for two cases: caching and No Caching (NC). We limited the NC tests to 1 MB files

4.3.2.2 Recovery Tests

In these tests we measure how long it takes for the storage system to reconfigure itself after a node failure. We measure recovery performance by reformatting one storage node. When a storage node is reformatted the file copies stored on that node are lost. We measure the time until the system has created new copies corresponding to the copies that were lost.

4.4 Read and Write Performance

In this section we look at the read and write performance of each of the four configurations. In Section 4.5 we compare the different configurations.

4.4.1 Compuverde Unstructured

Figures 4.2(a) and 4.2(b) show that the throughput is low when number of clients and the size of the files are small; the throughput increases when number of clients and the size of the files increase. It can also be noted that the performance in case of using cache in the storage nodes, e.g., 1 MB files, does not differ much compared to the case that using NC, i.e., 1 MB (NC).

4.4.2 Compuverde Structured

Figures 4.3(a) and 4.3(b) show that the data transfer rate is low when the number of clients and the size of the files are small and it increases when number of clients and size of files increase. It can also be noted that the performance difference between using caching in the storage nodes, e.g., 1 MB files, and using NC, i.e., 1 MB (NC), is approximately a factor of 1.5 when writing; there is no significant difference between caching and NC when reading.

4.4.3 OpenStack's Swift

Figures 4.4(a) and 4.4(b) show that in cases of writing/reading the files of large size (10 MB), the data transfer rate increases rapidly when the number of the clients increases. While in case of writing files with size of 1 MB and less the curve is quite stable.

4.4.4 Gluster

Figures 4.5(a) and 4.5(b) show that the data transfer rate for large files increases when the number of clients increases. However, for smaller files the transfer rate does not increase so much when the number of clients increases.

In fact, when the number of clients exceeds a certain values the transfer rate starts to decrease. The reason for this is probably that Gluster contains contention bottlenecks internally. According to the performance test results, the utilization for the storage nodes never exceeds 50% for Gluster. For the other test configurations we get much higher utilization values. This is an indication that there are internal performance bottlenecks in Gluster.

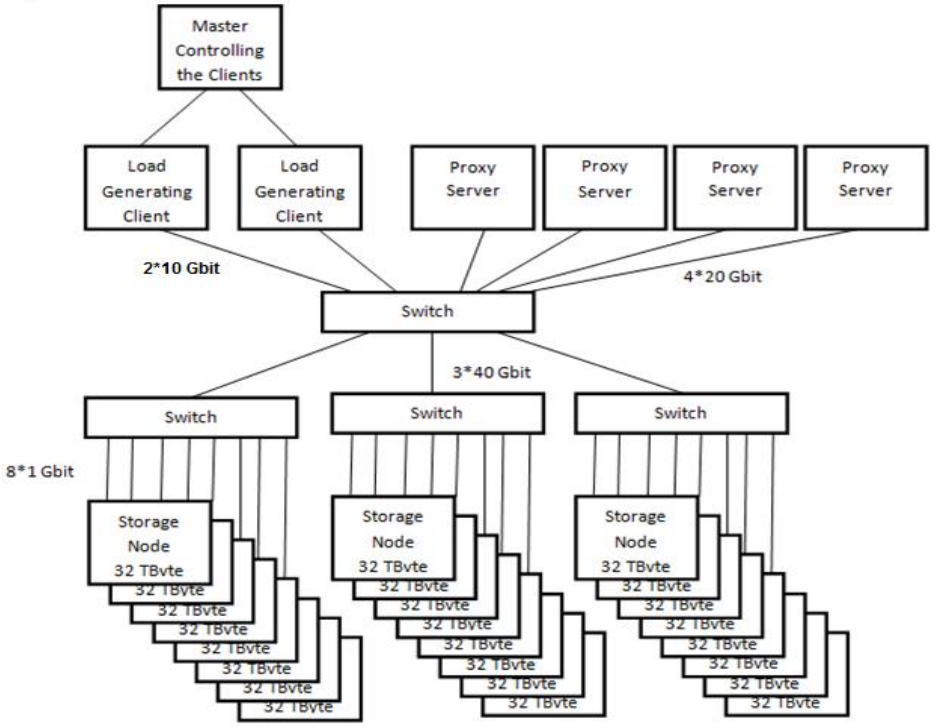


Figure 4.1. The physical structure of the test configuration

4.5 Comparing the Distributed Storage Systems

We have evaluated two unstructured storage systems (OpenStack's Swift and Compuverde Unstructured) and two structured storage systems (Gluster and Compuverde Structured). In Section 4.5.1 we compare the performance of the two unstructured systems and in Section 4.5.2 we compare the performance of the two structured systems. In Section 4.5.3 we compare the time to recreate all the file copies in a storage system in case one of the storage nodes fails.

4.5.1 Compuverde Unstructured vs. OpenStack's Swift

We talked to several cloud storage providers and it turned out that most of their users store small files with an average size of 1 MB. Therefore, the performance tests are compared only for 1 MB. Figure 4.6(a) shows that the write performance of Compuverde Unstructured for 256 clients (both when using caching and NC) was roughly 800 MB/s, while for OpenStack's Swift it was around 200 MB/s. Figure 4.6(b) shows that the read performance of Compuverde Unstructured for 256 clients (both when using caching and NC) was 1600 MB/s to 1900 MB/s, while for OpenStack's Swift it was around 600 MB/s. Figure 4.6(c)

shows that the create files performance of Compuverde Unstructured for 256 clients was 10118 operations/s in case of caching and 6500 operations/s in case of NC; for OpenStack's Swift it was 600 operations/s. Figure 4.6(d) show that the open files performance of Compuverde Unstructured for 256 clients was 11153 operations/s in case of caching and 12826 operations/s in case of NC; for OpenStack's Swift it was 4500 operations/s. The delete files performance test has been done by deleting files with a size of 1 MB. Figure 4.6(e) shows that the delete files performance of Compuverde Unstructured for 256 clients was 9956 operations/s in case of caching and 8145 operations/s in case of NC; for OpenStack's Swift it was 498 operations/s.

4.5.2 Compuverde Structured vs. Gluster

The write/read/delete performance tests have been done for 1 MB file size. Figure 4.7(a) shows that the write performance of Compuverde Structured for 256 clients was 655 MB/s in case of caching and 450 MB/s in case of NC; for Gluster it was 164 MB/s. Figure 4.7(b) shows that the read performance of Compuverde Structured for 256 clients was 780 MB/s in case of caching and 821 MB/s in case of NC; for Gluster it was 270 MB/s. Figure 4.7(c) shows that the performance for Compuverde Structured for 256 clients was 7370 operations/s in case of caching and 1239 operations/s in case of NC; for Gluster it was 241 operations/s. Figure 4.7(d) shows that the performance for Compuverde Structured for 256 clients was 11116 operations/s in case of caching and 12458 operations/s in case of NC; for Gluster it was 1029 operations/s. The delete files performance test has been done by deleting files of 1 MB size. Figure 4.7(e) shows that the performance for Compuverde Structured for 256 clients was 3548 operations/s in case of caching and 3367 operations/s in case of NC; for Gluster it was 441 operations/s.

The test results using the Spec2008sfs tool are shown in Figures 4.8(a) and 4.8(b). Figure 4.8(a) shows that both Compuverde Structured and Gluster meet the number of requested IOPS for 3000 IOPS and 4000 IOPS. However, when the requested numbers of IOPS increased to 5000 and above, Compuverde Structured delivered a number of IOPS relatively near to the requested one, while Gluster delivers a number of IOPS that is significantly lower than the requested number. Figure 4.8(b) shows the result of response time test that has been obtained using the Spec2008sfs performance evaluation tool. Compuverde's response time is in the range of 3.5 ms to 17 ms, while for Gluster the response time is between 10.1 ms and 33.3 ms.

4.5.3 Recovery Test

We did the recovery test for all four different configurations. The same recovery test has been run twice for each configuration.

Table 4.2: Recovery Test Results

Compuverde Unstructured	19 minutes (1140 s)	18 minutes (1080 s)
Compuverde Structured	22 minutes (1320 s)	22 minutes (1320 s)
OpenStack	9 hours 27 minutes (34020 s)	10 hours 16 minutes (36960 s)
Gluster	18 hours 27 minutes (66420 s)	18 hours 29 minutes (66540 s)

As shown in Table 4.2, the recovery time for Compuverde Unstructured was 18-19 minutes and the recovery time for OpenStack's Swift was approximately 10 hours. This means that the recovery time for Compuverde Unstructured is approximately 30 times faster than that of OpenStack's Swift. One reason for this difference is that Compuverde uses multicasting whereas OpenStack's Swift uses DHT. Another reason could be that OpenStack uses the rsync[2] command that is responsible for maintaining object replicas, consistency of objects and perform update operations. It seems that using rsync command introduces a significant overhead which causes a performance decrease. The situation is similar for Compuverde Structured with a recovery time of 22 minutes compared to Gluster with recovery time of approximately 18.5 hours. Compuverde Structured recovery time is thus approximately 50 times faster than Gluster recovery time. As discussed before, Gluster uses DHTs instead of multicasting. Gluster also uses rsync for replication. Another reason for the low performance of Gluster compare to Compuverde Structured is the architecture that is used by Gluster for replication. In Gluster the proxy servers are doing the self-healing while in Compuverde Structured storage nodes are performing the self-healing by themselves without involving any proxy servers which results in a many-to-many replication pattern.

4.6 Discussion and Related Work

Compared to conventional centralized storage systems, a distributed storage system allows for not only increased performance and redundancy, but also affords improved energy efficiency and lowering the carbon footprint of the system. For instance, by removing the need for a central, high-powered storage controller and replacing it with low wattage storage nodes, such as the ones used in the experiments presented in this paper. Furthermore, a distributed storage system built from standard hardware components also makes it possible to exchange the individual nodes with nodes with a lower carbon footprint as

[2] rsync is a file transfer program for Unix-like systems.

technology advances. Reducing the carbon footprint and enabling green computing are two important aspects of Cloud Computing.

In recent years, many research, and development efforts have been done in cloud computing, specifically on distributed file systems. In [37] the authors have done a performance comparison between several distributed file systems such as Hadoop, MooseFS (MFS) and Lustre. They have compared functionalities as well as I/O performance of these three file systems.

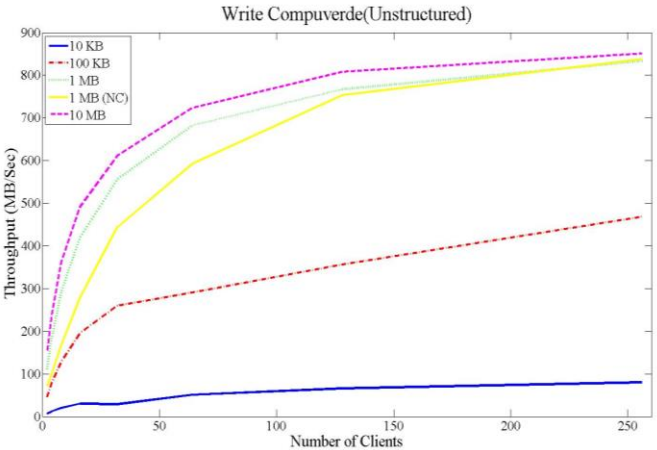
In [38] the authors have done a performance comparison between Google File System (GFS) and MFS in terms of reliability, file performance and scalability. According to their comparison GFS and MFS are both reliable since resource files are backed up. But they found a single point of failure in master on GFS while it does not exist on MFS. In MFS there is a need for manual back up after a problem has occurred. Their comparison of the file performance indicates that GFS is used for large GB file size while MFS supports small files better.

4.7 Conclusion

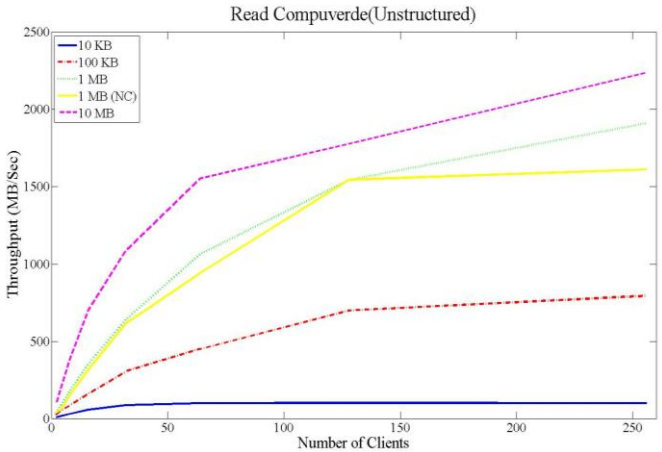
We have compared two unstructured storage systems for Cloud Computing (Compuvede Unstructured and Openstack's Swift) and two structured storage systems for Cloud Computing (Compuverde Structured and Gluster). Compuverde uses multicasting and Openstack's Swift and Gluster use Distributed Hash Tables (DHTs). The architectural advantage of DHTs compared to multicasting is that we do not need to broadcast requests; the hash table gives us the address of the nodes that store of the requested data and we avoid communication overhead. However, the obvious disadvantage with DHTs is that we need to run a hash function to obtain the address of the data, which introduces processing overhead. This means that the architectural decision, whether to use DHTs or multicasting will introduce different kinds of overhead: processing overhead for DHTs and communication overhead for multicasting.

We have compared the performance using a large storage system and realistic workloads, including the well-known Spec2008sfs test tool. Our experiments show that Compuverde has higher performance than the two systems that use DHTs. The performance advantage of Compuverde is particularly clear when the number of clients that issue simultaneous accesses to the system is high, which is typical in Cloud Computing. The performance advantage of Compuverde is not a result of caching in the storage nodes, i.e., the performance of Compuverde using NC is still significantly higher than that of the other two systems. We believe that the main reason for the higher performance is that Compuverde uses multicast instead of DHTs. The communication overhead introduced by multicasting does obviously not affect the performance as negatively as the processing overhead introduced by DHTs.

The recovery tests show that Compuverde recovers from a storage node failure much faster than OpenStack's Swift and Gluster. Again, we believe that the use of multicast instead of DHTs is the main reason. However, this cannot be the only reason for the significant difference in recovery times. One additional reason for Gluster to perform slower than Compuverde Structure could be that Gluster involves proxy servers in self-healing while Compuverde uses the many-to-many replication pattern and only involves storage nodes in self-healing. Another reason could be that Compuverde has built its own recovery protocol from scratch, whereas OpenStack's Swift and Gluster base their protocols on existing applications (e.g., rsync). Moreover, the processor utilization for Gluster never exceeds 50%, even for high loads. This indicates that there are internal performance bottlenecks in Gluster, which probably contributes to the relatively long time for self-healing.

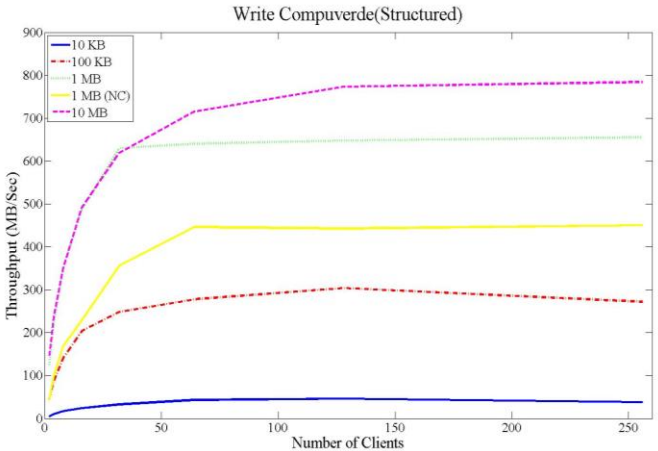


a. Write Performance Test Results (Compuverde Unstructured)

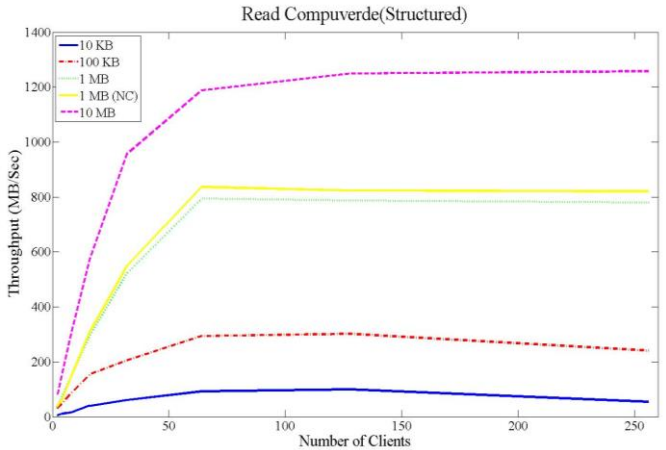


b. Read Performance Test Results (Compuverde Unstructured)

Figure 4.2: In figures (a) and (b) the y-axis denotes the data transfer rate in MB/s, while the x-axis denotes the number of clients that are writing/reading simultaneously.

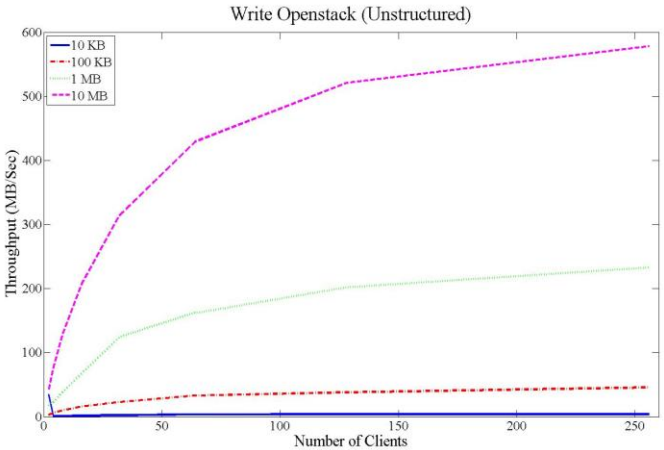


a. Write Performance Test Results (Compuverde Structured)

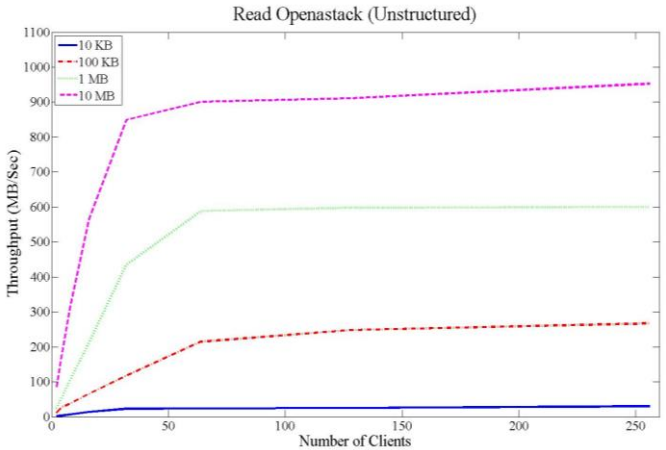


b. Read Performance Test Results (Compuverde Structured)

Figure 4.3: In figures (a) and (b) the y-axis denotes the data transfer rate in MB/s, while the x-axis denotes the number of clients that are writing/reading simultaneously.

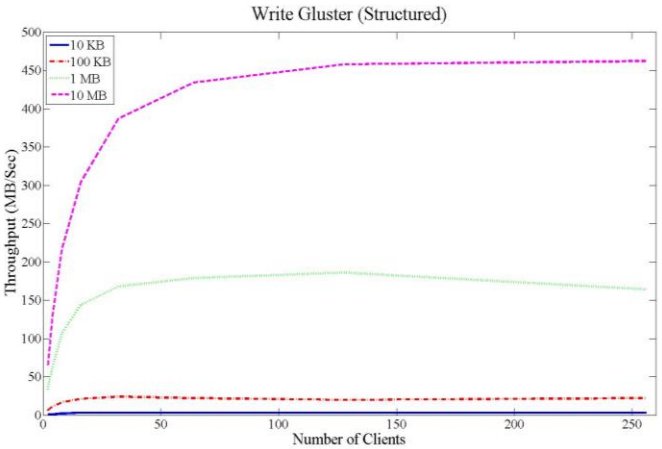


a. Write Performance Test Results (Openstack)

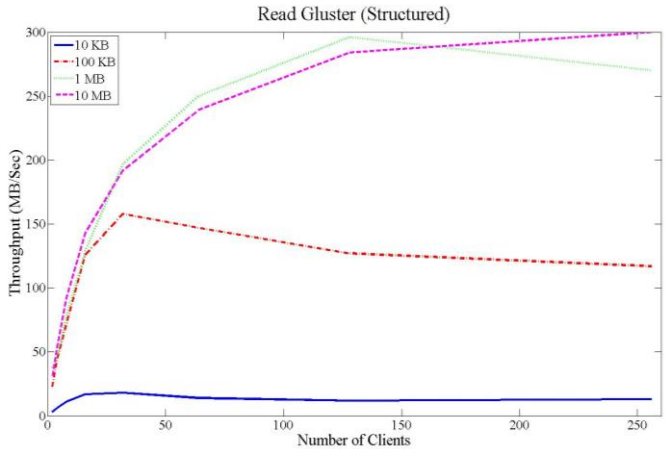


b. Read Performance Test Results (Openstack)

Figure 4.4: In figures (a) and (b) the y-axis denotes the data transfer rate in MB/s, while the x-axis denotes the number of clients that are writing/reading simultaneously.

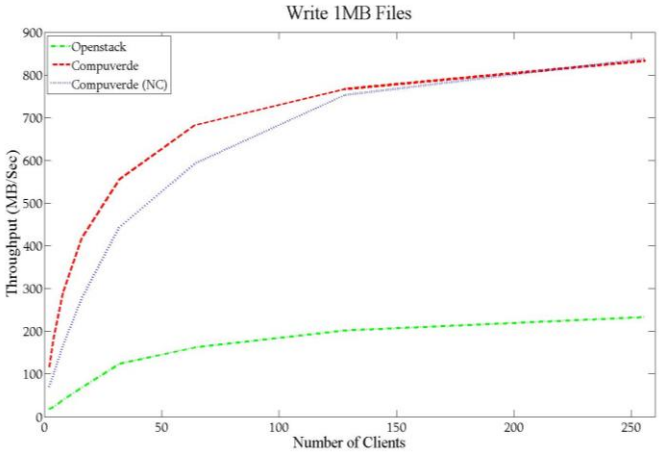


a. Write Performance Test Results (Gluster)

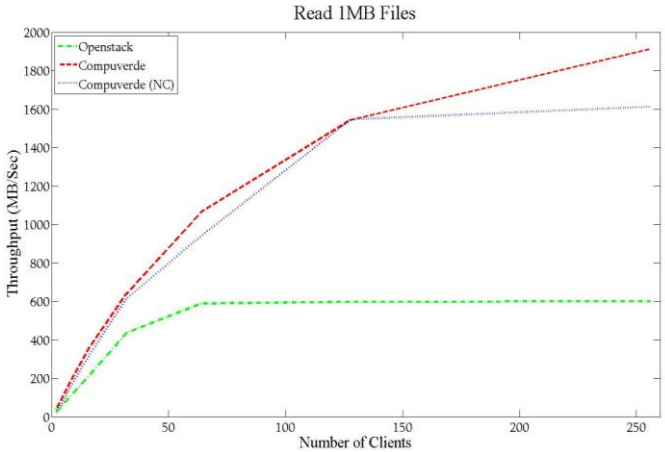


b. Write Performance Test Results (Gluster)

Figure 4.5: In figures (a) and (b) the y-axis denotes the data transfer rate in MB/s, while the x-axis denotes the number of clients that are writing/reading simultaneously.

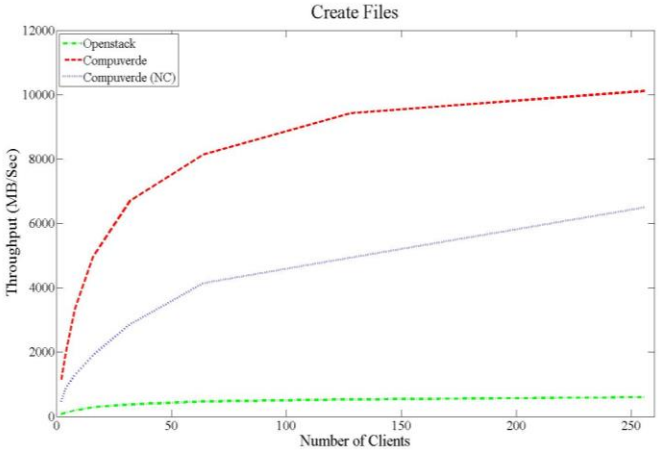


a. Write Performance Compuverde Unstructured vs. Openstack's Swift

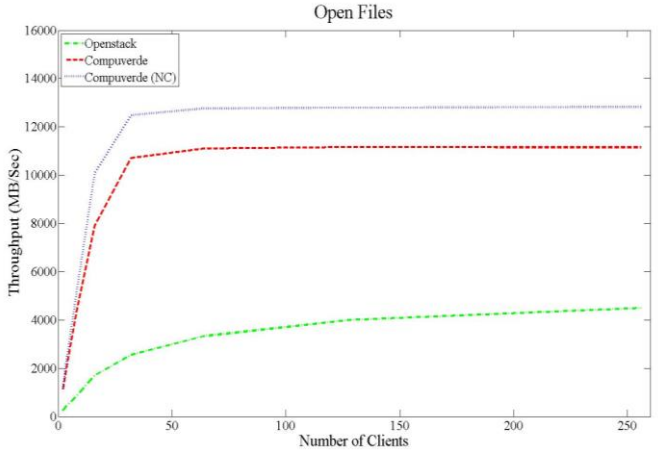


b. Read Performance Compuverde Unstructured vs. Openstack's Swift

Figure 4.6: Comparison between the performance of Compuverde Unstructured and OpenStack's Swift for files of 1 MB.

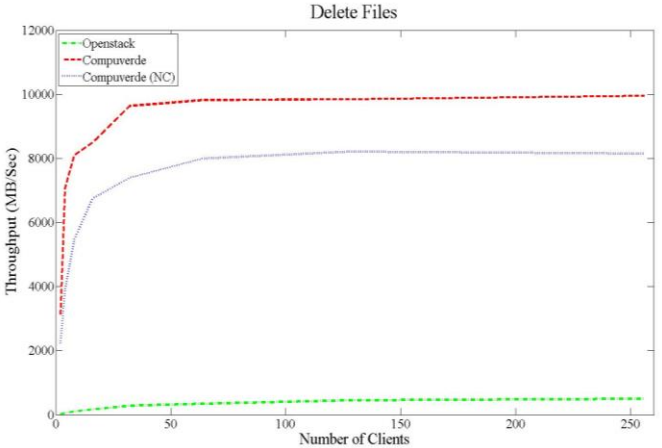


c. Create Files Performance Compuverde Unstructured vs. Openstack's Swift



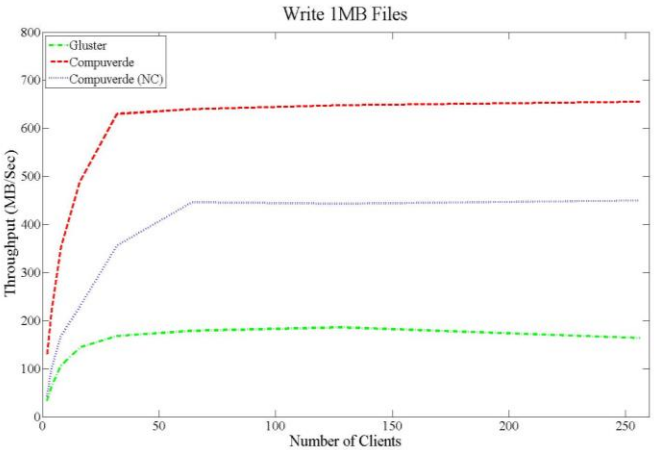
d. Open Files Performance Compuverde Unstructured vs. Openstack's Swift

Figure 4.6: Comparison between the performance of Compuverde Unstructured and OpenStack's Swift for files of 1 MB.

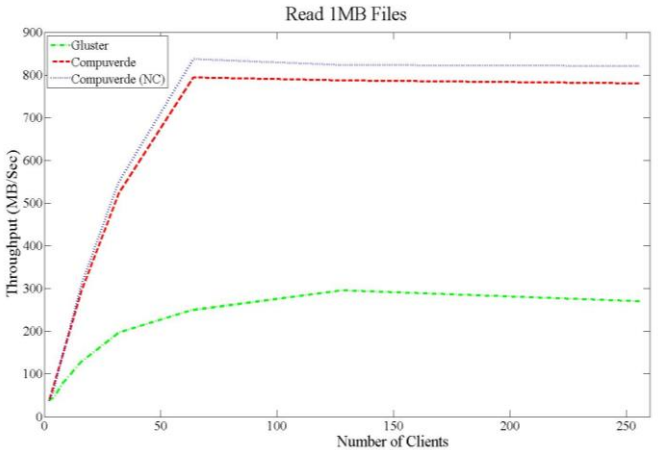


e. Delete Files Performance Compuverde Unstructured vs. Openstack's Swift

Figure 4.6: Comparison between the performance of Compuverde Unstructured and OpenStack's Swift for files of 1 MB.

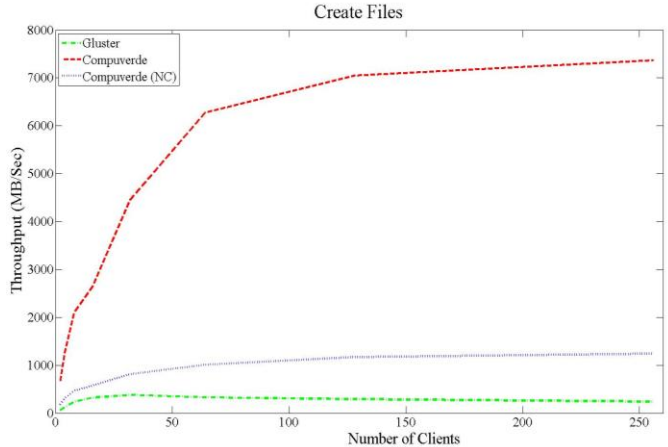


a. Write Performance Compuverde Structured vs. Gluster

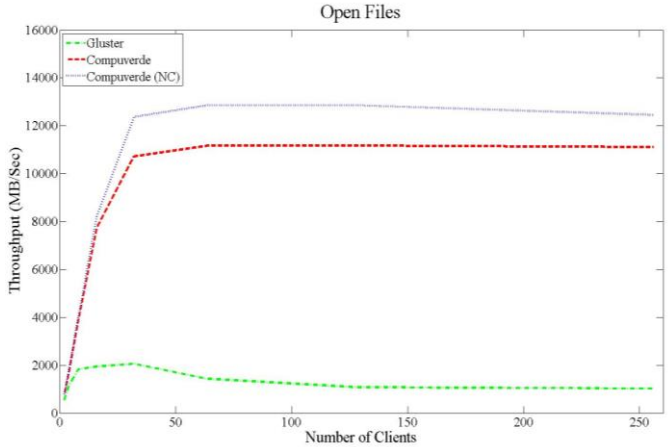


b. Read Performance Compuverde Structured vs. Gluster

Figure 4.7: Comparison between the performance of Compuverde Structured and Gluster for files of 1 MB.

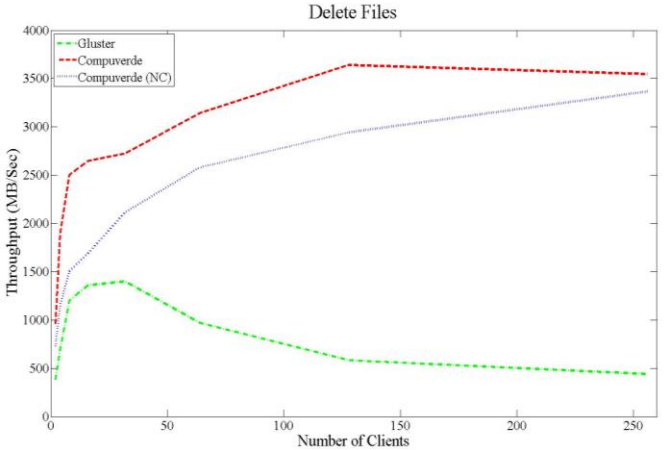


c. Create Files Performance Compuverde Structured vs. Gluster



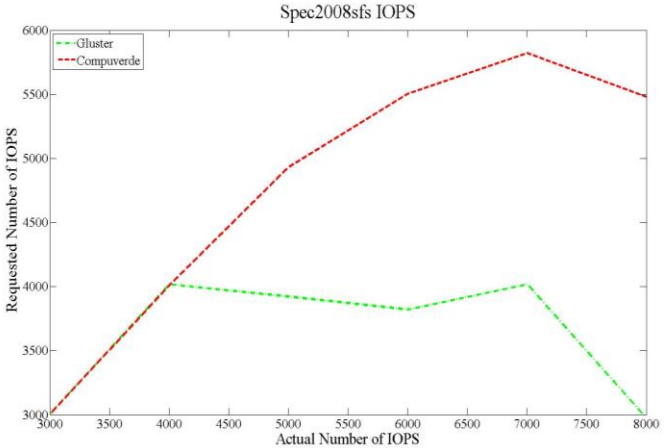
d. Open Files Performance Compuverde Structured vs. Gluster

Figure 4.7: Comparison between the performance of Compuverde Structured and Gluster for files of 1 MB.

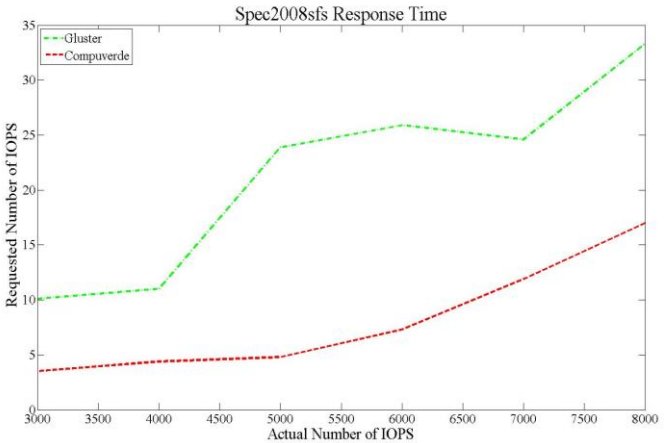


e. Delete Files Performance Compuverde Structured vs. Gluster

Figure 4.7: Comparison between the performance of Compuverde Structured and Gluster for files of 1 MB.



a. Performance Evaluation Compuverde Structured vs. Gluster



b. Performance Evaluation Compuverde Structured vs. Gluster

Figure 4.8: Comparison between the performance of Compuverde Structured and Gluster when using the Spec2008sfs tools.

4.8 References

[1] William Geisel, “Tutorial on Reed–Solomon Error Correction Coding”, Technical Memorandum, NASA, TM-102162, August, 1990.

[2] Liuis Pamies I Juarez, “On the Design and Optimization of Heterogeneous Distributed Storage Systems”, University Rovira in Virgili, Department of Engineering Information in Mathematic, PHD thesis, July, 2011.

[3] Garth A. Gibson, Rodney Van Meter, “Network Attached Storage Architecture”, Magazine, Communications of the ACM, New York, November, 2000.

[4] Michael Factor, Kalman Meth, Dalit Naor, Ohad Rodeh, Julian Satran, ”Object storage: The future building block for storage systems”, published in: in 2nd International IEEE Symposium on Mass Storage Systems and Technologies, Sardinia,2005.

[5] Santa Clara, “Amplidata Demonstrates Highly Scalable and Reliable Storage Solutions for Massive Cloud Deployments at Intel Development Forum”, Article at PRNewswire, Calif., September, 2011.

[6] Caringo CAStor (2011, September 15). ”Castor the Market Leading Object Storage Engine” [Online]. Available: http://www.caringo.com/downloads/datasheets/Caringo-CAStor-Object-Storage.pdf

[7] Scott A. Brandt, Darrell D. E. Long, Carlos Maltzahn, Ethan L. Miller, Sage A. Weil, “Ceph: A Scalable, High-Performance Distributed File System”, University of California, Santa Cruz, Appeared in Proceeding of the 7th Conference on Operating Systems Design and Implementation (OSDI’06), November, 2006.

[8] EMC Atmos, “EMC Atmos Cloud Optimize Storage for Web Services” Whitepaper, April, 2010.

[9] Gluster Inc. “An Introduction to Gluster Architecture” Whitepaper, 2011.

[10] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. ”The Google File System”, Appeared in 19th ACM Symposium on Operating Systems Principles, Lake George, NY, October, 2006.

[11] Robert Chansler, Hairong Kuang, Sanjay Radia, Konstantin Shvachko,“The Hadoop Distributed File System”, Yahoo! Sunnyvale, California USA, 2010.

[12] Sarp Oral, Galen Shipman, Feiyi Wang, “Understanding Lustre FileSystem Internals”, OAK RIDGE, 2009.

[13] Zainul Abbasi, Garth Gibson, Brian Mueller, Jason Small, Marc Unangst, Brent Welch, Jim Zelenka, Bin Zhou. “Scalable Performance of The Panasas Parallel File System”, FAST’08 proceedings of the 6th USENIX

Conference on File and Storage Technologies, USENIX Assosiation Berkeley, CA, USA, 2008.

[14] Drew Robb, “Gluster Brings Open Source to Unstructured Data”, Storage Technology Features Article Published, August, 2010.

[15] OpenStack, “OpenStack Compute Admin Manual”, Manual, November, 2011.

[16] Joe Arnold, Dr. Jaesuk Ahn, Dr. Jinkyung Hwang, “Commercialization of OpenStack: Object Storage”, OpenStack conference commercialization of object storage, Korea, April, 2010.

[17] Pepple Ken,”Deploying OpenStack”. O’Reilly Media. ISBN 1449311059, August, 2011.

[18] OpenStack, “OpenStack Object Storage: An Overview” white paper, 2010.

[19] OpenStack, LLC, “Welcome to Swift’s documentation!”, Swift v1.4.8-dev documentation, 2011.

[20] Shunsuke Mikami, Kazuki Ohta, Osamu Tatebe, ”Using the Gfarm File System as a POSIX Compatible Storage Platform for Hadoop MapReduce Applications”, Published in: GRID’11 Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing, IEEE Computer Society Washington, DC, USA, 2011.

[21] Lei Chai, Ranjit Noronha, Dhabaleswar K. Panda, Thomas Talpey,”Designing NFS with RDMA for Security, Performance and Scalability”, Technical Report OSU-CISRC-6/07-TR47, The Ohio State University, 2007.

[22] Julian Dymcek, ”Survey of Distributed Hash Tables”, Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown WV, 2011.

[23] Roy T. Fielding, Richard N. Taylor, “Principles design of the modern Web architecture”, published in ICSE’00 Proceedings of the 22nd international conference on software engineering, ACM New York, NY, USA, 2000.

[24] Michael Jakl, ”REST: Representational State Transfer”, University of Technology Vienna, 2008.

[25] Wang, P., “IP SAN- from iSCSI to IP-addressable Ethernet disks” Appears in: Mass Storage Systems and Technologies. Proceedings, 20th IEEE/11th NASA Goddard Conference, 2003.

[26] Angelos Bilas, Michail D. Flouris, Yannis Klonatos, Thanos Makatos, Manolis Marazkis, “Using transparent compression to improve SSD-based I/O Caches”,Published in EuroSys’10 Proceedings of the 5th European conference on Computer systems, ACM NewYork, NY, USA, 2010.

[27] Curbera F., Duftler M., Khalaf R., Mukhi N., Nagy W., Weerawarana S. “Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI”, published in Internet Computing, IEEE, NY, USA, 2002.

[28] Dhabaleswar K. Panda, Ranjit Noronha, “IMCa: High Performance Caching Front-end for GlusterFS on InfiniBand” Network-Based Computing Laboratory, Computer Science and Engineering, The Ohio State University, 2008.

[29] OpenStack Object Storage Admin Manual, OpenStack, “Consideration and Tunning”, 2011.

[30] Amplidata,“Amplistor: Unbreakable Object Storage for Petabyte-Scale Unstructured Data” Whitepaper, April, 2011.

[31] Caringo CAStor, “CAStor: The Market Leading Object Storage Engine” Product Brief, September, 2011.

[32] Andrew Beekhof. Christine Caulfield, Steven C. Dake, “The Corosync Cluster Engine”, Proceedings of the Linux Symposium, Ottawa, Ontrio, Canada, July, 2008.

[33] George Parisis, “DHTbd: A Reliable Block-based storage system for High Performance clusters”, Proceedings of the IEEE/ACM CCGRID, UK, 2011.

[34] Roberto Lucchi, Michel Millot, “Resource Oriented Architecture and REST”, JRC Scientific and Technical Reports, European Communities, Luxembourg, 2008.

[35] Bin Fan, Garth Gibson, Wittawat Tantisiriroj, Lin Xiao, ”DiskReduce: Replication as a Prelude to Erasure Coding in Data-Intensive Scalable Computing”, Parallel Data Laboratory, Carnegie Mellon University, Pittsburgh, 2011.

[36] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, R. E. Gruber, “BigTable: A Distributed Storage System for Structured Data”, Journal: ACM Transactions on Computer Systems (TOCS), New York, USA, June, 2008.

[37] Wu Hao, Bai Songlin, “The Performance Study on Several Distributed File Systems”. Proceedings of the International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, Beijing, China, pp 226-229, 2011.

[38] Ping Chen, Xuerong Gou, Jianwei Li,”Research of Distributed File System Based on Massive Resource and Application in the Network Teaching System”. Proceedings of the International Conference on Advanced Intelligence and Awareness Internet, pp 154-158, Beijing, China, 2011.

5 Performance Comparison of KVM, VMware and XenServer using a Large Telecommunication Application

Abstract

One of the most important technologies in cloud computing is virtualization. This paper presents the results from a performance comparison of three well-known virtualization hypervisors: KVM, VMware and XenServer. In this study, we measure performance in terms of CPU utilization, disk utilization and response time of a large industrial real-time application. The application is running inside a virtual machine (VM) controlled by the KVM, VMware and XenServer hypervisors, respectively. Furthermore, we compare the three hypervisors based on downtime and total migration time during live migration. The results show that the Xen hypervisor results in higher CPU utilization and thus also lower maximum performance compared to VMware and KVM. However, VMware causes more write operations to disk than KVM and Xen, and Xen causes less downtime than KVM and VMware during live migration. This means that no single hypervisor has the best performance for all aspects considered here.

5.1 Introduction

Virtualization has many advantages over non-virtualized solutions, e.g., flexibility, cost and energy savings [19][34]. As a more specific example, consider the cost associated with test hardware used during professional software development. This includes the initial price for purchasing the equipment, as well as operational costs in the form of maintenance, configuration and consumed electricity. For economic reasons, organizations often choose to use virtualized test servers, so that the test hardware can be shared and maintained in a cost-effective way [20]. In order to provide maximum resource utilization, there should be no restrictions on the mapping of VMs to physical computers, i.e., it should be possible to run a VM on any physical server. In order to balance the load, it is desirable that a VM running on a physical host could be restarted on another physical host, i.e., there is a need for migrating VMs from one physical server to another [21][22][23]. There is support for migration in many commonly used virtualization systems, e.g., KVM Live Migration [16], VMware's vMotion [18] and XenServers's XenMotion [17].

There are three different approaches to VM migration: cold migration, hot migration and live migration. When cold migration is used the guest Operating System (OS) is shut down, the VM is moved to another physical server and then

the guest OS is restarted there. Hot migration suspends the guest OS instead of shutting it down. The guest OS is resumed after the VM is moved to the destination host. The benefit of hot migration is that application running inside the guest OS can preserve most of their state after migration (i.e., they are not restarted from scratch). In the live migration approach [13], the VM keeps running while its memory pages are copied to a different host. Live migration reduces the downtime dramatically for applications executing inside the VM. Live migration is thus suitable for high-availability services.

In this paper, we compare the performance of KVM, VMware and XenServer, for two different scenarios: when no VM is migrated, and when a VM is migrated from one physical server to another. The work load is, for both scenarios, a large real-time telecommunication application. In the case when no VM is migrated, we measure the CPU utilization, the disk utilization (the number of write operations), and the average application response time. When a VM is migrated we measure the CPU utilization, the disk utilization (the number of write operations), and the down time due to live migration.

The rest of the paper is organized as follows. In Section 5.2 the state of the art is summarized. Section 5.3 describes the experimental setup for the different hypervisors, and, in Section 5.4, we compare and analyze the results for KVM, VMware and XenServer. Finally, related work is discussed in Section 5.5. Section 6.5 concludes the paper.

5.2 State of The Art

5.2.1 Virtualization

In its simplest form, virtualization is a mechanism for several virtual OS instances on a single physical system. This is typically accomplished using a Hypervisor or Virtual Machine Monitor (VMM), which lies between the hardware and the OS. Virtualization is often beneficial for environments consisting of a large number of servers (e.g., a datacenter).

A virtualization solution relies on several components, such as CPU virtualization, memory virtualization, I/O virtualization, storage virtualization, and so on. In this paper we focus specifically on CPU and memory virtualization.

Current approaches to virtualization can be classified into: full virtualization, paravirtualization and hardware assisted virtualization [11][12].

Full virtualization uses binary translation which translates the kernel code so that privileged instructions can be converted to user-level instructions during run-time. Detection and translation of privileged instructions typically carries a large performance penalty. KVM and VMware support this approach.

Paravirtualization attempts to alleviate the performance of full virtualization by replacing privileged instructions with specific function calls to the hypervisor, so called hypercalls. This requires changes to the guest OS source code, which is not always possible. In particular, access to the source code of commercial OSs is heavily restricted. Both XenServer and KVM support paravirtualization.

Recent innovations in hardware, particularly in CPU, Memory Management Unit (MMU) and memory components (notably the Intel VT-x and AMD-V architectures [12]), provide some direct platform-level architectural support for OS virtualization. Hardware assisted virtualization offers one key feature: it avoids the need to trap and emulate privileged instructions by enabling guests to run at their native privilege levels. VMware and KVM support this approach.

5.2.2 Live Migration

Live migration is a mechanism that allows a VM to be moved from one host to another while the guest OS is running. This type of mobility provides several key benefits, such as fault tolerance, hardware consolidation, load balancing and disaster recovery. Users will generally not notice any interruption in their interaction with the application, especially in the case of non-real-time applications. However, if the downtime becomes too long, users of real-time applications, in particular interactive ones may experience serious service degradation [4].

To achieve live migration, the state of the guest OS on the source host must be replicated on the destination host. This requires migrating processor state, memory content, local storage and network state. The focus of our study is on network state migration.

Pre-copy is the memory migration technique adopted by KVM live migration, vMotion and XenMotion [13][28][27][32]. With this approach, memory pages belonging to the VM are transferred to the destination host while the VM continues to run on the source host. Transferred memory pages that are modified during migration are sent again to the destination to ensure memory consistency. When the memory migration phase is done the VM is suspended on the source host, and then any remaining pages are transferred, and finally the VM is resumed on the destination host [8]. The pre-copy technique captures the complete memory space occupied by the VM (dirty pages), along with the exact state of all the processor registers currently operating on the VM, and then sends the entire content over a TCP connection to a hypervisor on the other host. Processor registers at the destination are then modified to replicate the state at the source, and the newly moved VM can resume its operation [7][27][31].

The Kernel-based Virtual Machine (KVM) is a bare-metal (Type 1) hypervisor. The approach that KVM takes is to turn the Linux kernel into a VMM (or hypervisor). KVM provides a dirty page log facility for live migration, which

provides user space with a bitmap of modified pages since the last call [5][6]. KVM uses this feature for memory migration.

VMware is bare-metal (Type 1) hypervisor that is installed directly onto a physical servers without requiring a host OS. In VMware vSphere, vCenter Server provides the tools for vMotion (also known as Live Migration). vMotion allows the administrator to move a running VM from one physical host to another physical host by relocating the contents of the CPU registers and memory [9][10].

XenServer is bare-metal (Type 1) hypervisor and runs directly on server hardware without requiring host OS. XenMotion is a feature supported by XenServer, which allows live migration of VMs. XenMotion works in conjunction with Resource Pools. A Resource Pool is a collection of multiple similar servers connected together in a unified pool of resources. These connected servers share remote storage and common networking connections [1][15][30].

KVM, VMware and XenServer aim to provide high utilization of the hardware resources with minimal impact on the performance of the applications running inside the VM. In this study, we compare their performance by measuring downtime and total migration time during live migration as well as their CPU utilization, when running large telecommunication applications in the VMs.

5.3 Experimental Setup

Two HP DL380 G6x86 hosts have been used to test the performance of KVM and VMware ESXi 5.0. On top of the VMware ESXi 5.0, RedHat Enterprise Linux, Version 6.2 has been installed as a guest OS. The same hardware was used to test the performance of Xen for Linux Kernel 3.0.13 running as part of the SUSE Linux Enterprise Server 11 Service Pack 2. Each server is equipped with 24 GB of RAM, two 4-core CPUs with hyperthreading enabled in each core (i.e., a total of 16 logical cores) and four 146 GB disk. Both servers are connected via 1 Gbit Fibre Channel (FC) to twelve 400 GB Serial Attached SCSI (SAS) storage units. All devices are located in a local area network (LAN) as shown in Figure 5.1.

5.3.1 Test Configurations

Three different test setups were evaluated:

- KVM-based setup
- VMware-based setup
- XenServer-based setup

In each setup, two VMs are created inside hypervisor1 and hypervisor2, resulting in a total of four VMs (see Figure 5.1). One large industrial real-time telecommunication application is installed in the VMs. The application, referred to as server in the reminder of this paper, handles billing related requests. The server instances running on the VMs controlled by hypervisor1 are active in the sense that they are the primary consumers of the requests. The remaining two VMs under the control of hypervisor2 are running one passive instance of the server.

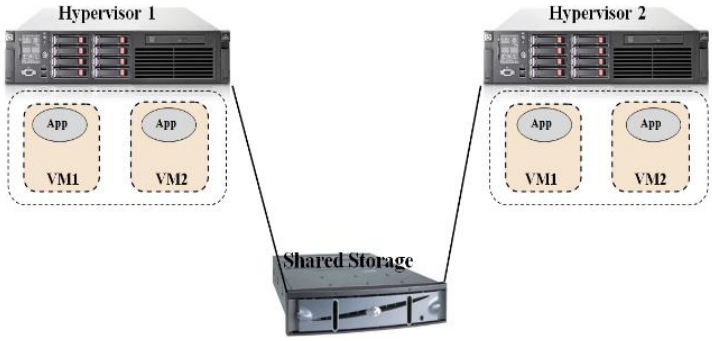


Figure 5.1. Network Plan

Each active server is clustered together with one passive server. Thus, two clusters are created. Both the active and the passive server in a cluster can receive requests. However, all traffic received by the passive server is forwarded to the corresponding active server. The active server then sends the response back to the passive server.

Finally, the passive server sends the response to the requesting system. Traffic going directly to the active server is handled without involving the passive server.

Another separate server runs a simulator that impersonates a requesting system in order to generate load towards the servers running in the clusters. The simulator is also located in the same LAN, but is not shown in Figure 5.1.

5.3.2 Test Cases

Two kinds of tests are considered in this study: performance tests and live migration tests.

5.3.2.1 Performance tests

In these tests, we vary the number of CPU cores (logical cores) in the VMs as well as the load towards the application.

We have three different core configurations: 6, 12 and 16 cores. For test cases with 12 cores and 16 cores the RAM for the VM is set to 24 GB, but for test case with 6 cores, the RAM size set to 14 GB for each of the VMs. This is an application specific setting that is recommended by the manufacturer. A single cluster is used for the case with 12 and 16 cores, respectively. Both clusters are used when testing the 6 cores configuration in order to assess the performance of two 6-core systems versus the performance a single 12-core system.

There are five load levels used in this test: 500, 1500, 3000, 4300, and 5300 incoming requests per second (req/s).

For each setup the following metrics are measured: CPU utilization, disk utilization and response time.

CPU utilization and disk utilization are measured inside the hypervisor on both servers using the commands presented in Table 5.1. For disk utilization, we consider only write operations to the shared storage shown in Figure 5.1. The response time is measured inside the simulator as the duration from the instant a request is sent from the simulator to the application until the simulator receives the corresponding reply.

5.3.2.2 Live Migration tests

In these tests, we measure CPU and disk utilization during live migration. Four VMs with 6 cores CPU and 14 GB of RAM were created. For each configuration, a single VM (active server, e.g., VM1 on Hypervisor1 in Figure 5.1) is migrated from the source host to the destination host while the simulator creates a load of 100 req/s for the VM. At the same time the other VM (e.g. VM2 on Hypervisor1 in Figure 5.1) on the source host is receiving 1500 req/s. The other VMs (VM1 and VM2 on Hypervisor2 in Figure 5.1) on the destination host receive negligible traffic in the form of 100 req/s and thus are not completely idle.

Table 5.1. CPU and disk utilization command API

Virtualization System	**Command Interface**	
	CPU Utilization	***Disk Utilization***
KVM	ssh + sar	ssh + iostat
VMware	vCenter Server-performance graphs	vCenter Server-performance graphs
XenServer	ssh + xentop	ssh + iostat
Non-virtualized Server	ssh + sar	ssh + iostat

The application manufacturer considered this as a realistic example when one would like to migrate a VM to load-balance the system.

In addition to CPU and disk utilization, we measure the downtime and the total migration time. The total migration time is obtained from the hypervisor for KVM and XenServer, and from vCenter for VMware (see Table 5.1). Downtime is

defined as the time from the instant when the VM is suspended on the source host (Hypervisor1 in Figure 5.1) until the VM is restarted on the destination host (Hypervisor2 in Figure 5.1). We measured the downtime inside the simulator and our results indicate that it corresponds to the maximum response time of the application.

5.4 Comparison between KVM, VMware and XenServer

In Section 5.4.1, the KVM, VMware and XenServer virtualization systems are compared in terms of CPU utilization (6 cores, 12 cores, and 16 cores), disk utilization and response time. These values have been measured for different loads (500, 1500, 3000, 4300, 5300 req/s) except for XenServer, which could not handle the highest load (5300 req/s).In Section 5.4.2, we compare the CPU utilization and the disk utilization during live migration, and in Section 5.4.3, we compare the total migration time and downtime of the VMs during live migration for the KVM, VMware and Xen Server setups, respectively.

5.4.1 CPU, Disk Utilization and Response Time (6 cores, 12 cores, 16 cores)

CPU and disk utilization are measured inside the hypervisors. We also performed the same measurements on the non-virtualized (target) server in order to establish a baseline for our results (see

Table). The response time is measured in the simulator.

As shown in Figure 5.2, Xen has the highest CPU utilization (approximately 80%) in the test case with 16 cores. Because of this high CPU utilization the application failed for traffic loads higher than 4300 req/s. KVM and VMware CPU utilization increases proportional to the load with an increase rate similar to that of the target. In Figure 5.3, we can observe that again Xen CPU utilization is significantly higher compared to VMware, KVM and the target in case of 12 cores. As shown in Figure 5.4, KVM, VMware and the target CPU utilization in case of 6 cores, are almost identical while Xen CPU utilization is the highest and at the highest point is around 70% which is the 20% higher CPU utilization compared to KVM, VMware and the target.

In Figure 5.5, we can observe that in case of 16 cores, VMware has the highest disk utilization, up to 25000 KB/s. KVM and Xen the disk utilization is linearly increasing with a rate similar to that of the disk utilization of the target. However, for KVM's and Xen's disk utilization is always around 2000 KB/s higher compared to the target. As shown in Figure 5.6, in case of 12 cores, Xen and KVM disk utilization is 5000 KB/s higher compared to the target while disk utilization for VMware is the highest, with a maximum around 30000 KB/s. In Figure 5.7, we can observe that VMware has the highest disk utilization compared to KVM and

Xen, which show 34000 KB/s at the highest point. Xen's disk utilization in case of 6 cores is higher than KVM. The maximum disk utilization for Xen is around 25000 KB/s while the maximum KVM disk utilization is around 20000 KB/s. That is 5000 KB/s higher compared to the target but still the lowest compared to other virtualization systems.

Figure 5.8 shows that the response time of the application when using Xen is the highest for all traffic loads except for loads higher than 4300 req/s. Since for loads higher than 4300 req/s the application failed when using Xen, KVM has the highest response time after Xen, and at the highest point is around 25 ms in case of 16 cores. The response times of the application when using VMware is similar to the response times we had on the target. As shown in Figure 5.9, the response time of the application when using Xen reaches 26 ms at the highest point. In case of KVM the application has also high response times with a maximum of around 20 ms, which is higher than VMware's. The application response times when using VMware is similar to the response times of the application on the target. In Figure 5.10, we can observe that response time of the application when using Xen at the highest point is more than 25 ms, which is twice the application response time in case of the non-virtualized target. In the case of the 6 cores configuration using KVM the response time increases with a similar rate to the case when using VMware. However, for KVM and VMware the response times are around 5 ms higher compared to the target.

5.4.2 CPU, Disk Utilization and ResponseTime during Live Migration

CPU utilization is measured inside the hypervisors on both the source and the destination servers, during the live migration. Disk utilization is also measured inside both hypervisors. We initiate a migration after the system has been running for 15 minutes.

As shown in Figure 5.12, KVM's CPU utilization on the source is around 26% before the live migration begins. The CPU utilization on the destination is around 6%. After the live migration has been started, the CPU utilization first increases to 35% and then decreases to 18% on the source. However, on the destination server the CPU utilization settles around 13% after the live migration. As shown in Figure 5.12, VMware's CPU utilization before live migration is around 20% on the source hypervisor and around 4% on the destination hypervisor. When the live migration has been started, the CPU utilization on source increases to about 34% and remains at that level during the live migration. On the destination, the CPU utilization becomes around 15% after the live migration has started. After the live migration has stopped, the CPU utilization decreases to around 15% on the source hypervisor and to around 10% on the destination hypervisor. In Figure 5.12, we can observe that the Xen CPU utilization before live migration is around 34% on the source hypervisor and around 7% on the destination hypervisor. In the beginning of the live migration, the CPU utilization on source increases to around 40% and on the destination the Xen CPU utilization increases to around 13%. After the live

migration is completed, the CPU utilization on the source decreases to around 29%, while on the destination's CPU utilization increases to 15%.

As shown in Figure 5.13, KVM's disk utilization on the source is around 10000 KB/s before live migration. On the destination, the disk utilization is around 6000 KB/s before live migration. After the live migration has started, the disk utilization on the source decreases to 9000 KB/s, while on the destination's disk utilization increases to 7000 KB/s. As shown in Figure 5.13, VMware's disk utilization is around 15000 KB/s on the source before live migration while on the destination the disk utilization is around 7000 KB/s. After the live migration, the disk utilization on the source decreases to around 13000 KB/s and on the destination it increases to around 9000 KB/s. In Figure 5.13 we can observe that the Xen disk utilization before the live migration is around 13000 KB/s on the source and around 6000 KB/s on the destination. When the live migration has started, the disk utilization increases to around 30000 KB/s on the source and to around 23000 KB/s on the destination. After the live migration has completed, the disk utilization on the source decreases to around 9000 KB/s and on the destination the disk utilization increases to around 10000 KB/s.

5.4.3 Downtime and Total Migration Time

The downtime has been obtained from the maximum response time, which is measured inside simulator during the live migration. Downtime corresponds to the time that application is not available and the VM is suspended.

As shown in Figure 5.11, the response time of the application when using KVM as hypervisor is around 1 ms before the live migration is started, but when the VM is suspended the response time increases to 700 ms. So the application was down for less than 700 ms. In Figure 5.11, we can observe that the response time of the application when using VMware as hypervisor is around 1 ms, but when the VM is totally down the application response time increases to 3000 ms. So the application downtime was around 3000 ms. As shown in Figure 5.11, before the live migration starts the application response time when using Xen is around 4 ms. When the live migration begins, the response time increases to 280 ms. So the application was down for less than 4 ms.

The total migration time is calculated inside the source hypervisor. It corresponds to the time that the VM started to be migrated until the complete VM state has been transferred to the destination hypervisor (see Figures 5.12-5.13). The total migration time for VMware, KVM and Xen is around 2 minutes.

5.5 Related Work

In recent years, there have been several efforts to compare different live migration technologies. Xiujie et al. [1] compare the performance of vMotion and

XenMotion under certain network conditions defined by varying the available bandwidth, link latency and packet loss. Their results show that vMotion produces less data traffic than XenMotion when migrating identical VMs. However, in networks with moderate packet loss and delay, which are typical in a Virtual Private Network (VPN), XenMotion outperforms vMotion in total migration time.

Tafa et al. [2] compare the performance of three hypervisors: XEN-PV, XEN-HVM and Open-VZ. They simulated the migration of a VM using a warning failure approach. The authors used a CentOS tool called "Heartbeat" that monitors the well-being of high-availability hosts through periodic exchanges of network messages. When a host fails to reply to messages the tool issues a failure notification that causes the hypervisor to migrate the VM from the "dead" host to one that is "alive". Further, they compared CPU usage, memory utilization, total migration time and downtime. The authors have also tested the hypervisor's performance by changing the packet size from 1500 bytes to 64 bytes. From these tests they concluded that Open-VZ has a higher CPU usage than XEN-PV, but the total migration time is smaller for Open-VZ (3.72 seconds for packet size of 64 bytes) than for XEN-PV (5.12 seconds for packet size of 64 bytes). XEN-HVM has lower performance than XEN-PV; especially regarding downtime. XEN-HVM had16 ms downtime while XEN-PV had 9 ms downtime for packet size of 64 bytes compared to our results with the large application we have got 300 ms downtime for Xen and total migration time of around 2 minutes.

In Chierici et al. [3] and Che et al. [29] present a quantitative and synthetically performance comparison between Xen, KVM and OpenVZ. They used several benchmarks (NetIO, IOzone, HEP-Spec06, Iperf and bonnie++) to measure CPU, network and disk accesses. According to their measurements, the OpenVZ has the best performance; also Xen hypervisor offers good performance while KVM has apparently low performance than OpenVZ and Xen.

There has been a similar study to our work carried out by Hicks, et al. [14], in which the authors focused only on memory migration and storage migration in the KVM, XenServer, VMware and Hyper-V virtualization systems. However, they did not consider CPU utilization of hypervisor during live migration in their study.

Clark et al. [27] introduced a method for the migration of entire operating system when using Xen as a hypervisor. They have tested different applications and recorded the service downtime and total migration time. Their results show 210 ms downtime for SPECweb99 (web-server) and 60 ms downtime for Quake3 (game server) during the migration.

Du et al. [24] proposed new method called Microwiper which makes less dirty pages for live migration. They implemented their method on the pre-copy based live migration in Xen hypervisor. They've tested two different programs with one with fixed memory writes and the other one with very quick memory writes. They compared the downtime and total migration time when using their method (Microwiper) versus the original Xen live migration (XLM). Their results show the

original Xen live migration gets 40 ms downtime for VM memory size of 1024 MB when running quick memory writes program and total migration time of 11 seconds while their technique (Microwiper) decreases the downtime so it became around 10 ms but they got the same total migration time.

Web 2.0 application [33] has been evaluated by Voorsluys et al. [25] in terms of downtime and total migration time during live migration. They run XenServer as a hypervisor on their VM hosts. According to their experiments downtime of their system when serving 600 concurrent users is around 3 seconds and their total migration time is around 44 seconds which is much higher compared to our results because of the application that they've used also their setup is different.

Jo et al. [26] implemented a technique to reduce the duplication of data on the attached storage. They used different applications, RDesk I and II, Admin I, etc. and they measured the down time and total migration time during live migration when using XenServer as hypervisor. Their experiment shows 350 seconds total migration time for the original Xen live migration when the maximum network bandwidth is 500 megabits per second while using their proposed technique reduces this number to 50 seconds when duplication ratio is up to 85 percent.

5.6 Conclusion and Future Work

The results of the performance tests for different configurations of number of CPU cores show that KVM and VMware CPU utilization is almost identical and similar to CPU utilization on the target machine (non-virtualized) while XenServer has the highest CPU utilization with a maximum around 80%. In terms of disk utilization, the results indicate that KVM and Xen have similar disk utilization while VMware has the highest disk utilization (around 30000 KB/s for the highest load). The response time of the application is the highest when using Xen as hypervisor showing around 25 ms at the highest point. For KVM and VMware, the response time is almost similar (around 20 ms).

In general, KVM and VMware perform better in terms of CPU utilization while Xen CPU utilization is the highest. In terms of disk utilization KVM and Xen have similar performance while VMware has the highest disk utilization. Further, in terms of response time Xen has the longest response times compared to KVM and VMware.

As the results have shown, the CPU utilization during live migration is lower for KVM than for VMware while Xen had the highest CPU utilization during live migration. The disk utilization when KVM is used is 1000 KB/s lower compared to VMware during the migration.

For VMware, the downtime is measured to 3 seconds during live migration. For KVM and Xen the measured downtime are only 0.7 seconds and 0.3 seconds, respectively.

In general, the results presented in this study show that both VMware and KVM perform better in terms of application response time and CPU utilization for a configuration of two VMs with 6 cores each, compared to a configuration with a single VM with 16 or 12 cores. Xen's performance is below that of the two other virtualization systems tested. However, Xen's live migration technology, XenMotion, performs better than VMware's vMotion and KVM live migration technology in terms of downtime.

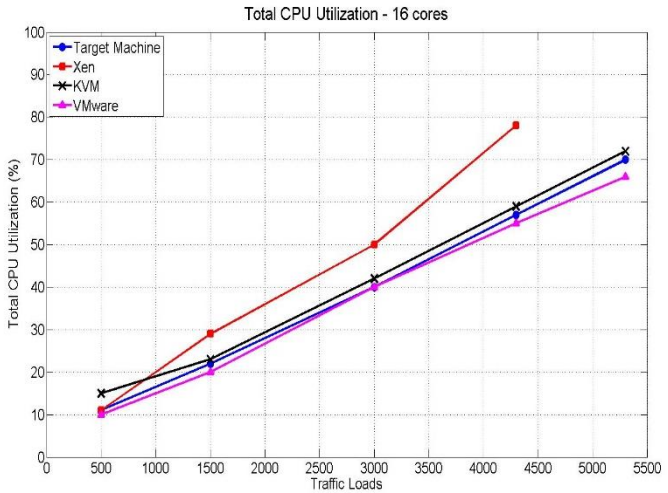


Figure 5.2. KVM, VMware and Xen CPU utilization for 16 cores

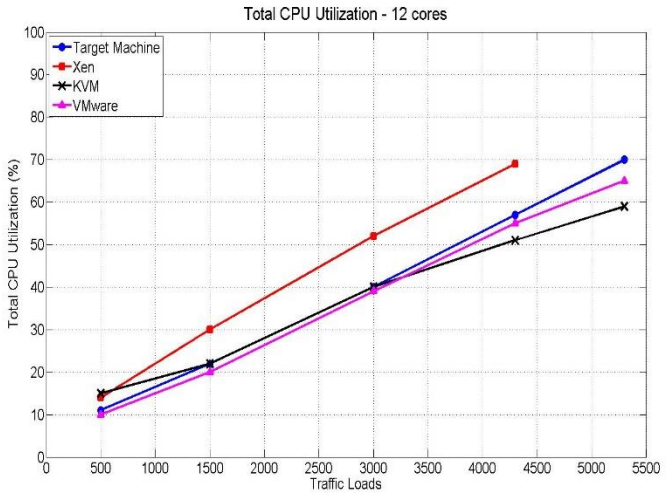


Figure 5.3. KVM, VMware and Xen CPU utilization for 12 cores

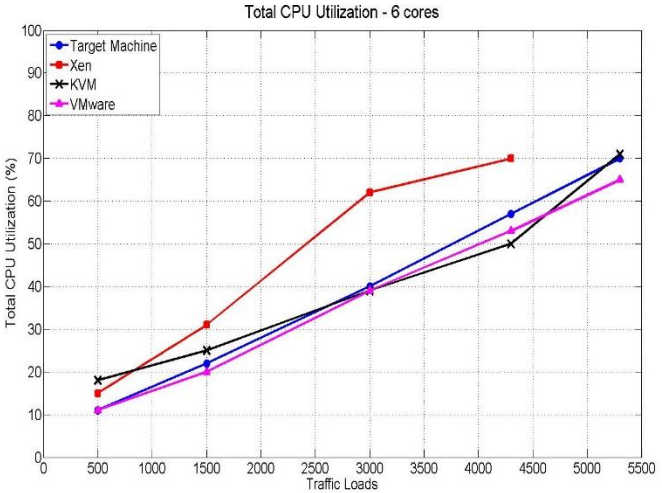


Figure 5.4. KVM, VMware and Xen CPU utilization for 6 cores

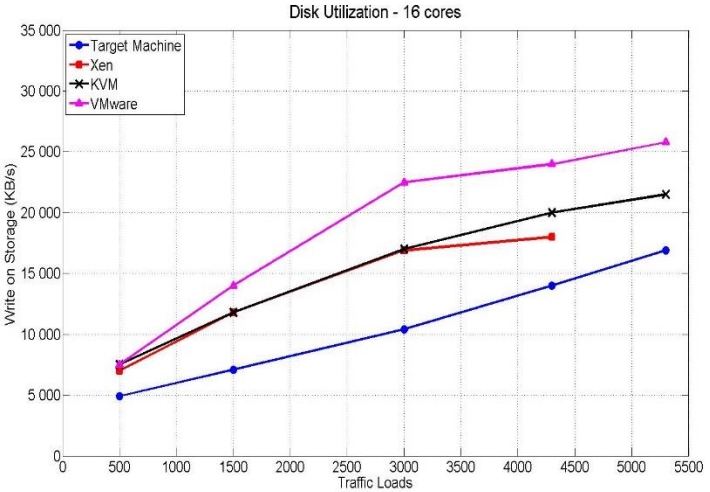


Figure 5.5. KVM, VMware and Xen disk utilization for 16 cores

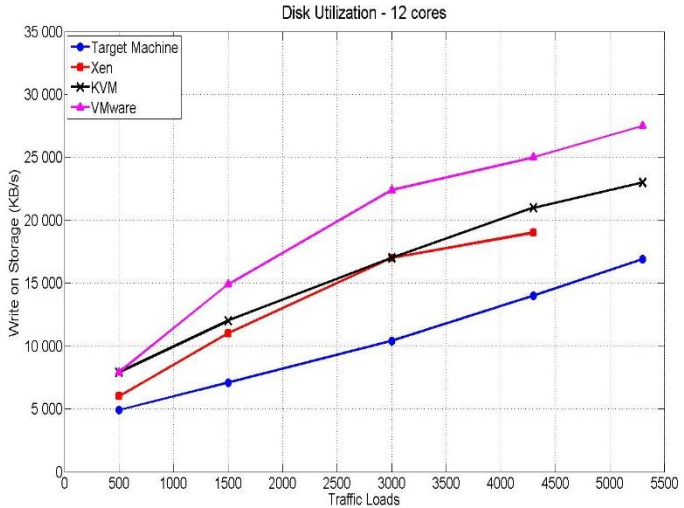


Figure 5.6. KVM, VMware and Xen disk utilization for 12 cores

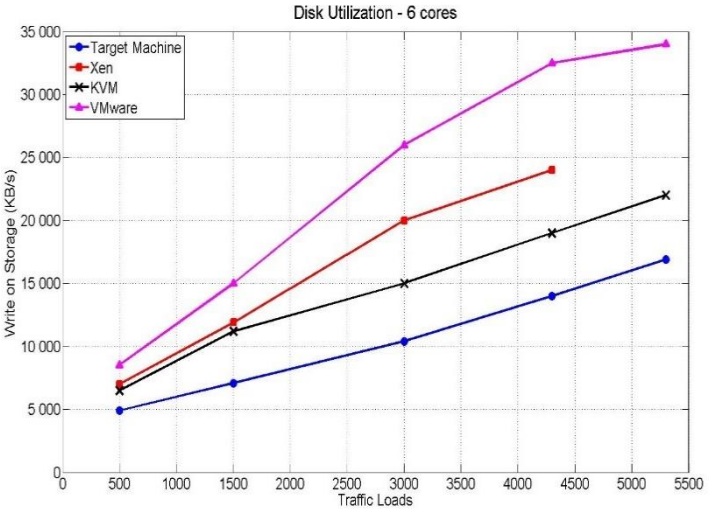


Figure 5.7. KVM, VMware and Xen disk utilization for 6 cores

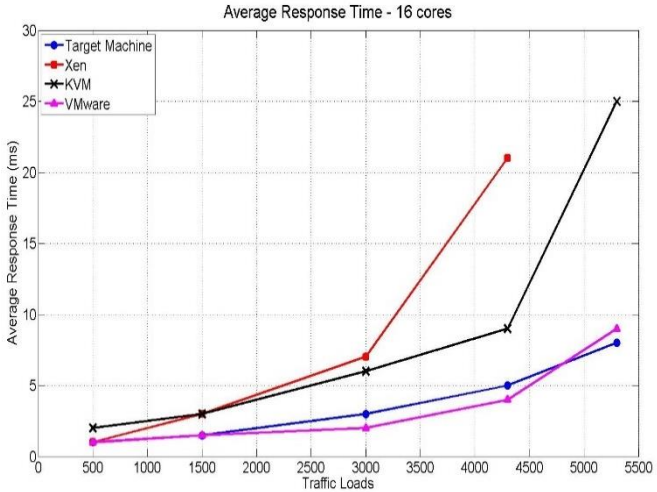


Figure 5.8. KVM, VMware and Xen response time for 16 cores

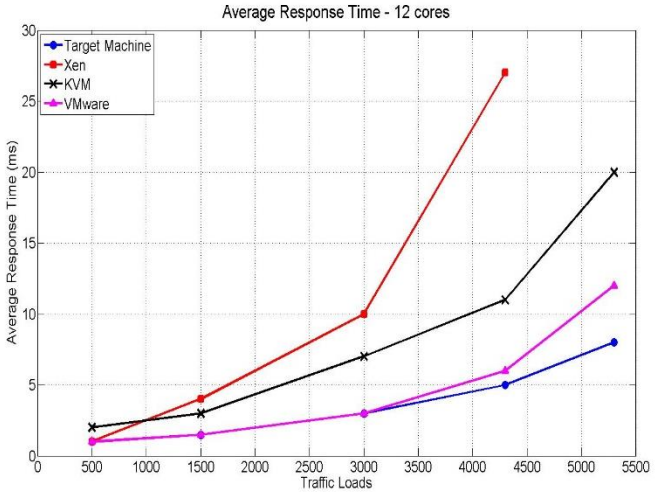


Figure 5.9. KVM, VMware and Xen response time for 12 cores

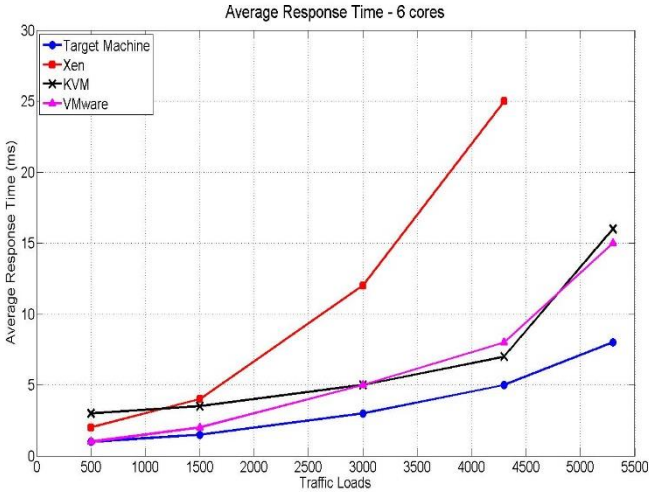


Figure 5.10. KVM, VMware and Xen response time for 6 cores

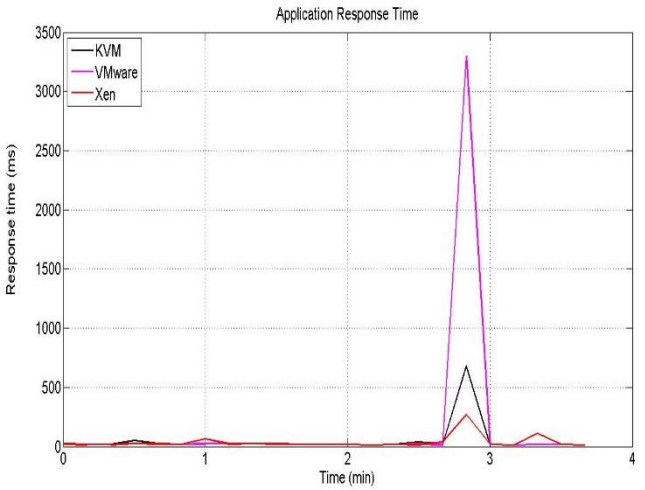


Figure 5.11. KVM , VMware and Xen response time during live migration

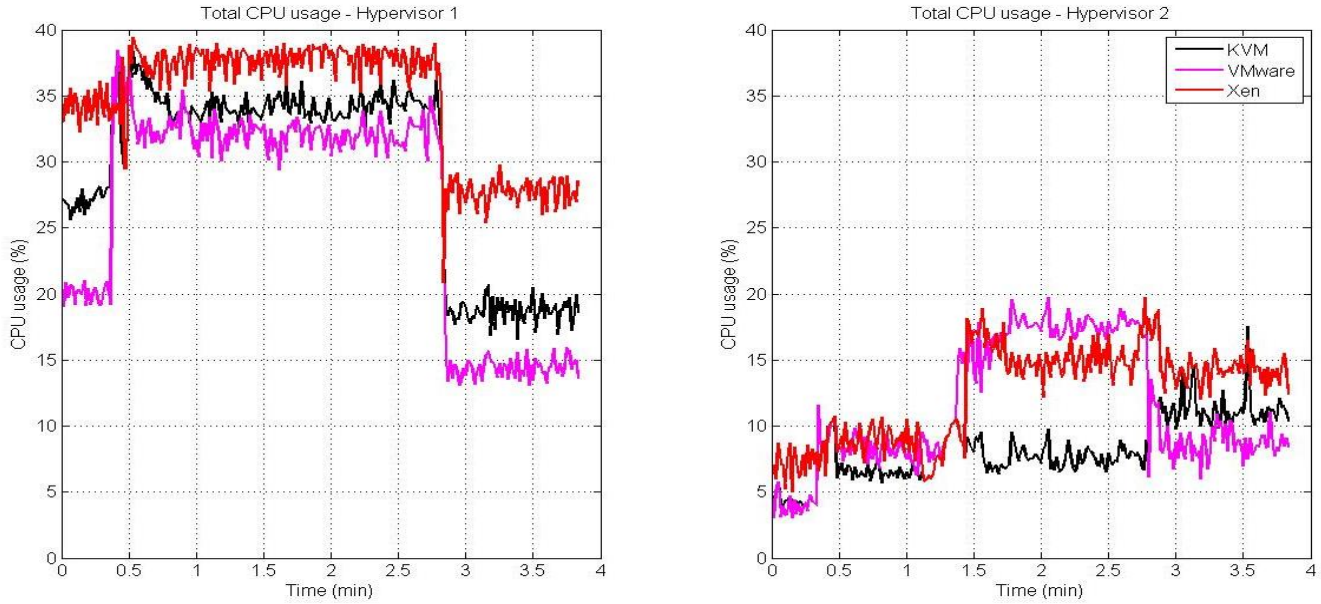


Figure 5.12. KVM , VMware and Xen CPU utilization during live migration

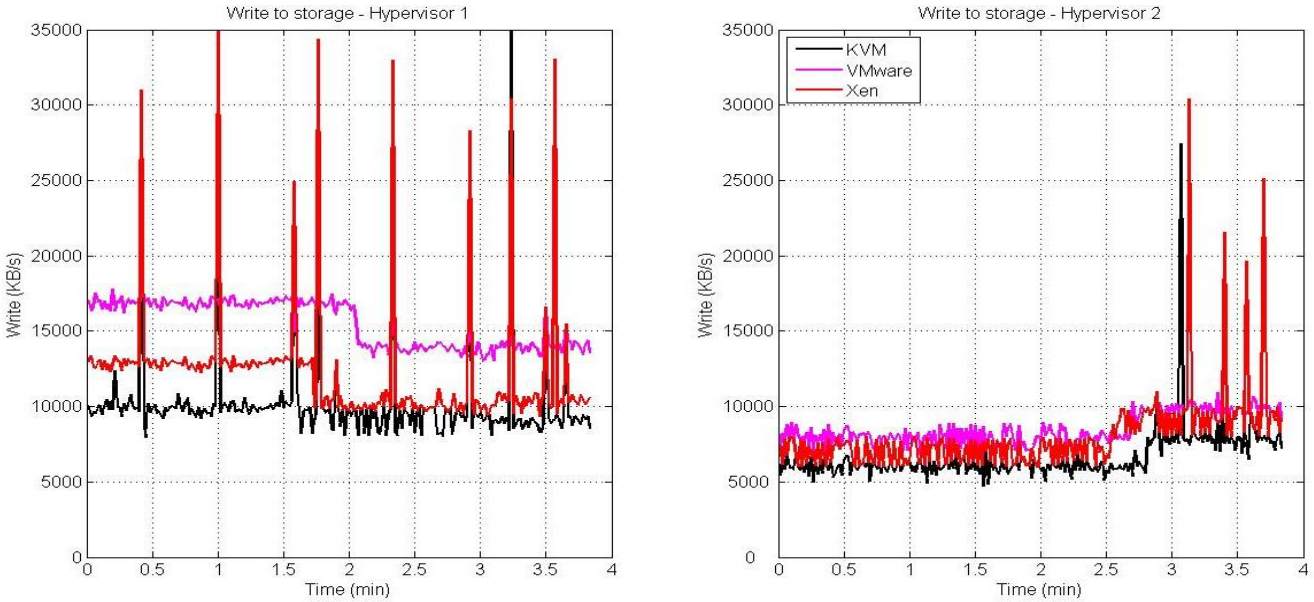


Figure 5.13. KVM , VMware and Xen disk utilization during live migration

5.7 References

[1] F. Xiujie, T. Jianxiong, L. Xuan, and J. Yaohui, "A Performance Study of Live VM Migration Technologies: vMotion vs XenMotion," Proceedings of the International Society for Optical Engineering, Shanghai, China, 2011, pp. 1-6.

[2] I. Tafa, E. Kajo, A. Bejleri, O. Shurdi, and A. Xhuvani, "The Performance between XEN-HVM, XEN-PV And OPEN-VZ During Live Migration," International Journal of Advanced Computer Science and Applications, 2011, pp. 126-132.

[3] A. Chierici and R. Veraldi, "A Quantitative Comparison between Xen and KVM," 17th International Conference on Computing in High Energy and Nuclear Physics, Boston, 2010, pp. 1-10.

[4] D. Huang, D. Ye, Q. He, J. Chen, and K. Ye, "Virt-LM: a benchmark for live migration of virtual machine," ACM SIGSOFT Software Engineering Notes, USA, 2011, pp. 307-316.

[5] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "KVM: the linux virtual machine monitor," OLS, Ottawa, 2007, pp. 225-230.

[6] Red Hat. (2009). "KVM- Kernel based virtual machine,"[Online]. Available from:http://www.redhat.com/rhecm/rest-rhecm/jcr/repository/collaboration/jcr:system/jcr:versionStorage/5e7884ed7f00000102c317385572f1b1/1/jcr:frozenNode/rh:pdfFile.pdf , 2014-03-10

[7] M.R. Hines and K. Gopalan, "Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning," international conference on virtual execution environments, USA, 2009, pp. 51-60.

[8] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmorth, "Evaluation of delta compression techniques for efficient live migration of large virtual machines," 7th ACM SIGPLAN/SIGOPS International conference on Virtual execution environments, USA, 2011, pp. 111-120.

[9] S. Lowe, "Mastering VMware vSphere 5," Book, 2011.

[10] E. L. Haletky, "VMware ESX and ESXi in the Enterprise:Planning Deployment of Virtualization Servers," Upper Saddle River, NJ: Prentice Hall, 2011.

[11] A.Kovari and P.Dukan, "KVM & OpenVZ virtualization based IaaS Open Source Cloud Virtualization Platform," 10th Jubilee International Symposium on Intelligent Systems and Informatics, Serbia, 2012, pp. 335-339.

[12] A.J. Younge, R. Henschel, and J.T. Brown, "Analysis of Virtualization Technologies for High Performance Computing Environments," 4th IEEE

International Conference on Cloud Computing, Washington, DC, 2011, pp. 9-16.

[13] A. Warfield, et al., ”Live Migration of Virtual Machines,” Proceedings of the 2nd conference on Symposium on Network Systems Design and Implementation, USA, 2005, pp. 273-286.

[14] A. Hicks, et al., “A Quantitative Study of Virtual Machine Live Migration,” Proceedings of the ACM Cloud and Autonomic Computing Conference, USA, 2013, pp. 1-10.

[15] J. Wang, L. Yang, M. Yu, and S. Wang, ”Application of Server Virtualization Technology Based on Citrix XenServer in the Infromation Center of the Public Security Bureau and Fire Service Department,” Proceedings of the Computer Science and Society, Kota Kinabalu, 2011, pp. 200-202.

[16] KVM. [Online]. Available from: http://www.linux-kvm.org/page/Main_Page , 2014-03-10

[17] XenServer. [Online]. Available from: http://www.citrix.com/products/xenserver/overview.html , 2014-03-10

[18] VMware. [Online]. Available from: http://www.vmware.com/ , 2014-03-10

[19] Ch. Cai and L. Yuan, ”Research on Energy-Saving-Based Cloud Computing Scheduling Strategy,” Journal of Networks, 2013, pp. 1153-1159.

[20] E.Michael and F. Janos, “A Survey of Desktop Virtualiztion in Higher Education: An Energy-and Cost-Savings Perspective,” 19th Americas conference on Information Systems, 2013, pp. 3139-3147.

[21] X. Li, Q. He, J. Chen, K. Ye, and T. Yin, “Informed Live Migration Strategies of Virtual Machines for Cluster Load Balancing” Proceedings of the 8th IFIP International Conference, 2011, pp. 111-122.

[22] Z. Wenyu, Y. Shaoubao, F.Jun, N. Xianlong, and S. Hu, “VMCTune: A Load Balancing Scheme for Virtual Machine Cluster Based on Dynamic Resource Allocation” Proceedings of the 9th International Conference on Grid and Cloud Computing, 2010, pp. 81-86.

[23] P. Riteau, C. Morin, and T. Priol, “Shrinker: Efficient Live Migration of Virtual Machines” Concurrency and Computation: Practice and Experience, 2013, pp. 541-555.

[24] Y. Du, H. Yu, G. Shi, J. Chen, and W. Zheng, ”Microwiper: Efficient Memory Propagation in Live Migration of Virtual Machines,” 39th International Conference on Parallel Computing, 2010, pp. 141-149.

[25] W. Voorsluys, J. Broberg, S.Venugopal, and R. Buyya, “Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation,”

Proccedings of the 1st International Conference on Cloud Computing, 2009, pp. 254-265.

[26] Ch. Jo, E. Gustafsson, J. Son, and B. Egger, “Efficient Live Migration of Virtual Machines Using Shared Storage,” Proceedings of the 9th International Conference on Virtual Execution Environments, 2013, pp. 41-50.

[27] C. Clark, et al., “Live Migration of Virtual Machines,” Proceedings of the 2nd symposium on Networked Systems Design and Implementation, 2005, pp. 273-86.

[28] S. Akoush, R. Sohan, A. Rice, A.W. Moore, and A. Hopper, “Predicting the Performance of Virtual Machine Migration,” Proceedings of the 18th IEEE/ACM international symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, 2010, pp. 37-46.

[29] J. Che, Y. Yu, C. Shi, and W. Lin, “A Synthetical Performance Evaluation of OpenVZ, Xen and KVM,” Proceedings of the IEEE conference on Asia-Pacific Services Computing, 2010, pp. 587-594.

[30] S. Kikuchi and Y. Matsumoto, “Impact of Live Migration on Multi-tier Application Performance in Clouds,” Proceedings of the 5th IEEE international conerence on Cloud Computing, 2012, pp. 261- 268.

[31] H. Liu, H. Jin, Ch. Xu, and X. Liao, “Performance and Energy Modelling for Live Migration of Virtual Machines,” Proceedings of the conference on Cloud Computing, 2013, pp. 249-264.

[32] S. Kikuchi and Y. Matsumoto, “Performance Modelling of Concurrent Live Migration Operations in Cloud Computing Systems using PRISM Problemabilistic Model Checker,” Proceedings of the IEEE 4th international conference on Cloud Computing, 2011, pp. 49-56.

[33] L. Wang, et al., “Cloud Computing: a Prespective Study,” Proceedings of the New Generation Computing conference, 2010, pp. 137-146.

[34] J. Che, Q. He, Q. Gao, and D. Huang, “Performance Measuring and Comparing of Virtual Machine Monitors,” Proceedings of the 5th interantional conference on Embedded and Ubiquitous Computing, 2008, pp. 381-386.

6 Performance Implications of Over-Allocation of Virtual CPUs

Abstract

A major advantage of cloud environments is that one can balance the load by migrating virtual machines (VMs) from one server to another. High performance and high resource utilization are also important in a cloud. We have observed that over-allocation of virtual CPUs to VMs (i.e. allocating more vCPUs to VMs than there CPU cores on the server) when there are many VMs running on one host can reduce performance. However, if we do not use any over-allocation of virtual CPUs we may suffer from poor resource utilization after VM migration. Thus, it is important to identify and quantify performance bottlenecks when running in virtualized environment. The results of this study will help virtualized environment service providers to decide how many virtual CPUs should be allocated to each VM.

6.1 Introduction

There has been many studies on resource allocation in virtualized environments [1]. A virtual machine (VM) is usually configured with a number of virtual CPUs (vCPUs). One can decide to use the physical CPU cores in different ways, e.g., one can have a small number of VMs with a large number of vCPUs each, or a large number of VMs with a small number of vCPUs each. Previous studies show that different ways of using the physical CPU cores resources affects system performance [10]. If we sum up the number of vCPUs in all VMs on a physical server, we could end up in three situations: the total number of vCPUs exceeds the number of physical CPU cores (over-allocation), the total number of vCPUs is the same as the number of physical CPU cores (balanced allocation), or the number of vCPUs is smaller than the number of physical cores (under-allocation). Under-allocation clearly results in sub-optimal resource utilization since some physical cores are not used. One reason for over-allocating resources in virtualized environments is live migration. In modern data centers virtual machines migrate from host to host based on the pre-defined rules without involving a human operator. This type of fully automatic load balancing can provide high and even resource utilization over a cluster of physical servers, e.g., VMware's Distributed Resource Scheduler (DRS) [11]. One important challenge in such automatic environments is efficient resource utilization. If we do not use over-allocation, we may not be able to use all the physical cores after live migration. E.g., assume there are two hosts each with 24 CPU cores, and there are four virtual machines running on each host and each VM has 6 virtual CPUs (vCPUs), i.e., balanced allocation.

If one VM is migrated to the other host, the total number of VMs on the source host becomes three, and they will only have a total number of 18 vCPUs, i.e., six of the physical CPU cores will not be used. However, if each VM had 24 vCPUs allocated to it, there would be enough vCPUs to utilize the physical cores. There is a risk of achieving poor performance due to over-allocation of resources. One reason for this is that when we use over allocation the VMs will time share the CPU resources; in the case of no overallocation there is no or very little time sharing. Time sharing increases the number of context switches between VMs. The overhead due to excessive context switching between VMs, and other contention related activities, will decrease performance. These performance bottlenecks should be identified, quantified, and avoided. We have built a testbed using VMware. We have tested and analyzed vCPU over-allocation as well as no over-allocation scenarios; we have also compared the performance of a small number of VMs with many vCPUs with a larger number of VMs with fewer vCPUs each. In this study we have used a large industrial telecommunication application and measured the performance of the application under different conditions. The results of this study will help virtual environment service providers to decide how many virtual CPUs should be allocated to each VM.

6.2 Related Work

Over-allocating of resources has been used to increase the application performance during worst case demand. However, since servers operate most of the time at very low utilization level, most of these resources become wasted in non-peak times. Zhu et al. [12] and Liu et al. [13] designed resource controllers to modify CPU cycles in order to control CPU utilization of VMs. Padala et al. [14] studied over-allocation of the CPU resources on virtualized servers and build a model to control the CPU allocation dynamically to the individual VM. Espadas et al. [4] establish measurements for under and over provisioning of virtualized resources. They have also proposed a resource allocation mechanism in order to deploy Software-as-a-Service (SaaS) applications. Our work is distinct from their experiment in terms of the application that we have used; we have used a real-time telecommunication application while they have used web applications. Yu et al. [5] proposed try-before-buy technique which is a thin-provisioning approach for resource allocation on the Xen hypervisor. In their approach, the try-step is when a minor share of physical resources will be allocated to the VM and performance will be measured. Then according to the conducted results, in the buy-step the amount of allocated resources will be increased if necessary. Watson et al. [15] proposed a probabilistic performance model which can based on performance measurements on the application, predicts how much CPU resources needs to be allocated to VMs to get better performance. Both [5] and [15], have used the Xen hypervisor while we have used VMware. Garg et al. [16] have considered resource allocation, and proposed a model in a datacenter that runs different type of applications. However, none of their applications were real-time

telecommunication applications. Wang et al. [17] investigate the impact of software resource allocation such as thread pool size on throughput in various servers (e.g., Apache web server). Later, Li et al. [18] designed an algorithm to find the best software resource allocations. Heo et al. [19] studied memory over-allocation and presented their experimental results. Pooja et al. [1] provided performance measurements of memory intensive applications under different conditions, e.g., size of the memory allocated to each virtual machine changes under the given load, using windows server 2008 R2 and 2012. Their results indicate that when the number of page faults per second becomes constant, allocating more memory will not improve performance. In our experiment we have used a CPU intensive application. Larcheveque and Dubois [20] proposed an algorithm to keep the CPU utilization of a physical machine at 66% and when a physical machine becomes overloaded they migrate VMs to balance the load.

6.3 Experimental Setup

6.3.1 Testbed

Eight servers have been used as the hosts. On these hosts we run VMware ESXi 5.5.0. On top of VMware, RedHat Enterprise Linux, version 6.2 has been installed as the guest OS. Each server is equipped with 128 GB of RAM, two 6-core CPUs (2x Intel XEON 2.0 GHz) with hyperthreading enabled in each core (i.e., a total of 24 logical cores) [21][22][23].

In order to reduce the time for live migration one would like that all servers in a cluster share the same disk; in this case no files need to be moved during a live migration of a VM. Having one large physical disk (or disk array) for a large cluster can, however, become a single point of failure and a performance bottleneck. A strong trend in virtualization and cloud computing is to use the so called distribute storage systems.

In a distributed storage system the disks are physically distributed to the servers in the cluster, thus avoiding the single point of failure and performance bottleneck problems. However, a software layer creates an image of one shared virtual disk, thus avoiding the need to migrate files during live migration. One high performing distributed storage systems is the Compuverde system [24].

All servers in our cluster are connected to Compuverde distributed storage system. As shown in Figure 6.1, the distributed storage system consists of two components: 1) Compuverde software, that is installed inside a VM and 2) data storage which consists of SSD cache (7x Intel 330 60 GB (RAID 10+hotspare)) and disk persistent storage (8x Intel 330 60 GB (RAID 5(7+1))). As shown in

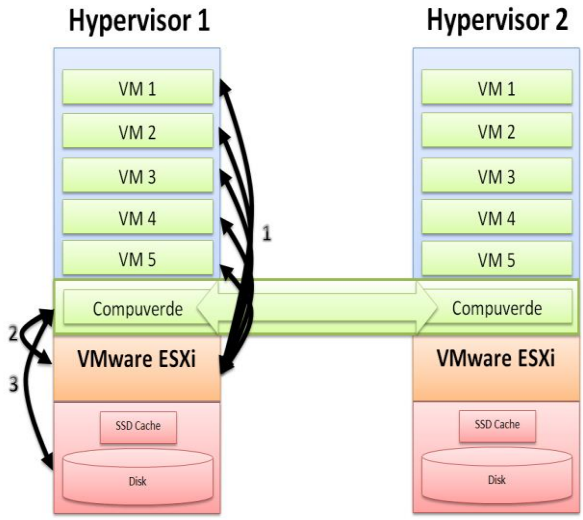


Figure 6.1. Compuverde storage system. The numbers (1-3) indicate the order in which an access to disk is handled. 1) a VM makes a disk access. 2) the hypervisor forwards the request to the Compuverde software that runs in a VM. 3) the Compuverde software makes the actual disk access possibly including accesses to multiple distributed disks. In case of a read, the result is returned back to the requesting VM in reversed order.

Figure 6.2, all Compuverde VMs communicate with each other in order to maintain and share information about stored data.

The Compuverde software is responsible for managing distribution of data between all storage nodes in different hosts and it has been designed so that it treats all the SSD caches and disks as a virtual Network Attached Storage (vNAS). E.g., during write process, each virtual machine will write data to VMware ESXi, VMware communicates with the Compuverde software through the NFS protocol, and then the Compuverde software will communicate with other Compuverde instances on the other hosts in order to distribute data, and finally it will send data to the storage which is combination of SSD cache and disk storage (see steps 1-3 on the Hypervisor 1 in Figure 6.1).

The same large real-time telecommunication application is installed in all VMs. The application, referred to as telecom server in the reminder of this paper, handles billing related requests. It consists of several hundreds of thousands lines of code.

Another separate server runs a simulator that impersonates a requesting system in order to generate load towards the telecom servers. The simulator is not shown in Figure 6.2. All communications are through Ethernets (Intel X520 SFP+ 10 GB). All clusters are monitored using VMware vCenter (see Figure 6.2).

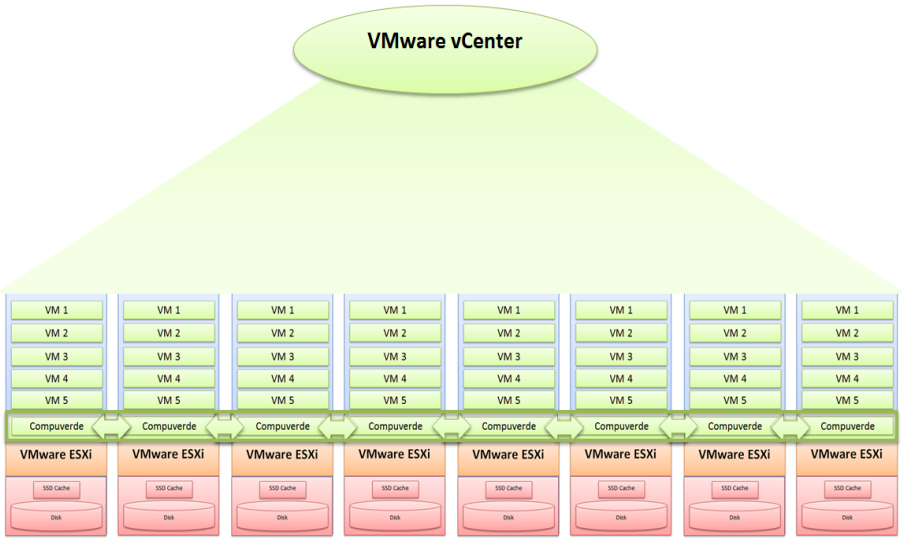


Figure 6.2. Testbed setup

For both the over-allocation and the no over-allocation test cases we used two VMware ESXi hosts and on each host we have created various numbers of VMs. As discussed previously, there are eight physical servers in the cluster, and all these servers contain disks and SSDs that are used by the distributed storage system, i.e., all eight servers are potentially involved in disk accesses. However, in these measurements we only run the application workload on two of the servers.

Each host has 128 GB RAM in total, for both over-allocation and no over-allocation test cases we have allocated 14 GB RAM to each virtual machine; 14 GB RAM is the minimum RAM that is recommended for the application.

On each host 24 cores are available, in order to avoid any interference between the virtual machine that contains the Compuverde software and other VMs, we have created two resource pools, one with 4 cores and the other one with 20 cores. The VM containing Compuverde software was bound to the resource pool with 4 cores and the other VMs are bound to the resource pool that contains 20 cores.

6.3.2 Test cases

We have measured CPU utilization, disk utilization and average response time for cases with and without over-allocation (see Table 6.1 for CPU and disk utilization and response time command API).

6.3.2.1 No over-allocation

There are 20 CPU cores available for virtual machines on each host. For the case with no over-allocation we have divided 20 vCPUs equally between the VMs. Here we defined two test cases: one with a small (two) number of VMs and one with many (five) VMs.

6.3.2.1.1 Two Virtual Machines

In this case we have allocated 10 vCPUs to each virtual machine.

6.3.2.1.2 Five Virtual Machines

In this case we have allocated 4 vCPUs to each virtual machine.

6.3.2.2 Over-allocation

In this case we have allocated 10 or 20 vCPUs to each virtual machine.

6.3.2.2.1 Two Virtual Machines

In this case we have allocated 20 vCPUs to each virtual machine.

6.3.2.2.2 Five Virtual Machines

In this case we have allocated 10 vCPUs to each virtual machine.

6.3.2.2.3 Five Virtual Machines

In this case we have allocated 20 vCPUs to each virtual machine; we have called this case, massive over-allocation.

6.4 Experimental Results

Figure 6.3 shows the CPU utilization on the hypervisor level for all five cases. The CPU utilization on the hypervisor level shows how much the VMs are using the CPU resources that are available on the physical machine. In addition, we have measured CPU usage inside VMs, and we have observed that for the case with no over-allocation (solid blue line in Figure 6.3), CPU usage inside VMs and CPU utilization on the hypervisor level were more or less identical, which was expected since there was no over-allocation in this case. In the case with over-allocation (dashed green line in Figure 6.3) the average CPU usage inside VMs was roughly half of the CPU utilization on hypervisor level, which was also expected since there were twice as many vCPUs as physical cores in this case. Also for the case with

massive over-allocation (dashed blue line in Figure 6.3) the CPU utilization on hypervisor was roughly four times higher than the average CPU usage inside VMs, and in this case there were four times as many vCPUs as physical cores.

If we compare the two red lines (solid and dashed) in Figure 6.3, we see that, in both cases (over-allocation and no over-allocation), the CPU utilizations are almost identical. However, the case with over-allocation has slightly higher CPU utilization for the same workload. When we compare two blue lines (solid and dashed), we see that the CPU utilization when there is over-allocation is much higher compare with the case with no over-allocation (see Figure 6.3). Figure 6.3 also shows that the dashed green line is roughly in the middle between the solid blue line and the dashed blue line. One reason that we could not reach higher load than 5600 req/s in case of five virtual machines and 20 vCPUs could be the high CPU utilization on hypervisor, because when the load is 5600 req/s the CPU utilization is already 82%. If we compare the case with less over-allocation (dashed green line), we can observe that the CPU utilization is less than the case with massive over-allocation (dashed blue line), therefore it reaches the limit (82%) later so we could increase the load up to 6125 req/s.

In Figure 6.4, the write rate to disk is measured on the hypervisor level; we have measured the write rate towards the NFS shared storage. For each VM, the application itself will add an extra write on disk even if there is no load coming in to the telecom application. The extra write for each VM can be observed in the graph and this explains the difference between red lines and the blue/green lines (red lines are representing the cases with two virtual machines while blue/green lines are representing the cases with five virtual machines).

Table 6.1. Command API used for performance measurements

VMware Virtualization System	**Command Interface**		
	CPU Utilization (%)	***Disk Utilization (kb/s)***	***Response Time (ms)***
Inside Hypervisor	vCenter Server-performance graphs	vCenter Server-performance graphs	Write Latency from vCenter Server-performance graphs
Inside Virtual Machine	ssh + sar	ssh + iostat	Inside the application

The application is designed so that when the disk write latency is low (when writing to the disk is fast), it writes more frequently and may write the same data several times (the application keeps track of the write latency and throttles the write frequency according to that). If we compare the two red lines in Figure 6.4, we see that for the case with no over-allocation, the VMs write around 2000 kb/s more compared to the case with over-allocation. The reason is that the write latency is

lower in the case of no over-allocation. The higher write latency in case of over-allocation is probably due to the fact that the two VMs time share the physical cores (in the case of no over-allocation, there is no or very little time sharing). Time sharing and overhead for context switching increase the write latency.

The same happened when we had five VMs. When comparing the two blue lines (dashed line with solid line), we see that when we had no over-allocation the write rate was higher compared to the case with over-allocation. The same discussion applies for the case with five VMs and 10 vCPUs (dashed green line); if we compare this case with the case with five virtual machines and 20 vCPUs (dashed blue line), we see that they are almost identical. While we expect the green dashed line with less over-allocation to be higher than the case with massive over-allocation (dashed blue line). The reason for this behavior is the disk write latency. We have measured the disk write latency and observed that for all three cases when we had over-allocation the write latency is higher compared to other cases with no over-allocation.

This means that the write latency decreases when we have five VMs compared to having two VMs. When there are only two VMs the load is divided between those two VMs, e.g., when the load is 3000 req/s and we have two VMs, each VM will receive 1500 req/s. However, when there are five VMs the same load will be distributed equally between these five VMs, i.e., when the load is 3000 req/s and we have five virtual machines, each VM will receive 600 req/s. Figure 6.4 shows that the write latency is smaller for many VMs with lower load on each VM compared to few VMs with higher load on each VM. Also, Figure 6.4 shows that the cases with no over-allocation (the solid lines) have smaller write latency compare to the cases with over-allocation.

Figure 6.5 shows the average response time of the application. If we compare the cases with five VMs with the cases with two VMs, we see that for five VMs the average response time is a bit lower than the cases with two VMs except for the case with less over-allocation (dashed green line), the response time is lower that all other cases. However, the response time differences are rather limited.

6.5 Conclusion

In this paper, we have presented a detailed comparison between over-allocation and no over-allocation of vCPU resources to the VMs and how it affects the performance. We have measured the performance in terms of CPU and disk utilization inside hypervisor as well as response time of the large industrial telecommunication application.

By allocating more virtual CPUs (vCPUs) than there are physical processor cores to the VMs we can allow live migration of VMs without ending up in a situation where there are less vCPUs than physical cores on some server, i.e., by having over-allocation we can avoid under-allocation after live migration.

However, we have shown that over-allocation of vCPU can result in significant CPU overhead. The overhead increases with the degree of over-allocation. Our measurements show that the cases with 2.5 and 5 times over-allocation results in more overhead compared to the case with 2 times more vCPUs compared to physical cores.

The reason for the increased overhead is that when we have over-allocation the VMs need to time-share the physical cores. In the case of a balanced allocation (or under-allocation) a physical core can be permanently allocated to a VM. Time sharing of physical cores results in context switches, and context switches result in overhead.

In this paper, we have quantified the cost of having different amounts of over-allocation. Providers of virtualized service can use this quantification in order to do a balanced trade-off between the flexibility offered by over-allocation and the performance penalty. Our results indicate that it is in many cases wise to use a moderate level of over-allocation (not exceeding a factor of two) which gives some flexibility at a very modest performance cost.

Our measurements also show that the write latency decreases with the number of VMs sharing the physical server (seen indirectly as by considering that the amount of writes to disk increases with the number of VMs sharing the same physical server - due to write throttling in the application). Over-allocation increases the write latency (seen indirectly as by considering that the amount of writes to disk increases with over-allocation - due to write throttling in the application). There was no clear connection between the application level response time with neither the number of VMs nor with the degree of over-allocation.

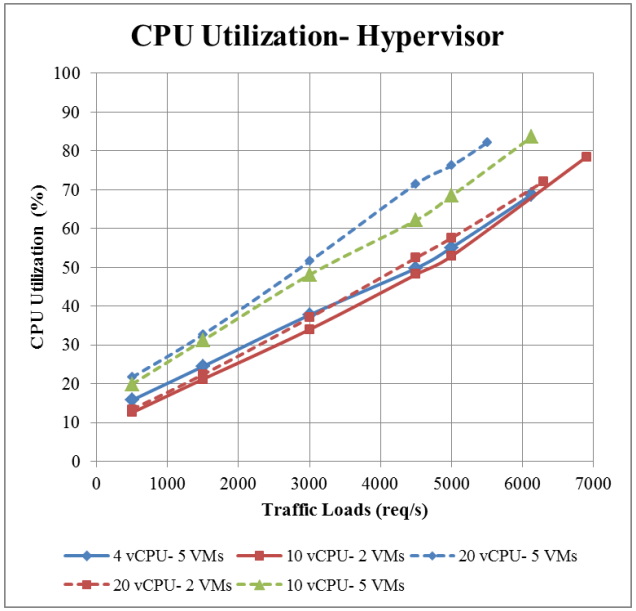


Figure 6.1. CPU utilization inside the hypervisor

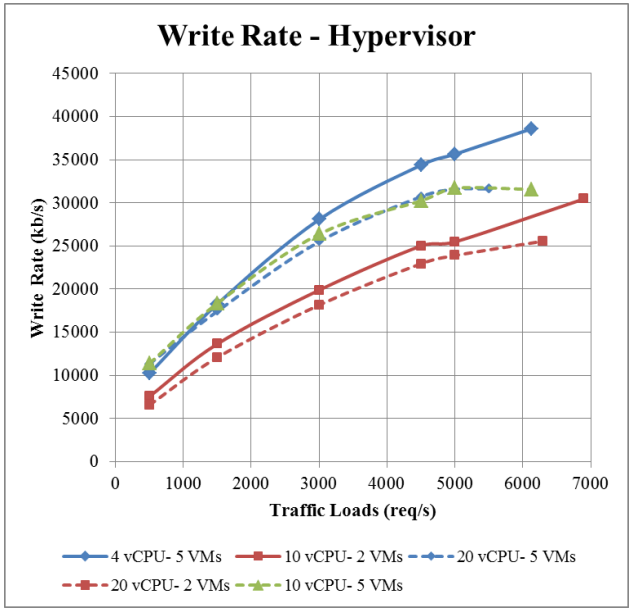


Figure 6.2. Write-Rate inside the hypervisor

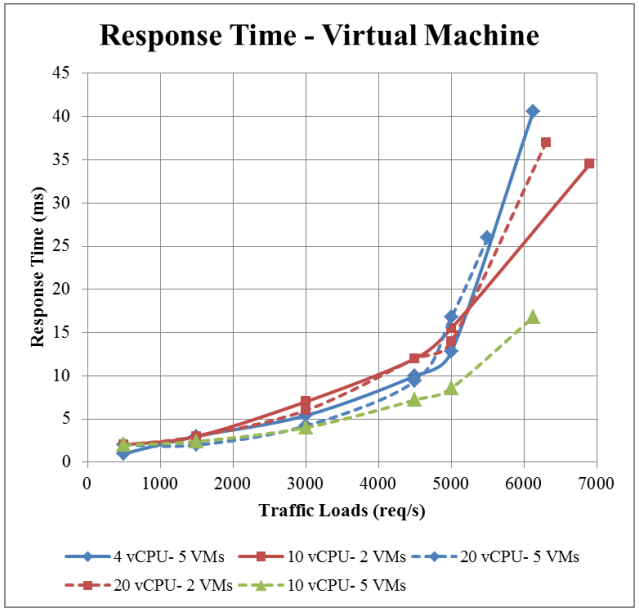


Figure 6.3. Average response time of the application

6.6 References

[1] P. Pooja, A. Pandey, “Impact of Memory Intensive Applications on Performance of Cloud Virtual Machine,” Published in: Engineering and Computational Sciences conference, 2014, pp 1-6.

[2] A. Kandalintsev, R. Lo Cigno, D. Kliazovich, P. Bouvry, “Profiling Cloud Applications with hardware Performance Counters,” Published in: Information Networking Conference, 2014, pp 52-57.

[3] M. Guzek, D. Kliazovich, P. Bouvry, “A Holistic Model for Resource Representation in Virtualized Cloud Computing Data Centers,” Published in: Cloud Computing Technology and Science, 2013, pp 590-598.

[4] J. Espades et al. ”A tenant-based resource allocation model for scaling Software-as-a-Service applications over cloud computing infrastructures,” Published in journal of Future Generation Computer Systems, 2013, pp 273-286.

[5] R. Yu et al. “Resource Allocation in Virtualized Systems Based on Try-Before-Buy Approach,” Published in 2nd International conference on Challenges in Environmental Science and Computer Engineering, 2011, pp 193-199.

[6] F. Diaz, E. Doumith, M. Gagnaire, “Impact of Resource Over-Reservation (ROR) and Dropping Policies on Cloud Resource Allocation,” Published in Cloud Computing Technology and Science conference, 2011, pp 470-476.

[7] M. Mithani and S. Rao, “Improving Resource Allocation in Multi-Tier Cloud Systems,” Published in Systems conference, 2012, pp 1-6.

[8] W. Dawoud, I. Takouna, C. Meinel, “Elastic VM for Rapid and Optimum Virtualized Resources Allocation,” Published in Systems and Virtualization Management academic workshop, 2011, pp 1-4.

[9] T. Wo, Q.Sun, B. Li, C. Hu, “Overbooking-based Resource Allocation in Virtualized Data Center,” Published in Object/Component/Service-Oriented Real-Time Distributed Computing workshops, 2012, pp 142-149.

[10] S. Shirinbab, L. Lundberg, D. Ilie, ”Performance Comparison of KVM, VMware and XenServer using a Large Telecommunication Application,” Published in the fifth International Conference on Cloud Computing, GRIDs, and Virtualization, 2014, pp 114-122.

[11] K. Lazri, S. Laniepce, J. Ben-Othman, ”When Dynamic VM Migration Falls under the Control of VM Users,” Published in 5th International Conference on Cloud Computing Technology and Science, 2013, pp 395-402.

[12] X. Zhu, Z. Wang, S. Singhal, “Utility-Driven Workload Management using Nested Control Design,” Proceedings of the American Control Conference, 2006, pp 6033-6038.

[13] X. Liu, X. Zhu, S. Singhal, M. Arlitt, “Adaptive entitlement control of resource partitions on shared servers,” 9th International Symposium on Integrated Network Management, 2005, pp 163-176.

[14] P. Padala et al. “Adaptive Control of Virtualized Resources in Utility Computing Environments,” Proceedings of the ACM Conference on Computer Systems, 2007, pp 289-302.

[15] B. Watson et al. “Probabilistic Performance Modeling of Virtualized Resource Allocation,” Published in 7th International conference on Autonomic Computing, 2010, pp 99-108.

[16] S. Garg, S. Gopalaiyengar, R. Buyya, ”SLA-Based Resource Provisioning for Heterogeneous Workloads in a Virtualized Cloud Datacenter,” Published in proceedings of the 11th international conference on Algorithms and Architectures for Parallel Processing, 2011, pp 371-384.

[17] Q. Wang, S. Malkowski, Y. Kanemasa, D. Jayasinghe,”The Impact of Soft Resource Allocation on n-Tier Application Scalablity,” Published in: Parallel and Distributed Processing Symposium (IPDPS), 2011, pp 1034-1045.

[18] J. Li, Q. Wang, D. Jayasinghe, S. Malkowski,”Profit-Based Experimental Analysis of IaaS Cloud Performance: Impact of Software Resource Allocation,” Published in: Services Computing (SCC), 2012, pp 344-351.

[19] J. Heo, X. Zhu, P. Padala, Z. Wang, ”Memory Overbooking and Dynamic Control of Xen Virtual Machines in Consolidated Environments,” Published in: Integrated Network Management Conference, 2009, pp 630-637.

[20] L. Dubois and H. Larcheveque, “Optimizing Resource Allocation while Handling SLA Violations in Cloud Computing Platforms,” Published in Parallel and Distributed Processing conference, 2013, pp 79-87.

[21] VMware, ESXi 5.5, vCenter Server 5.5, “vSphere Resource Management,” VMware’s Technical Document, available at: http://pubs.vmware.com/vsphere-55/topic/com.vmware.ICbase/PDF/vsphere-esxi-vcenter-server-55-resource-management-guide.pdf.

[22] S. Lowe, “Best Practices for Oversubscription of CPU, Mamory and Storage in vSphere Virtual Environments,” VMware’s White paper, available at: https://communities.vmware.com/servlet/JiveServlet/previewBody/21181-102-1-28328/vsphere-oversubscription-best-practices[1].pdf .

[23] S. Patel, R. Bhujade, A. Sinhal, S. Kathortia, ”Resource Optimization and Cost Reduction by Dynamic Virtual Machine Provisioning in Cloud,” Published in Advances in Computing, Communication and Informatics, 2013, pp 857-861.

[24] S. Shirinbab, L. Lundberg, D. Erman, ”Performance evaluation of distributed storage systems for cloud computing”, Published in International Journal of Computers and their applications, 2013, pp 195-207.

7 Comparing Automatic Load Balancing using VMware DRS with a Human Expert

Abstract

In recent years, there has been a rapid growth of interest in dynamic management of resources in virtualized systems. Virtualization provides great flexibility in terms of resource sharing but at the same time it also brings new challenges for load balancing using automatic migrations of virtual machines. In this paper, we have evaluated VMware's Distributed Resource Scheduler (DRS) in a number of realistic scenarios using multiple instances of a large industrial telecommunication application. We have measured the performance on the hosts before and after the migration in terms of CPU utilization, and compared DRS migrations with human expert migrations. According to our results, DRS with the most aggressive threshold gave us the best results. It could balance the load in 40% of cases while in other cases it could not balance the load properly. DRS did completely unnecessary migrations back and forth in some cases.

7.1 Introduction

There are number of benefits with virtualization. One of the advantages of virtualization is dynamic migration of virtual machines (VMs) on a cluster of physical machines. VM migrations [1][2] can be used for balancing the utilization of server host resources in order to avoid having heavily loaded hosts while lightly loaded are available [3]. Load balancing helps to maximize resource by optimizing the mapping of VMs to hosts [4].

Virtualization technologies such as VMware [6][7] try to address load balancing issues. VMware's Distributed Resource Scheduler (DRS) is a tool provided by VMware to automatically balance resource utilization across hardware resources.

7.1.1 Distributed Resource Scheduler (DRS)

VMware's DRS [6][7] is a tool that monitors utilization of system resources and migrates VMs to balance the load using VMware VMotion commands. In order for DRS to balance the entire system, all hosts should be added to a DRS cluster. VMware Virtual Center (vCenter) continuously monitors CPU and memory usage for all hosts and VMs in the cluster. DRS migrates VMs within the cluster ensuring an even distribution of load among the hosts. When the utilization of a physical machine is beyond a fixed threshold, the machine is deemed

overloaded, and DRS will automatically select a VM on that physical machine to be moved to a lightly loaded physical machine [8].

DRS takes resource management decisions according to metrics related to VMs, hosts and cluster both for memory and CPU (network and storage not taken into consideration). According to different levels of aggressiveness for DRS the threshold for migration will change. VMware defines five DRS aggressiveness levels, namely: conservative (level 1), moderately conservative (level 2), moderate (default) (level 3), moderately aggressive (level 4), and aggressive (level 5). In this paper, we have built a testbed with number of VMs using VMware virtualization technology. In order to trigger VM migrations we did one or more load shifts were we used load generators to change the work that each WM is doing. We have compared migrations triggered by DRS algorithm with human expert migration decisions. In this study we have used a large industrial telecommunication application and measured the performance of the application in different conditions; before load shift when DRS is turned off, after migration when DRS is still turned off, after DRS is tuned on and after human expert migration.

This paper is organized as follows: in Section 7.2 we present some related works. In Section 7.3, we present our experimental setup and different test scenarios configurations. Section 7.4 is the core part of the paper, in which we present our results and make a comparison. Section 7.5 concludes the paper.

7.2 Related Work

Arzuaga and Kaeli [5] propose an algorithm for load-balancing and compared their algorithm with VMware's DRS. Their results indicate that they can outperform VMware's DRS and improve performance up to 5%.

In [7], the authors tested the effectiveness of VMware's DRS algorithm. The difference between their test cases and our test cases is that here we have used a more complex application and run our tests using three hosts with more complex load scenarios; while in their work they have only tested the VMware's DRS algorithm during one test case (when they add a new host to the VMware's DRS cluster).

Lazri et al. [8] did experiments on how an attacker can influence the resource management system to make it trigger VM migrations using VMware's DRS. Parts of their study related to VMware's DRS analysis are similar to our study. However, they have only considered live migrations from a security perspective while in our work we have looked at performance and focused on comparison between human expert decisions and live migrations triggered by DRS.

Lu et al. [9] proposed a performance management tool and compared it with VMware's DRS. The authors only considered resource settings at the individual VM level or at the resource pool level. Our work differs from their work; we have

compered how smart the VMware's DRS algorithm is compared to human expert migration decisions.

In [10], the authors proposed an approach which automatically finds thresholds close to the optimal, i.e., threshold which yields an optimal number of VM migrations. However in their experiment they have done their experiment using UML (User Mode Linux) VMs while we have used VMware.

In [11], the authors proposed an algorithm for automated VM migration in order to balance the load. Kochut and Beaty [11], also presented an analytical model of virtual machine migration. The authors did their experiments using VMware and XenServer hypervisors. However, none of them considered VMware's DRS.

In [13], the authors explain that a physical host needs to have a higher remaining capacity in order to accommodate incoming VMs; therefore they have designed their migration metric based on maximizing the variance of remaining capacities of the physical servers and also consider the cost of migration. Their work is very similar to [5], one difference is that in [13], they have considered cost of migration.

In addition to these studies, there are several DRS products and projects that have been developed by research institutions, like: Entropy [14], Sandpiper [15], Smart-DRS [16] and so on.

7.3 Experimental Setup

7.3.1 Testbed Setup

The experimental setup is shown in Figure 7.4. It consists of three physical hosts running VMware ESXi 5.5.0. On top of VMware ESXi 5.5.0, RedHat Enterprise Linux, Version 6.2 has been installed as a guest OS. Each host is equipped with 128 GB of RAM, two 6-core CPUs (2x Intel XEON 2.0 GHz) with hyperthreading enabled in each core (i.e., a total of 24 logical cores). These hosts create a DRS cluster. All hosts in our DRS cluster are connected to the Compuverde [17] distributed storage system which provides 2.55 terabyte (TB) vNAS storage. The Compuverde distributed storage system consists of two components, 1) Compuverde software, that is installed inside a virtual machine and 2) data storage which consists of SSD cache (7x Intel 330 60 GB (RAID 10+hotspare)) and disk persistent storage (8x Intel 330 60 GB (RAID 5(7+1))).

A large industrial real-time telecommunication application is installed in all the VMs. The application handles billing related requests in a telecommunication system. The application works as an active-passive cluster which means that each active server is clustered together with one passive

server (in Figure 7.4, VM 1-1 and VM 1-2 are one application cluster, VM 1-1 is the active server and VM 1-2 is the passive server, and same pattern used for the rest of the VMs). Both the active and the passive server can receive requests. However, all traffic received by the passive server is forwarded to the corresponding active server. The active server then sends the response back to the passive server. Finally, the passive server sends the response to the requesting system. Traffic going directly to the active server is handled without involving the passive server. A separate host runs a simulator that generates load towards the servers running in the clusters. The simulator is located in the same LAN, but is not shown in Figure 7.4. All communications are done using Ethernet (Intel X520 SFP+ 10 GB). All hosts in the DRS cluster are monitored using VMware Virtual Center 5.5.0.

We have created 12 VMs. Inside these VMs we have installed our application, i.e., in total we have 6 active-passive application clusters. We have allocated 10 vCPU and 14 GB RAM (14 GB RAM is the minimum that is recommended for the application) to each of these VMs. Inside each host 24 vCPUs are available.

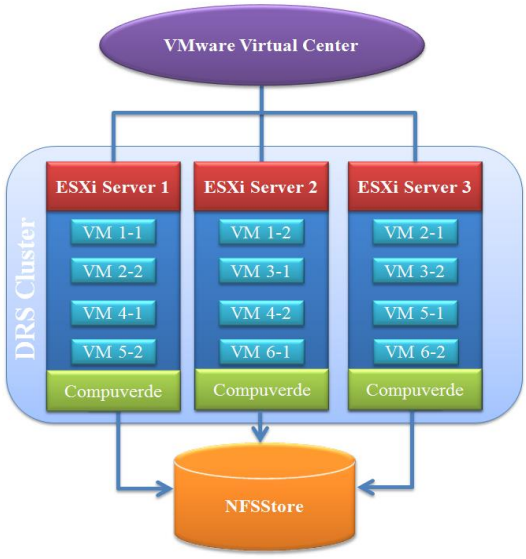


Figure 7.4: Experimental Setup

In order to avoid any interference between the VM that contains the Compuverde software and other VMs, we have created two different resource pools, one with 4 vCPUs and the other one with 20 vCPUs. The VM containing the Compuverde software has been bound to the resource pool with 4 vCPUs and the rest of the VMs are bound to the resource pool that contains 20 vCPUs.

7.3.2 Test Scenarios

We have designed our test scenarios in such a way that there is always a good solution to balance the load. Here are our test scenarios:

1. There are three hosts and each host contains 4 VMs. All VMs receive the same number of requests per second (700, 500, and 300). Suddenly we increase the load on one VM (up to 2100, 1500, and 900 req/s).

2. There are three hosts and each host contains 4 VMs. All VMs receive the same number of requests per second (600, 500, and 300). Suddenly we increase the load on two VMs on two different hosts (up to 2500, 2000, and 900 req/s).

3. There are two hosts and each host contains 6 VMs. All VMs receive the same number of requests per second (600, 500 and 300). Suddenly we add a new empty host to the DRS cluster.

4. There are three hosts and each host contains 4 VMs. All VMs receive the same number of requests per second (700, 500 and 300). Suddenly we decrease the load on one VM and increase the load on another VM, on two different hosts. This test scenario simulates the situation when one VM stops working. As we have mentioned earlier the application works as an active-passive cluster, so if the active server stops working the passive server will become active and handle all requests.

5. There are three hosts and each host contains 4 VMs. All VMs receive the same number of requests per second (700, 500 and 300). Suddenly we place one of the hosts in maintenance mode.

For all test scenarios, we have used the DRS automatic placement and repeated each test for three different levels of aggressiveness, level 1 (conservative), level 3 (default/moderate), and level 5 (most aggressive).

We have measured the CPU utilization for each test scenario. The measurements were done during four different states:

1. Sending the same amount of load to all virtual machines while DRS is turned off.

2. Shifting the load on one or two VMs (depending on the scenario) while DRS is still turned off.

3. After having turned on DRS (using three different levels of aggressiveness).

4. After a human expert has done the migrations manually.

For test scenarios 3 and 5, there was no second state (load shift); we have only measured the performance at states 1, 3, and 4.

7.4 Comparing VMware's DRS Migrations with Human Expert Migrations

7.4.1 Test Case 1

In this case, there are four VMs inside all three hosts. All VMs receive the same number of requests per second, then suddenly we increase the load on one of the VMs (VM 1-1) on Host 1 (ESXi Server 1) and send two times higher load to this VM. We have done this test for three different load scenarios and three different DRS migration threshold levels.

As seen in Table 7.1, after the load shift, DRS only performed migration in two of the cases (500 and 300 req/s) with migration threshold set to level 5 (aggressive). In both cases only one VM was migrated, and it migrated from Host 1 to Host 3. Table 7.2 shows that for the lowest load (300 req/s), the CPU utilization is around 27% before load shift on all three hosts. After the increase of the load on VM 1-1 to 600 req/s, the CPU utilization on the Host 1 (ESXi Server 1) increases to 34%. Since the passive server (VM 1-2) is running on Host 2 (ESXi Server 2), Table 7.2, shows that the CPU utilization on Host 2 has increased to 28%. For level 5, and 500 and 300 req/s, a human expert would do the same as DRS (migrate one VM from Host 1 to Host 3) (see Table 7.1). So, in this case DRS did a good job.

For the case of medium load (500 req/s), Table 7.2 shows that when the load was the same on all VMs, the CPU utilization was around 38% on all three hosts. After we increased the load on VM 1-1 to 1000 req/s, the CPU utilization on Host 1 and Host 2 increased to 49% and 40%, respectively. Then we have enabled DRS with its threshold set to level 1, and then we have changed migration threshold to level 3 (moderate/default) and level 5 (aggressive). DRS only started the migration when the threshold was set to the level 5 (aggressive) (see Table 7.1). In this case, same as previous case (low load 300 req/s), DRS migrated one VM from Host 1 to Host 3 (see Table 7.1). If we compare the CPU utilization on three hosts, we can observe that after DRS migration CPU utilization on Host 1 has been decreased to 44% and CPU utilization on Host 3 has been increased up to 44%, while CPU utilization on Host 2 remained unchanged. As we have discussed earlier this is similar to what human expert would do in this situation. Therefore, the CPU utilization on all hosts after human expert migration is identical with the CPU utilization after DRS migration (see Table 7.2) in this case.

However, when the load was high (700 req/s), DRS did not migrate any of the VMs (see Table 7.1) which was unexpected, because as it can be observed from Table 7.2, CPU utilization on the Host 1 was very high, 67%, while on Host 3 the CPU utilization was quiet low 49%, so obviously one VM should have been migrated from Host 1 to Host 3. In this case, a human expert would migrate one VM from Host 1 to Host 3 (see Table 7.1), to balance the CPU utilization across all hosts; so CPU utilization became around 57% on Host 3, 58% on Host 1 and 52% on Host 2 (see Table 7.2). The reason why DRS did not migrate any VM could be that the DRS system did not find any "good" host for receiving a VM, i.e., it did not find any host with a low enough load. However, in this case DRS did not do a very good job.

7.4.2 Test Case 2

In Test case 2, we do a more complex load shift. In this case, we have increased the load on two VMs running on two different hosts at the same time, while all other VMs are receiving the same amount of load. In this case, all hosts could not support our previous high load which was 700 req/s, therefore we needed to decrease the high load to 600 req/s, but the other load scenarios are remained unchanged. As seen in Table 7.3, VMware's DRS performed the migration only for migration threshold level 5 (aggressive). For the case with high load (600 req/s), after we have increased the load on two VMs (VM 1-1 and VM 2-1), up to 1800 req/s, and turned on the DRS, we have observed that DRS started to migrate one VM from Host 1 to Host 2 and after some time it started to migrate another VM from Host 2 back to Host 1 (see Table 7.3), we call this the "ping-pong" effect. DRS repeated this behavior for a while and at the end we concluded that it had difficulties to balance the load. Here we should mention that nothing happened when we have selected DRS migration threshold level 1 (conservative) and level 3 (moderate) (see Table 7.3). In this case a human expert would migrate one VM from Host 1 to Host 2 (e.g., VM 3-2) and one VM from Host 3 to Host 2 (e.g., VM 6-2) (see Table 7.3). It can also be observed from Table 7.4, that the CPU utilization after DRS migration did not change, while after human expert migration the CPU utilization became around 62% on all hosts. For the case with medium load (500 req/s), DRS had the same behavior, it migrated VMs back and forth, i.e., the "ping-pong" effect (see Table 7.3). As can be seen in Table 7.4, the CPU utilization remained unchanged after DRS migration while after human expert migration the CPU utilization became around 53%. In these cases, DRS did not do a very good job.

For the case with low load (300 req/s), we have increased the load on both VM 1-1 and VM 2-1) to 900 req/s. DRS performed a migration only when the migration threshold was set to level 5 (aggressive); in this case it migrated one VM from Host 1 to Host 2 but no migration from/to Host 3 (see Table 7.3). In this case we can say that DRS did a reasonably good job but it only could balance part of the load. As can be seen in Table 7.4, the CPU utilization after the DRS migration became 34% on both Host 1 and Host 2 while it was still high (38%) on Host 3. In order to

balance the load completely DRS should have migrated a VM from Host 3 to Host 2 in addition to the previous migration, which was what a human expert would do in this situation (see Table 7.3). As can be observed from Table 7.4, after the human expert migrations, the CPU utilization on all hosts became around 34%.

7.4.3 Test Case 3

In Test case 3, we had two hosts and each host contained six VMs; we sent the same load to all VMs, so there was no load shift in this case, instead we have added an empty server. (Similar to the previous case, we could not reach 700 req/s, so instead we have used 600 req/s as the high load.)

In this case for the highest load (600 req/s), when the migration threshold for DRS was level 3 (moderate/default) it started to migrate two VMs from Host 1 to Host 3 and two VMs from Host 2 to Host 3, and at the end it migrated back one of the VMs from Host 3 to Host 2, i.e., the "ping-pong" effect (see Table 7.5). As can be seen in Table 7.5, the CPU utilization of Host 1, Host 2, and Host 3 after DRS migration became 36%, 53%, and 48% when the threshold for migration was set to moderate (level 3). The human expert migrations were two VMs from Host 1 to Host 3 and two VMs are migrated from Host 2 to Host 3 (see Table 7.5). At the end we can observe that the CPU utilization has been evenly distributed to all hosts after human expert migration (see Table 7.6). For the same load (600 req/s), when the DRS migration threshold has been set to level 5 (aggressive), DRS migrated two VMs from Host 1 to Host 3 and two VMs from Host 2 to Host 3, but after sometime migrated back one of the VMs from Host 3 to Host 2, i.e., the "ping-pong" effect (see Table 7.5).

In the case with medium load (500 req/s), when the DRS migration threshold was set to level 3 (moderate/default), DRS migrated two VMs from Host 1 to Host 3 and one VM from Host 2 to Host 3 (see Table 7.5). Table 7.6 shows that after DRS migration the CPU utilization on Host 1 decreased to 31% while on Host 2 it became 47% and on Host 3 it became 42%. If we compare this with what a human expert would do, we can observe that after human expert migration the CPU utilization will become around 40% on all hosts. For the same load (500 req/s) and the DRS migration threshold, level 5 (aggressive), DRS migrated two VMs from Host 1 to Host 3 and one VM from Host 2 to Host 3. After some time DRS migrated one more VM from Host 2 to Host 3 (see Table 7.5).

So at the end we can observe from the Table 7.6, that CPU utilization on the Host 1 became 37% while on the Host 2 and the Host 3, CPU utilization became 41%. Although DRS migration was different from what human expert would migrate, DRS could almost balance the CPU utilization on all hosts (see Table 7.6). If we compare DRS migration with human expert migration, we can say that after human expert migration the CPU utilization on all three hosts became around 40% which was slightly better than DRS.

For the case of low load (300 req/s), when the migration threshold was set to level 3 (moderate), DRS only migrated two VMs to Host 3, one VM from Host 1 and one VM from Host 2 (see Table 7.5). As it can be observed from Table 7.6, the CPU utilization after DRS migration became 33% on Host 1, 34% on Host 2 and 20% on Host 3. So DRS migration did not balance the CPU utilization properly. As seen in Table 7.6, after human expert migration the CPU utilization on all hosts became around 28%. For the same load (300 req/s) and the DRS migration threshold, level 5 (aggressive), we have observed that DRS migrated two VMs from Host 1 to Host 3 and two VMs from Host 2 to Host 3 (see Table 7.5). Although DRS migrated two VMs from each host to Host 3 which is the same decision that a human expert would make the result was slightly different. The reason was that DRS migrated one active server and one passive server from Host 1 to Host 3 and at the same time two active servers from Host 2 to Host 3, while a human expert would migrate one active and one passive server from each hosts to Host 3 to balance the load (see Table 7.5). Therefore, at the end we can observe from Table 7.6 that, after human expert migration the CPU utilization became around 28% on all hosts while after the DRS migration, CPU utilization on Host 3 is different.

7.4.4 Test Case 4

In Test case 4, we have simulated the scenario where one VM (the active server in an application cluster) stops working and the passive server becomes active and handles all requests. So, in our test environment we have turned off VM 1-1, in this way all requests will be forwarded to VM 1-2.

For the case with high load, 700 req/s, and when we set the threshold level to the most aggressive (level 5), DRS migrated one VM from Host 2 to Host 1 (see Table 7.7). Table 7.8 shows that after we have turned off VM 1-1 on Host 1, the CPU utilization became 40% on Host 1 and 63% on Host 2. After the DRS migration, the CPU utilization became 48% on Host 1 and 55% on Host 2. A human expert would migrate one active server from Host 2 to Host 1 and one passive server from Host 1 to Host 2 (see Table 7.7). If we compare the results of human expert migrations and DRS, we see that after human expert migrations the CPU utilization on all hosts became around 52% while after DRS migrations, the CPU utilization on Host 2 was still higher than the two other hosts (see Table 7.8). Although DRS was not able to balance the load completely, we see that DRS could balance the load to some extent. For the case with medium load (500 req/s), when the DRS migration threshold was set to level 5 (aggressive), DRS migrated one active server from Host 2 to Host 1 and one passive server from Host 1 to Host 2, which was the same as what human expert would do in this situation (see Table 7.7). Table 7.8 shows that the CPU utilization on all hosts after both the human expert and the DRS migration became around 40%. For the case with low load, 300 req/s, DRS did not migrate any VM for all three different migration threshold levels (see Table 7.4). In this case, as it can be observed from the Table 7.8, that CPU utilization on all hosts after the load shift is remained unchanged. While after

human expert migration we can observe from the Table 7.8 that the CPU utilization became around 29% on all hosts. Here we should mention that DRS was not able to make proper decisions comparing to human expert decisions in most of the cases with different migration threshold levels. Only in one of the cases it worked when the load was medium (500 req/s) and the migration threshold level was 5 (aggressive).

7.4.5 Test Case 5

In Test case 5, we wanted to put Host 3 in maintenance mode. In this scenario all VMs on Host 3 should be migrated to other hosts.

In the case with high load (700 req/s), after we have turned on the DRS and set the migration threshold to level 1 (conservative), it started to migrate two VMs from Host 3 to Host 1 and two VMs from Host 3 to Host 2. DRS migrated the same VMs to the same hosts for other levels of aggressiveness (level 3 and level 5) (see Table 7.9). For the case with medium load (500 req/s), DRS migrated again the same VMs to the same hosts for all three different levels of aggressiveness (see Table 7.9). In both of these cases we can say that DRS did a good job and made the same decision as a human expert. As seen in Table 7.10, after DRS migration the CPU utilization on both Host 1 and Host 2 became the same.

However, when the load was low (300 req/s), after we have turned on the DRS and set the migration threshold to level 1 (conservative) and level 3 (moderate), DRS migrated three VMs from Host 3 to Host 2 and one VM from Host 3 to Host 1 (see Table 7.9). Table 7.10 shows that after DRS migration the CPU utilization on Host 2 is 47% and the CPU utilization on Host 1 is 38%. This means that DRS could not distribute the load between the hosts equally and DRS migration was not successful in this case. However, when the migration threshold was set to level 5 (aggressive) for the same load (300 req/s), we see that DRS migrated two VMs from Host 3 to Host 1 and two VMs from Host 3 to Host 1, similar to what a human expert would do (see Table 7.9). From Table 7.10, it can be seen that the CPU utilization both after DRS migration and after human expert migration became around 40%. Although DRS decisions did not work in two of the cases, we see that in the other cases, DRS was able to make good decisions.

7.5 Conclusions

In this study, our goal was to compare VMware's DRS migrations versus human expert migrations using various test scenarios. The test scenarios were designed so that there are always optimal solutions for balancing the load all over the hosts; we wanted to see how close DRS migrations are compared to human expert migrations.

We considered five test cases, and three different loads for each of these test cases. For each load and test case we tested three different levels of aggressiveness in DRS. This means that we looked at 5*3*3 = 45 cases. In 23 of these cases DRS did nothing. In 11 cases it did (more or less) the same decision as a human expert. In 7 cases DRS could balance the load to some extent, but in 4 cases DRS suffered from the "ping-pong" effect and did completely unnecessary migrations back and forth.

When the migration threshold was set to level 1 (conservative) or level 3 (moderate/default) DRS did not migrate any virtual machine in most of the cases (Test cases 1, 2, and 4). However, in Test case 5, DRS starts to migrate virtual machines even when the migration threshold was set to level 1 (conservative). One reason could be that VMware have considered similar test scenarios when they have designed the DRS algorithm (evacuating a physical machine for maintenance is a very common scenario); however for some more complex test scenarios VMware's DRS was unreliable, according to our results (e.g., Test case 2). Overall if we compare the system performance after DRS migration we can observe that we have obtained better results - more close to human expert migrations - with the aggressive threshold (level 5). In the 15 cases were we uses level 5, we got no migrations in two cases, good (human expert quality) migrations in 7 cases, reasonably good migrations in three cases, and the undesirable "ping-pong" effect in three cases. So even if the migrations are better in general using level 5, the risk of suffering from the "ping-pong" effect is also considerably higher compared to the other levels of aggressiveness.

Our study shows that there is still considerable room for improvement of VMware's state-of-the-art DRS load balancing systems. In particular the load balancing needs to be more robust in the sense that completely unnecessary migrations such as the "ping-pong" effect should be avoided.

Table 7.1. Summary of the Results of the Test Case 1

Load (req/s)		**Level of Aggresivness**			**Human Expert**
All VMs	***One VM***	***Level 1***	***Level 3***	***Level 5***	
700	1400	-	-	-	VM 2-2 from Host 1 to Host 3
500	1000	-	-	VM 2-2 from Host 1 to Host 3	
300	600	-	-	VM 5-2 from Host 1 to Host 3	

Table 7.2. Test Case 1 Results, CPU Utilization

	CPU Utilization (%)											
	a. Low Load (300req/s)				***b. Medium Load (500req/s)***				***c. High Load (700req/s)***			
	Same Load	Load Shift	DRS Migration	Human Expert	Same Load	Load Shift	DRS Migration	Human Expert	Same Load	Load Shift	DRS Migration	Human Expert
Host 1	27	34	31	31	38	49	44	44	49	67	67	58
Host 2	27	28	28	28	38	40	40	40	49	51	51	52
Host 3	26	26	30	30	37	38	44	44	48	49	49	57

Table 7.3. Summary of the Results of the Test Case 2

Load (req/s)		**Level of Aggresivness**			**Human Expert**
All VMs	***Two VMs***	***Level 1***	***Level 3***	***Level 5***	
600	1800	-	-	VM 3-2 from Host 1 to Host 2 VM 5-2 from Host 2 to Host 1 (ping-pong)	VM 3-2 from Host 1 to Host 2 VM 6-2 from Host 3 to Host 2
500	1500	-	-	VM 3-2 from Host 1 to Host 2 VM 5-2 from Host 2 to Host 1 (ping-pong)	
300	900	-	-	VM 2-2 from Host 1 to Host 2	

Table 7.4. Test Case 2 Results, CPU Utilization

	CPU Utilization (%)											
	a. Low Load (300req/s)				***b. Medium Load (500req/s)***				***c. High Load (600req/s)***			
	Same Load	Load Shift	DRS Migration	Human Expert	Same Load	Load Shift	DRS Migration	Human Expert	Same Load	Load Shift	DRS Migration	Human Expert
Host 1	28	38	34	34	40	60	60	54	46	72	72	63
Host 2	28	29	34	36	40	41	41	53	47	47	47	62
Host 3	27	38	38	34	39	59	59	51	46	72	72	62

Table 7.5. Summary of the Results of the Test Case 3

Load (req/s)	Level of Aggresivness			Human Expert
All VMs	Level 1	Level 3	Level 5	
600	-	VM 2-1 from Host 2 to Host 3 VM 1-1 from Host 1 to Host 3 VM 4-1 from Host 2 to Host 3 VM 3-1 from Host 1 to Host 3 VM 4-1 from Host 3 to Host 2 (ping-pong)	VM 2-2 from Host 1 to Host 3 VM 6-1 from Host 2 to Host 3 VM 1-1 from Host 1 to Host 3 VM 2-1 from Host 2 to Host 3 VM 2-2 from Host 3 to Host 2 (ping-pong)	VM 1-2 from Host 2 to Host 3 VM 2-1 from Host 2 to Host 3 VM 5-1 from Host 1 to Host 3 VM 6-2 from Host 1 to Host 3
500	-	VM 5-1 from Host 1 to Host 3 VM 2-1 from Host 2 to Host 3 VM 1-1 from Host 1 to Host 3	VM 1-1 from Host 1 to Host 3 VM 3-1 from Host 1 to Host 3 VM 2-1 from Host 2 to Host 3 VM 3-2 from Host 2 to Host 3	
300	-	VM 3-1 from Host 1 to Host 3 VM 2-1 from Host 2 to Host 3	VM 6-1 from Host 2 to Host 3 VM 3-1 from Host 1 to Host 3 VM 2-1 from Host 2 to Host 3 VM 6-2 from Host 1 to Host 3	

Table 7.6. Test case 3 results, CPU utilization

	CPU Utilization (%)											
	a. Low Load (300req/s)				*b. Medium Load (500req/s)*				*c. High Load (600req/s)*			
	Same Load	DRS Migration (moderate)	DRS Migration (aggressive)	Human Expert	Same Load	DRS Migration (moderate)	DRS Migration (aggressive)	Human Expert	Same Load	DRS Migration (moderate)	DRS Migration (aggressive)	Human Expert
Host 1	43	33	29	28	64	31	37	41	73	36	46	46
Host 2	44	34	24	29	63	47	41	40	73	53	42	45
Host 3	3	20	34	28	4	42	41	39	4	48	49	44

Table 7.7. Summary of the Results of the Test Case 4

Load (req/s)	**Level of Aggresivness**			**Human Expert**
All VMs	**Level 1**	**Level 3**	**Level 5**	
700	-	-	VM 4-2 from Host 2 to Host 1	VM 6-1 from Host 2 to Host 1 VM 2-2 from Host 1 to Host 2
500	-	-	VM 3-1 from Host 2 to Host 1 VM 1-1 from Host 1 to Host 2	
300	-	-	-	

Table 7.8. Test Case 4 Results, CPU Utilization

	CPU Utilization (%)											
	a. Low Load (300req/s)				***b. Medium Load (500req/s)***				***c. High Load (700req/s)***			
	Same Load	Load Shift	DRS Migration	Human Expert	Same Load	Load Shift	DRS Migration	Human Expert	Same Load	Load Shift	DRS Migration	Human Expert
Host 1	28	22	22	29	40	31	40	40	53	40	48	52
Host 2	28	34	34	29	39	49	41	41	51	63	55	52
Host 3	27	27	27	28	38	38	38	38	51	50	50	51

Table 7.9. Summary of the Results of the Test Case 5

Load (req/s)	**Level of Aggresivness**			**Human Expert**
All VMs	**Level 1**	**Level 3**	**Level 5**	
700	VM 3-2 from Host 3 to Host 2 VM 6-2 from Host 3 to Host 1 VM 2-1 from Host 3 to Host 1 VM 5-1 from Host 3 to Host 2	VM 3-2 from Host 3 to Host 2 VM 6-2 from Host 3 to Host 1 VM 2-1 from Host 3 to Host 1 VM 5-1 from Host 3 to Host 2	VM 3-2 from Host 3 to Host 2 VM 6-2 from Host 3 to Host 1 VM 2-1 from Host 3 to Host 1 VM 5-1 from Host 3 to Host 2	
500	VM 3-2 from Host 3 to Host 2 VM 6-2 from Host 3 to Host 1 VM 2-1 from Host 3 to Host 1 VM 5-1 from Host 3 to Host 2	VM 3-2 from Host 3 to Host 2 VM 6-2 from Host 3 to Host 1 VM 2-1 from Host 3 to Host 1 VM 5-1 from Host 3 to Host 2	VM 3-2 from Host 1 to Host 2 VM 5-2 from Host 2 to Host 1 VM 5-2 from Host 1 to Host 2 VM 3-2 from Host 2 to Host 1	VM 3-2 from Host 3 to Host 2 VM 6-2 from Host 3 to Host 1 VM 2-1 from Host 3 to Host 1 VM 5-1 from Host 3 to Host 2
300	VM 3-2 from Host 3 to Host 2 VM 6-2 from Host 3 to Host 1 VM 2-1 from Host 3 to Host 2 VM 5-1 from Host 3 to Host 2	VM 3-2 from Host 3 to Host 2 VM 6-2 from Host 3 to Host 1 VM 2-1 from Host 3 to Host 2 VM 5-1 from Host 3 to Host 2	VM 3-2 from Host 3 to Host 2 VM 6-2 from Host 3 to Host 1 VM 2-1 from Host 3 to Host 1 VM 5-1 from Host 3 to Host 2	

Table 7.10. Test Case 5 Results, CPU Utilization

	CPU Utilization (%)									
	a. Low Load (300req/s)				***b. Medium Load (500req/s)***			***c. High Load (700req/s)***		
	Same Load	DRS Migration (level 1 and 3)	DRS Migration (aggressive)	Human Expert	Same Load	DRS Migration	Human Expert	Same Load	DRS Migration	Human Expert
Host 1	28	38	42	42	40	63	63	52	85	85
Host 2	28	47	38	38	40	63	63	51	85	85
Host 3	27	4	4	4	39	4	4	50	4	4

7.6 References

[1] C. Clark, et al., “Live migration of virtual machines,” 2nd symposium on Networked Systems Design and Implementation, 2005, pp 273-286.

[2] S. Sotiriadis, N. Bessis, P. Gepner, N. Markatos, “Analysis of Requirements for virtual machine migrations in dynamic clouds,” 12th IEEE International conference on Parallel and Distributed Computing, 2013, pp 116-123.

[3] C. A. Waldspurger, “Memory resource management in vmware esx server,” SIGOPS Oper.Sys.Rev., vol36, 2002, pp 181-194.

[4] H. Nagamani and P. Jayarekha, “Load balancing with optimal cost scheduling algorithm,” International conference on Computation of Power, Energy, Information and Communication, 2014, pp 24-31.

[5] E. Arzuaga, and D. Kaeli, “Quantifying load imbalance on virtualized enterprise servers,” proceedings of the first international conference on Performance engineering, 2010, pp 235-242.

[6] VMware Inc. Resource Management with VMware DRS. http://www.vmware.com/pdf/vmware_drs_wp.pdf

[7] VMware Inc. DRS Performance and Best Practices. https://www.vmware.com/files/pdf/drs_performance_best_practices_wp.pdf

[8] K. Lazri, S. Laniepce, J. Ben-Othman, ”When Dynamic Vm Migration Falls Under the Control of VM Users,” IEEE international conference on Cloud Computing Technology and Science, 2013, pp 395-402.

[9] Lei Lu, et al., “Application-driven dynamic vertical scaling of virtual machines in resource pools,” IEEE Network Operations and Management, 2014, pp 1-9.

[10] H. W. Choi, H. Kwak, A. Sohn, K. Chung, “Autonomous learning for efficient resource utilization of dynamic vm migration,” 22nd annual international conference on Supercomputing, 2008, pp 185-194.

[11] J. Park, J. Kim, H. Choi, Y. Woo, “Virtual machine migration in self-managing virtualized server environments,” 11th International conference on Advanced Communication Technology, 2009, pp 2077-2083.

[12] A. Kochut, K. Beaty, “On strategies for dynamic resource management in virtualized server environments,” 15th International symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2007, pp 193-200.

[13] G. Khanna, K. Beaty, G. Kar, A. Kochut, ”Application performance management in virtualized server environments,” 10th IEEE conference on Network Operations and Management, 2006, pp 373-381.

[14] F. Hermenier, et al. “Entropy: a Consolidation Manager for Clusters,” Proceedings of the ACM international conference on virtual execution environments, 2009, pp 41-50.

[15] T. Wood, P. Shenoy, A. Venkataramani, M. Yousif, “Black-box and Gray-box Strategies for Virtual Machine Migration,” Proceedings of the 4th USENIX conference on Networked systems design and implementation, 2007, pp 1-14.

[16] L. Xu, W. Chen, Z. Wang, Sh. Yang, “Smart-DRS: Astrategy of Dynamic Resource Scheduling in Cloud Data Center,” Proceedings of the IEEE international conference on Cluster Computing Workshops, 2012, pp 120-127.

[17] S. Shirinbab, L. Lundberg, D. Erman,” Performance evaluation of distributed storage systems for cloud computing,” Published in International Journal of Computers and their applications, 2013, pp 195-207.

8 Performance Implications of Resource Over-Allocation During the Live Migration

Abstract

As the number of cloud users are increasing, it becomes essential for cloud service providers to allocate the right amount of resources to virtual machines, especially during live migration. In order to increase the resource utilization and reduce waste, the providers have started to think about the role of over-allocating the resources. However, the benefits of over-allocations are not without inherent risks. In this paper, we conducted an experiment using a large telecommunication application that runs inside virtual machines, here we have varied the number of vCPU resources allocated to these virtual machines in order to find the best choice which at the same time reduces the risk of under-allocating resources after the migration and increases the performance during the live migration. During our measurements we have used VMware's vMotion to migrate virtual machines while they are running. The results of this study will help virtualized environment service providers to decide how much resources should be allocated for better performance during live migration as well as how much resource would be required for a given load.

8.1 Introduction

Live migration is the process of moving a virtual machine from one physical machine to another physical machine, while the virtual machine is executed with (almost) no interruption [9]. Live migration is an essential feature in virtual environments. Load balancing, online maintenance, fault tolerance and energy reduction are all dependent on live migration of virtual machines [8]. This feature is supported by VMware (vMotion), Xen (XenMotion), Microsoft Hyper-V and Redhat KVM [10]. There are several studies comparing live migration in VMware, KVM, Xen and Hyper-V [5] [19].

The common approach for live virtual machine migration is pre-copy [9]. During the pre-copy process in VMware vMotion, a shadow virtual machine created on the destination host, and then each used memory page (known as dirty pages) is copied from the source to the destination. , as each round takes some time and in the meantime the virtual machine is still running on the source host, some pages may be dirtied and have to be re-sent, so this iterative memory copying is continued until no changed pages remain, at this point the last stage is stopping the virtual machine on the source and resume it on the destination [11].

Live migration cost is classified into performance overhead on the physical machine and live migration execution cost [18]. In order to measure the performance overhead on the physical machine, there are a number of parameters that need to be considered, such as CPU and disk utilization, and network bandwidth. Execution costs are total migration time and migration downtime. The total migration time is the time it takes from when the migration initiated for a virtual machine on the source host until the virtual machine is resumed on the destination host, while the downtime is the time it takes for the source host to suspend execution of virtual machine until the destination host resumes it [9].

In data centers, migrations are based on pre-defined rules and without involving a human operator. VMware's Distributed Resource Scheduler (DRS) [11] is one example of automatic load balancing which can provide high and even resource utilization over a cluster of physical servers. Although live virtual machine migration is generally fast, it is highly resource-intensive; therefore it can affect the application performance and resource usage of the migrating virtual machine as well as other virtual machines which are sharing the same physical hardware [15]. One important challenge in modern data centers where virtual machines migrate from host to host all the time is efficient resource utilization.

One solution could be to over-allocate resources to the virtual machines in a way such that after live migration we still are able to use all physical resources, E.g., assume there are two hosts each with 24 CPU cores, and there are four virtual machines running on each host and each VM has 6 virtual CPUs (vCPUs). If one VM is migrated to the other host, the total number of VMs on the source host becomes three, and they will only have a total of 18 vCPUs, i.e., six of the virtual CPU cores will not be used. However, if each VM had 24 vCPUs allocated to it, there would be enough vCPUs to utilize the physical cores.

An important issue in data centers is management of virtual machines, in terms of resource allocation and de-allocation, and virtual machine migration. The system administrator in a data center has control over executing applications and resource requirements. Therefore the system administrator has a good opportunity to make sure that different virtual machines meet their performance and service level objectives while hardware resources are utilized effectively [14].

The system administrator can benefit from over-allocation of resources by allocating more resources to the virtual machines than they required, so in this way the virtual machine will still perform well even after migration.

As we have discussed there are some advantages with over-allocation of resources especially during live migration. However, there is a risk of achieving poor performance which we have discussed in our previous study [12]. In [12], we have quantified the cost of having different amounts of over-allocation. According to our results, using a moderate level of over-allocation gives some flexibility at a very modest performance cost. In our previous study we did not consider the effect of over-allocating resources on the live migration overhead.

In this paper, we have built a testbed containing ten virtual machines hosted on two physical servers and varied the number of virtual CPUs allocated to the virtual machines. We have used a large industrial telecommunication application to run inside these virtual machines and we have measured the performance in terms of CPU utilization, downtime and total migration time of the application during the live migration.

This paper is organized as follows: in Section 8.2 we present some related works. In Section 8.3, we present our experimental setup and different test cases configurations. Section 8.4 is the core part of the paper, in which we present our results and discuss the impact of over-allocation during live migration. And Section 8.5 concludes the paper.

8.2 Related Work

In [1], the authors developed a new technique for resource allocation in virtualized data centers. However they did not consider the over-allocation of resources.

M. Elsaid and C. Meinel [7] studied the impact of live migration on the datacenter. The authors have considered network resources and power consumption. In our work we have focused on CPU utilization, migration down time and total migration time.

Both Xiao et al. [16] and Vishnupriya et al. [17] have used virtual machine live migration for dynamic resource allocation in a virtualized environment. They did not consider the over-allocation.

Elsaid and Meinel [18] have done some experiments using different migration cost models for VMware vMotion. Their focus was to predict the live migration CPU and network overhead.

However, little research has been done in considering the over-allocation of resources during the live migration. Therefore, in this study we will investigate the effect of resource over-allocation on the down time and total migration time of a large industrial real-time application.

8.3 Experimental Setup

In this section, we present our experimental setup and test cases.

8.3.1 Testbed setup

Two servers have been used as hosts to test the performance of VMware ESXi 5.5.0. On top of VMware, RedHat Enterprise Linux, version 6.2 has been installed as the guest OS. Each server is equipped with 128 GB of RAM, two 6-core CPUs (2x Intel XEON 2.0 GHz) with hyperthreading enabled in each core (i.e., a total of 24 logical cores [2] [3] [4]).

In order to reduce the time for live migration one would like that all servers in a cluster share the same disk; in this case no files need to be moved during a live migration of a VM. Having one large physical disk (or disk array) for a large cluster can, however, become a single point of failure and a performance bottleneck. The latest trend in virtualization and cloud computing is that one uses so called distribute storage systems. In a distributed storage system the disks are physically distributed to the different servers in the cluster, thus avoiding the single point of failure and performance bottleneck problems. Based on the physically distributed disks, a software layer creates an image of one shared virtual disk, thus avoiding the need to migrate files during a live migration. One of the high performing distributed storage systems is the Compuverde system [5] [6].

All servers in our cluster are connected to the Compuverde distributed storage system which provides vNAS storage. The Compuverde distributed storage system consists of two components, 1) Compuverde software, that is installed inside a virtual machine and 2) data storage which consists of SSD cache (7x Intel 330 60 GB (RAID 10+hotspare)) and disk persistent storage (8x Intel 330 60 GB (RAID 5(7+1))). All virtual machines containing Compuverde software are communicating with each other in order to maintain and share information about data. The Compuverde software is responsible for managing distribution of data between all storage nodes in different hosts and it has been designed so that it treats all the SSD caches and disks as a virtual Network Attached Storage (vNAS).

The same large industrial real-time telecommunication application is installed in all the VMs. The application, referred to as telecom server in the reminder of this paper, handles billing related requests in a telecommunication system.

The application consists of several hundreds of thousands lines of code and it works as an active-passive cluster. Each active telecom server is clustered together with one passive telecom server. Both the active and the passive telecom servers in a cluster can receive requests. However, all traffic received by the passive telecom server is forwarded to the corresponding active telecom server. The active telecom server then sends the response back to the passive telecom server. Finally, the passive telecom server sends the response to the requesting system. Traffic going directly to the active telecom server is handled without involving the passive telecom server.

Another separate server runs a load generator that impersonates a requesting system in order to generate load towards the telecom servers running in the clusters. The load generator is also located in the same LAN. All communications are done through Ethernets (Intel X520 SFP+ 10 GB). The two hosts are configured in a cluster that is managed by VMware vCenter.

8.3.2 Test configurations

For different live migration and over-allocation test cases we used two VMware ESXi hosts and on each host we have created five virtual machines. As discussed previously, these two physical servers contain disks and SSDs that are used by the distributed storage system. Each host has 128 GB RAM in total and we have allocated 14 GB RAM to each virtual machine; 14 GB RAM is the minimum RAM that is recommended for the application.

In addition, allocating a lot of RAM resources to the virtual machines could affect the total performance, since the hypervisor may not get enough RAM. Over provisioning of RAM resources is not the focus of this study. Inside each host 24 vCPUs are available, in order to avoid any interference between the virtual machine that contains the Compuverde software and other virtual machines, we have isolated them by creating two different resource pools, one with 4 vCPUs and the other one with 20 vCPUs. The virtual machine containing Compuverde software has been bound to the resource pool with 4 vCPUs and the rest of the virtual machines are bound to the other resource pool that contains 20 vCPUs.

8.3.3 Test cases

We have measured performance during live migration in terms of migration downtime and total migration time, as well as, CPU utilization (see Table for CPU utilization, downtime/response time and total migration command APIs). Our goal was to measure how different allocations of vCPUs to each virtual machine affect the performance during live migration. We also measure how other virtual machines are affected during migration of one virtual machine. In order to measure these aspects, we defined six different test cases (see Table) and for each test case, we varied the number of vCPUs allocated to each virtual machine and measured the performance during live migration of one virtual machine.

In addition, we varied the load that is sent to each virtual machine (here we only sent the load to the active telecom servers). So one scenario was when we sent low load (100 req/s) to one virtual machine (one active telecom server) and higher load (1900 req/s) to other virtual machines (another active telecom servers) and then migrate the virtual machine with the low load (100 req/s) from the source host to the destination host. Another scenario was when we sent heavy load (1900 req/s) to one virtual machine and lower load (1000 req/s) to the other virtual machines and migrate the virtual machine with the heavy load (1900 req/s) from the source host to the destination host. As it can be observed from Table , we varied the

number of vCPUs allocated to each virtual machine (5, 10, 20 vCPUs) and repeated the two different load scenarios for each case.

8.3.3.1 Test case 1

Here we allocated 5 vCPUs to each virtual machine, in total we have allocated 25 vCPUs which is higher than number of vCPUs that we have available (20 vCPUs). So here we have little over-allocation. And we tested two different load scenarios,

Table 8.1. Command API used for performance measurements

VMware Virtualization System	**Command Interface**		
	CPU Utilization (%)	***Downtime/ Maximum Response Time (ms)***	***Total Migration Time (ms)***
Inside Hypervisor	vCenter Server-performance graphs	Inside the application	vCenter Server-performance graphs
Inside Virtual Machine	ssh + sar	Inside the application	-

Table 8.2: Test cases

Test ID	**Load Scenarios (req/s)**		**vCPUs**
	One Virtual Machine	***Other Virtual Machines***	
1.1	1900	1000	5 vCPUs
1.2	100	1900	
2.1	1900	1000	10 vCPUs
2.2	100	1900	
3.1	1900	1000	20 vCPUs
3.2	100	1900	

8.3.3.1.1 Test case 1.1

In this case we sent 1900 req/s to one virtual machine and 1000 req/s to the other virtual machines and we migrated the virtual machine with 1900 req/s load from the source host to the destination host.

8.3.3.1.2 Test case 1.2

In this case we sent 100 req/s to one virtual machine and 1900 req/s to the other virtual machines and we migrated the virtual machine with 100 req/s load from the source host to the destination host.

8.3.3.2 Test case 2

Here we allocated 10 vCPUs to each virtual machine, in total we have allocated 50 vCPUs which is higher than number of vCPUs that we have available (20 vCPUs). So here we have medium over-allocation. And we tested two different load scenarios,

8.3.3.2.1 Test case 2.1

In this case we sent 1900 req/s to one virtual machine and 1000 req/s to the other virtual machines and we migrated the virtual machine with 1900 req/s load from the source host to the destination host.

8.3.3.2.2 Test case 2.2

In this case we sent 100 req/s to one virtual machine and 1900 req/s to the other virtual machines and we migrated the virtual machine with 100 req/s load from the source host to the destination host.

8.3.3.3 Test case 3

Here we allocated 20 vCPUs to each virtual machine to in total we have allocated 100 vCPUs which is higher than number of vCPUs that we have available (20 vCPUs). So here we have massive over-allocation. And we tested two different load scenarios,

8.3.3.3.1 Test case 3.1

In this case we sent 1900 req/s to one virtual machine and 1000 req/s to the other virtual machines and we migrated the virtual machine with 1900 req/s load from the source host to the destination host.

8.3.3.3.2 Test case 3.2

In this case we sent 100 req/s to one virtual machine and 1900 req/s to the other virtual machines and we migrated the virtual machine with 100 req/s load from the source host to the destination host.

8.4 Impact of over-allocation during live migration

8.4.1 Experimental Results

The CPU utilization for the test cases 1.1 and 2.1 and 3.1 are shown in Figure 8.1.a, c and e. From the Figure 8.1.a, it can be observed that, the CPU utilization on both the source and the destination host is around 50% before the migration starts. Since the load on the virtual machine that is going to be migrated to the other host is around 100 req/s the difference between CPU utilization on the source and the destination host is not significant. During the virtual machine migration it can be observed from the Figure 8.1.a that the CPU utilization is increased to around 65% both on the source and the destination host. After the migration is completed, we can see that the CPU utilization on the source host decreases again to around 50%, while it increases on the destination host to around 55%. The reason for the higher CPU utilization on the destination host is that one virtual machine has been added, so there are in total six virtual machines running on the destination host and four virtual machines are remained on the source host. If we compare this figure with two other figures (Figure 8.1.c and e), it can be observed that the CPU utilization before the migration starts is higher for the

Table 8.3: Downtime and Total migration time results

Test ID	*Downtime/ Maximum Response Time (sec)*	*Total Migration Time (sec)*
1.1	2.214	30
1.2	4.013	38
2.1	2.166	48
2.2	4.938	52
3.1	2.923	48
3.2	4.482	53

cases with 10 vCPU (60%) and 20 vCPU (65%) compared with the case with 5 vCPU (50%) and the reason for that is the amount of over-allocation that we have and the variation is the cost of this over-allocation. After the migration completed CPU utilization on the source host became around 55% (Figure 8.1.c) and 60% (Figure 8.1.e), while on the destination host it became around 63% (Figure 8.1.c) and 68% (Figure 8.1.e). In Figure 8.1.b we can observe that the CPU utilization before the migration starts is higher on the source host than on the destination host.

The variation is caused by the load (around 1900 req/s) that we are sending to the virtual machine that we are going to migrate to the other host. As it can be observed from Figure. 8.1.b, after the migration is completed the CPU utilization on the source host decreases to 30% while it increases on the destination host (becomes around 55%). Comparing Figure. 8.1.a and b, we can observe that when the virtual machine is heavily loaded (Figure. 8.1.b), the CPU utilization has more fluctuation compare to a low loaded VM.

The same testing sequence has been followed for all other test cases with the different number of vCPU allocation and different load scenarios. In general, it can be observed after comparison between Figure 8.1.a, c and e, that the CPU utilization on the source host has not been changed dramatically after the migration for the test case 1.1, because the over-allocation is medium and after the migration four virtual machines with 5 vCPUs allocated to each are remained which will use in total 20 vCPUs and that is all the vCPUs that we have available in the source host. However, for test case 2.1 and 3.1, the difference becomes more noticeable, around 5% after the migration has finished, which still is not very significant (see Figure 8.1.c and e). So the conclusion would be that when there is a need to migrate a low loaded virtual machine, there is no dramatic difference between having 5vCPU, 10vCPU or 20vCPUs allocated to virtual machines.

Table shows that, for the cases with little over-allocation (Test cases 1.1 and 1.2), when the load increases it effects both the downtime and the total migration time. The downtime was increased by 2 seconds and the total migration time was increased by 8 seconds. For the cases with medium and massive over-allocation, Table shows that, in general, the total migration time and downtime is higher for both the low and heavy loaded VM. However, in the case with medium over-allocation, the total migration time was increased by only 4 seconds for the heavy load and which is half of the time increase compared to the case with little over-allocation. This means that the case medium over-allocation has improved the live migration performance which is very important for the real-time applications. While the downtime has not highly affected by the heavy load, it has been increased by only 2.7 seconds. For the case with the massive over-allocation (Test cases 3.1 and 3.2), the results of total migration time and downtime for both heavy and low loaded VM is very similar to the case with medium over-allocation.

8.5 Conclusion

In this study, we have measured the performance in terms of CPU utilization, migration down time and total migration time of a large telecommunication application during the live migration. We have built a testbed containing ten virtual machines hosted on two physical servers and varied the number of virtual CPUs allocated to the virtual machines. We have designed six different test cases in order to figure out how different allocation of vCPUs to each virtual machine will affect the performance during live migration, also how other virtual machines in the background are affected during the live migration of one virtual machine from the source host to the destination host.

According to our results, over-allocation has a small effect on the CPU utilization of the low loaded VMs, while it highly effects the downtime and total migration time. However, once we have reached a certain amount of over-allocation, then having more over-allocation does not have noticeable effect even on the downtime and the total migration time. This means that in the case of low loaded VMs it would be possible to have a massive over-allocation. Having

massive over-allocation gives more flexibility in terms of number of VMs that can be migrated and also reduce the risk of ending up in the situation when there is an under-allocation of resources.

However, according to our results, for the test cases with heavy loaded VMs, CPU utilization fluctuation is a lot for both before and after the live migration. Except for one of the test cases with heavy loaded VM and small amount of over-allocation (Test DI 1.2), for two other test cases with medium and high over-allocation we have got lots of request failures due to the very long downtime. This means that live migration of a heavy loaded VM is not recommended when the amount of over-allocation is medium or massive and there is a very high risk of getting request failures especially for large real-time applications.

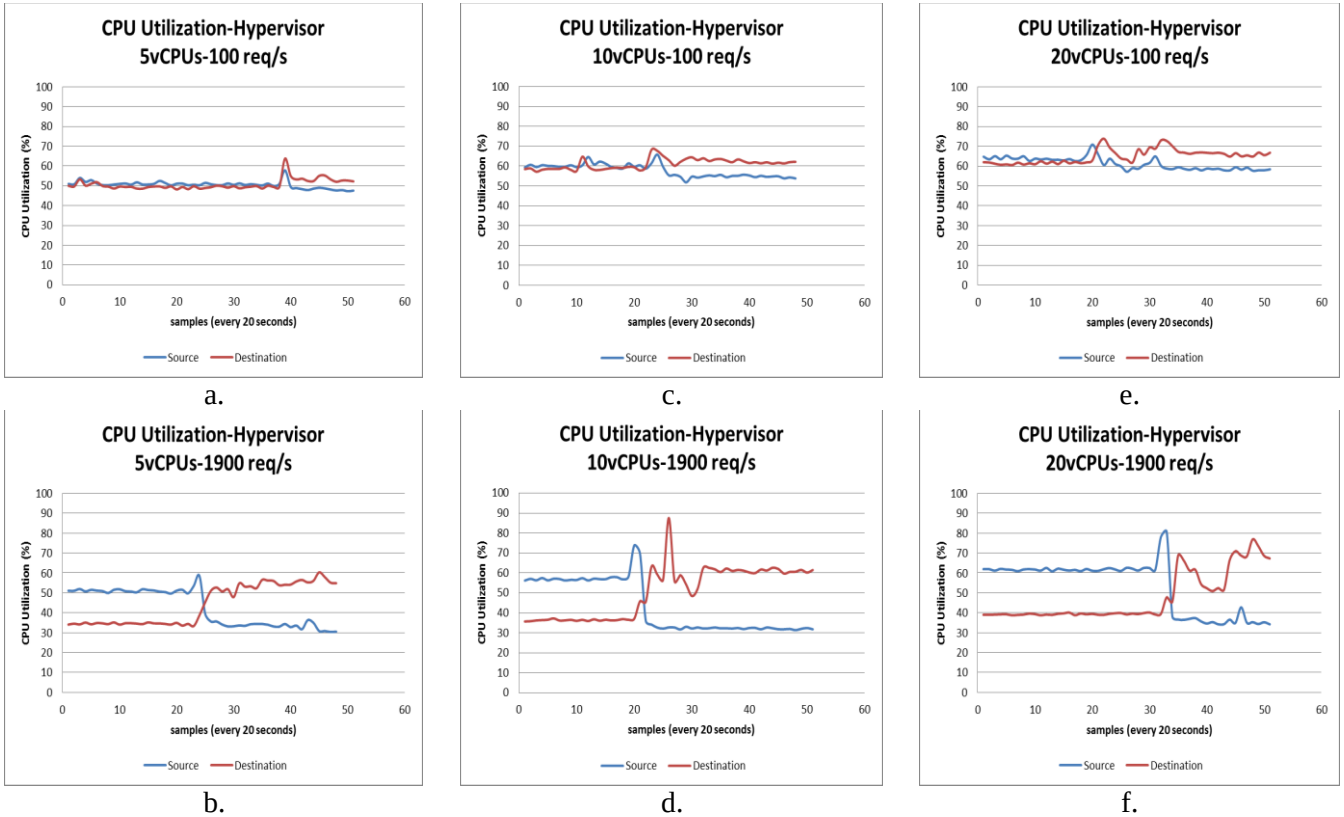


Figure 8.1. CPU utilization on the source host and the destination host before and after the live migration: Test cases 1.1 and 1.2 are shown in a and b. Test cases 2.1 and 2.2 are shown in c and d, Test cases 3.1 and 3.2 are shown in e and f.

8.6 References

[1] W. Zhang et al. “Autonomic Resource Allocation in Virtualized Data Centers,” Published in: Parallel and distributed Processing with Applications, 2012, pp 192-198.

[2] X. Liu, X. Zhu, S. Singhal, M. Arlitt, “Adaptive entitlement control of resource partitions on shared servers,” Published in: 9th International Symposium on Integrated Network Management, 2005, pp 163-176.

[3] VMware, ESXi 5.5, vCenter Server 5.5, “vSphere Resource Management,” VMware’s Technical Document, available at: http://pubs.vmware.com/vsphere-55/topic/com.vmware.ICbase/PDF/vsphere-esxi-vcenter-server-55-resource-management-guide.pdf.

[4] B. Watson et al. “Probabilistic Performance Modeling of Virtualized Resource Allocation,” Published in: 7th International conference on Autonomic Computing, 2010, pp 99-108.

[5] S. Shirinbab, L. Lundberg, D. Ilie, ”Performance Comparison of KVM, VMware and XenServer using a Large Telecommunication Application,” Published in: the fifth International Conference on Cloud Computing, GRIDs, and Virtualization, 2014, pp 114-122.

[6] S. Shirinbab, L. Lundberg, D. Erman,”Performance evaluation of distributed storage systems for cloud computing,” Published in: International Journal of Computers and their applications, 2013, pp 195-207.

[7] M. Elsaid, C. Meinel, “Live Migration Impact on Virtual Datacenter Performance: VMware vMotion Based Study,” Published in: Future Internet of Things and Cloud, 2014, pp 216-221.

[8] A. Strunk, “Costs of Virtual Machine Live Migration: A Survey,” Published in: Services, 2012, pp 323-329.

[9] C. Clark et al., “Live Migration of Virtual Machines,” Published in: 2nd conference on Networked Systems Design and Implementation, 2005, pp 273-286.

[10] W. Hu et al., “A Quantitative Study of Virtual Machine Live Migration,” Published in: Cloud and Autonomic Computing conference, 2013, pp 1-10.

[11] VMware, Inc. “VMware vSphere vMotion Architecture, Performance and Best Practices in VMware vSphere 5,” Technical Paper, 2011, pp 1-26.

[12] S. Shirinbab, L. Lundberg, “Performance Implications of Over-allocation of Virtual CPUs,” Published in: International Symposium on Networks, Computers and Communications, 2015, pp 1-6.

[13] K. Lazri, S. Laniepce, J. Ben-Othman, “When Dynamic VM Migration Falls under the Control of VM Users,” Published in 5th International Conference on Cloud Computing Technology and Science, 2013, pp 395-402.

[14] R. Birke, A. Podzimek, L.Y. Chen, E. Smirni, “State-of-the-Practice in Data Center Virtualization: Towards a Better Understanding of VM Usage,” Published in: Dependable Systems and Networks, 2013, pp 1-12.

[15] Y. Wu, M. Zhao, “Performance Modeling of Virtual Machine Live Migration,” Published in: Cloud Computing Conference, 2011, pp 492-499.

[16] Z. Xiao, W. Song, Q. Chen, “Dynamic Resource Allocation Using Virtual Machines for Cloud Computing Environment,” Published in: IEEE Transactions on Parallel and Distributed Systems, 2013, pp 1107-1117.

[17] S. Vishnupriya, P. Saranya, P. Suganya, “Effective Management of Resource Allocation and Provisioning Cost Using Virtualization in Cloud,” Published in: Advanced Communication Control and Computing Technologies, 2014, pp 1726-1731.

[18] M. Elsaid, C. Meinel, “Live Migration Impact on Virtual Datacenter Performance,” Published in: Future Internet of Things and Cloud, 2014, pp 216-221.

[19] W. Hu et al., “A Quantitative Study of Virtual Machine Live Migration,” Published in: ACM Cloud and Autonomic Computing Conference, 2013, pp 1-10.

9 Performance Evaluation of Container and Virtual Machine Running Cassandra Workload

Abstract

Today, scalable and high-available NoSQL distributed databases are largely used as Big Data platforms. Such distributed databases typically run on a virtualized infrastructure that could be implemented using Hypervisor-based virtualization or Container-based virtualization. Hypervisor-based virtualization is a mature technology but imposes overhead on CPU, memory, networking, and disk. Recently, by sharing the operating system resources and simplifying the deployment of applications, container-based virtualization is getting more popular. Container-based virtualization is lightweight in resource consumption while also providing isolation. However, disadvantages are security issues and I/O performance. As a result, today these two technologies are competing to provide virtual instances for running big data platforms. Hence, a key issue becomes the assessment of the performance of those virtualization technologies while running distributed databases.

This paper presents an extensive performance comparison between VMware and Docker container, while running Apache Cassandra as workload. Apache Cassandra is a leading NoSQL distributed database when it comes to Big Data platforms. As baseline for comparisons we used the Cassandra's performance when running on a physical infrastructure. Our study shows that Docker had lower overhead compared to the VMware when running Cassandra. In fact, the Cassandra's performance on the Dockerized infrastructure was as good as on the Non-Virtualized.

9.1 Introduction

Hypervisor-based virtualization began in 1960s and since then it has been widely used in Cloud Computing. Hypervisors, also called Virtual Machine Monitors (VMM) share the hardware resources of a real machine between multiple Virtual Machines (VMs). By virtualizing system resources such as CPUs, memory and interrupts, it became possible to run multiple Operating Systems (OS) concurrently. Most commonly used hypervisors are Kernel Virtual Machine (KVM), Xen Server, VMware and Hyper-V. Hypervisor-based virtualization enables new features such as performance management, elastic resource scaling, and reliability services to be applied without requiring modifications to applications or operating systems. It also enables virtual machine migration for load balancing to eliminate hotspots and consolidation to improve resource utilization and energy efficiency. However hypervisor level virtualization

introduce performance overheads as studied in [4][5][6][7][8] and still limits it from being used in performance critical domains [1][2][3].

Recently, container-based virtualization has gained more popularity than hypervisor-based virtualization. A container is a light weight operating system running inside the host system. An application running in a container has an unshared access to a copy of the operating system. In other words, containers virtualize the operating system while hypervisors virtualize the hardware resources. Therefore, container-based virtualization is well-known for providing savings in resource consumption without the overhead of hypervisor-based virtualization while also providing isolation [9]. The main difference between Virtual Machine and Container architecture is that, for the virtual machines, each virtualized application includes an entire guest operating system and necessary Binaries/Libraries, while for the containers; the Container engine contains just the application and its dependencies (Binaries/Libraries).

The Container-based virtualization is not a new concept; it has been offered before by FreeBSD Jails (available since 2000) and Solaris Zones (available since 2004). In the beginning of 2008 a new Linux kernel was released replacing the earlier variations in the form of Linux container (LXC) [10][11][12]. Other alternatives to Linux-based containers are Open VZ [13][14][15] and Docker [16][17][18]. Recently, there have been several studies on performance of container-based virtualization technologies, especially Docker containers [19][20]. Docker containers are designed to run a single application per container while LXC containers are more like virtual machines with a fully functional operating system [16]. The Container-based architecture is rapidly becoming a popular development and deployment paradigm because of advantages such as low overhead and portability. Disadvantages in using containers are: security issues (which will not be the focus in this study [21]); the limitations of not being able to run a different OS or kernels; and the maturity level of management tools (e.g. for live migration, snapshotting and resizing) is lower than for virtual machines.

Since both containers and hypervisors have their set of benefits and drawbacks, one of the key point to select the proper virtualization technology for big data platforms is to assess the performance of virtualized or containerized databases and how this relates to physical ones [22]. This paper answer to this need providing a detailed performance comparison running a distributed database on a cluster of physical server, on a VMware cluster of virtual machine, and on a Docker cluster. As database, we selected Apache Cassandra [23][24][25] an open-source NoSQL distributed database widely adopted by companies using Docker and VMware for production and widely used in Big Data applications. Cassandra is a leading transactional, scalable, and highly-available. It is known to manage some of the world's largest datasets on clusters with many thousands of nodes deployed across multiple data centers. Also, Cassandra Query Language (CQL) is user-friendly and declarative [26][27].

Our experimental results show that the Dockerized version had lower overhead compared to the VMware Virtualized. In fact, the performance of the Dockerized version was as good as Non-Virtualized.

The presented work is organized as follows: In Section 9.2 we discuss related work. Section 9.3 describes the experimental setup and test cases. Section 9.4 presents the experimental results and we conclude our work in Section 9.5.

9.2 Related Work

Virtual machine is a mature technology which was introduced by IBM mainframes in the 70s [28]. Virtual Machines have benefited from decades of hardware and software optimizations. Hypervisor-based cloud computing platforms such as KVM, VMware, Xen, and Hyper-V introduce an overhead which has been well-studied [4][5][6][7][8].

Recently, container-based virtualization technologies such as, Docker and Linux Container (LXC) have gained a lot of interest [29][30][31]. The major advantage of container is in achieving near-native performance. There are a number of studies comparing hypervisor-based virtualization versus container-based virtualization. In [32] the authors compared the performance of KVM and Xen hypervisors with Docker container on the ARM architecture. According to their results containers had better performance in CPU bound workloads and request/response networking, while hypervisors had better performance in disk I/O operations (because of the hypervisor's caching mechanisms) and TCP streaming benchmark. Similarly, in [33] and [34], the authors compared the performance of container-based virtualization with hypervisor-based virtualization for HPC. According to their result, container-based solution delivered better performance than hypervisor-based solution. Our work has some similarities to these works however in our study we considered performance of the Cassandra a distributed NoSQL database.

In [35] the authors characterized the performance of three common hypervisors KVM, Xen, VMware ESXi and an open source container LXC across two generations of GPUs and two host microarchitectures, and across 3 sets of benchmarks. According to their results KVM achieved 98-100% of the base system's performance while VMware and Xen achieved only 96-99%. The difference between their work and our work is that we considered performance of Cassandra database in terms of CPU utilization, disk utilization and latency while they were more interested in performance of different generations of GPUs. In [13] the authors compared the performance of an open source container technology OpenVZ and Xen hypervisor. According to their results, container-based virtualization outperforms hypervisor-based virtualization. Their work is similar to our study. However, they considered Wide-Area Motion Imagery (WAMI), Full Motion Video (FMV), and text data, while in our case study we were more interested in the performance of the Cassandra database. Also in our study, we used KVM as a hypervisor-based virtualization and as a container-based virtualization

we considered Docker and LXC which are different from what they have considered in their study.

In [2] the authors compared the execution times of AutoDock3 (a scientific application) for both Docker container and in virtual machines created using OpenStack. According to their results, the overall execution times for container-based virtualization systems are less than in hypervisor-based virtualization systems due to differences in start-up times. In [9] the authors analyzed the process handling, file systems and namespace Isolation for container-based virtualization systems such as Docker, Linux Containers (LXC), OpenVZ and Warden. From their assessment, containers have an advantage over VMs because of performance improvements and reduced start-up times. In [3] the authors demonstrated that container-based systems are more suitable for usage scenarios that require high levels of isolation and efficiency such as HPC Clusters. Their results indicate that container-based systems perform two times better for server-type workloads than hypervisor-based systems.

Databases are often chosen to store and query large amount of data. Traditionally, SQL databases were used in most of datacenters. However, because of scalability issues, NoSQL databases have gained popularity since 2007 [36]. NoSQL databases support large demands of data in a scalable and flexible manner. Currently, in the majority of datacenters online applications with NoSQL databases are hosted on virtual machines. Therefore, there is a need to evaluate NoSQL database performance in virtual environments. There are various studies trying to assess the performance impacts of virtualization on SQL and NoSQL databases.

In [37] the authors compared performance of three databases: a SQL database PostgreSQL, and two NoSQL databases MongoDB and Cassandra using a sensor data storage. They also compared running these databases on a Physical machine and a Virtual machine. According to their results, virtualization has a huge effect on Cassandra read performance while it has a moderate performance impact on MongoDB, and increase the write performance on PostgreSQL. The difference between their work and our work is that, we were more interested in comparison between performance impacts of container-based virtualization and hypervisor-based virtualization on the Cassandra database. Another difference is that they have used only one machine to run their experiment while in our work we have used multiple machines. In [38] the authors compared the performance of Docker container and KVM hypervisor using the MySQL database. According to their results, container show equal or better performance than virtual machine in almost all cases. Their work is similar to our work. However, we chose to evaluate Cassandra because it is a popular NoSQL database and it is widely used in the cloud. In [39] the authors compared the performance of two NoSQL databases MongoDB and Cassandra using different virtualization techniques, VMware (full virtualization) and Xen (paravirtualization). According to their results VMware provides better resource utilization for NoSQL databases. Their work is different from our work; in our case study we considered performance of Cassandra on KVM hypervisor, and container-based virtualization.

9.3 Evaluation

The goal of the experiment was that of comparing the performance of VMware virtual machines and Docker containers when running Cassandra. As baseline for the comparison we used the performance of Cassandra running on physical servers.

9.3.1 Experimental Setup

All our tests were performed on three HP servers DL380 G7 with a total of 16 processor cores (plus HyperThreading), 64 GB of RAM and disk of size 400 GB. Cassandra 3.0.8 run on RHEL7 and this setup was identical for all test cases: Non-Virtualized, Virtualized, and Dockerized. In total, three Cassandra nodes were configured in a cluster with default settings. The same version of Cassandra was used on the load generators as well. VMware ESXi 6.0.0 was installed in case of Cassandra-Virtualized.

9.3.2 Workload

To generate workload, we used Cassandra-stress tool. The Cassandra-stress tool is a Java-based stress utility for basic benchmarking and load testing of a Cassandra cluster. Creating the best data model requires significant load testing and multiple iterations. The Cassandra-stress tool helps us in this endeavor by populating our cluster and supporting stress testing of arbitrary CQL tables and arbitrary queries on tables. The Cassandra package comes with a command-line stress tool (Cassandra-stress tool) to generate load on the cluster of servers, the cqlsh utility, a python-based command line client for executing Cassandra Query Language (CQL) commands and the nodetool utility for managing a cluster. These tools are used to stress the servers from the client and manage the data in the servers.

The Cassandra-stress tool creates a keyspace called keyspace1 and within that, tables named standard1 or counter1 in each of the nodes. These are automatically created the first time we run the stress test and are reused on subsequent runs unless we drop the keyspace using CQL. A write operation inserts data into the database and is done prior to the load testing of the database. Later, after the data are inserted into the database, we run the mix workload, and then split up the mix workload and run the write only workload and the read only workload.

Table 9.1. Cassandra-stress Tool Sample Commands

	Command
Populate the Database	dsc-cassandra-3.0.8/tools/bin/cassandra-stress write n=40000000 -pop seq=1..40000000 -node -schema "replication(strategy=NetworkTopologyStrategy, datacenter1=3)"

Mix-Load	dsc-cassandra-3.0.8/tools/bin/cassandra-stress mixed ratio\(write=1, read=3\) duration=30m -pop seq=1..40000000 -schema keyspace="keyspace1" -rate threads=100 limit=op/s –node
Read-Load	dsc-cassandra-3.0.8/tools/bin/cassandra-stress read duration=30m -pop seq=1..40000000 -schema keyspace="keyspace1" -rate threads=100 limit=op/s –node
Write-Load	dsc-cassandra-3.0.8/tools/bin/cassandra-stress write duration=30m -pop seq=1..40000000 -schema keyspace="keyspace1" -rate threads=100 limit=op/s –node

Below we described in detail each workload as well as the commands we used for generating the workloads:

9.3.2.1 Mix-Load

To analyze the operation of a database while running both read and write operations during one operation, a mixed load command is used to populate the cluster. A mixed load operation consists of 75% read requests and 25% write requests (three reads and one write) and generated for duration of 30 minutes onto the three-node Cassandra cluster. The command used for generating the mixed load is described in Table 9.1.

9.3.2.2 Read-Load

In addition to the mix workload, we measured the performance of the database for the read-only workload. In this case, a read load command is used to populate the cluster. A read load operation consists of 100% read requests generated for duration of 30 minutes onto the three-node Cassandra cluster. The command used for generating the read-load is described in Table 9.1.

9.3.2.3 Write-Load

In addition to the mix and read workload, we measured the performance of the database for the write-only workload. In this case, a write load command is used to populate the cluster. A write load operation consists of 100% write requests generated for duration of 30 minutes onto the three-node Cassandra cluster. The command used for generating the write-load is described in Table 9.1.

Short descriptions of the Cassanrda-stress tools input parameters (used in Table 9.1) is noted below:

- mixed: Interleave basic commands with configurable ratio and distribution. The cluster must first be populated by a write test. Here we selected a mixed load operation of 75% reads and 25% write.
- write: Multiple concurrent writes against the cluster.
- read: Multiple concurrent reads against the cluster.

- n: Specify the number of operations to run. Here we chose n=40000000 to generate 10 GB database.

- pop: Population distribution and intra-partition visit order. In this case we chose seq= 1…40000000.

- node: To specify the address of the node to which data is to be populated.

- schema: Replication settings, compression, compaction, and so on. Here for the write operation we have modified the replication strategy to "NetworkTopologyStrategy" and set the number of replication for datacenter 1 to 3. Later for the mixed load we just set the name of the default keyspace which is "keyspace1".

- duration: It specifies the time in minutes to run the load.

- rate: Thread count, rate limit, or automatic mode (default is auto). In order to control the incoming traffic, we set the number of threads to 100. We have also limited the number of operations per second, so that we can measure the CPU utilization, write rate and the latency mean for different number of Transactions Per Seconds(tps) (40k, 80k, 120k, 160k, and 200k tps).

9.3.3 Performance metrics

The performance of Docker container and VMware versus the non-virtualized solution are measured using the following metrics:

- CPU utilization,

- disk and write rate, and

- mean latency.

The CPU utilization and the disk throughput are measured directly on the server nodes by means of sar and iostat respectively. The latency is measured on the client side, that is measured by the stress test tool.

9.3.4 Test cases

9.3.4.1 Cassandra-Non-Virtualized:

In this case, three servers are allocate and on each server we run Cassandra application. All servers are connected and create a three-node Cassandra cluster.

9.3.4.2 Cassandra-Virtualized:

In this case, one virtual machine is created on each host, in total three virtual machines.

9.3.4.3 Cassandra-Dockerized:

In this case, one container is created on each server, in total three containers. In each container we run Cassandra application. All Cassandra nodes are connected and a three node Cassandra cluster in configured.

In each test case, we first experiment with Cassandra different workload scenarios, Mix, Read and Write workload. Second, we experiment with various replication-factor configurations for the Cassandra cluster. In Cassandra, the input splits are replicated among the nodes based on a user-set replication-factor (RF). This design prevents data loss and helps with fault tolerance in case of node failure [40]. In our experiments, we investigate three different replication-factor setup, RF=1, RF=2, and RF=3. In our test environment with three Cassandra node cluster, replication factor three means that each node should have a copy of the input data splits. For the case of RF=2, the seed node decides based on some algorithms where to store replicas. In the case of RF=1, each node only receives part of the input data and no replicas. We repeat the same test cases for all three different technologies, Non-virtualized, Virtualized, and Dockerized.

9.4 Experimental Results

In this section, we present the results of the experiments.

9.4.1 Mix-Load

Figure 9.2 shows the results of running the three node Cassandra cluster in a Non-virtualized, Virtualized and Dockerized environment while having a Mix-Load. The result show that both Cassandra Non-Virtualized and Dockerized with RF=1 can handle maximum 200k (tps). While Cassandra Virtualized with RF=1 can only handle 160k (tps). Considering the CPU overhead for the virtual machines when run the mix workload. CPU overhead measured from subtraction of CPU utilization of Virtualized-Cassandra and Non-Virtualized-Cassandra. For the Virtualized-Cassandra case the CPU overhead is up to 29%. One reason for getting very high overhead could be due to additional layers of processing required by the virtualization software e.g. VMware [41].

For the case with RF=2, we could reach a higher throughput than Non-virtualized with Docker. One reason could be that the write latency was higher on the Non-virtualized compared with the Docker. It can be observed from the Figure

9.2 (g, h, i). This slightly higher latency could be one reason for the Cassandra to not perform as good on the Non-virtualized as it did on the Docker.

In general, for all different deployments, RF=1 is always performing better than RF=2 and RF=3. One reason is that when the RF in set to one, only one copy of data is written and this process is very fast in Cassandra. As it can be seen also from the Write rate figures (Figure 9.2 (d, e, f)), less number of packets are written to the disk when the RF is equal to one. Respectively, latency mean for RF=1 is lower than RF=2 and RF=3. Although RF=1 is fast but from high availability point of view the data center providers prefer to have RF=2 or RF=3. However, Cassandra is using different Consistency Levels in order to wait for the respond from several nodes, and in our case we set the consistency level to be Quorum. Quorum means that Cassandra node returns the record after a quorum of replicas has responded from any datacenter. So, in our case as it can also be seen from the figures, for cases with RF=2 and RF=3 the performance is almost identical in terms of CPU utilization. Except for RF=2 in Dockerized which performed better than RF=3 in Dockerized (Figure 9.2 (a)). In terms of Write rate and latency, RF=2 is lower than RF=3.

9.4.2 Write-Load

Figure 9.3 shows the results of running the Cassandra application inside VMware virtual machines, Docker containers and bare-metal while running the write-load. Comparing Figure 9.3 and Figure 9.2, we can observe that, the CPU utilization is very low in general in Figure 9.3 (less than 50%) compared with Figure 9.2 (more than 90%). Respectively, as it can be seen from latency mean figures in Figure 9.3, latency mean is also very low (around 1) compared with the latency mean in Figure 9.2.

In conclusion we can say that write-load is very quick in Cassandra. One reason could be the way data is written in Cassandra. During a write process in Cassandra, data is written in the commit log and mem table, then flushing data from the mem table and storing data on disk in SSTables. In this process, Cassandra does not need to wait for any acknowledgment from other nodes regarding replicas.

9.4.3 Read-Load

In Figure 9.4, we show the result of running Cassandra inside a three node cluster using different deployments such as: Dockerized, Virtualized and bare-metal while running the read-load. Comparing the results of the Read-load presented in Figure 9.4 with Figures 9.2 and 9.3, we can observe that the read process is consuming a lot of CPU resources. And according to the latency mean figures, read process is much slower than write process in Cassandra. One reason for that could be the read process in Cassandra. During a read process, Cassandra processes data at several stages on the read path to discover where data is stored, such as checks the mem table, checks the row cache, checks Bloom filter, checks partition key cache. All

these processes consume CPU resources and has major impact on the latency. In summary, Docker container performed better than Cassandra-Virtualized and except one case, it performed as good as Non-Virtualized in terms of throughput.

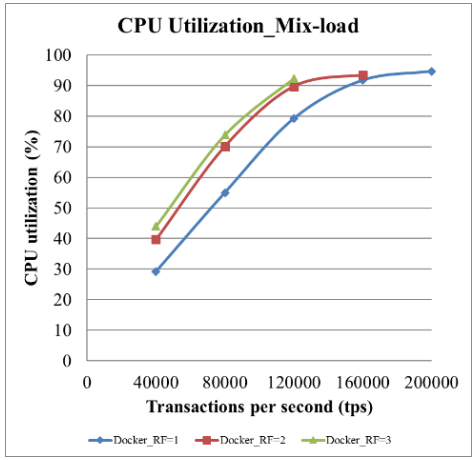
In terms of CPU utilization, Non-virtualized show the lowest CPU utilization and then Docker show a slightly higher CPU utilization with the overhead of around 10% and then Cassandra-Virtualized show the highest CPU utilization with the overhead of around 30%. In terms of the write rate, Cassandra-Virtualized write rate is slightly higher than the Cassandra-Non-Virtualized and the Cassandra-Dockerized. In terms of the latency mean, Cassandra-Virtualized has higher latency compared to the Non-Virtualized and the Docker.

9.5 Conclusions and Future Work

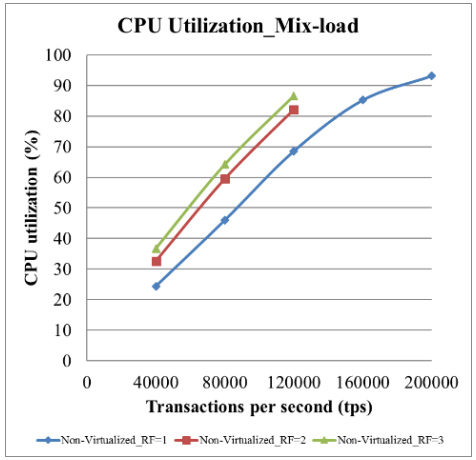
In this paper, we try to address the problem of which solution is better for distributed databases such as Cassandra, Non-Virtualized, Virtualized (VMware) or Docker? The overall result showed that the biggest issue with running the Cassandra-Virtualized, is the significant resource and operational overheads of the virtualization layer which affects the performance of the application too. However, Cassandra-Dockerized seems to address the challenges of virtualization and that is done mostly by packaging the applications and their dependencies into lightweight containers. According to the result, Cassandra-Dockerized consumed fewer resources and operational overheads compared to the Cassandra-Virtualized (see Table 9.2). In addition, the performance of Cassandra-Dockerized was as good as Cassandra-Non-Virtualized.

Even though container solution is showing very low overhead and system resource consumption, it suffers from securing stored data which is crucial for database protection. Comparing containers architecture with virtual machines, containers cannot be secure candidate for databases because all containers share the same kernel and are therefore less isolated than virtual machines. A bug in the kernel affects every container and results in data loss. On the other hand, hypervisor-based virtualization is a mature and secure technology. Virtual machines are able to partition and distribute resources viably in the hypervisor without relying on kernel support or separate hardware.

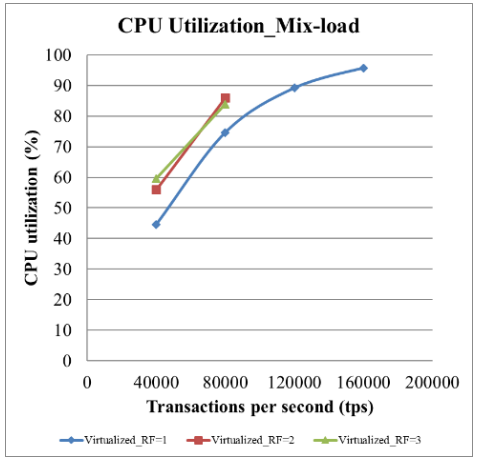
However, according to our results, hypervisor-based virtualization suffers from noticeable overhead which effects the performance of the databases. Since both containers and virtual machines have their set of benefits and drawbacks, an alternative solution could be to combine the two technologies. In the future, we plan to investigate the alternative solution by running containers inside virtual machines running Cassandra workload. In this way, we can get the benefits of both security of the virtual machine with the execution speed of containers.



a.

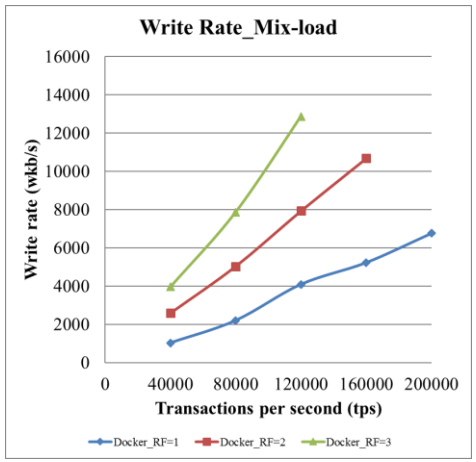


b.

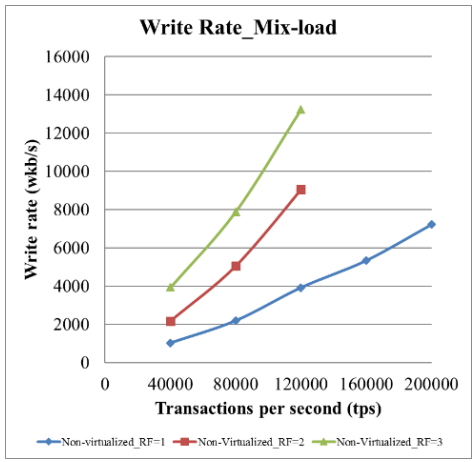


c.

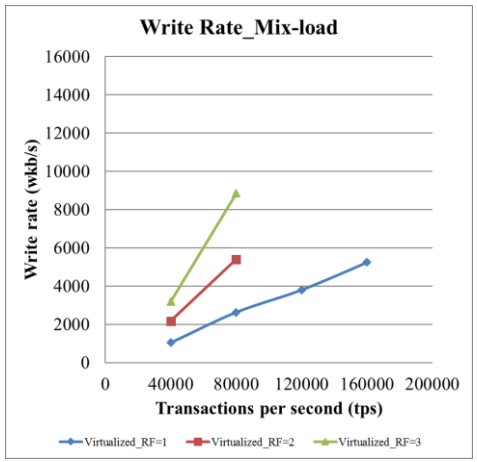
Figure. 9.2: Mix-load workload. Docker, Non-virtualized and Virtualized scenarios are compared using the mean value of CPU utilization, write rate and latency.



d.

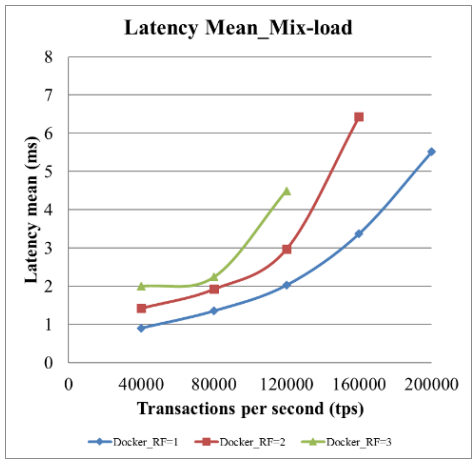


e.

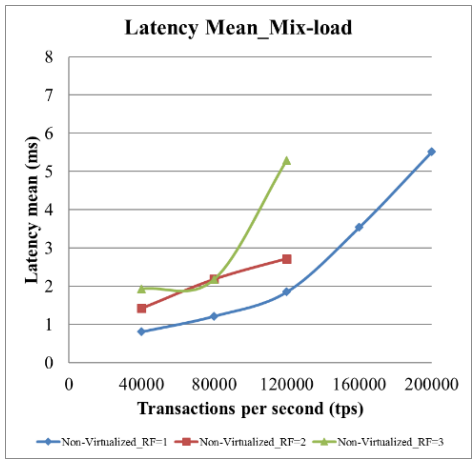


f.

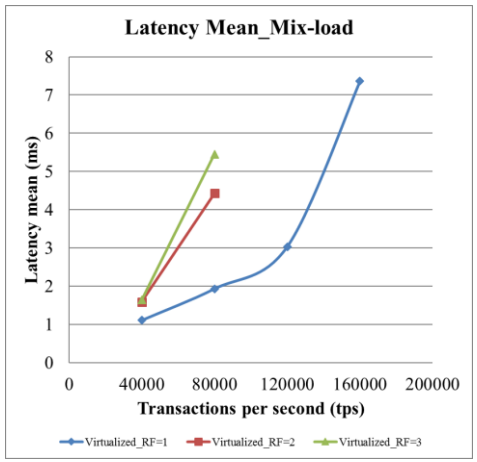
Figure. 9.2: Mix-load workload. Docker, Non-virtualized and Virtualized scenarios are compared using the mean value of CPU utilization, write rate and latency.



g.

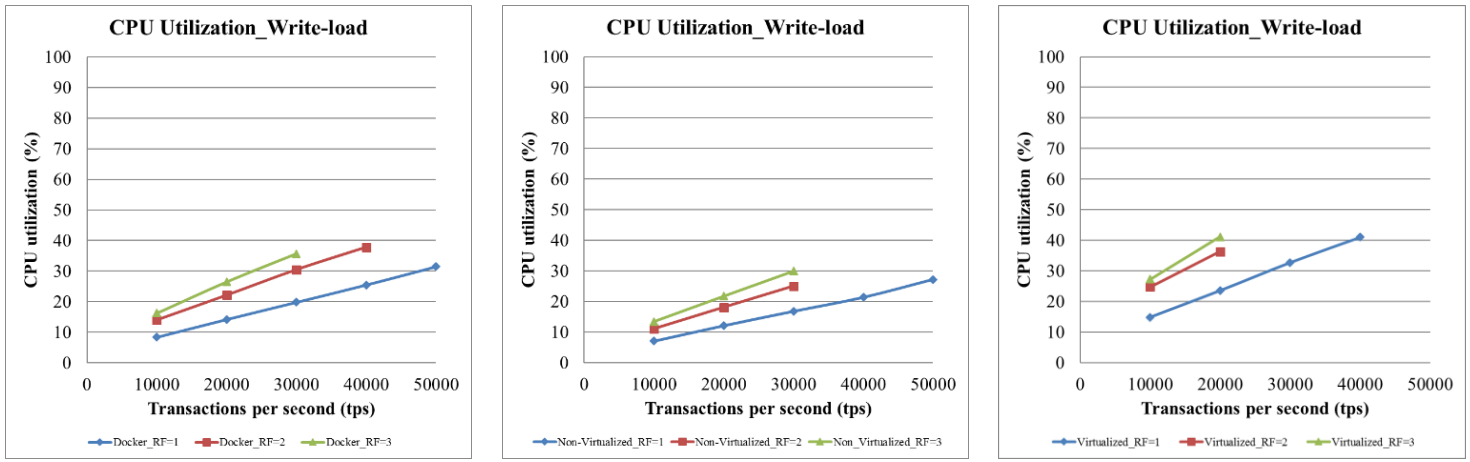


h.



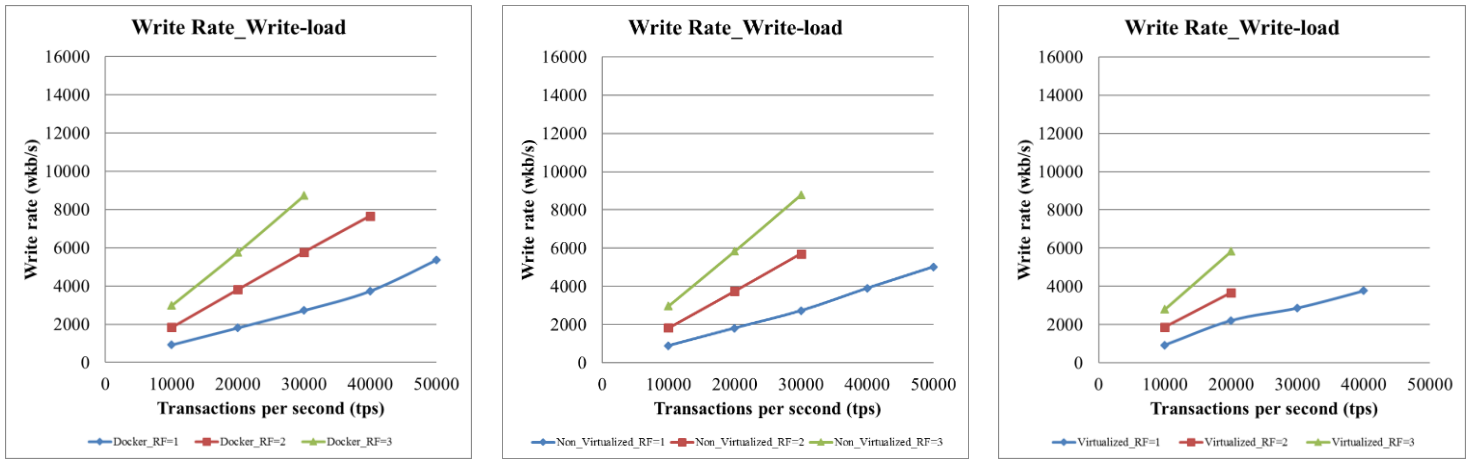
i.

Figure. 9.2: Mix-load workload. Docker, Non-virtualized and Virtualized scenarios are compared using the mean value of CPU utilization, write rate and latency.



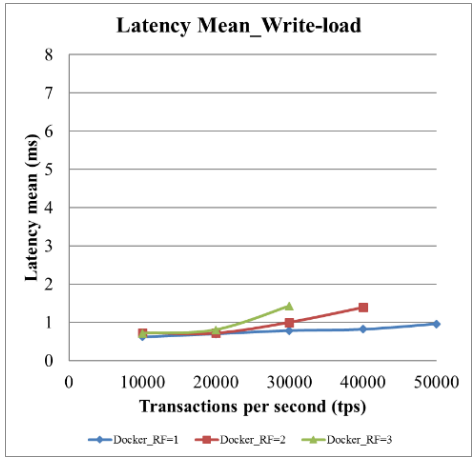
a. *b.* *c.*

Figure. 9.3: Write-load workload. Docker, Non-virtualized and Virtualized scenarios are compared using the meand value of CPU utilization, write rate and latency.

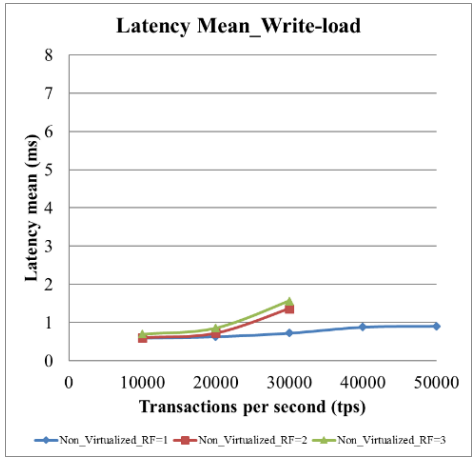


d. *e.* *f.*

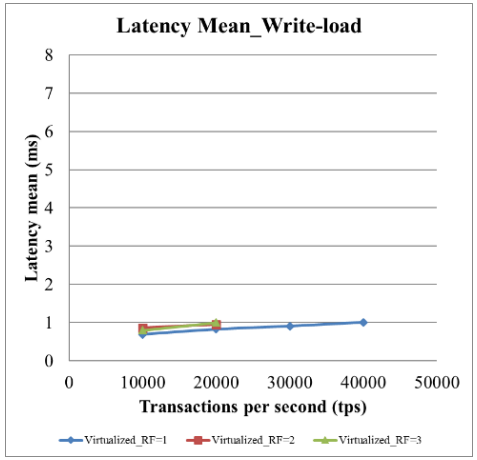
Figure. 9.3: Write-load workload. Docker, Non-virtualized and Virtualized scenarios are compared using the meand value of CPU utilization, write rate and latency.



g.

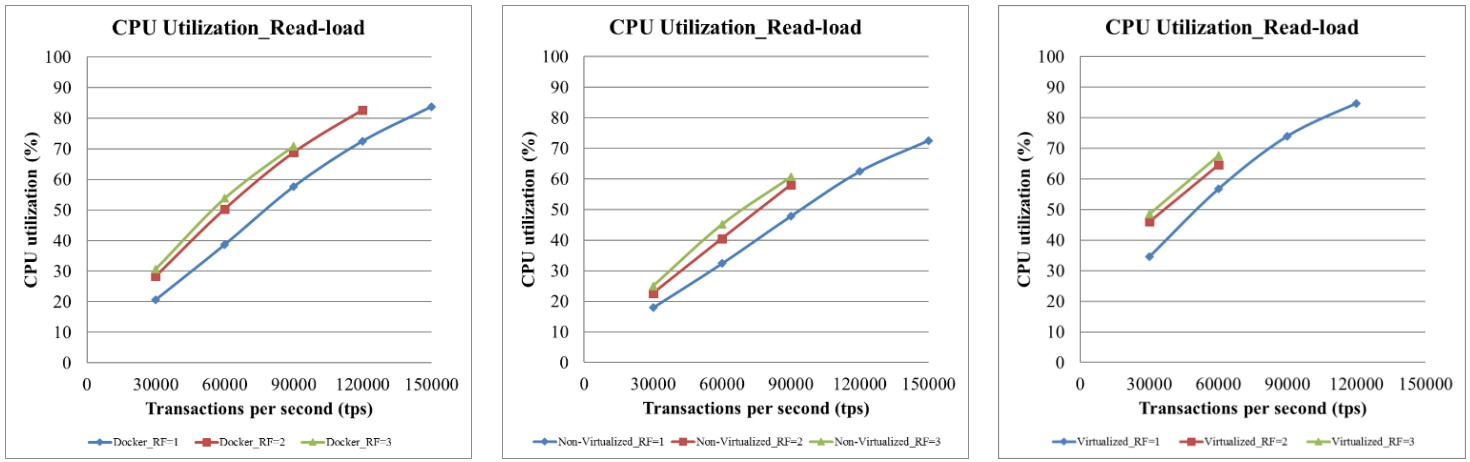


h.



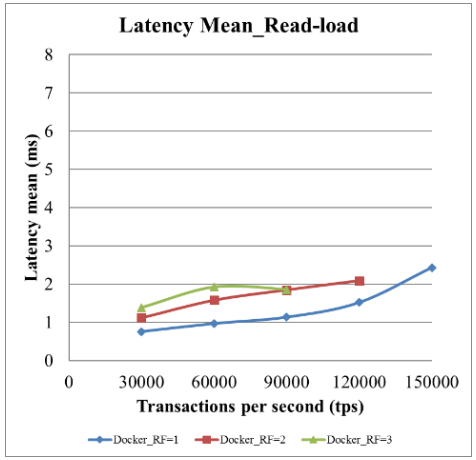
i.

Figure. 9.3: Write-load workload. Docker, Non-virtualized and Virtualized scenarios are compared using the meand value of CPU utilization, write rate and latency.

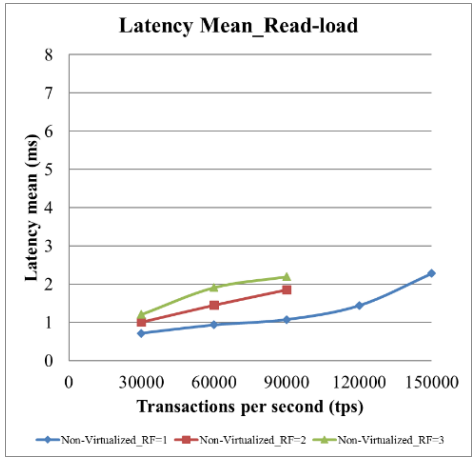


a. *b.* *c.*

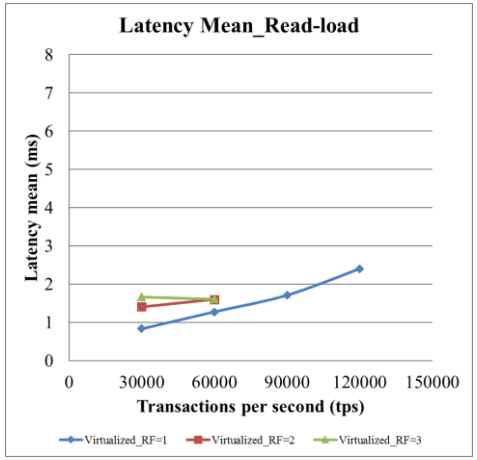
Figure. 9.4: Read-load workload. Docker, Non-virtualized and Virtualized scenarios are compared using the mean value of CPU utilization, write rate and latency.



d.



e.



f.

Figure. 9.4: Read-load workload. Docker, Non-virtualized and Virtualized scenarios are compared using the mean value of CPU utilization, write rate and latency.

Table 9.2. Summary of the Performance Results

RF	Workload	Docker			Non-Virtualized			Virtualized		
		CPU Util (%)	***Write Rate (wkb/s)***	***Latency (msec)***	***CPU Util (%)***	***Write Rate (wkb/s)***	***Latency (msec)***	***CPU Util (%)***	***Write Rate (wkb/s)***	***Latency (msec)***
1	***Mix (80K tps)***	55	2201	1.35	46	2190	1.2	75	2633	1.9
	Read (60k tps)	39	2	0.9	32	3	0.9	57	15	1.2
	Write (20k tps)	14	1811	0.7	12	1816	0.6	24	2210	0.8
2	***Mix (80k tps)***	70	5032	1.9	59	5050	2.1	86	5407	4.4
	Read (60k tps)	50	2	1.5	41	3	1.4	65	16	1.6
	Write (20k tps)	22	3815	0.7	18	3748	0.7	36	3667	0.9
3	***Mix (80k tps)***	74	7860	2.2	64	7888	2.1	84	8849	5.45
	Read (60k tps)	54	2	1.9	45	2	1.9	68	18	1.6
	Write (20k tps)	26	5772	0.8	22	5835	0.8	41	5828	1

9.6 References

[1] L. Yu, L. Chen, Z. Cai, H. Shen, Y. Liang, Y. Pan, "Stochastic Load Balancing for Virtual Resource Management in Datacenters," Published in IEEE Transactions on Cloud Computing (Volume: PP, Issue: 99), 2016, pp 1-14.

[2] T. Adufu, J. Choi, Y. Kim, "Is Container-Based Technology a Winner for High Performance Scientific Applications?" Published in APNOMS, 2015, pp 507-510.

[3] S. Soltesz, H. Potzl, M. E. Fiuczynski, A. Bavier, L. Peterson, "Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors," Published in EuroSys conference, 2007, pp. 275-287.

[4] J. Li, et al. "Performance Overhead among Three Hypervisors: An Experimental Study using Hadoop Benchmarks," Published in IEEE international congress on Big Data, 2013, pp 9-16.

[5] S. G. Soriga and M. Barbulescu, "A Comparison of Performance and Scalability of Xen and KVM Hypervisors," Published in RoEduNet international conference 12th edition, 2013, pp 1-6.

[6] J. Hwang, S. Zeng, F. Wu, T. Wood, "A Component-Based Performance Comparison of Four Hypervisors," Published in IFIP/IM, 2013, pp 269-276.

[7] J. Che, Q. He, Q. Gao, D. Huang, "Performance Measuring and Comparing of Virtual Machine Monitors," Published in EUC international conference, 2008, pp 381-386.

[8] S. Shirinbab, L. Lundberg, D. Ilie, "Performance Comparison of KVM, VMware and XenServer using a Large Telecommunication Application," Published in the 5th international conference on Cloud Computing, GRIDs, and Virtualization, 2014, pp 114-122.

[9] R. Dua, A. R. Raja, D. Kakadia, "Virtualization vs Containerization to support PaaS Cloud Engineering," Published in IC2E conference, 2014, pp. 610- 614.

[10] LXC, "Linux Containers," 2016, https://linuxcontainers.org/

[11] C. Pahl, "Containerization and the PaaS Cloud," Published in IEEE Cloud Computing (Volume: 2, Issue: 3), 2015, pp 24-31.

[12] R. Rosen, "Linux Containers and The Future Cloud," Linux J, 2014.

[13] R. Wu, A. Deng, Y. Chen, E. Blasch, B. Liu, "Cloud Technology Applications for Area Surveillance," Published in NAECON conference, 2015, pp 89-94.

[14] OpenVZ, “OpenVZ Virtuozzo Containers,” 2016, https://openvz.org/

[15] K. Kolyshkin, “Virtualization in Linux,” Whitepaper, OpenVZ, 2006.

[16] B. I. Ismail et al. “Evaluation of Docker as Edge Computing Platform,” Published in ICOS conference, 2015, pp 130-135.

[17] Docker, “Docker,” 2016, https://www.docker.com/

[18] C. Anderson, ”Docker: Software Engineering”, Software IEEE, (Volume:32, Issue: 3), 2015.

[19] C. Kan, “DoCloud: An Elastic Cloud Platform for Web Applications Based on Docker,” Published in ICACT, 2016, pp 478-483.

[20] E. N. Preeth, J. Mulerickal, B. Paul, Y. Sastri, “Evaluation of Docker Containers Based on Hardware Utilization,” Published in ICCC conference, 2015, pp 697-700.

[21] A. Gkortzis, S. Rizou, D. Spinellis, “An Empirical Analysis of Vulnerabilities in Virtualization Technologies,” Published in Cloud Computing Technology and Science conference, 2016, pp. 533-538.

[22] C. Mohmet and M. Buyukkececi, “Big Data Challenges in Information Engineering Curriculum," Published in EAEEIE conference, 2014, pp 1-4.

[23] R. Jain, S. Iyengar, A. Arora, “Overview of Popular Graph Databases,” Published in: ICCCNT conference, 2013, pp 1-6.

[24] Cassandra, “Cassandra,” 2016, http://cassandra.apache.org/

[25] E. Hewitt, Cassandra: The Definitive Guide. O’Reilly Media, 2010.

[26] J. R. Lourenco, et al. “NoSQL in Practice: A Write-Heavy Enterprise Application,” Published in IEEE international congress on Big Data, 2015, pp 584-591.

[27] A. Chebotko, A. Kashlev, S. Lu, ”A Big Data Modeling Methodology for Apache Cassandra,” Published in IEEE international congress on Big Data, 2015, pp 238-245.

[28] R. P. Goldberg, “Survey of Virtual Machines Research,” Computer, 1974.

[29] D. Liu, L. Zhao, “The Research and Implementation of Cloud Computing Platform Based on Docker,” Published in ICCWAMTIP conference, 2014, pp 475-478.

[30] D. Bernstein, “Containers and Cloud: From LXC to Docker to Kubernetes,” Published in IEEE Cloud Computing (Volume: 1, Issue: 3), 2015, pp 81-84.

[31] Nitin Naik, “Migrating from Virtualization to Dockerization in the Cloud Simulation and Evaluation of distributed Systems,” Published in a conference of the Maintenance and Evolution of Service-Oriented and Cloud-based Environments, 2016, pp. 1-8.

[32] M. Raho, A. Spyridakis, M. Paolino, D. Raho, “KVM, Xen and Docker: A Performance Analysis for ARM based NFV and Cloud Computing,” Published in AIEEE conference, 2015, pp 1-8.

[33] J. Zhang, X. Lu, D.K. Panda, “Performance Characterization of Hypervisor -and Container-Based Virtualization for HPC or SR-IOV Enabled InfiniBand Clusters,” Published in Parallel and Distributed Processing Symposium Workshops, 2016, pp. 1777-1784.

[34] M. T. Chung, N. Q. Hung, M. T. N. Thoai, “Using Docker in high Performance Computing Applications,” Published in Communications and Electronics conference, 2016, pp. 52-57.

[35] J. P. Walters, et al. “GPU Passthrough Performance: A Comparison of KVM, Xen, VMware ESXi, and LXC for CUDA and OpenCL Applications,” Published in the 7th international conference on Cloud Computing, 2014, pp 636-643.

[36] N. Leavit, “Will NoSQL Database Live Up to Their Promise?” Published in IEEE Computer, 2010, pp 12-14.

[37] J. Sipke, B. waaij, R. Meijer, “Sensor Data Storage Performance: SQL or NoSQL, Physical or Virtual,” Published in CLOUD, 2012, pp 431-438.

[38] W. Felter, A. Ferreira, R. Rajamony, J. Rubio, “An Updated Performance Comparison of virtual Machines and Linux Containers,” Published in ISPASS conference, 2015, pp 171-172.

[39] G. Martins, P. Bezerra, R. Gomes, F. Albuquerque, “Evaluating Performance Degradation in NoSQL Databases Generated by Virtualization,” Published in LANOMS, 2015, pp 84-91.

[40] E. Dede, B. Sendir, P. Kuzlu, J. Hartog, M. Govindaraju, ”An Evaluation of Cassandra for Hadoop,” Published in CLOUD, 2013, pp. 494-501.

[41] J. Heo, R. Taheri, “Virtualization Latency-Sensitive Applications: Where Does the Overhead Come From?, ” Published in VMware Technical Journal, 2013, https://labs.vmware.com/vmtj/virtualizing-latency-sensitive-applications-where-does-the-overhead-come-from

10 Performance Comparison between Horizontal Scaling of Hypervisor and Container Based Virtualization using Cassandra NoSQL Database

Abstract

Cloud computing promises customers the on-demand ability to scale in face of workload variations. There are different ways to accomplish scaling, one is vertical scaling and the other is horizontal scaling. The vertical scaling refers to buying more power (CPU, RAM), buying a more expensive and robust server, which is less challenging to implement but exponentially expensive. While, the horizontal scaling refers to adding more servers with less processor and RAM, which is usually cheaper overall and can scale very well. The majority of cloud providers prefer the horizontal scaling approach, and for them would be very important to know about the advantages and disadvantages of both technologies from the perspective of the application performance at scale. In this paper, we compare performance differences caused by scaling of the different virtualization technologies in terms of CPU utilization, latency, and the number of transactions per second. The workload is Apache Cassandra, which is a leading NoSQL distributed database for Big Data platforms. Our results show that running multiple instances of the Cassandra database concurrently, affected the performance of read and write operations differently; for both VMware and Docker, the maximum number of read operations was reduced when we ran several instances concurrently, whereas the maximum number of write operations increased when we ran instances concurrently.

10.1 Introduction

Today's modern data centers are increasingly virtualized where applications are hosted on one or more virtual servers that are then mapped onto physical servers in the data center. Virtualization provides a number of benefits such as flexible allocation of resources and scaling of applications. Scalability corresponds to the ability of a system uniformly to handle an increasing amount of work [1] [2] [3]. Nowadays, there are two types of server virtualization technologies that are common in data center environments, hardware-level virtualization and operating system level virtualization. Hardware-level virtualization involves embedding virtual machine software (known as hypervisor or Virtual Machine Monitor (VMM)) into the hardware component of a server. The hypervisor controls processor, memory, and other components by allowing several different operating systems to run on the same machine without the need for a source code. The

operating system running on the machine will appear to have its own processor, memory, and other components. Virtual machines are extensively used in today's practice. However, during the last few years, much attention has been given to operating system level virtualization (also known as container-based virtualization or containerization). Operating system level virtualization refers to an operating system feature in which the kernel allows the existence of multiple isolated user-space instances (also known as partitions or containers) instead of just one. As has been shown in Figure 10.2, containers are more light weight than virtual machines, various applications in container share the same operating system kernel rather than launching multiple virtual machines with separate operating system instances. Therefore, container-based virtualization provides better scalability than the hypervisor-based virtualization [4].

Currently, two concepts are used to scale virtualized systems, vertical and horizontal scaling [5] [6] [7][8]. Vertical scaling corresponds to the improvement of the hardware on which application is running, for example addition of memory, processors, and disk space. While horizontal scaling corresponds to duplication of virtual servers to distribute the load of transactions. Horizontal scaling approach is almost always more desirable because of its advantages such as, no limit to hardware capacity, easy to upgrade, and easier to run fault-tolerance.

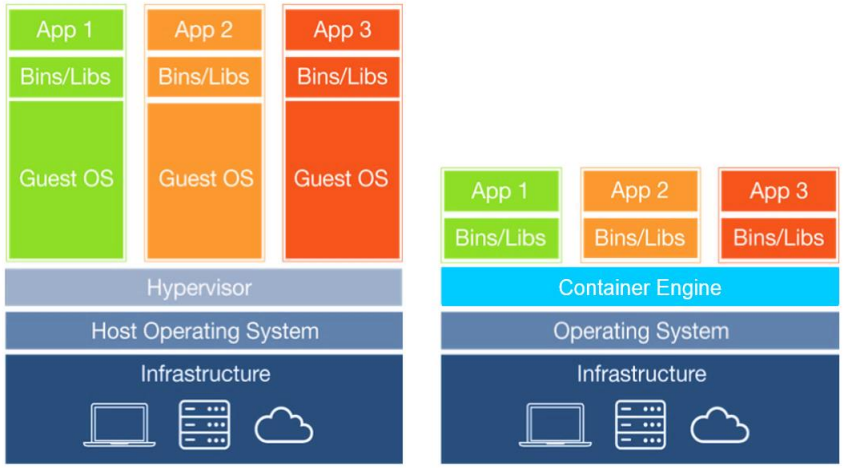


Figure 10.2. Different of Virtual Machines and Containers Architecture

In our previous study, we explored the performance of a real application, Cassandra NoSQL database, on the different environments. Our goal was to understand the overhead introduced by virtual machines (specifically VMware) and containers (specifically Docker) relative to non-virtualized Linux [9]. In this study, our goal is to provide an up-to-date comparison of containers and virtual machine environments using recent software versions. In addition, explore how much horizontal scaling of virtual machines and containers will improve the

performance in terms of the system CPU utilization, latency, and throughput. In this work, we have used multiple instances of the Cassandra running concurrently on the different environments.

The presented work is organized as follows: In Section 10.2 we discuss related work. Section 10.3 describes the experimental setup and test cases. Section 10.4 presents the experimental results and we conclude our work in Section 10.5.

10.2 Related Work

Both container-based and virtual machine-based virtualization technologies have been growing at a rapid space, and research work evaluating the performance aspects of these platforms provides an empirical basis for comparing their performance. Our previous research [9], has compared performance overheads of Docker containers, VMware virtual machines versus Non-virtualized. We have shown that, Docker had lower overhead compared to the VMware. In this paper, we try to expand our previous work and compare the two technologies; Container-based and Virtual Machine-based virtualization in terms of their scalabilities running Cassandra workload. There have not been many studies on both scalability and performance comparison between the two technologies. A comparison between Linux containers and AWS ec2 virtual machines is performed in [10]. According to their results, containers outperformed virtual machines in terms of both performance and scalability.

In [11], the authors evaluated the performance differences caused by the different virtualization technologies in data center environments where multiple applications are running on the same servers (multi-tenancy). According to theirs study, containers may suffer from performance in multi-tenant scenarios, due to the lack of isolation. However, containers offer near bare-metal performance and low footprint. In addition, containers allow soft resource limits which can be useful in resource over-utilization scenarios. In [12], the authors studied performance implications on the NoSQL MongoDB during the horizontal scaling of virtual machines. According to their results, horizontal scaling affects the average response time of the application by 40%.

10.3 Evaluation

The goal of the experiment was that of comparing the performance scalability of the Cassandra while running it on multiple virtual machines versus on multiple containers concurrently.

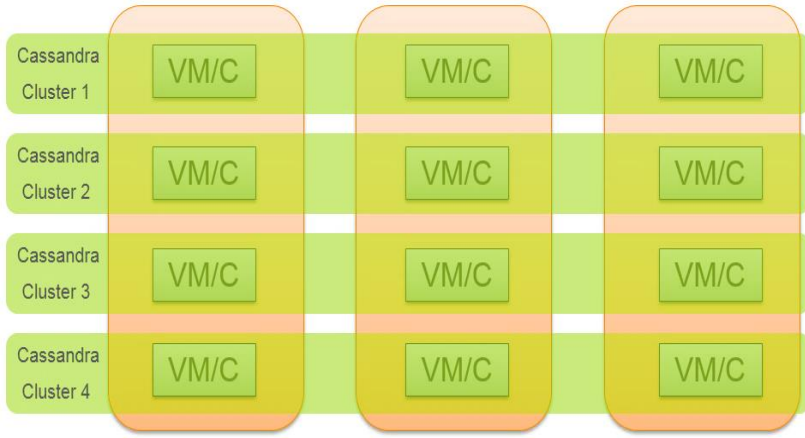


Figure 10.3. Experimental Setup

10.3.1 Experimental Setup

All our tests were performed on three HP servers DL380 G7 with processors for a total of 16 cores (plus HyperThreading) and 64 GiB of RAM and disk of size 400 GB. Red Hat Enterprise Linux Server 7.3 (Maipo) (Kernel Linux 3.10.0-514.e17.x86_64) and Cassandra 3.11.0 are installed on all hosts as well as virtual machines. Same version of Cassandra used on the load generators. To test containers, Docker version 1.12.6 installed and in case of virtual machines VMware ESXi 6.0.0 installed. In total 4 times 3-node Cassandra clusters configured for this study (see Figure 10.3).

10.3.2 Workload

To generate workload, we used Cassandra-stress tool. The Cassandra-stress tool is a Java-based stress utility for basic benchmarking and load testing of a Cassandra cluster. Creating the best data model requires significant load testing and multiple iterations. The Cassandra-stress tool helps us in this endeavor by populating our cluster and supporting stress testing of arbitrary CQL tables and arbitrary queries on tables. The Cassandra package comes with a command-line stress tool (Cassandra-stress tool) to generate load on the cluster of servers, the cqlsh utility, a python-based command line client for executing Cassandra Query Language (CQL) commands and the nodetool utility for managing a cluster. These tools are used to stress the servers from the client and manage the data in the servers.

The Cassandra-stress tool creates a keyspace called keyspace1 and within that, tables named standard1 or counter1 in each of the nodes. These are automatically created the first time we run the stress test and are reused on subsequent runs unless we drop the keyspace using CQL. A write operation inserts data into the database

and is done prior to the load testing of the database. Later, after the data are inserted into the database, we run the mix workload, and then split up the mix workload and run the write only workload and the read only workload. In [9] [1], we described in detail each workload as well as the commands we used for generating the workloads, in this paper we have used the same approach for generating the workload.

10.3.3 Performance Metrics

The performance of Docker containers and VMware virtual machines are measured using the following metrics:

- CPU Utilization (percentage),
- Maximum Transactions Per Second (TPS), and
- Mean Latency (milisecond).

The CPU utilization is measured directly on the server nodes by means of sar command. The latency and maximum transactions per second (TPS) are measured on the client side, that are measured by the stress test tool. The term transactions per second refers to the number of database transactions performed per second.

10.3.4 Test Cases

10.3.4.1 One-Cassandra-three-node-cluster

In this case, one virtual machine/container deployed on each host running Cassandra application. All virtual machines/containers configured as one 3-node cluster.

10.3.4.2 Two-Cassandra-three-node-clusters

In this case, two containers/virtual machines deployed on each host running Cassandra application. Each container/virtual machine on each host belongs to its own 3-node cluster, so in total two 3-node clusters configured to run concurrently.

10.3.4.3 Four-Cassandra-three-node-clusters

In this case, four containers/virtual machines deployed on each host running Cassandra application. Each container/virtual machine on each host belongs to its own 3-node cluster, so in total four 3-node clusters configured to run concurrently.

In this experiment, we compare the performance of virtual machines and containers running different Cassandra workload scenarios, Mix, Read and Write. However, unlike our previous study [9], here we decided to set the replication-factor as three. In our test environment with three-node clusters, replication factor three means that each node should have a copy of the input data splits.

10.4 Performance and Scalability Comparison

10.4.1 Transactions per second (tps)

Figure 10.3 shows transactions per second (tps) during write, read and mixed load. In this figure we summarized the total transactions per second from different number of Cassandra clusters running on Docker containers and VMware virtual machines. According to the results, overall in all cases Docker containers could handle higher number of database transactions per second than VMware virtual machines. In the case of the mixed load, Docker containers could handle around 25% more transactions per second than VMware virtual machines. In the case of only write load the difference is around 19% more for containers than virtual machines. While in the case of only read load, there is a huge difference of around 40% in the number of transactions per second between virtual machines and containers. Another aspect to consider according to the transactions per second results is that, running multiple instances of the Cassandra database concurrently, affected the performance of read and write operations differently; for both VMware and Docker, the maximum number of read operations was reduced when we ran several instances concurrently, whereas the maximum number of write operations increased when we ran instances concurrently. Note that increasing the number of Cassandra clusters did not have any significant impact on the number of transactions per second in the case of the mixed-load.

10.4.2 CPU utilization

Figure 10.4 shows the results of CPU utilization of multiple number of Cassandra clusters running on virtual machines and containers during write, read, and mix workloads. According to the results, in general CPU utilization of one cluster of virtual machines/containers are lower than two clusters and CPU utilization of two clusters is less than three clusters. It can be observed from the figures that, the overhead of running multiple clusters in terms of CPU utilization is around 10% for both containers and virtual machines. This overhead decrease as the load increases, one reason for this can be the background jobs that are running in Cassandra and as the load increases Cassandra by default delays these jobs since there are not enough resources available for executing the jobs. In addition, it can be observed from the figures that, the overall CPU utilization of containers is lower than virtual machines for all different workloads. Considering the mix workload CPU utilization of containers is around 15% lower than CPU utilization of virtual machines. The difference between CPU utilization of containers and virtual

machines is around 12% for the write workload which is very close to the difference that we saw for the mix workload case. However, this difference is significantly higher for the read workload up to around 40%. According to these results, read operations utilize more CPU cycles on virtual machines than on containers.

10.4.3 Latency

Figure 10.5 shows the results of latency mean of multiple number of Cassandra clusters running on virtual machines and containers during write, read, and mix workloads. As it can be observed from the figures, in general, the latency of containers is 50% lower than virtual machines as the load increases.

In the case of the mixed workload, the latency difference between having one cluster and two clusters is negligible. However, the latency difference between having one or two clusters compared with four clusters is around 33%. In the case of the write workload the difference between having containers.

However, for virtual machines, the latency becomes around 10ms in the case of 4 clusters when the tps is only 80k. Also, in the case of two clusters and 1cluster, since the cluster did not handle the load of 80k tps the latency is only shown for 40k tps which is around 2-3 ms. In the case of read workload, for the virtual machines the latency increases up to around 50% higher for the case with two clusters compared with one cluster. The latency increases up to around 20% for the case of four clusters compared with the case of two clusters and there is an increase of up to around 60% compared to the case of only one cluster. According to these results scaling would be very expensive for virtual machines in terms of latency mean which will have a negative impact on the application performance. However, in the case of containers the cost in terms of latency difference for having multiple clusters compared with one cluster is up to around 23%. According to the results, running multiple clusters inside containers will have less impact on the latency and the performance of the application (in this case Cassandra) than running multiple clusters inside virtual machines. The latency difference increases exponentially as the number of clusters increases as well as the load increases. The latency difference increases up to around 23% on containers and up to around 60% on virtual machines while having 100% read workload. The latency difference is negligible in the case of write workload. Also, there is a moderate latency difference in the case of mixed workload which is up to around 20% for virtual machines when the tps is 80k and up to around 25% for containers when the tps is 120k.

10.5 Discussions and Conclusions

In this study, we have compared the performance of running multiple clusters of the NoSQL Cassandra database inside Docker containers and VMware virtual

machines. We have measured the performance in terms of CPU utilization, Latency mean and the maximum number of Transactions Per Second (TPS). According to our results, running Cassandra inside multiple clusters of VMware virtual machines was showing less performance in terms of maximum number of transactions per second compared to the Docker containers. The performance difference was around 20% lower during the mixed workload, around 16% lower during the write-only workload and around 29% lower during read-only workload. One reason for this could be that containers are lighter-weight compared to virtual machines, therefore there is a less overhead of the virtualization layer and this helps the application to get more resources and performs better on containers than virtual machines. Another reason can be how a write and a read operation procedure works in Cassandra. In Cassandra, a write operation in general performs better than a read operation because it does not involve too much I/O. A write operation is completed when the data has been both written in the commit log (file) and in memory (memtable). However, a read operation may require more I/O for different reasons. A read operation first involves reading from a filter associated to sstable that might save I/O time saying that a data is surely not present in the associated sstable and then if filter returns a positive value, Cassandra starts seeking the sstable to look for data. In terms of CPU Utilization, the Cassandra application performs better on containers than on virtual machines. According to our results, the difference between CPU utilization on virtual machines is around 16% higher than containers during the mixed workload, around 8% higher during the write-only workload and around 32% higher during the read-only workload. In addition, the Cassandra application running inside virtual machines got up to around 50% higher latency than containers during the mixed workload. The difference became up to around 40% higher on virtual machines during the write-only workload compared to containers, also up to around 30% higher on virtual machines during the read-only workload compared to containers. As it has been discussed before in general the read-only workload is showing less performance than the write-only workload, and the impact of the different types of workloads on the performance in terms of CPU utilization is higher on virtual machines than containers.

However, considering the scalability aspects of the virtual machines and the containers, according to our results, containers scale better without loosing too much performance while virtual machines overhead is very high, and it has a negative impact on the performance of the application. This might differ depending on the application and the type of workload as we have seen during our experiments. Therefore, cloud providers need to investigate this issue while deploying both virtual machines and containers across data centers also at larger scale.

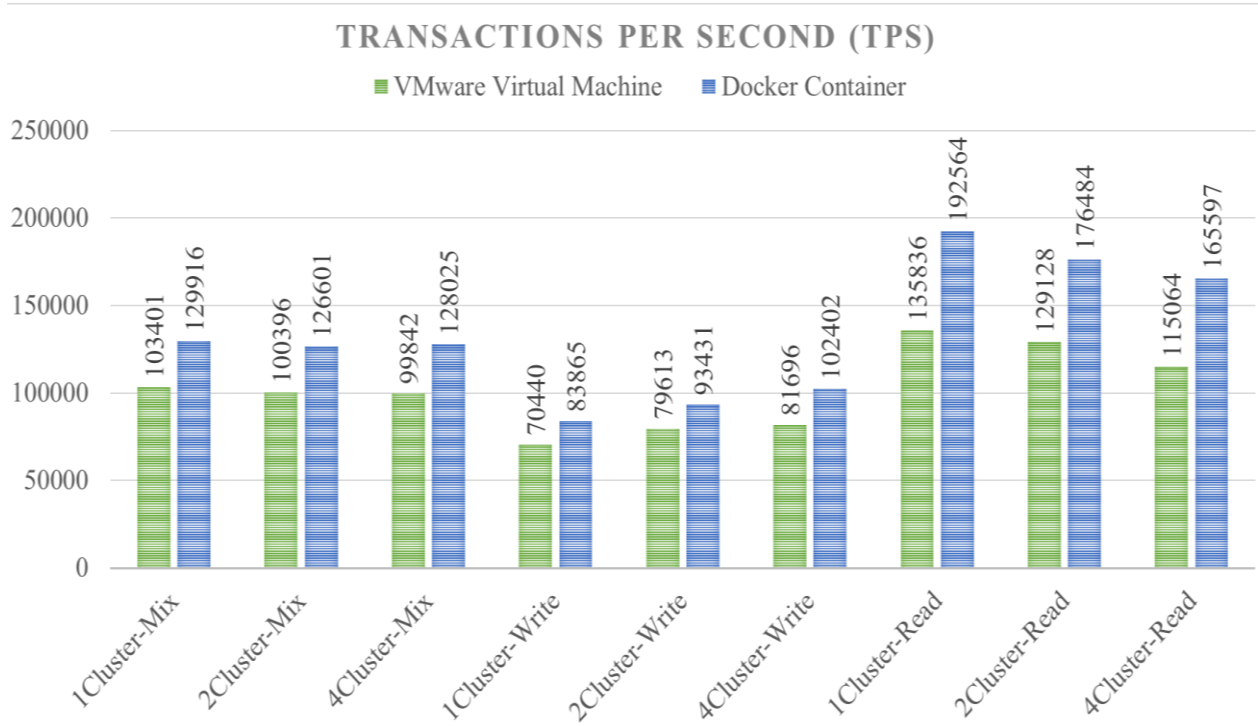
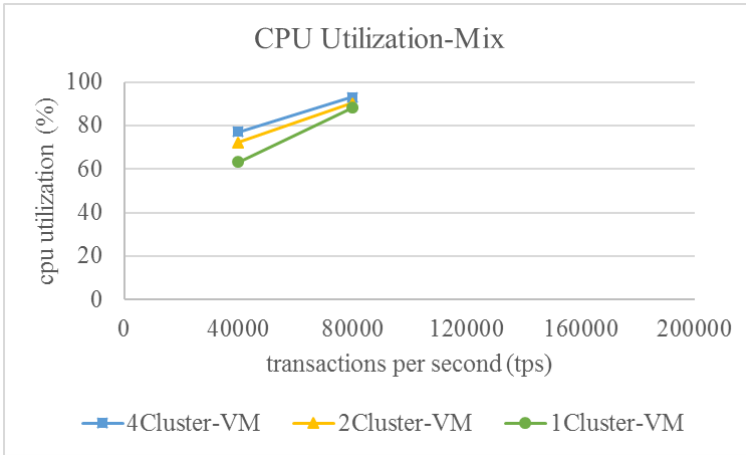
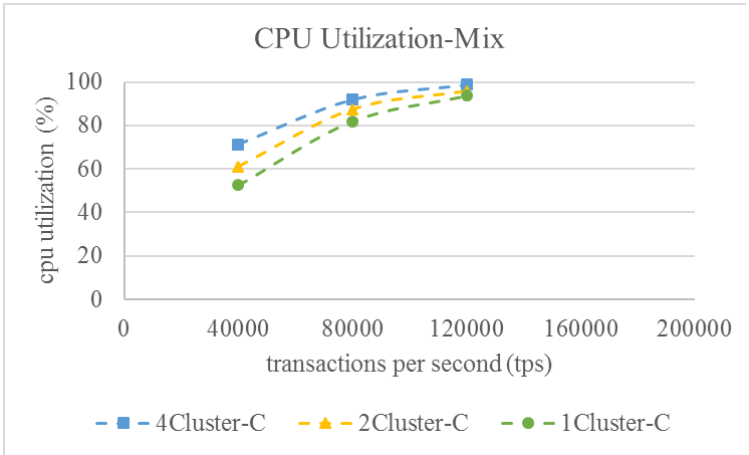


Figure 10.4. Transactions per second (tps)

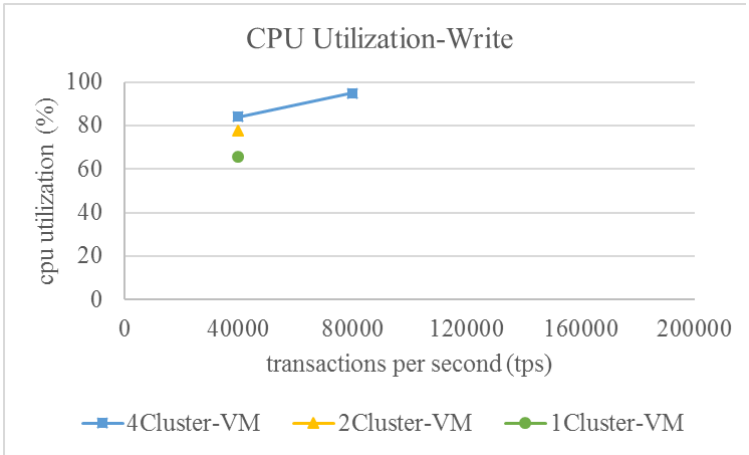


a.

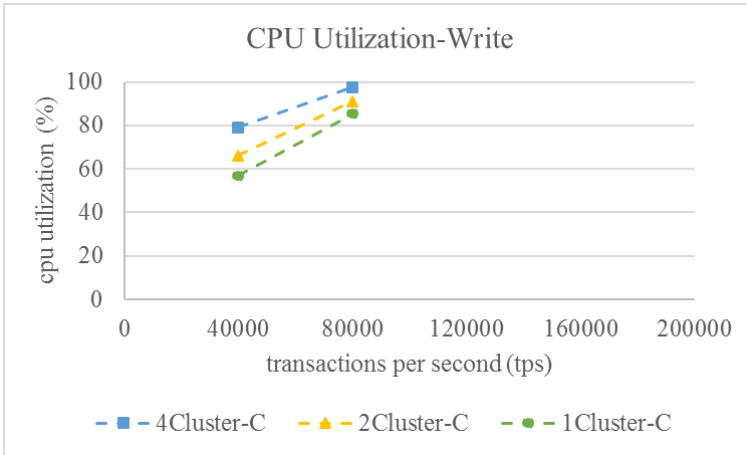


b.

Figure 10.4. CPU utilization results for Write, Read and Mix workload for multiple Cassandra clusters running on virtual machines and containers concurrently.

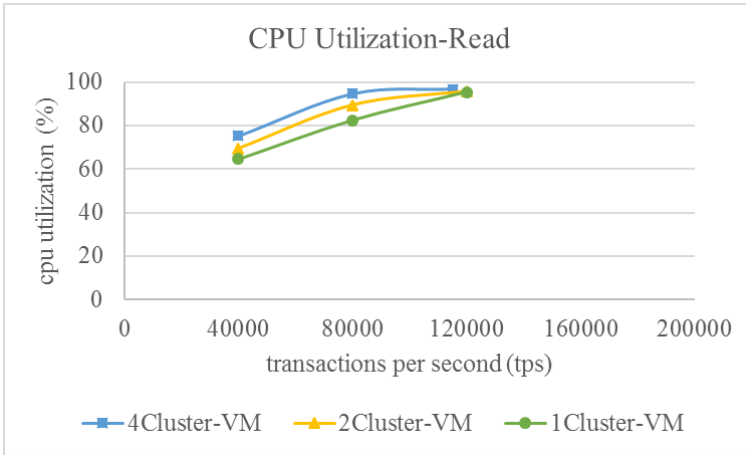


c.

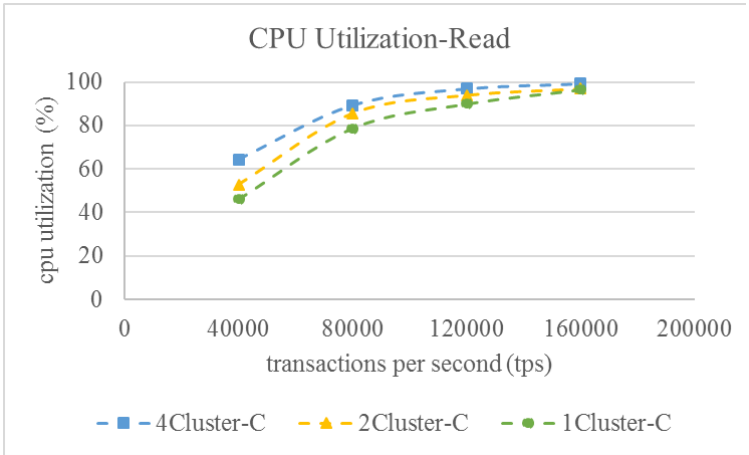


d.

Figure 10.4. CPU utilization results for Write, Read and Mix workload for multiple Cassandra clusters running on virtual machines and containers concurrently.

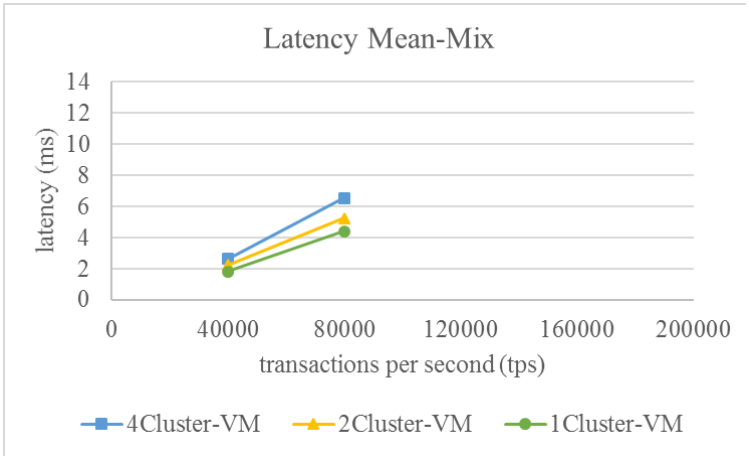


e.

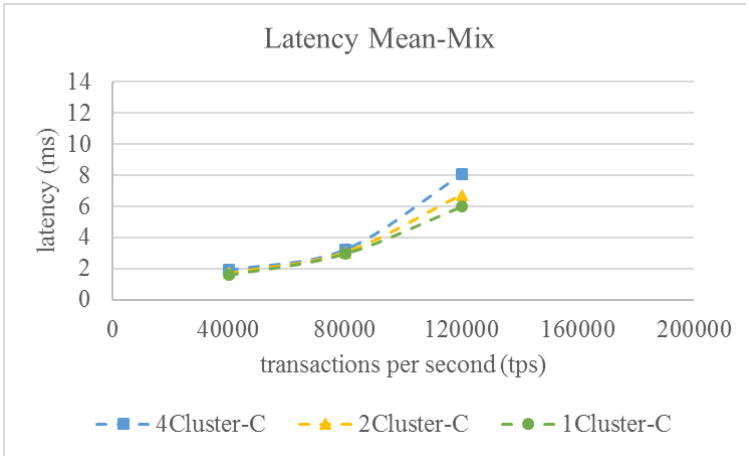


f.

Figure 10.4. CPU utilization results for Write, Read and Mix workload for multiple Cassandra clusters running on virtual machines and containers concurrently.

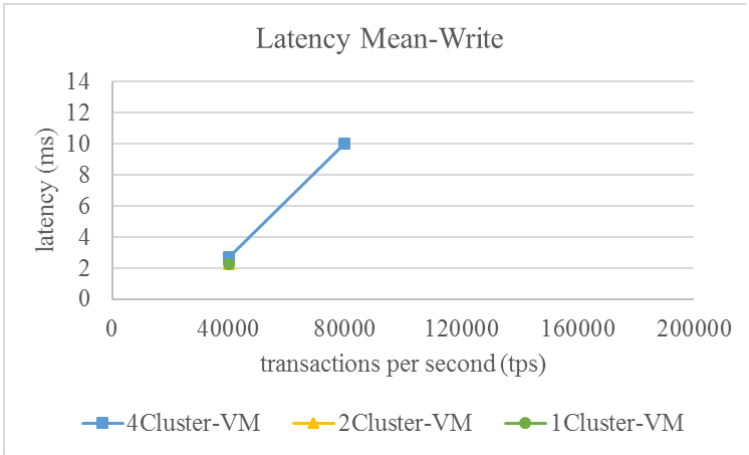


a.

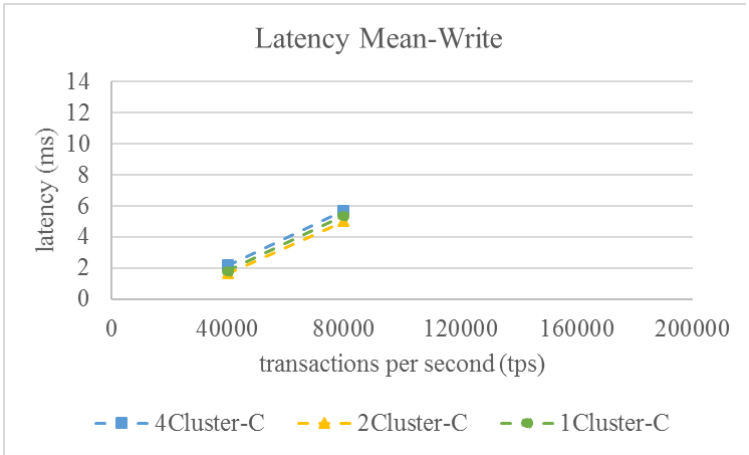


b.

Figure 10.5. Latency mean results for Write, Read and Mix workload for multiple Cassandra clusters running on virtual machines and containers concurrently

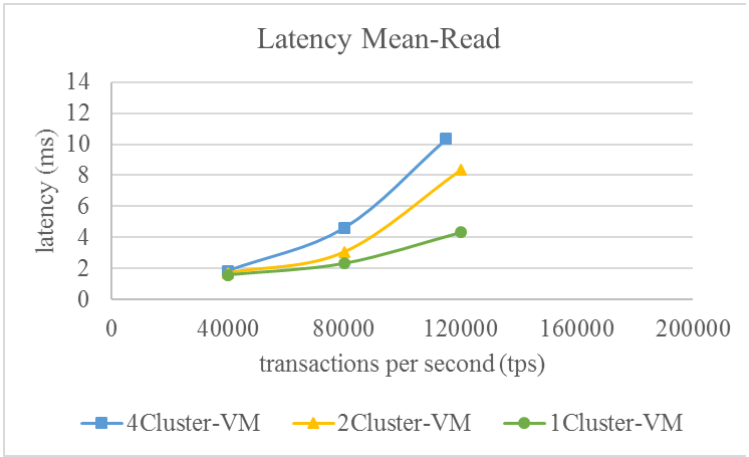


c.

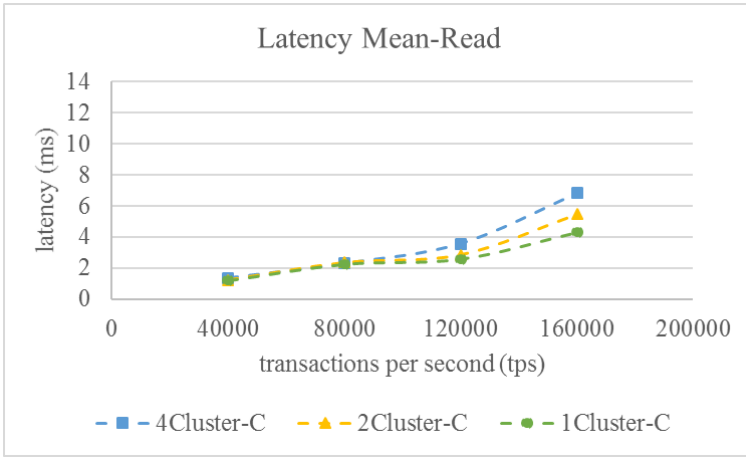


d.

Figure 10.5. Latency mean results for Write, Read and Mix workload for multiple Cassandra clusters running on virtual machines and containers concurrently



e.



f.

Figure 10.5. Latency mean results for Write, Read and Mix workload for multiple Cassandra clusters running on virtual machines and containers concurrently

10.6 References

[1] G. Huang et al., “Auto Scaling Virtual Machines for Web Applications with Queuing Theory,” in ICSAI conference, pp. 433-438, 2017.

[2] S. He et al., “Elastic Application Container: A Lightweight Approach for Cloud Resource Provisioning,” in AINA conference, pp. 15-22, 2012.

[3] A. Horiuchi, and K. Saisho, “Development of Scaling Mechanism for Distributed Web System,” in SNPD conference, pp. 1-6, 2015.

[4] F. Tseng et al., ”A Lightweight Auto-Scaling Mechanism for Fog Computing in Industrial Applications,” in IEEE Transactions on Industrial Informatics Journal, vol. PP, no. 99, pp. 1-1, 2018.

[5] W. Wenting, C. Haopeng, C. Xi, “An Availability-Aware virtual Machine Placement Approach for Dynamic Scaling of Cloud Applications,” in UIC/ATC conference, pp. 509-516, 2012.

[6] L. Chien-Yu, S. Meng-Ru, L. Yi-fang L. Yu-Chun, L. Kuan-Chou, ”Vertical/Horizontal Resource Scaling Mechanism for Federated Clouds,” in ICISA conference, pp.1-4 , 2014.

[7] S. Sotiriadis, N. Bessis, C. Amza, R. Buyya, “Vertical and Horizontal Elasticity for Dynamic Virtual Machine Reconfiguration,” in IEEE Transactions on Services Computing Journal, vol. PP, no. 99, pp. 1-14, 2016.

[8] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, P. Merie, ”Elasticity in Cloud Computing: State of the Art and Research Challenges,” in IEEE Transactions on Services Computing Journal, vol. PP, Issue. 99, pp 1-1, 2017.

[9] S. Shirinbab, L. Lundberg, E. Casalicchio, “Performance Evaluation of Container and Virtual Machine Running Cassandra Workload, ” in CloudTech conference, pp. 1-8, 2017.

[10] A.M. Joy, "Performance Comparison between Linux Containers and Virtual Machines," in ICACEA Conference, pp. 342-346, 2015.

[11] L. Chaufournier, P. Sharma, P. Shenoy, Y.C. Tay, "Containers and Virtual Machines at scale: A Comparative Study," in Middleware Conference, pp. 1-13, 2016.

[12] C. Huang et al., “The Improvement of Auto-Scaling Mechanism for distributed Database- A Case Study for MongoDB,” in APNOMS conference, pp. 1-3, 2013.

11 Scheduling Tasks with Hard Deadlines in Virtualized Software Systems

Abstract

There is scheduling on two levels in real-time applications executing in a virtualized environment: traditional real-time scheduling of the tasks in the real-time application, and scheduling of different Virtual Machines (VMs) on the hypervisor level. In this paper, we describe a technique for calculating a period and an execution time for a VM containing a real-time application with hard deadlines. This result makes it possible to apply existing real-time scheduling theory when scheduling VMs on the hypervisor level, thus making it possible to guarantee that the real-time tasks in a VM meet their deadlines. If overhead for switching from one VM to another is ignored, it turns out that (infinitely) short VM periods minimize the utilization that each VM needs to guarantee that all real-time tasks in that VM will meet their deadlines. Having infinitely short VM periods is clearly not realistic, and in order to provide more useful results we have considered a fixed overhead at the beginning of each execution of a VM. Considering this overhead, a set of real-time tasks, the speed of each processor core, and a certain processor utilization of the VM containing the real-time tasks, we present a simulation study and some performance bounds that make it possible to determine if it is possible to schedule the real-time tasks in the VM, and in that case for which periods of the VM that this is possible.

11.1 Introduction

Most real-time services were originally designed for physical (un-virtualized) computer systems. However, the trend towards virtualization pushes, for cost reasons, more and more systems onto virtualized machines, and at some point one would also like to run real-time systems with hard deadlines in a virtualized environment. Moving a real-time system with hard deadlines to a virtualized environment where a number of Virtual Machines (VMs) share the same physical computer is a challenging task. The original real-time application was designed so that all tasks were guaranteed to meet their deadlines provided that the physical computer was fast enough. In a system with faster processors, and more cores, one would like to put several VMs on the same hardware and some (or all) of these VMs may contain real-time tasks with hard deadlines. In such a system there will be scheduling at two levels [6]: traditional real-time scheduling of the tasks within a VM, and hypervisor controlled scheduling of several VMs on the same physical server. In [25] and [32] the authors refer to this technique as Component-based

design. This technique is also known as Hierarchical scheduling [21] [22] [31] [32] [34].

Traditional scheduling of tasks on a physical uni-processor computer is well understood, and a number of useful results exist [9], e.g., it is well known that Earliest Deadline First (EDF) is optimal when we allow dynamic task priorities. Similarly, it is well-known that Rate-Monotonic Scheduling (RMS) where tasks are assigned priorities based on their deadlines is optimal for the case when we use static task priorities. These priority scheduling algorithms are based on a number of parameters for each task τ_i. These parameters are typically, the period T_i the worst-case execution time C_i, and the deadline D_i, for task τ_i. Often, we assume that $D_i = T_i$, and in that case we only need two parameters for each task, namely, T_i and C_i. Priority assignment schemes such as EDF and RMS are typically used in the original real-time scheduling applications, i.e., in the applications that will be running in a VM.

If we ignore the overhead for context switching from one VM to another and if we use (infinitely) small time slots, we could let a VM get a certain percentage of the physical computer, e.g., two VMs where each VM uses every second time slot. This kind of situation could be seen as two VMs running in parallel with 50% of full speed each. In that case, the real-time application would meet all deadlines if the processor on the physical computer is (at least) two times as fast as the processor for which the original real-time application was designed for. However, the overhead for switching from one VM to another cannot be ignored and the time slot lengths for this kind of switching can obviously not be infinitely small. In order to minimize the overhead due to switching between VMs we would like to have relatively long time periods between switching from one VM to another VM. In order to share the physical hardware between as many VMs as possible we would also like to allocate a minimum percentage of the physical CPU to a VM, i.e., we would only like to allocate enough CPU resources to a VM so that we know that the real-time application that runs in that VM meets all its deadlines.

In order to use EDF, RMS or similar scheduling algorithms also on the hypervisor level, i.e., when scheduling the different VMs to the physical hardware, we need to calculate a period T_{VM} and a (worst-case) execution time C_{VM} for each VM that share a physical computer. It can be noted that also most real-time multiprocessor scheduling algorithms are based on the period and the worst-case execution time [8] [20]. This is important since most modern hardware platforms, i.e., most platforms on which the VMs will run, are multiprocessors.

VMs with one virtual processor will, for several reasons, be a very important case. Many existing real-time applications with hard deadlines have been developed for uniprocessor hardware. Moreover, even when using state-of-the-art multiprocessor real-time scheduling algorithms, one may miss deadlines for task

sets with processor utilization less than 40% [8]. For the uni-processor case it is well known that when using RMS we will always meet all deadlines as long as the processor utilization is less than $\ln(2) = 69.3\%$ [9]. This indicates that, compared to having a small number of VMs with many virtual cores each, it is better to use a larger number of VMs with one virtual core each on a multicore processor (we will discuss this in Section 10.2). We will present our results in the context of VMs with one virtual core. However, the results could easily be extended to VMs with multiple virtual cores as long as each real time task is allocated to a core (we will discuss this in Section 11.9). Systems that use global multiprocessor scheduling of real-time tasks, i.e., systems that that allow tasks to migrate freely between processors, are not considered here.

In this paper we will, based on an existing real-time application and the processor speed of the physical hardware, calculate a period T_{VM} and an execution time C_{VM} such that the existing real-time application will meet all deadlines when it is executed in a VM, provided that the VM executes (at least) C_{VM} time unites every period of length T_{VM}. We will show, and it is also well known from previous studies, that if overhead for switching from one VM to another is ignored, it turns out that (infinitely) short VM periods minimizes the utilization that each VM needs to guarantee that all real-time tasks in that VM will meet their deadlines. Having infinitely short VM periods is clearly not realistic, and in order to provide more useful results we consider a fixed overhead at the beginning of each execution of a VM. Considering this overhead, a set of real-time tasks, the speed of the each processor core, and a certain processor utilization of the VM containing the real-time tasks, we present a simulation study and some performance bounds that make it possible to determine if it is possible to schedule the real-time tasks in the VM, and in that case for which periods of the VM that this is possible. We will base our calculations on the case when we use static priorities, and thus RMS, in the original real-time applications. However, we expect that our approach can easily be generalized to cases when other scheduling policies, such as EDF, are used in the original real-time applications (we will discuss this in Section 11.2).

11.2 Related Work

Today, most physical servers will contain multiple processor cores. Modern virtualization systems, such as KVM, VMware and Xen, make it possible to define VMs with a number of (virtual) cores, thus allowing parallel execution within a VM. This means that one can use the physical hardware in different ways: one can have a large number of VMs with one (virtual) core each on a physical (multi-core) server, or a smaller number of VMs with multiple (virtual) cores each (or a combination of these two alternatives). It is also possible to make different design decisions in the time domain, e.g., allowing a VM with one virtual core to execute for relatively long time periods, or restricting a VM with multiple cores to relatively short execution periods. Real-time scheduling theory (for non-virtualized systems)

shows that the minimum processor utilization for which a real-time system can miss a deadline, using fixed priority scheduling, decreases as the number of processors increases, e.g., 69.3% for one processor systems [7] (using RMS) and 53.2% for two processor systems [8] and then down to as little as 37.5% for systems with (infinitely) many processors [8].

Consequently, compared to multiprocessor systems, the processor utilization is in general higher for systems with one processor. This is one reason why we have assumed that the VM containing the original real-time application only has one (virtual) processor. Also, most existing real-time applications are developed for systems with one processor.

In this paper we have assumed that the real-time application in the VM uses RMS. If we assume some other scheduling policy, e.g., EDF we can use the same technique. The only difference is that the formula $R_i = C_i + \sum_{j=1}^{i-1} \lceil R_i/T_j \rceil C_j$ (see Section 11.3), needs to be replaced with the corresponding analysis for EDF.

Very little has been done in the area of scheduling real-time tasks with hard deadlines in virtualized systems. Some results on real-time tasks with soft deadlines exist [1] [16].

There are a number of results concerning so called proportional-share schedulers [2] [3] [4] [18]. These results look at a real-time application that runs inside an operating system process. The proportional-share schedulers aim at dividing the processor resource in predefined proportions to different processes.

In [10] the authors look at a model for deciding which real-time tasks to discard when the cloud system's resources cannot satisfy the needs of all tasks. This model does, however, not address the problems associated with hard deadlines.

In [11] the authors ran an experiment using a real-time e-learning distributed application for the purpose of validating the IRMOS approach. The IRMOS uses a variation of the Constant Bandwidth Server (CBS) algorithm based on EDF. Furthermore in [17] the authors developed their particular strategy in the context of IRMOS project. They tried to consider isolation and CPU scheduling effects on I/O performance. However, in IRMOS they do not consider hard real-time tasks scheduled using the RMS.

Reservation-based schedulers are used as hypervisor level schedulers. In [12] and [19] the authors used CPU reservation algorithm called, Constant Bandwidth Server (CBS) in order to prove that the real time performance of the VMs running on the hypervisor is affected by both the scheduling algorithm (CBS) and VM technology (in this case KVM). However, the authors do not present a method for how to schedule different VMs running on the hypervisor.

In [13] the authors presented two algorithms for real-time scheduling. One is the hypervisor scheduling algorithm and the other is the processor selection algorithm. However they only consider scheduling VMs on the hypervisor level, they do not investigate scheduling of the hard real-time tasks that run inside the VMs.

Eucalyptus is open-source software for building private and hybrid clouds. There are several algorithms already available in Eucalyptus for scheduling VMs with some advantages and disadvantages. In [14] the authors proposed a new algorithm for scheduling VMs based on their priority value, which varies dynamically based on their load factors. However they consider dynamic priority based scheduling not static priority.

In [15], a priority based algorithm for scheduling VMs is proposed. The scheduler is first distinguishing the best matches between VMs and empty places and then deploying the VMs onto the corresponding hosts. The authors did a comparison between their priority algorithm and First Come First Serve (FCFS) algorithm, they concluded that the resource performance of their algorithm is not higher than the FCFS algorithm all the time but it has higher average resource performance. Nevertheless, they do not consider periodic tasks and static priority assignment.

The VSched system, which runs on top of Linux, provides soft real-time scheduling of VMs on physical servers [5]. However, the problems with hard deadlines are not addressed in that system.

In [21], the authors proposed a hierarchical bounded-delay resource model that constructs multiple levels resource partitioning. Their approach is designed for the open system environment. Their bounded-delay resource partition model can be used for specifying the real-time guarantees supplied from a parent model to a child model where they have different schedulers, while in [22] and [32], the authors proposed a resource model that can provide a compositional manner such that if the parent scheduling model is schedulable, if and only, its child scheduling models are schedulable. However, none of the proposed resource models consider scheduling in virtualized environment.

In [23] and [31], the authors presented a methodology for computing exact schedulability parameters for two-level framework while in [24] they did an analysis on systems where the fixed priority pre-emptive scheduling policy is used on both level. Further in [26], the authors presented a method for analysis of platform overheads in real-time systems. Similar work by [33] represents that their proposed approach can reduce pre-emption and overhead by modifying the period and execution time of the tasks.

In [25], the authors developed compositional real-time scheduling framework based on the periodic interface, they have also evaluated the overheads that this periodic interface incur in terms of utilization increase. Later in [41], the authors proposed an approach to eliminate abstraction overhead in composition. In their latest study the authors have improved their previous works and proposed a new technique for the cache-related overhead analysis [40].

In [27], the authors implemented and evaluated a scheduling framework that built on Xen virtualization platform. Another similar work has been done by [29]; they represent an implementation of compositional scheduling framework for virtualization using the L4/Fiasco micro kernel which has different system architecture compared to Xen. The authors calculated clock cycle overhead for the L4/Fiasco micro kernel. In [28], the authors proposed and compare the results of overhead of an external scheduler framework called ExSched that is designed for real time systems. In [30], the authors presented and compared several measurements of overheads that their implemented hierarchical scheduling framework imposes through its implementation over VxWorks.

Compositional analysis framework based on the explicit deadline periodic resource model has been proposed by [38]. They have used EDF and Deadline Monotonic (DM) scheduling algorithm and their model supports sporadic tasks. In [39], the authors present the RM schedulability bound in a periodic real time system which is an improvement to the earlier bound that has been proposed by [7]. However, none of these works consider the overhead in their models.

11.3 Problem Definition

We consider a real-time application consisting of n tasks. Task τ_i $(1 \leq i \leq n)$ has a worst-case execution time C_i $(1 \leq i \leq n)$, and a period T_i $(1 \leq i \leq n)$. This means that task τ_i generates a job at each integer multiple of Ti and each such job has an execution requirement of C_i time units that must be completed by the next integer multiple of T_i. We assume that each task is independent and does not interact (e.g., synchronize or share data) with other tasks. We also assume that the first invocation of a task is unrelated to the first invocation of any other task, i.e., we make no assumptions regarding the phasing of tasks with equal or harmonic periods. We assume that the deadline Di is equal to the period, i.e., $D_i = T_i$ $(1 \leq i \leq n)$. The tasks are executed using static task priorities, and we use RMS scheduling, which means that the priority is inversely proportional to the period of the task (i.e., tasks with short periods get high priority). This static priority assignment scheme is optimal for the uni-processor case [9].

The real-time application is executed by a VM with one virtual processor. The real-time tasks may miss their deadlines if the VM containing the tasks is not scheduled for execution by the hypervisor during a certain period of time. For

instance, if some period that the VM is not running exceeds some T_i, it is clear that the corresponding task τ_i will miss a deadline. Also, if the VM gets a too low portion of a physical processor, the tasks may also miss their deadlines since there will not be enough processor time to complete the execution time before the next deadline.

In a traditional real-time application, a task τ_i will voluntarily release the processor when it has finished its execution in a period, and C_i denotes the maximum time it may execute before it releases the processor. In the case with real-time scheduling of VMs on the hypervisor level it is more natural to assume that the hypervisor preempts VM_j and puts VM_j in the blocked state when it has executed for C_{VM_j} time units in a period. The hypervisor then moves VM_j to the ready state at the start of the next period. As mentioned before, the length of the period for VM_j is T_{VM_j}.

On the hypervisor level one may use any scheduling policy as long as one can guarantee that each VM is executed C_{VM}, during each period T_{VM}. On multicore processors one could for instance bind each VM to a core and let the VMs that share the same core share it using RMS, or one could let the VMs share a global ready queue, i.e., a VM could be executed on different cores during different time periods.

11.4 Defining Tvm And Cvm

Without loss of generality, we order the tasks τ_i $(1 \leq i \leq n)$ such that $T_i \leq T_{i+1}$. This means that τ_1 has the highest priority and τ_n has the lowest priority using RMS. Let R_i denote the worst-case response time for taskτ_i. From previous results we know that

$$R_i = C_i + \sum_{j=1}^{i-1} \lceil R_i/T_j \rceil C_j \qquad (1)$$

on a physical uni-processor server (or when the VM has uninterrupted access to a physical processor). In order to obtain R_i from Equation (1) one needs to use iterative numeric methods [9]. In order to meet all deadlines, we must make sure that $R_i \leq T_i$ $(1 \leq i \leq n)$.

Consider a time period of length t, which may extend over several periods T_{VM}. The scenario with minimum execution of the VM during period t, starts with a period of $2(T_{VM} - C_{VM})$ with no execution (see Figure 11.1) [37][25], i.e., the period starts exactly when the VM has executed C_{VM} time units as early as possible in one of its periods. Following this line of discussion, it is also clear that for the worst-case scenario $\lfloor (t - 2(T_{VM} - C_{VM}))/T_{VM} \rfloor$ is the number of whole periods of length T_{VM} (each containing a total execution of C_{VM}) that is covered by t.

Let t' denote the minimum amount of time that the VM is running during a time period of length t. From Figure 10.1 we get the minimum t' as:

$$t' = \left\lfloor \frac{(t-2(T_{VM}-C_{VM}))}{T_{VM}} \right\rfloor C_{VM} + \min\left(\left(t - 2(T_{VM} - C_{VM}) - \left\lfloor \frac{(t-2(T_{VM}-C_{VM}))}{T_{VM}} \right\rfloor T_{VM} \right), C_{VM} \right) \quad (2)$$

In Equation (2), the first term $(\lfloor (t - 2(T_{VM} - C_{VM}))/T_{VM} \rfloor C_{VM})$ corresponds to the full periods, and the last term to the remaining part. The term $t - 2(T_{VM} - C_{VM}) - \lfloor (t - 2(T_{VM} - C_{VM}))/T_{VM} \rfloor T_{VM}$ is the time that the VM has access to a physical processor during the part of t that exceeds the full periods.

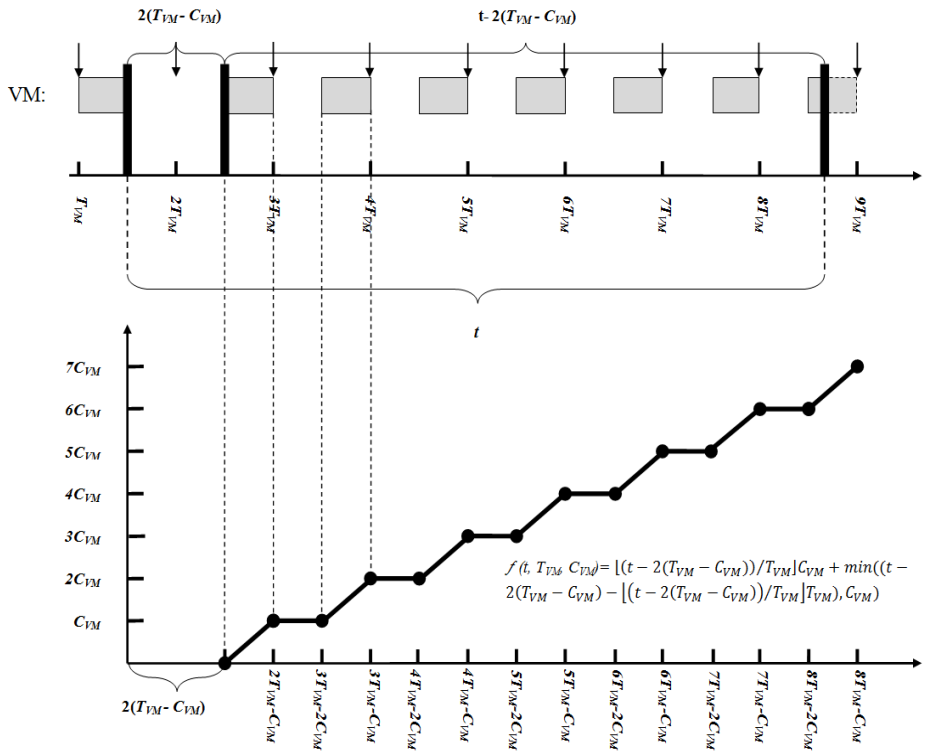


Figure 11.1 : Worst-case scenario when scheduling a VM with period T_{VM} and (worst-case) execution time C_{VM}.

The minimum comes from the fact that time that the VM has access to a physical processor during the time interval that exceed the full periods cannot be more than C_{VM}. This means that t' is a function of three parameters, i.e., $t' = f(t, T_{VM}, C_{VM})$. For fixed T_{VM} and C_{VM}, $t' = f(t, T_{VM}, C_{VM})$ is a continuous increasing function in t, consisting of straight line segments from $((2(T_{VM} -$

$C_{VM}) + nT_{VM}), nC_{VM})$ to $((2(T_{VM} - C_{VM}) + (n+1)T_{VM}), (nC_{VM} + T_{VM}))$ for any $n = 0, 1, 2, \ldots$ and horizontal lines connecting them.

Figure 11.1 displays a general piece of the curve, and the points $P_n = ((2(T_{VM} - C_{VM}) + nT_{VM}), C_{VM})$ are the lower corners in the graph.

We now define the inverse function

$$t = f^{-1}(f(t, T_{VM}, C_{VM}), T_{VM}, C_{VM}) \qquad (3)$$

By looking at Figure 10.2 we see that

$$f^{-1}(t, T_{VM}, C_{VM}) = 2(T_{VM} - C_{VM}) + t + \lfloor t/C_{VM} \rfloor (T_{VM} - C_{VM}) \qquad (4)$$

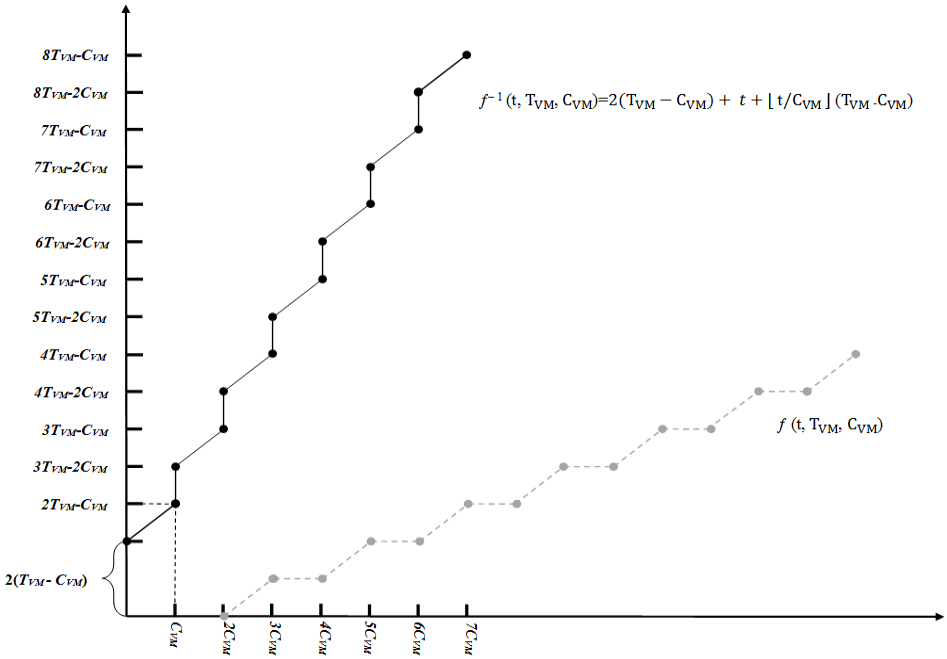


Figure 11.2 : The function $\boldsymbol{f^{-1}(t, T_{VM}, C_{VM})}$; $\boldsymbol{t}$ *is the parameter on the x-axis in the graph.*

From previous results on R_i (see [9] and above), and from the definition of f^{-1} we get that the worst-case response time for task τ_i is

$$R_i = f^{-1}\left(\left(C_i + \sum_{j=1}^{i-1} \lceil R_i/T_j \rceil C_j\right), T_{VM}, C_{VM}\right) \qquad (5)$$

For example if we have two tasks and T1 = 8, C1 = 1, and T2 = 15, C2 = 3, and $T_{VM} = 6$ and $C_{VM} = 3$ we get $7 = R_1 = f^{-1}(1, 6, 3)$ and $14 = R_2 = f^{-1}\big((3 + \lceil 14/8 \rceil * 1), 6, 3\big)$.

In order to solve Equation (5), one needs to use numeric and iterative methods, i.e., a very similar approach as the well-known method used for obtaining R_i in the non-virtualized case [9] (this approach can easily be implemented in a program that calculates the R_i values). In order to meet all deadlines for all tasks τ_i, we need to select T_{VM} and C_{VM} so that Equation (5) $\leq T_i (1 \leq i \leq n)$.

11.5 Example

Consider the following small real-time application with three tasks.

Table 11.1: Example of a small real-time application with three tasks.

Task	Period (T_i)	Worst-case execution time (C_i)	Utilization (U_i)
τ_1	16	2	2/16 = 0.125
τ_2	24	1	1/24 = 0.042
τ_3	36	4	4/36 = 0.111
			$\sum$ = 0.278

As discussed above, we use fixed priorities and RMS priority assignment. If we let the VM that executes this application use 40% of a CPU resource, i.e., if $C_{VM}/T_{VM} = 0.4$, we can use Equation (4) to calculate the maximum T_{VM} so that all three tasks will meet their deadlines. When $C_{VM}/T_{VM} = 0.4$ we can replace C_{VM} with $0.4T_{VM}$ in Equation (4), thus obtaining the function $f^{-1}(t, T_{VM}) = 1.2T_{VM} + t + \lfloor t/0.4T_{VM} \rfloor (0.6T_{VM})$.

We start by looking at τ_1. We need to find the maximal T_{VM} so that $R_1 = f^{-1}\left(\left(C_1 + \sum_{j=1}^{0} \lceil R_1/T_j \rceil C_j\right), T_{VM}\right) = f^{-1}(C_1, T_{VM}) = f^{-1}(2, T_{VM}) \leq T_1 = 16$. In general, f^{-1} is solved using a numeric and iterative approach in a similar way as R_i is obtained in the non-virtualized case [9]. However, we will see that for this τ_1 the $\lfloor t/C_{VM} \rfloor (T_{VM} - C_{VM})$ part of f^{-1} can be ignored. In that case, we get the following equation for the maximum T_{VM}: $1.2T_{VM} + 2 = 16$, and from this we get $T_{VM} = 14/1.2 = 11.7$. If we have a period of 11.7 we get a $C_{VM} = 0.4 \times 11.7 = 4.68$, and (as predicted above) since $C_{VM} > C_1$, we know that we do not have to consider the $\lfloor t/C_{VM} \rfloor (T_{VM} - C_{VM})$ part of f^{-1}.

We now look at τ_2. We want to find the maximal T_{VM} so that $R_2 = f^{-1}\left(\left(C_2 + \sum_{j=1}^{1} \lceil R_2/T_j \rceil C_j\right), T_{VM}\right) \leq T_2 = 24$. It is clear that τ_2 will miss its deadline with $T_{VM} = 14/1.2 = 11.7$ (which is the maximal T_{VM} period for which

τ_1 will meet its deadlines); if we use $T_{VM} = 14/1.2 = 11.7$, the first execution period will (in the worst-case, see Figure 11.1) start at time $2(T_{VM} - C_{VM}) = 1.2T_{VM} = 14$. Since $T_1 = 16$ and $C_1 = 2$ we see that τ_1 will execute two times back-to-back in this interval, i.e., after the first execution of τ_1 it will be released again at time 16. Consequently, τ_2 cannot start executing until time 18, and the first execution period of the VM will end at $2T_{VM} - C_{VM}$ (see Figure 10.1) $= 1.6T_{VM} = 1.6 \times 11.7 = 18.7$, and since $C_1 = 1, \tau_2$ cannot complete during the first execution period of the VM. The second period of the VM starts at time $3T_{VM} - 2C_{VM}$ (see Figure 11.1) $= 2.2T_{VM} = 2.2 \times 11.7 = 25.7$, which is after the deadline of $\tau_2 (T_2 = 24)$.

By using our formulas, we see that in order for τ_2 to meet its deadlines T_{VM} cannot be larger than $13/1.2 = 10.8$. This means that we now know that the real-time application can at most have $T_{VM} = 10.8$ when $C_{VM}/T_{VM} = 0.4$. For $T_{VM} = 10.8$ and $C_{VM}/T_{VM} = 0.4$, the corresponding C_{VM} is $0.4 \times 10.8 = 4.33$.

We finally look at τ_3. We need to find the maximum T_{VM} so that $R_3 = f^{-1}\left(\left(C_3 + \sum_{j=1}^{2}\lceil R_3/T_j \rceil C_j\right), T_{VM}\right) \leq T_3 = 36$. In this case we see that τ_3 will not meet its deadline when $T_{VM} = 13/1.2 = 10.8$. The reason for this is that both τ_1 and τ_2 will cause interference on τ_3, and τ_3 will as a consequence of this not complete in the first T_{VM} cycle, since $C_1 + C_2 + C_3 = 2 + 1 + 4 = 7 > 4.33$. The second T_{VM} cycle will complete at time $3T_{VM} - C_{VM}$(see Figure 10.1) $= 3 \times 10.8 - 4.33 = 28.07$. Before the end of this cycle both τ_1 and τ_2 will have had one new release each (τ_1 at time 16 and τ_2 at time 24). This means that τ_3 will not complete during the second cycle of T_{VM} since$2C_1 + 2C_2 + C_3 = 4 + 2 + 4 = 10 > 2 \times 4.33 = 8.66$.

In the worst-case scenario (see Figure 11.1), the third cycle of T_{VM} will start at time $4T_{VM} - 2C_{VM} = 4 \times 10.8 - 2 \times 4.33 = 34.54$. At time 32 there is a new release of task τ_1, and since τ_1 has higher priority than τ_3, task τ_1 will execute for two time units starting at time 34.54.

Since $T_3 = 36$, we see that τ_3 will miss its deadline. This means that we need a shorter period T_{VM} in order to guarantee that also τ_3 will meet its deadlines.

When using our formulas, we see that $T_{VM} = 10$ is the maximal period that τ_3 can tolerate in order to meet its deadline when $C_{VM}/T_{VM} = 0.4$, i.e., for

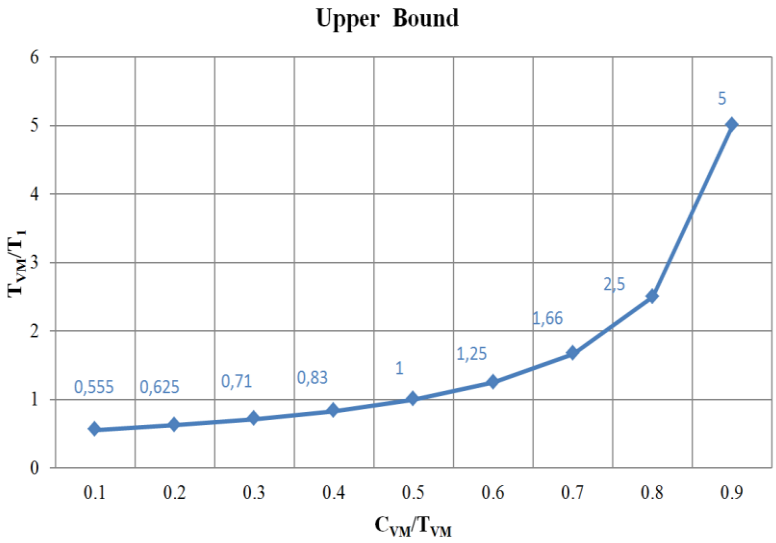


Figure 11.2 : The upper bound for $\boldsymbol{T_{VM}}$ */ T1*

$C_{VM}/T_{VM} = 0.4$ we get $T_{VM} = 10$, and τ_3 is the task that requires the shortest period T_{VM}.When $C_{VM}/T_{VM} = 0.5$ we can use our formulas to calculate a T_{VM}. In this case we get a maximal T_{VM} of 14 for task τ_1, and the calculations for tasks τ_2 and τ_3 will result in larger values on the maximal T_{VM}.

This means that τ_1 is the task that requires the shortest period T_{VM}, i.e., $T_{VM} = 14$ when $C_{VM}/T_{VM} = 0.5$. In general, the period T_{VM} will increase when the utilization C_{VM}/T_{VM} increases, and the task that is "critical" may change when C_{VM}/T_{VM} changes (e.g., task τ_3 when $C_{VM}/T_{VM} = 0.4$ and task $\tau_1$when $C_{VM}/T_{VM} = 0.5$).

11.6 Simulation Study

In Section 11.5 we saw that the maximal T_{VM}, for which a task set inside the VM is schedulable increases when C_{VM}/T_{VM} increases. In this section we will quantify the relation between the maximal $T_{VM}$the utilization C_{VM}/T_{VM}.

We will do a simulation study where we consider two parameters:

- n – the number of tasks in the real-time application
- u – the total utilization of the real-time application

The periods T_I are taken from a rectangular distribution between 1000 and 10000. The worst-case execution time C_I for task τ_I is initially taken from a rectangular distribution between 1000 and T_i.

All worst-case execution times are then scaled by a factor so that we get a total utilization u. For each task, we then find the maximum T_{VM} using Equation (5) so that all tasks meet their deadlines. We refer to this period as T_{max}, and we then select the minimum of the n different T_{max} values (one value for each task). For each pair of n and u, we calculate the minimum T_{max} for $(C_{VM}/T_{VM} = 0.9)$, $(C_{VM}/T_{VM} = 0.8)$, $(C_{VM}/T_{VM} = 0.7)$, $(C_{VM}/T_{VM} = 0.6)$, $(C_{VM}/T_{VM} = 0.5)$, $(C_{VM}/T_{VM} = 0.4)$, $(C_{VM}/T_{VM} = 0.3)$, $(C_{VM}/T_{VM} = 0.2)$, $(C_{VM}/T_{VM} = 0.1)$ for 10 randomly generated programs (we generate the programs using the distribution and the technique described above). We look at n = 10, 20, and 30, and u = 0.1, 0.2 and 0.3. This means that we look at 3*3 = 9 combinations of n and u, and for each of these nine combinations we look at 9 different values on C_{VM}/T_{VM}. This means that we looked at 3*3*9 = 81 different scenarios. For each such scenario we generated 10 programs using a random number generator.

11.7 Results

In the worst-case scenario (see Figure 11.1), the maximum time that a virtual machine may wait before its first execution is $2(T_{VM} - C_{VM})$. In order for the real-time tasks not to miss their deadlines the maximum waiting time, $2(T_{VM} - C_{VM})$ must be less than the shortest period, T1 (i.e., $2(T_{VM} - C_{VM}) < T_1$). For each value of C_{VM}/T_{VM}, we can replace C_{VM} and calculate the T_{VM}/T_1 using the formula above. For example, if $C_{VM}/T_{VM} = 0.7$ then we can replace C_{VM} by $0.7T_{VM}$ in $2(T_{VM} - C_{VM}) < T_1$ so we have $2(T_{VM} - 0.7T_{VM}) < T_1$, from that we get $0.6T_{VM} < T_1$, this means that $T_{VM}/T_1 < 1/0.6 = 1.66$. By continuing this we can calculate the values of T_{VM}/T_1 for each value of C_{VM}/T_{VM}. The corresponding graph is presented in Figure 11.2. These values are clearly the upper bound for all the values of T_{VM}/T_1, and our simulation study shows that, for all combinations of u, n, and C_{VM}/T_{VM}, T_{VM}/T_1 is less than this upper bound.

11.7.1 Total Utilization of 0.1

In Figure 11.4, we see that the standard deviation is very small for u = 0.1. For n = 10 we get values around 0 and for n = 20, we get 0.006 and for n = 30, we get 0.009.

As shown in Figure 11., for different number of the tasks (n = 10, 20 and 30) T_{VM} / T1 increases when C_{VM}/T_{VM} increases. The first observation is thus that the maximum T_{VM} for which the task set inside the VM is schedulable increases when the VM gets a larger share of the physical processor, i.e., when C_{VM}/T_{VM} increases.

The second observation is that the curves in Figure 11. are below, but very close to, the upper bound in Figure 11.2. Also, when total utilization (u) is 0.1, we observe that T_{VM} / T1 is zero when $C_{VM}/T_{VM} = 0.1$. This means that the task set inside the VM is not schedulable when $C_{VM}/T_{VM} = 0.1$.

11.7.2 Total Utilization of 0.2

Figure 11. shows that the standard deviation divided with the average is almost zero except when $C_{VM}/T_{VM} = 0.3$. This means that for $C_{VM}/T_{VM} = 0.3$ (and n = 10, 20, and 30) some task sets are schedulable with T_{VM} / T1 close to the upper bound (0.71, see Figure 11.2), but other task sets need a much shorter T_{VM} . When is C_{VM}/T_{VM} larger than 0.3, all task sets are schedulable with T_{VM} / T1 close to the upper bound.

Figure 11. shows that the maximum T_{VM}, for which the task set inside the VM is schedulable, increases when C_{VM}/T_{VM} increases. When C_{VM}/T_{VM} is larger than 0.3 the curves in Figure 11. are very close to the upper bound in Figure 11.2. When $C_{VM}/T_{VM} = 0.1$, and 0.2, Figure 11. shows that the task set inside the VM is not schedulable (i.e., T_{VM} / T1 = 0 for these values). When $C_{VM}/T_{VM} = 0.3$, Figure 11. shows that the average T_{VM} / T1 are significantly below the upper bound. As discussed above, the reason for this is that when $C_{VM}/T_{VM} = 0.3$ some task sets are schedulable with T_{VM} / T1 close to the upper bound, but other task sets need a much shorter T_{VM}.

11.7.3 Total Utilization of 0.3

Figure 11. shows that the standard deviation divided with the average is almost zero except when $C_{VM}/T_{VM} = 0.4$ (for n = 10, 20, and 30), and when $C_{VM}/T_{VM} = 0.5$ (for n = 10).

This means that for $C_{VM}/T_{VM} = 0.4$ (and n = 10, 20, and 30) some task sets are schedulable with T_{VM} / T1 close to the upper bound (0.83, see Figure 11.2), but other task sets need a much shorter T_{VM}. For n = 10, we have a similar situation when $C_{VM}/T_{VM} = 0.5$. In general, the standard deviation decreases when n increases.

11.8 Considering Overhead

In our previous model presented in Section 10.3, we neglected the overhead induced by switching from one VM to another. However, in reality this is not the case. So in this section we consider overhead at the beginning of execution of each VM.

11.8.1 Defining Overhead

By considering the overhead we can rewrite the Equation (4) as

$$f^{-1}(t, T_{VM}, C_{VM}, X) = (T_{VM} - C_{VM}) + t + \left\lceil \frac{t}{(C_{VM} - X)} \right\rceil (T_{VM} - C_{VM} + X) \quad (6)$$

where X denotes the overhead (see Figures 11.10 and 11.11). So, in the worst-case scenario considering this overhead model, the first execution of task τ_1 is after $2(T_{VM} - C_{VM}) + X$ time units (see Figure 11.10). Our model is obviously valid for non-preemptive scheduling since we have considered overhead at the beginning of the execution of each task [35]. The overhead model can also be used in systems with preemptive scheduling, since one can put a bound on the number of preemptions in RM and EDF schedulers [36]. By multiplying the maximum number of preemptions with the overhead for a preemption, and then making the safe (but pessimistic) assumption that all this overhead occurs at the start of a period, we arrive at the model considered here.

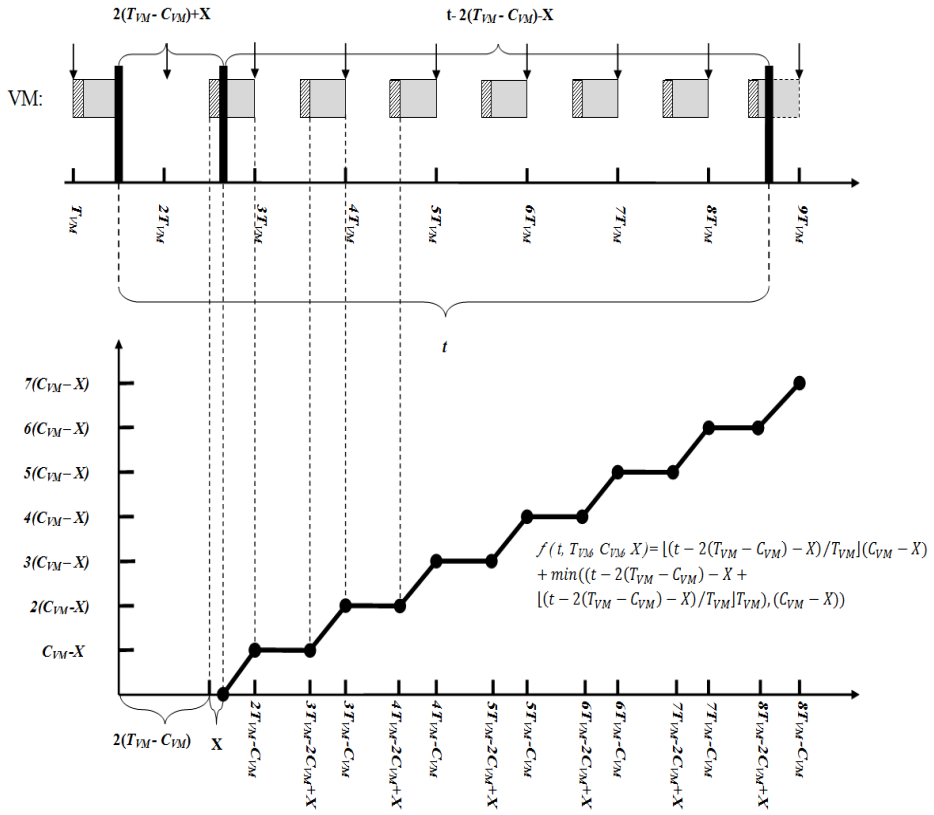


Figure 11.3 : Worst-case scenario when considering context switches overheads.

In [26][25] and [41] the authors calculated a different kind of overhead for periodic tasks; using our notation they calculated the ratio $((C_{VM}/T_{VM}) - u)/u$ (which for fixed C_{VM}/T_{VM} and u is an increasing function of T_{VM}).

In [26] and [40] the authors considered different overhead models (including overhead due to cache misses) on the task level in compositional analysis of real-time systems. Our model considers overhead on the hypervisor level, and our overhead analysis is in thus orthogonal to the overhead analysis on the task level (i.e., both models could be applied independently).

11.8.2 Prediction Model

For any task set, if C_{VM}/T_{VM} is given, it is possible to predict a range where we can search for values of T_{VM} that can make the task set inside the VM schedulable (i.e., a value of T_{VM} such that all tasks meet their deadlines). For values of T_{VM} that are outside of this interval, we know that there is at least one task that will not meet its deadline. In Figure 11.12(a) and (b), the solid line represents Maximum R_i/T_i $(1 \leq i \leq n)$ versus different values of T_{VM}, for given values of C_{VM}/T_{VM} and overhead X. As long as Maximum $R_i/T_i \leq 1$ we know that the task set is schedulable. If we can find two values of T_{VM} such that Maximum R_i/T_i = 1, Figure 11.12(a) indicates that the interval between these two values is the range of values of T_{VM} for which the task set is schedulable.

We now define a prediction model consisting of three lines that will help us to identify the range of values that will result in a schedulable task set. We will refer to these three lines as: the left bound, the lower bound and the schedulability limit. These three lines will produce a triangle. The intersection between the schedulability limit line and the left bound will give us the first point and the intersection between the schedulability limit line and the lower bound will give us the second point.

We consider the corresponding T_{VM} values of these two points on the x-axis and the interval between these two values (see Figure 11.12(a) and (b)). If the intersection between the lower bound and the schedulability limit is left of the intersection of the left bound and the schedulability limit (see Figure 11.12(b)), then it is not possible to find a T_{VM} that will make the task set schedulable.

In order to find the left bound we use the equation below:

$$\left(\frac{(C_{VM}-X)}{T_{VM}}\right) \geq \text{Total Utilization} \qquad (7)$$

According to Equation (7) a T_{VM} value must be selected so that $\left(\frac{(C_{VM}-X)}{T_{VM}}\right) \geq$ Total Utilization. Here X represents the amount of overhead and $(C_{VM} -$

X) represents the effective execution of the VM in one period. Obviously, $(\frac{(C_{VM}-X)}{T_{VM}})$ should be higher than or equal to total utilization since in order to successfully schedule all tasks in the VM, the utilization of the VM should be a value that is the same as the total utilization or higher.

In order to calculate the lower bound we consider the Maximum R_i/T_i $(1 \leq i \leq n)$ value. Our previous experiments showed that it is often (but not always) the task with the shortest period that restricts the length of T_{VM}. τ_1 is the task in the task set which has the shortest period T_1, and obviously $R_1/T_1 \leq$ Maximum R_i/T_i. We have $R_1/T_1 = (2(T_{VM} - C_{VM}) + X + C_1)/T_1$, and if we rewrite this equation and consider T_{VM} as a variable then we can calculate the lower bound using the equation below:

$$f(T_{VM}) = \frac{2(T_{VM}-C_{VM})}{T_1} + \frac{(C_1+X)}{T_1} \qquad (8)$$

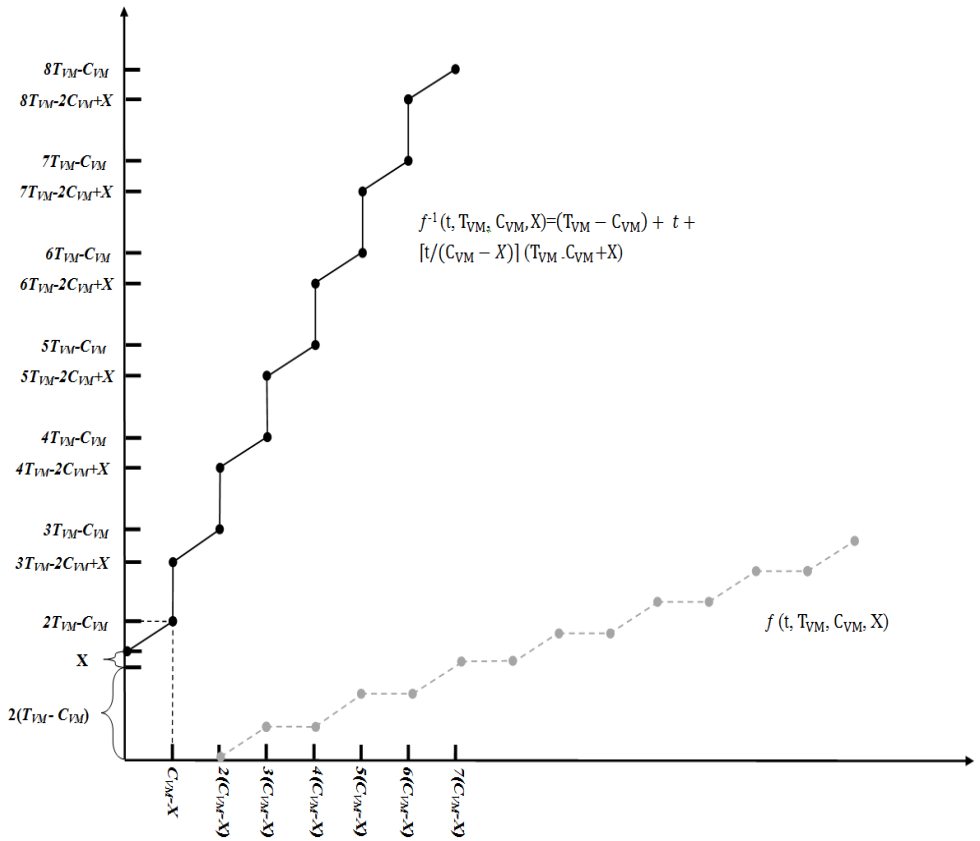


Figure 11.4 : The function $\boldsymbol{f^{-1}(t, T_{VM}, C_{VM}, X)}$; $\boldsymbol{t}$ *is the parameter on the x-axis in the graph.*

By considering the common form of a linear equation $f(x) = mx + b$ where m and b are constant values and m represents the slope of the line and b represents the offset, we can rewrite the Equation (8) if the value of C_{VM}/T_{VM} is given, e.g., $C_{VM}/T_{VM} = 0.54$, we can rewrite the Equation (8) as $f(T_{VM}) = \frac{2(1-0.54)}{T_1}(T_{VM}) + \frac{(C_1+X)}{T_1}$,. Thus the slope of the line becomes $m=\frac{2(1-0.54)}{T_1}$ and the offset is $b=\frac{(C_1+X)}{T_1}$.

The schedulability limit is represented by a horizontal line at Maximum $R_i/T_i = 1$. R_i/T_i $(1 \leq i \leq n)$.

In order to calculate the value Maximum R_i/T_i, we first calculate R_i/T_i for each task τ_i $(1 \leq i \leq n)$, and then Maximum R_i/T_i $(1 \leq i \leq n)$is selected for the entire task set.

For instance in Figure 11.12(a), total utilization $U = 0.3$, $C_{VM}/T_{VM} = 0.54$ and $X = 1$, so to calculate the left bound using Equation (7), for $T_{VM} = 1$, we will get $(0.54T_{VM} - 1)/T_{VM} = (0.54(1) - 1)/1 = -0.46$ for $T_{VM} = 2$ we get 0.04, for $T_{VM} = 3$ we get 0.206 and for $T_{VM} = 4$ we will have 0.29 while for $T_{VM} = 5$ we will get 0.34. So in this case for $T_{VM} = 5$ we get the value of 0.34 which is higher than total utilization (0.3) so we know that for T_{VM} values that are higher than 5 we have a chance to find the suitable T_{VM} for the entire task set.

However for different values of X the same T_{VM} value may not be valid anymore, e.g., if we consider X = 2, then for $T_{VM} = 5$ we will get $\frac{(C_{VM}-X)}{T_{VM}} = 0.14$. For $T_{VM} = 9$ we get $\frac{(C_{VM}-X)}{T_{VM}} = 0.31$ which is higher than total utilization (0.3).

In Figure 11.12(b), for total utilization $u = 0.1$, $C_{VM}/T_{VM} = 0.2$ and $X = 16$, if we calculate the left bound using Equation (7), then for $T_{VM} = 160$ we get $(0.2T_{VM} - 1)/T_{VM} = (0.2(160) - 16)/160 = 0.1$ which is equal to total utilization (0.1) (i.e., $T_{VM} = 160$ is the left bound).

For the lower bound, if we consider the first task in the task set has the shortest period $T_1 = 157$, and its execution time is $C_1 = 2$. Using Equation (8), we can get the slope of the line $\frac{2(1-0.54)}{T_1} = \frac{2(0.46)}{157} \approx 0.006$ and the offset will be $\frac{(C_1+X)}{T_1} = \frac{(2+1)}{157} \approx 0.02$ so we can plot a line which changes in proportional to T_{VM} values $f(T_{VM}) = 0.006(T_{VM}) + 0.02$ (see Figure 11.12 (a)).

In Figure 11.12 (b), we can consider a task in the task set with the shortest period, $T = 161$ and execution time $C = 2$, so we will have the linear equation $f(T_{VM}) = 0.009(T_{VM}) + 0.11$ which corresponds to the lower bound and changes

in proportional to T_{VM} values. As it can be observed from Figure 11.12 (b), if the intersection between the left bound and the lower bound become above the schedulability limit line there is no chance of finding any T_{VM} value that can schedule this task set when the overhead is $X = 16$.

In order to validate our model, in the next section we have presented different experiments with different number of task sets and various values of total utilizations and C_{VM}/T_{VM}. We have also considered different amounts of overhead (X).

11.8.3 Overhead Simulation Study

During overhead simulation we had 10 task sets of 10 tasks each, n = 10 (we saw no major difference when increasing the number of tasks). Each task $\tau_I\ (1 \leq I \leq n)$has a period $T_I\ (1 \leq I \leq n)$ and execution time $C_I\ (1 \leq I \leq n)$. Each task's period T_I is randomly generated in the range of [100, 1000] and the execution time C_I is randomly generated in the range of [100, T_I]. The C_I values are then multiplied with a factor to get the desired utilization u. Different values have been considered for overheads $X = 1, 2, 4, 8, 16$. Different values for total utilization u and C_{VM}/T_{VM} are also considered (see Table 11.2).

Given X, u and C_{VM}/T_{VM}, and using Equation (6) we calculate R_i for all different values of T_{VM} in the range of [0, 500]. We then calculate $R_i/T_i\ (1 \leq i \leq n)$ for each task for each value of T_{VM}, and then we obtain the Maximum $R_i/T_i\ (1 \leq i \leq n)$for each value of T_{VM} (see Figures 11.13 – 11.15). The figures show the average value of the 10 task sets.

Table 11.2 : Overhead simulation, different values for Total utilization (u) and C_{VM}/T_{VM} .

		C_{VM}/T_{VM}			
Total Utilization (u)	**0.1**	0.14	0.16	0.18	0.2
	0.2	0.28	0.32	0.36	0.4
	0.3	0.42	0.48	0.54	0.6

11.8.3.1 Total Utilization of 0.1

Figure 11.13 shows that for total utilization of 0.1 and overhead values are more than 4 ($X = 8$ and $X = 16$), the task sets are not schedulable.

However, for small value of overhead $X = 1$, the task sets are always schedulable for the values of C_{VM}/T_{VM} considered here. When the overhead values are $X = 2$ and $X = 4$, the task sets are schedulable only when C_{VM}/T_{VM} = 0.2.

11.8.3.2 Total Utilization of 0.2

Figure 11.14 shows the Maximum R_i/T_i values for different T_{VM} and overhead values when total utilization is 0.2. As we can observe in the figures, the task sets are not schedulable when $X = 16$ (even for C_{VM}/T_{VM} = 0.4). For other overhead values ($X = 1, 2, 4$), we see that in most of the cases the task sets are schedulable. When the value of C_{VM}/T_{VM} = 0.4, even task sets with overhead value of $X = 8$ are schedulable.

11.8.3.3 Total Utilization of 0.3

For total utilization of 0.3, Figure 11.15 shows that when C_{VM}/T_{VM} is 0.54 and 0.6, all tasks in the task set will meet their deadlines for at least some T_{VM} value.

11.9 Conclusions

We consider a real time application consisting of a set of n real-time tasks $\tau_i (1 \leq i \leq n)$ that are executed in a VM; the tasks are sorted based on their periods, and τ_1 has the shortest period. We have defined a function $f^{-1}(t, T_{VM}, C_{VM})$ such that a real-time application that uses fixed priorities and RMS priority assignment will meet all deadlines if we use a VM execution time C_{VM} and a VM period T_{VM} such that $R_i = f^{-1}\left(\left(C_i + \sum_{j=1}^{i-1}\lceil R_i/T_j \rceil C_j\right), T_{VM}, C_{VM}\right) \leq T_i (1 \leq i \leq n)$. This makes it possible to use existing real-time scheduling theory also when scheduling VMs containing real-time applications on a physical server.

The example that we looked at in Section 10.5 shows that there is a trade-off between on the one hand a long T_{VM} period (which reduces the overhead for switching between VMs), and low processor utilization (i.e., low C_{VM}/T_{VM}). The example also shows that the "critical" task, i.e., the task which puts the toughest restriction on the maximal length of T_{VM}, may be different for different values on C_{VM}/T_{VM} .

From the simulation results shown in Section 10.6, we see that increasing the number of the tasks (n) does not affect the maximum T_{VM} for which the task set inside the VM is schedulable (see Figure 11., Figure 11. and Figure 11.). The simulation results also show that the standard deviation of the maximum T_{VM} is almost zero except when C_{VM}/T_{VM} is slightly above the total utilization (u) of the task set (see Figure 11.4, Figure 11. and Figure 11.).

We have also presented an upper bound on the maximum T_{VM} for which the task set inside the VM is schedulable (see Figure 11.2). The simulation results show that the maximum T_{VM} is very close to this bound when C_{VM}/T_{VM} is (significantly) larger than the total utilization (u) of the task set inside the VM.

If overhead from switching from one VM to another is ignored, the simulation study in Section 11.6 shows those infinitely small periods (T_{VM}) are the best, since they minimize processor utilization. In order to provide more realistic results, we included and evaluated an overhead model that makes it possible to consider the overhead due to context switches between VMs. Along with the model we also defined two performance bounds and a schedulability line, each representing a straight line in a figure that plots the Maximum R_i/T_i as a function of the period of the VM (T_{VM}). These three lines form a triangle and we show that the intersection between the performance bounds and the schedulability lines defines an interval where valid periods (i.e., periods that could result in all tasks meeting their deadlines) can be found. This performance model also makes it possible to easily identify cases when no valid T_{VM} can be found.

We have also done a simulation study that shows how the overhead for switching from one VM to another affects the schedulability of task set running in the VM.

Our method is presented in the context of VMs with one virtual core. However, it is easily extendable to VMs with multiple cores as long as each real-time task is allocated to one of the (virtual) cores. In that case we need to repeat the analysis for each of the virtual cores and make sure that all real-time tasks on each core meet their deadlines.

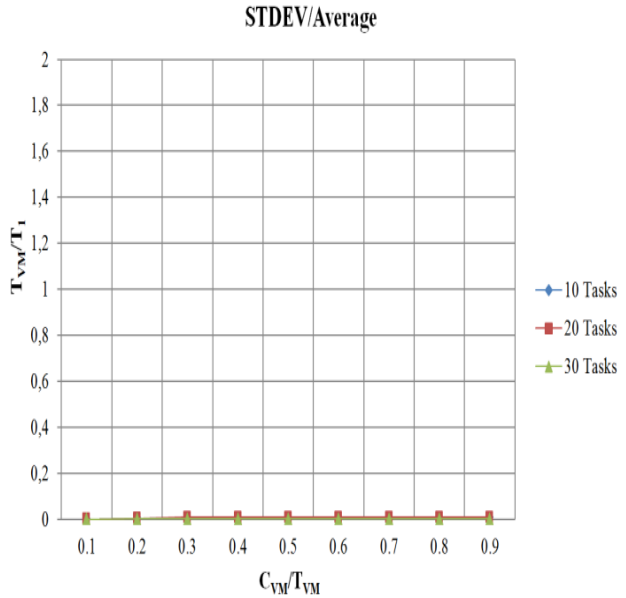


Figure 11.4 : Standard deviation for T_{VM} divided with average T_{VM} when the total utilization u = 0.1.

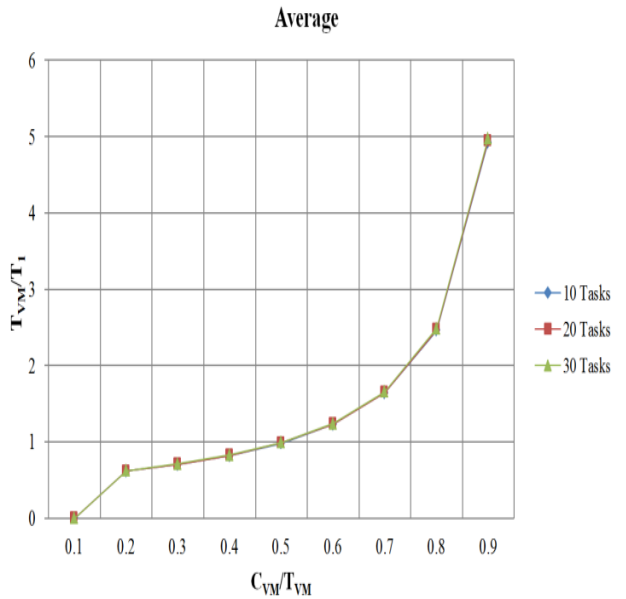


Figure 11.5 : Average T_{VM}/T_1 when the total utilization u = 0.1.

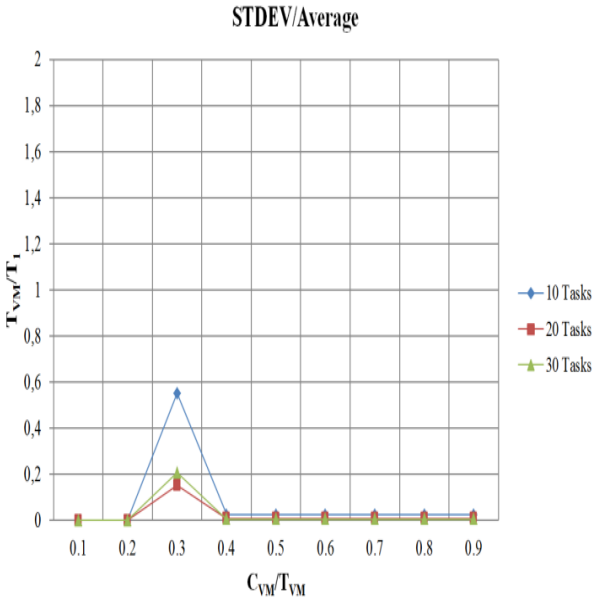


Figure 11.6 : Standard deviation for T_{VM} divided with average T_{VM} when the total utilization u = 0.2.

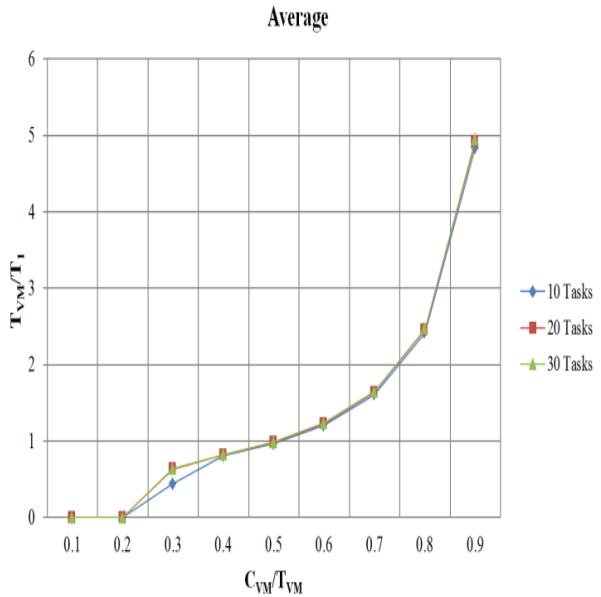


Figure 11.7 : Average T_{VM}/T_1 when the total utilization u = 0.2.

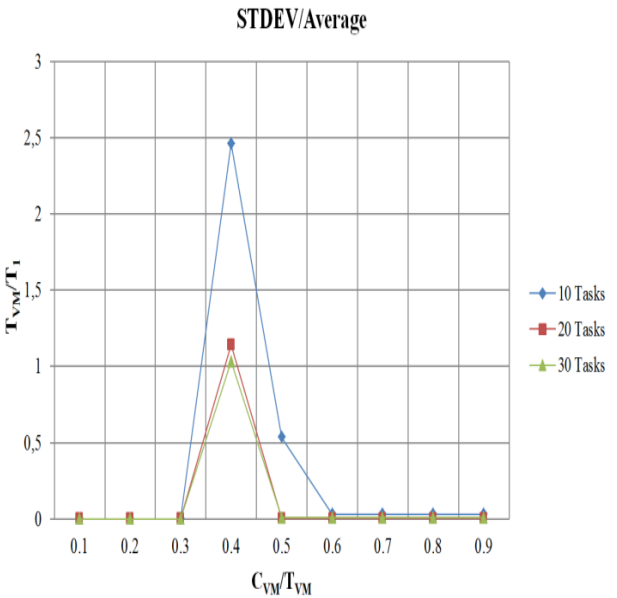


Figure 11.8 : Standard deviation for T_{VM} divided with average T_{VM} when the total utilization u = 0.3

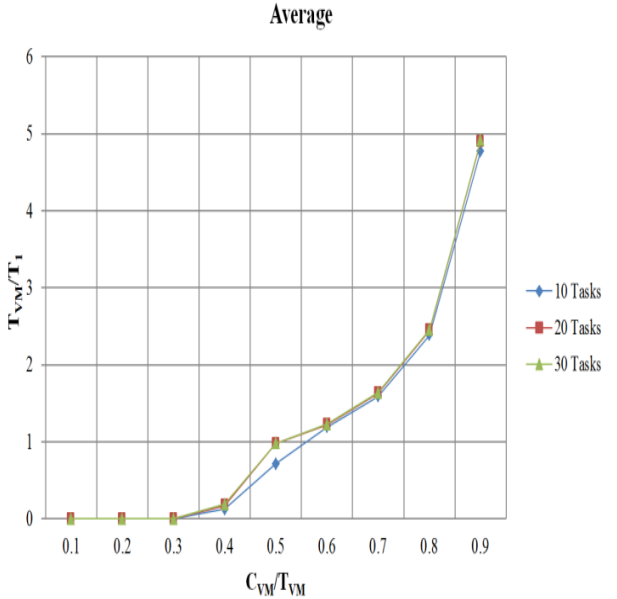
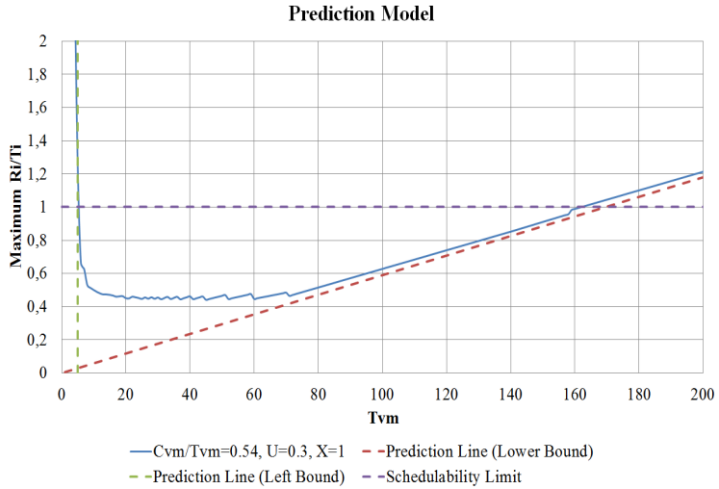
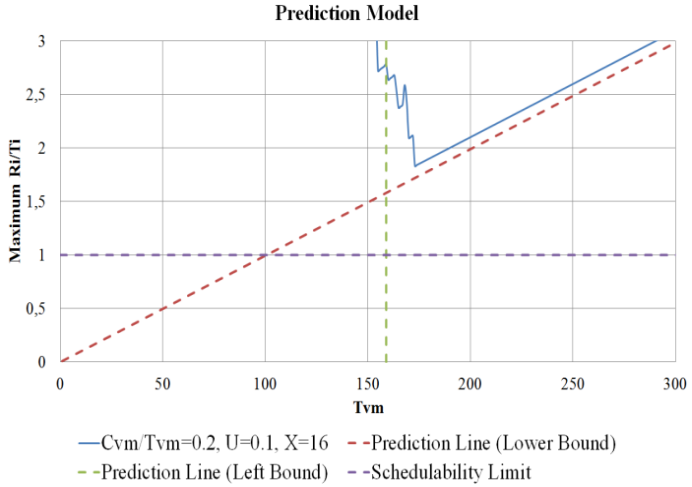


Figure 11.9 : Average T_{VM}/T_1 when the total utilization u = 0.3.

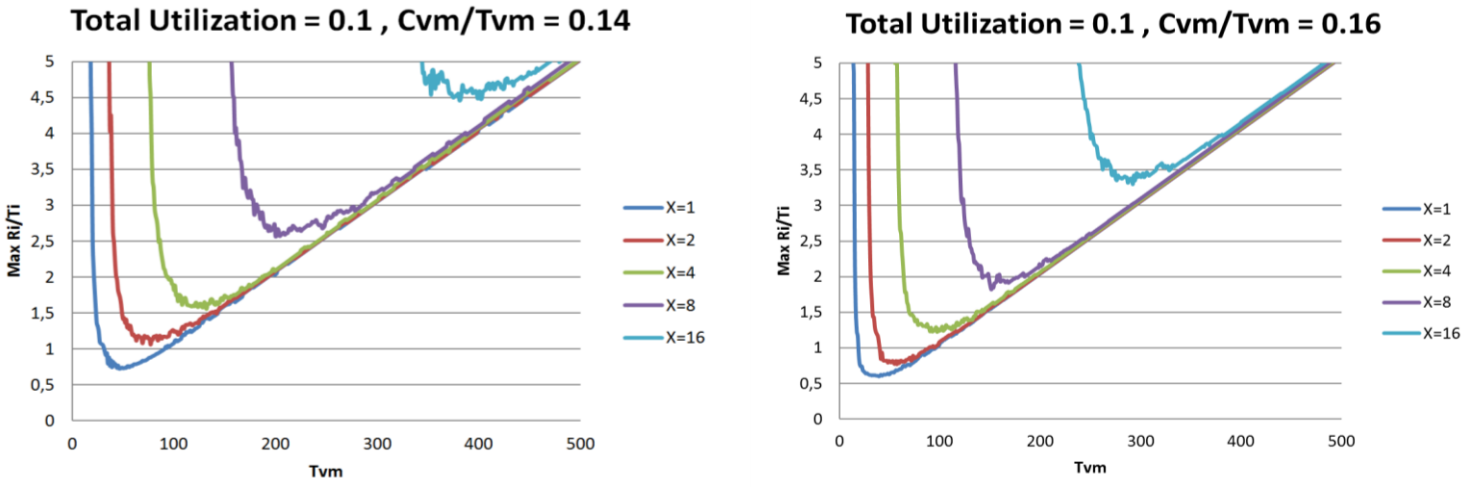


a.



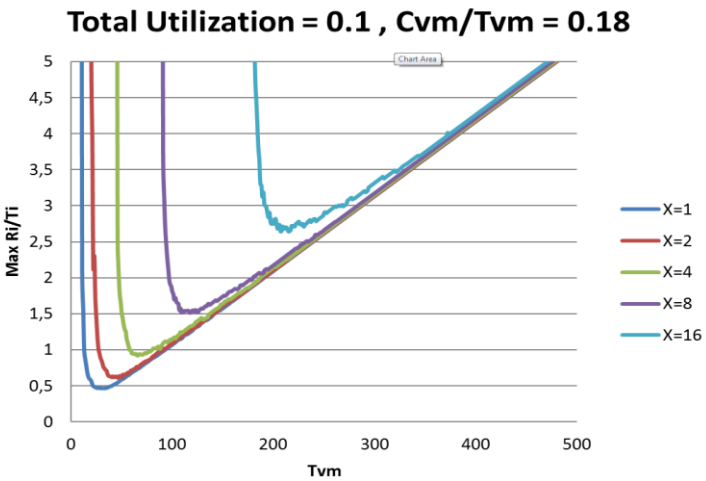
b.

Figure 11.12 : (a)Prediction Model for $\boldsymbol{C_{VM}/T_{VM} = 0.54, u = 0.3\ and\ X = 1}$ *and (b) Prediction Model for* $\boldsymbol{C_{VM}/T_{VM} = 0.2, u = 0.1\ and\ X = 16}$

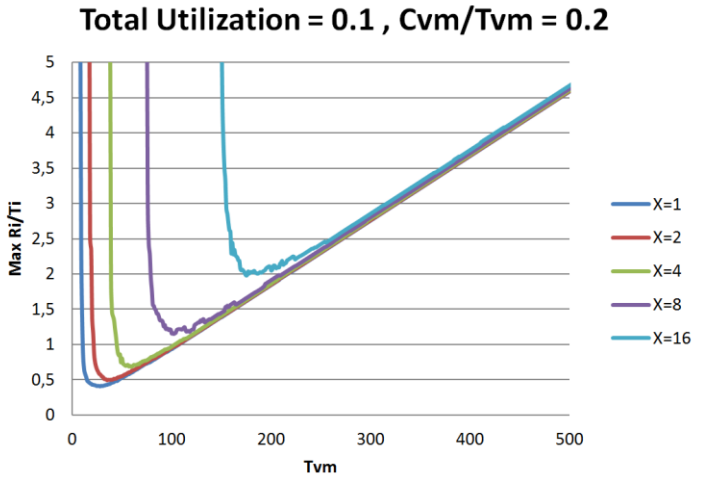


a. *b.*

Figure 11.13. Overhead simulation results for $\boldsymbol{C_{VM}/T_{VM} = 0.14, 0.16, 0.18, 0.2}$ *when total utilization* $\boldsymbol{u = 0.1}$

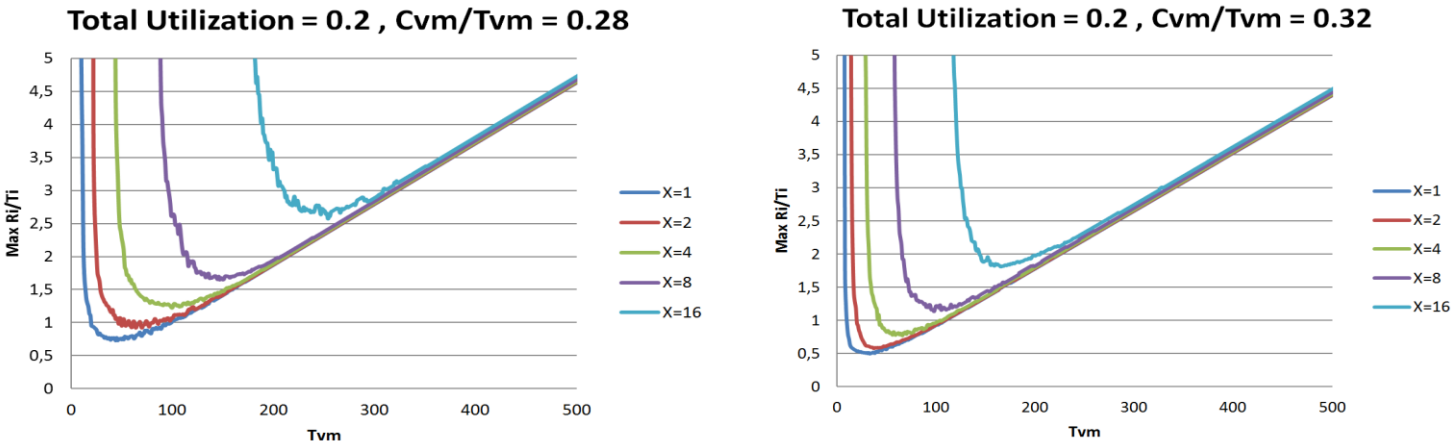


c.



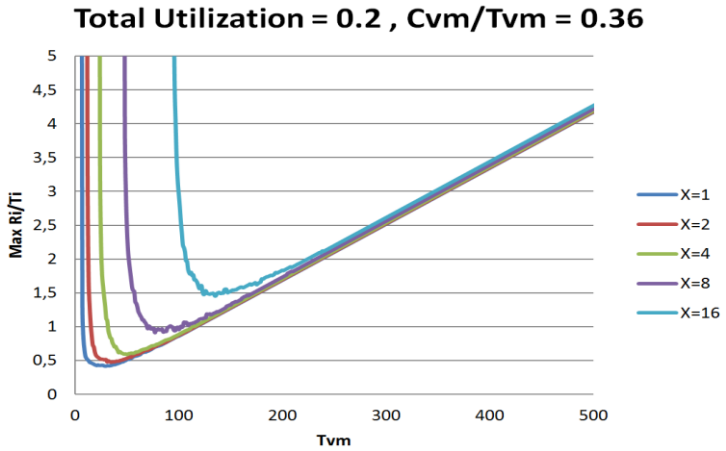
d.

Figure 11.13. Overhead simulation results for $\boldsymbol{C_{VM}/T_{VM} = 0.14, 0.16, 0.18, 0.2}$ *when total utilization* $\boldsymbol{u = 0.1}$

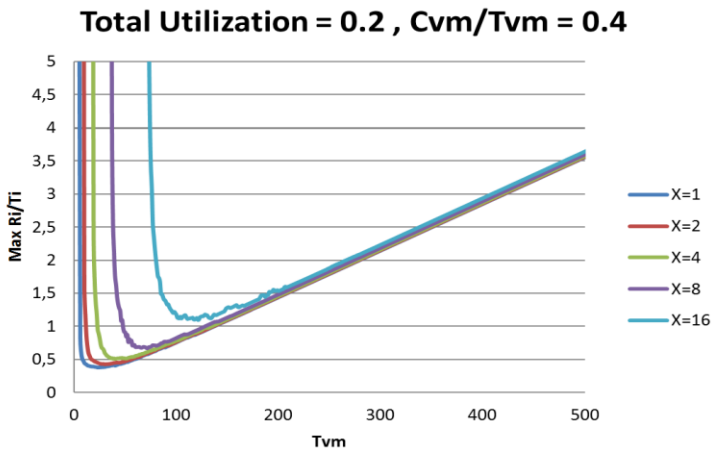


a. *b.*

Figure 11.14. Overhead simulation results for $\boldsymbol{C_{VM}/T_{VM} = 0.28, 0.32, 0.36, 0.4}$ *when total utilization is* $\boldsymbol{u = 0.2}$

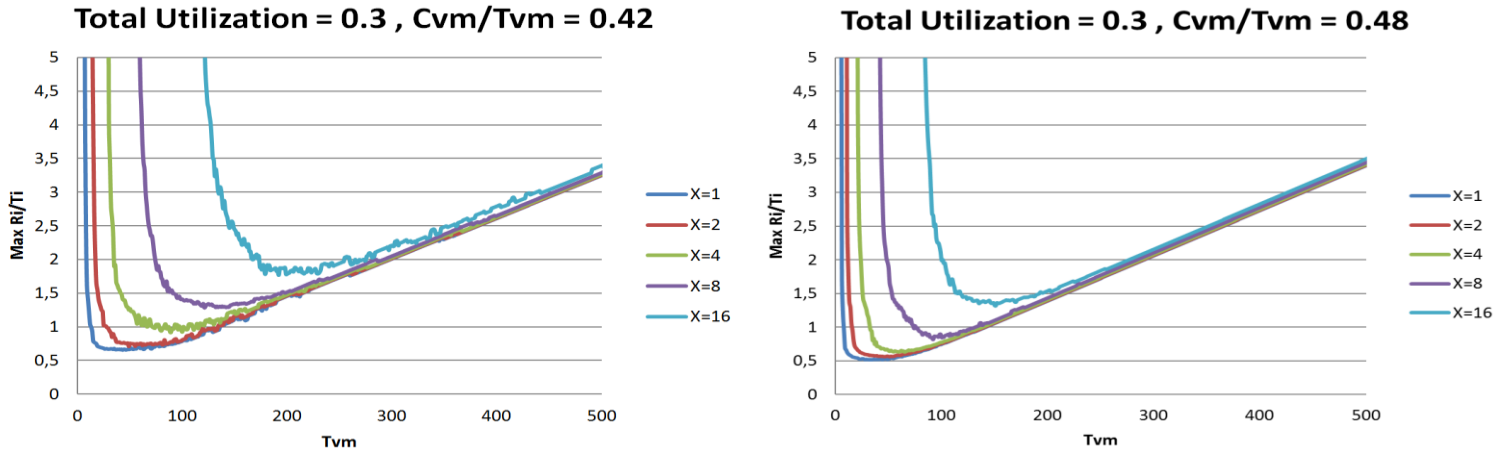


c.



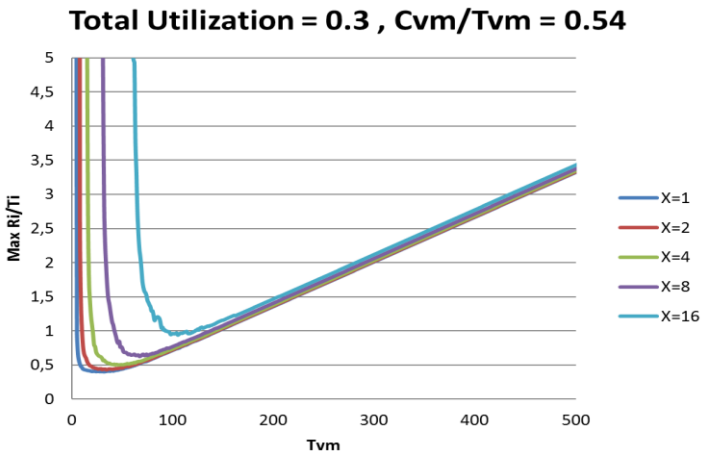
d.

Figure 11.14. Overhead simulation results for $\boldsymbol{C_{VM}/T_{VM} = 0.28, 0.32, 0.36, 0.4}$ *when total utilization is* $\boldsymbol{u = 0.2}$

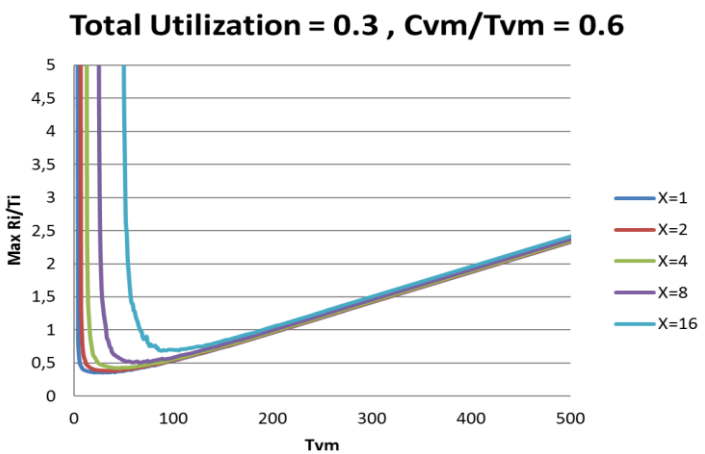


a. *b.*

Figure 11.15. Overhead simulation results for $\boldsymbol{C_{VM}/T_{VM} = 0.42, 0.48, 0.54, 0.6}$ *when total utilization is* $\boldsymbol{u = 0.3}$



c.



d.

Figure 11.15. Overhead simulation results for $\boldsymbol{C_{VM}/T_{VM} = 0.42, 0.48, 0.54, 0.6}$ *when total utilization is* $\boldsymbol{u = 0.3}$

11.10 References

[1] Lee M., Krishnakumar A.S., Krishnan P., Singh N., and Yajnik S. 2010. Supporting Soft Real-Time Tasks in the Xen Hypervisor. The 2010 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution (Pittsburg, Mar. 2010).

[2] Duda K. and Cheriton D. 1999, Borrowed-Virtual-Time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. ACM SIGOPS Operating Systems Review, 33 (5), December 1999.

[3] Stoica I., Abdel-Wahab H., Jeffay K., Brauha S., Gehrke J., and Plaxton G., 1996. A Proportional Share Resource Allocation Algorithm for Real-Time, Time-Shared Systems. 17th IEEE Real Time Systems Symposium, December 1996.

[4] Nieh J. and M. Lam. 2003. A SMART scheduler for multimedia applications. ACM Transactions on Computer Systems, vol. 21, No. 2, May 2003.

[5] Lin B. and DindaP.A. 2005. VSched: Mixing Batch and Interactive Virtual Machines Using Periodic Real-Time Scheduling. The 2005 ACM/IEEE SC05 Conference (Seattle, Nov. 2005).

[6] Salimi H, Najafzadeh M., and Sharifi M. 2012. Advantages, Challenges and Optimization of Virtual Machine Scheduling in Cloud Computing Environments, International Journal of Computer Theory and Engineering, vol. 4, no. 2, April 2012.

[7] Liu C. and Leyland J. 1973. Scheduling algorithms for multiprogramming in a hard real-time environment, Journal of the ACM, 20(1), 1973.

[8] Lundberg L. 2002. Analyzing Fixed-Priority Global Multiprocessor Scheduling. IEEE Real Time Technology and Applications Symposium, San Jose, USA, September 2002.

[9] Burns A. and Wellings A. 2009. Real-Time Systems and Programming Languages, Addison Wesley, ISBN 978-0-321-41745-9, 2009.

[10] Liu S., Quan G., and Ren S. 2010. On-line Scheduling of Real-time Services for Cloud Computing. IEEE 6th World Congress on Services, Miami, USA, July, 2011.

[11] Cucinotta T., Checconi F., Kousiouris G., Kyriazis D., Varvatigou T., Mazzetti A., Zlatev Z., Papay J., Boniface M., Berger S., Lamp D., Voith T., Stein M. 2010. Virtulised e-Learning with Real-Time Guarantees on the IRMOS Platform. IEEE International Conference on Service-Oriented Computing and Applications (SOCA).December 2010.

[12] Luca A. and Tommaso C. 2011, Efficient Virtualization of Real-Time Activities. IEEE International Conference on Service-Oriented Computing and Applications. USA, 2011, pp 1-4.

[13] Yunfa L., Xianghua X., Jian W., Wanqing L., Youwei Y. 2010. A Real-Time Scheduling Mechanism of Resource for Multiple Virtual Machine System. The ChinaGride Conference. Guangzhou, China, 2010, pp 137-143.

[14] Subramanian S., Nitish K., Kiran K. M., Sreesh P., Karpagam G. R. 2012. An Adaptive Algorithm for Dynamic Priority Based Virtual Machine Scheduling in Cloud. The IJCSI International Journal of Computer Science, November 2012, pp 397-383.

[15] Xiao J., Wang Z. 2012. A Priority Based Scheduling Strategy for Virtual Machine Allocations in Cloud Computing Environment. The International Conference on Cloud Computing and Service Computing, Shanghai, China, 2012, pp 50-55.

[16] Sisu X., Justin W., Chenyang L., Christopher G. 2011. RT-Xen: Towards Real-Time Hypervisor Scheduling in Xen. The International Conference on Embedded Software, Taipei, Taiwan, 2011, pp 39-48.

[17] Tommaso C., Dhaval G., Dario F., Fabio C. 2010. Providing Performance Guarantees to Virtual Machines. Proceedings of The 5th Workshop On Virtualization And Cloud Computing, Italy, 2010.

[18] Tommaso C., Gaetano A., Luca A. 2008. Real-Time Virtual Machines. The 29th Real Time Systems Symposium, Barcelona, Spain, December 2008.

[19] Tommaso C., Gaetano A., Luca A. 2009. Respecting temporal constraints in virtualized services. The Computer Software and Applications Conference, Seattle, U.S., July 2009, pp 73-78.

[20] Davis R., Burns A. A survey of Hard Real-Time Scheduling for Multiprocessor Systems, ACM Computing Surveys, Vol. 43, No. 4, October, 2011.

[21] Feng X. and Mok A. K. A Model of Hierarchical Real-Time Virtual Resources. In Proceedings of the 23rd IEEE Real-Time Systems Symposium, Austin, TX USA, Dec. 2002, pp 26-35.

[22] Shih I. and Lee I. Periodic Resource Model for Compositional Real-Time Guarantees. In Proceedings of the 24th Real-Time Systems Symposium, Cancun, Mexico, Dec. 2003, pp 2-13.

[23] Lipari G. and Bini E. A methodology for designing hierarchical scheduling systems. J. Embedded Computing, 2005, pp 257-269.

[24] Davis R. and Burns A. Hierarchical fixed priority pre-emptive scheduling. In 26th IEEE International Real-Time Systems Symposium. RTSS 2005.

[25] Shin I. and Lee I. Compositional real-time scheduling framework with periodic model. ACM Transactions on Embedded Computing Systems (TECS), 7(3):30, 2008.

[26] Phan L. T.X., Xu M., Lee J., Lee I., Sokolsky O. Overhead-Aware Compositional Analysis of Real-Time Systems. In 19th IEEE Real-Time and Embedded Technology and Applications Symposium, Philadelphia, PA, 2013, pp 237-246.

[27] Lee J., Xi S., Chen S., Phan L. T. X., Gill C., Lee I., Lu C., Sokolsky O. Realizing Compositional Scheduling through Virtualization. Proceedings of the IEEE 18th Real-Time and Embedded Technology and Applications Symposium, USA, 2012, pp 13-22.

[28] Asberg M., Nolte T., Kato S., Rajkumar R. ExSched: An External CPU Scheduler Framework for Real-Time Systems. 18th IEEE International conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Seoul, 2012, pp 240-249.

[29] Yang J., Kim H., Park S., Hong C., Shin I. Implementation of Compositional Scheduling Framework on Virtualization. Published in Newsletter ACM SIGBED, Vol 8 Issue 1, 2011, pp 30-37.

[30] Behnam M., Nolte T., Shin I., Asberg M., Bril R. Towards Hierarchical Scheduling in VxWorks. 4th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, Prague, Czech Republic, 2008, pp 63-72.

[31] Lipari G., Bini E. Resource Partitioning among Real-Time Applications. Proceedings of the 15th Euromicro conference on Real-Time Systems, 2003, pp 151-158.

[32] Shin I., Lee I. Compositional Real-Time Scheduling Framework. 25th IEEE International Real-Time Systems Symposium, 2004, pp 57-67.

[33] Zmaranda D., Gabor G., Popescu D.E., Vancea C., Vancea F. Using Fixed Priority Pre-emptive Scheduling in Real-Time Systems. Published in International Journal of Computers Communications and Control, 2011, pp 187-195.

[34] Saewong S., Rajkumar R., Lehoczky J., Klein M. Analysis of hierarchical fixed-priority scheduling. Proceedings of the 14th Euromicro Conference on Real-Time systems, CA, 2002, pp 173-181.

[35] Baruah S. The Non-preemptive scheduling of periodic tasks upon multiprocessors. Published in journal of real-time systems, USA, 2006, pp 9-20.

[36] Easwaran A., Shin I., Lee I., Sokolsky O. Bounding Preemptions under EDF and RM Schedulers. MS-CIS-06-07, Department of Computer and Information Science, University of Pennsylvania.

[37] Lundberg L., Shirinbab S. Real-time scheduling in cloud-based virtualized software systems. In proceedings of the Second Nordic Symposium on Cloud Computing, Oslo, Norway:ACM, 2013, pp 54-58.

[38] Easwaren A., Anand M., Insup L. Compositional analysis framework using EDP resource models. Published in Real-time systems symposium, 2007, pp 129-138.

[39] Lu W., Li K., Wei H., Shih W. Rate monotonic schedulability tests using period dependent conditions. Published in Journal Real-Time systems, 2007, pp 123-138.

[40] Meng X., Phan L., Lee I., Sokolsky O. Cache-aware compositional analysis of real-time multicore virtualization platforms. Published in Real-Time systems symposium, 2013, pp 1-10.

[41] Chen S., Phan L., Lee J., Lee I., Sokolsky O. Removing abstraction overhead in the composition of hierarchical real-time systems. Proceedings of the 17th IEEE Real-time and embedded technology and applications, 2011, pp 81-90.