ON IMPROVING
THE PERFORMANCE AND RESOURCE UTILIZATION OF
CONSOLIDATED VIRTUAL MACHINES:
MEASUREMENT, MODELING, ANALYSIS, AND PREDICTION

A thesis submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy
in the School of Information Technologies
at The University of Sydney

Maruf Ahmed
August 2018

(In the name of Allah, Most gracious, Most merciful)

Glorious Quran

Recite in the name of your Lord who created -
Created man from a clinging substance.
Recite, and your Lord is the most Generous -
Who taught by the pen -
Taught man that which he knew not.

Surah Al-'Alaq [96:1-5]

"My Lord, increase me in knowledge."

Surah Taha [20:114]

Elegy Written in a Country Churchyard (1751)

The boast of heraldry, the pomp of pow'r,
And all that beauty, all that wealth e'er gave,
Awaits alike th' inevitable hour.
The paths of glory lead but to the grave.

Thomas Gray (1716 –1771)

False Greatness (1706)

Tis true, my form is something odd
but blaming me, is blaming God,
Could I create myself anew
I would not fail in pleasing you.

Joseph Carey Merrick (1862 –1890)

Were I so tall to reach the pole,
Or grasp the ocean with my span,
I must be measured by my soul;
The mind's the standard of the man.

Isaac Watts (1674 –1748)

Abstract

"Being ignorant is not so much a shame, as being unwilling to learn"

— Benjamin Franklin (1706–1790)

This dissertation addresses the performance related issues of consolidated *Virtual Machines* (VMs). *Virtualization* is an important technology for the *Cloud* and data centers. In many ways, essential Cloud services are directly dependent on virtualization. Essential features of a data center like the fault tolerance and high-availability of services are implemented with the help of VMs. The fundamental principle of such schemes is to execute multiple copies of a VM on two or more physical servers; if one VMs fails, then another VM must take over the execution process.

Similarly, *pay-as-you-go* model, one of the most critical features of Cloud, is also made possible by VM deployment technique. Cloud users can rent VMs for any duration; the VMs are automatically created, managed, and destroyed by sophisticated Cloud management software. Furthermore, VMs consolidation helps to increase resource utilization and reduce energy costs by concurrently running multiple VMs on the same physical server. Thus, virtualization is an essential technology for the Cloud and data centers.

Cloud had become one of the significant innovations over the past decade. Research has been going on the deployment of newer and diverse set of applications like the *High-Performance Computing* (HPC), and parallel applications on the Cloud. Energy consumption efficiency is becoming a significant issue for the data centers, too. Nowadays, the big data centers house more servers than ever before; such data centers can consume megawatts of electricity per hour. Improvements in the energy efficiency of the servers can save much energy.

The primary technology behind the Cloud is virtualization; therefore, it is necessary to investigate the resource utilization characteristics of the VMs. The primary

method to increase the server resource utilization is VM consolidation, running as many VMs as possible on a server is the key to improving the resource utilization. On the other hand, consolidating too many VMs on a server can degrade the performance of all VMs.

What is more, different resource-intensive tasks like, CPU, memory, and I/O-intensive tasks have different effects on consolidation. The performance of VMs can change considerably depending on what types of tasks are being consolidated. Therefore, it is necessary to measure, analyze and find ways to predict the performance variation of consolidated VMs.

This dissertation investigates the causes of performance variation of consolidated VMs; the relationship between the resource contention and consolidation performance, and ways to predict the performance variation. Experiments have been conducted with real virtualized servers without using any simulation. All the results presented here are real system data. The VM performance is profiled for various stages of server consolidation.

The number of concurrently running VMs can change at any time on a server; moreover, different types of resource-intensive tasks may run on the VMs. In this dissertation, a methodology is introduced to do the experiments with a large number of tasks and VMs; it is called the *Incremental Consolidation Benchmarking Method* (ICBM). The experiments have been done with different types of resource-intensive tasks, parallel workflow, and VMs.

Furthermore, to experiment with a large number of VMs and collect the data; a scheduling framework is designed and implemented. The framework makes it easy to handle a large number of VMs at a time. The experiments are done in stages; at each stage different combination of tasks and VMs are concurrently run on the server to collected data. The data includes task execution time variation, system-interrupts, and page-faults.

Experiments have been done with various benchmark suites and a parallel workflow. Three popular hypervisors are used in the experiments; they are *VMware ESXi*, *Citrix XenServer*, and *Xen*. Analysis of the experimental data demonstrates some interesting relationship among the VM resource utilization and consolidation performance. Furthermore, it is shown that the profiled execution time variation data can be used as a performance measure and to train prediction models for the consolidated VMs.

Two machine learning techniques have been used in this dissertation, to build prediction models for the VM execution time variation; they are the *Least Square Regression* (LSR) and *Artificial Neural Network* (ANN). Experimental results show that the models can quite accurately predict the performance variation of VM consolidation on multiple hypervisors.

To conduct experiments, consolidated VMs have been divided into two categories; the target VM and co-located VMs. During experiments, data is separately collected from both sets of VMs, and separate sets of performance prediction models are built. Sperate data is used for training and testing the prediction models as well. Chapter 4 gives details about how the data is collected, and prediction models are built using LSR.

In this dissertation, the *Root Mean Squared Error* (RMSE), *Mean Squared Error* (MSE)/ *Mean Squared Deviation* (MSD), and *Mean Absolute Percentage Error* (MAPE)/ *Mean Absolute Percentage Deviation* (MAPD) are used as measures to compare the accuracy of prediction models. Three prediction models are built for three resource combinations; (i) CPU-memory, (ii) CPU-I/O, and (iii) memory-I/O. The prediction accuracies of the three models of each dataset are tested separately.

In Chapter 4, two sets of prediction models are created using two sets of *Execution Time Variation* (ETV) data. The first set of prediction models are built using target VM data, and prediction accuracies are tested on two tasks; Filebench and IOzone. In the case of Filebench, the RMSE of predictions for three resource combinations are 0.771, 2.131, and 2.605, respectively. On the other hand, for IOzone the RMSE of predictions for the same three resource combinations are 2.193, 2.475, and 3.083, respectively.

The second set of prediction models are created for the same three resource combinations (CPU-memory, CPU-I/O, and memory-I/O) with the data collected from the co-located VMs. Accuracies of the three models are tested again using Filebench and IOzone. In this case, for the Filebench the RMSE of predictions for three resource combinations are 0.657, 1.841, and 1.475, respectively. Lastly, for IOzone the RMSE of prediction models are 2.083, 4.572, and 3.898, respectively.

In Chapter 6, the effectiveness of the introduced methodology (ICBM) is tested for multiple hypervisors. In this chapter, experiments are performed with three different hypervisors; ESXi, XenServer, and Xen. Then, a set of unified prediction models are built using the experimental data from the three hypervisors. The accuracies of

the prediction models are also tested on three hypervisors using separate datasets for testing and training. The MAPE is used as a measure of prediction accuracy for the models. The MAPE of prediction models for ESXi, XenServer, and Xen hypervisors are 9.5255%, 29.7558%, and 12.5249%, respectively.

Furthermore, in Chapter 6, ANN prediction models for VM execution time variation are built and tested. The chapter describes how the experimental data are collected from multiple hypervisors and used to build the prediction models. Once again, models are trained and tested using separate datasets. The ANN prediction models are tested on two separate hypervisors and three sperate tasks.

The three tasks used for testing the prediction accuracy of ANN models are the Sysbench CPU test, Stream, and Filebench. For the Xen, the MAPEs of prediction for the above mentioned three tasks are 20.50%, 11.78%, and 21.02%, respectively. For ESXi, the MAPEs of prediction for the same three tasks are 22.17%, 12.31%, and 15.88%, respectively.

Experiments are also performed with the performance variation of tasks of a real-world scientific workflow on consolidated VMs. The *Galactic Arecibo L-band Feed Array HI* (GALFA-HI) Survey project employs the Arecibo 305 meter telescope to map the neutral hydrogen around the Milky Way Galaxy. A considerable amount of radio telescope and astrophysical data needs to be analyzed to produce mosaic images of the scanned section of the sky.

Additionally, experiments are also conducted to observe the performance impact of consolidation on the tasks of the above mentioned scientific workflow. Accuracies of the performance prediction models are tested on two hypervisors, Xen and ESXi. The *mAddCube* is one of the tasks of the workflow. In the case of Xen, the RMSE, and MAPE of prediction for the mAddCube task are 3.30, and 22.86%, respectively. Next, for the ESXi hypervisor, the RMSE, and MAPE of prediction for the same task are 3.35, and 19.52%, respectively.

I dedicate this dissertation to the memory of my beloved parents, and to my dear brother for all their support and sacrifices.

Acknowledgements

"Great minds discuss ideas
Average minds discuss events
Small minds discuss people"

— Anonymous / Disputed

This dissertation is my blood, sweat, and tears. I like to express my gratitude to the Almighty for all the blessings and favors, which He has bestowed on me. I was alone, unable to defend myself and betrayed by everyone. It is by only the mercy of Allah that I have been able to complete this thesis.

I thank the glorious Quran for providing me with the inspiration to carry on. Whenever I face adversity always there is a verse to cheer me up. I had no one to rely on and no way to support myself. Following verse gave me hope: "*whoever relies upon Allah - then He is sufficient for him.*"-Quran [65:3]. I face hostilities from everyone, and there was nowhere to go. Again, following Quranic verse inspired me: "*verily, with every difficulty, there is relief: Verily with every difficulty there is relief.*"- Quran [94:6].

Thanks to my parents, **Mesbah Uddin Ahmed** and **Maya Rahman**, for my upbringing and all the sacrifices that they made for me. My parents went through many difficulties in their lives. However, they always encouraged me to be truthful and honest. I was able to continue my education because they gave me a place to stay, provided me with food and all other expenses needed for my education. Without their support, I would have never been able to come this far. I pray that Allah gives them greater rewards for all the sacrifices they made and forgives them.

Thanks to my brother, **Mamun Ahmed**, for all the support he has given to me. During my Ph.D. studies, I had to go through much hardship. My brother gave me shelter and food despite his financial situation. Otherwise, I would not have been

able to complete this thesis. May Allah give my brother a generous reward for all his scarifies.

I like to thank Professor **Masud Hasan** for encouraging and guiding me during my Master's research. I found his teaching to be very valuable throughout my research career. Without that knowledge, I would have never been able to complete this dissertation.

The highest point of my academic career was when I got admission in the Computer Science and Engineering (CSE) Department at the Bangladesh University of Engineering and Technology (BUET). It is the toughest department in Bangladesh to get admission. Every single student is selected through a rigorous admission test. I like to give a shout-out to my batch-mates of "**csebuet98**". Once I knew them as some of the most talented young men and women, now they have made their mark in their respective fields. I feel proud to be able to know such a group of extraordinarily gifted people. I also like to thank the BUET batch of 98, the "**Pothik**" for all the fond memories.

I had to fight against difficult situations and people throughout my life. I was very joyous when I got admitted to the BUET after a very competitive entrance examination. Our classes started on December 6^{th}, 1998. My father died on the next day, 7^{th} December 1998. Our family was in a terrible situation, and I was unable to study in that year. I did not lose hope I retook that admission test next year and was able to get a place among the top tier students.

I started my Master's in 2005. I thought it would be a new beginning for me. I was getting perfect scores in every subject. Then, my mother was diagnosed with terminal stage of cancer. I took again took a break from studies to look after my mother. My mother died in January 10^{th}, 2006. I after my mother died I fall into a terrible situation; however, with the help of Allah I was able to complete the thesis.

By the grace of Allah, in 2011, I received full scholarships for Ph.D. from three top universities in Australia the University of Sydney, University of New South Wales, and University of Melbourne. I am thankful to the authorities of all three universities for such generous offers.

Throughout my life, I fall into difficulties due to the actions of some heartless people. I did not receive help, on many occasions my ideas were stolen, and I was in terrible financial shape. People mocked me and told me that I would not be able to do anything. I continued research against almost everyone's wish and paid dearly for that. It is my sheer determination and the grace of Allah that has brought me through. As

Quran says: "*Allah does not burden a soul beyond that it can bear*"-[2:286]. Quran also assures us that Allah is aware of the actions of those heartless people: "*And never think that Allah is unaware of what the wrongdoers do. He only delays them for a Day when eyes will stare*"-Quran[14:42].

Lastly, I want to thank my parents and my brother again. Without them, I would not have survived this long or reached this far. They provided me with food and shelter, enabling me to carry on with my dream of higher education. As an orphan, I know no one else would have helped me. On the other hand, my decision to peruse higher studies instead of a decent earning career has only brought miseries to their lives. As a son and a brother, I have not been able to do much for them. I pray to Allah that my family receives better rewards than the sacrifices they made for me. "*My Lord! Bestow on them Your mercy, as they did bring me up when I was small*"-[Quran 17:24].

List of Publications

1. Maruf Ahmed and Albert Y. Zomaya, The Effect of Resource Allocation and System Events on VM Consolidation, *In Proceeding of 2017 IEEE International Conference on Cluster Computing*, CLUSTER 2017, Hawaii, USA, September 5-8, pp. 557-562, 2017.

2. Maruf Ahmed and Albert Y. Zomaya, An Automated Lightweight Framework for Scheduling and Profiling Parallel Workflows Simultaneously on Multiple Hypervisors, *In Proceeding of The Eighth International Conference on Cloud Computing, GRIDs, and Virtualization*, CLOUD COMPUTING 2017, Athens, Greece, February 19-23, pp. 16-25, 2017.

3. Maruf Ahmed and Albert Y. Zomaya, A Framework for Scheduling and Profiling Task Execution Time Variations of Parallel Workflow on Virtual Machines, *Presented session poster at Seventh ACM Symposium on Cloud Computing*, SoCC 2016, Santa Clara, CA, USA, October 5-7, 2016.
(`http://acmsocc.github.io/2016/poster-sessions.html`)

4. Maruf Ahmed and Albert Y. Zomaya, Profiling and Predicting Task Execution Time Variation of Consolidated Virtual Machines, *In Proceeding of The Seventh International Conference on Cloud Computing, GRIDs, and Virtualization*, CLOUD COMPUTING 2016, Rome, Italy, March 20-24, pp. 103-112, 2016.
Selected for the International Journal On Advances in Intelligent Systems
(`https://www.iaria.org/conferences2016/AwardsCLOUDCOMPUTING16.html`)

Other Accomplishments

1. The Australian Mathematical Sciences Institute (AMSI) sponsored Internship. Topic: Virtualization Research. Institute: Integrated Research, North Sydney, NSW 2060, Australia. (From 29-Oct-2012 to 20-Feb-2013).

2. Technical Program Committee member, *The Tenth International Conference on Cloud Computing, GRIDs, and Virtualization*, CLOUD COMPUTING 2019, May 5 - 9, Venice, Italy, 2019.
(`https://www.iaria.org/conferences2019/ComCLOUDCOMPUTING19.html`)

3. Chair and Coordinator, special track, PERF-CLOUD: Performance Monitoring and Analysis of Virtual Machines and Cloud, *The Ninth International Conference on Cloud Computing, GRIDs, and Virtualization*, CLOUD COMPUTING 2018, February 18 - 22, Barcelona, Spain, 2018.
(`https://www.iaria.org/conferences2018/CLOUDCOMPUTING18.html`)

4. Technical Program Committee member, *The Ninth International Conference on Cloud Computing, GRIDs, and Virtualization*, CLOUD COMPUTING 2018, February 18 - 22, Barcelona, Spain, 2018.
(`https://www.iaria.org/conferences2018/ComCLOUDCOMPUTING18.html`)

5. Invited Panelist, Smart Cities and Cloud Computing, *The Eighth International Conference on Cloud Computing, GRIDs, and Virtualization*, CLOUD COMPUTING 2017, February 19 - 23, Athens, Greece, 2017.

6. Chair and Coordinator, special track, PERF-CLOUD: Performance Monitoring, Tracing and Improvement of Applications on the Cloud, *The Eighth International Conference on Cloud Computing, GRIDs, and Virtualization*, CLOUD

COMPUTING 2017, February 19 - 23, Athens, Greece, 2017.
(https://www.iaria.org/conferences2017/CLOUDCOMPUTING17.html)

7. Technical Program Committee member, *The Eighth International Conference on Cloud Computing, GRIDs, and Virtualization*, CLOUD COMPUTING 2017, February 19 - 23, Athens, Greece, 2017.
(https://www.iaria.org/conferences2017/ComCLOUDCOMPUTING17.html)

Table of Contents

List of Figures

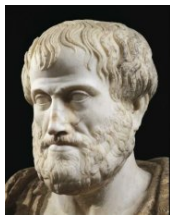
List of Tables

Chapter 1

Introduction

"Educating the mind without educating the heart is no education at all."

— Aristotle* (384-322 BC)



Cloud is one of the most innovative innovations of the last decade. It can fulfill the sudden or unexpected need of extra computing power often required by institutes and enterprises in a cost-effective way. The concept of Cloud rises from the demands from two opposite ends of the technological spectrum.

Small businesses and departments often find themselves in need of a significant amount of computing power. On the other hand, enterprises with a significant amount of hardware need to make good use of their resources. Cloud is the technology that offers benefits to the people on both ends of this spectrum. That is why it is one of the rare innovations that has generated interest among academia and enterprises alike.

Today, the Cloud provides various levels and types of services. However, every Cloud service is dependent on the *virtualization* in one way or another [5–12]. Virtualization divides the resources of a physical server into smaller units, which can be rented out as *Virtual Machines* (VMs). The Cloud users can rent VMs on demand and have complete control over the rented VMs. A physical server may host several VMs simultaneously with the help of a hypervisor.

*A Roman marble bust of Aristotle, copy of a Greek original; in the Museo Nazionale Romano, Rome. Image source: `http://www.abc.net.au/radionational/programs/scienceshow/a4.jpg/8106036`

The *Hypervisor* is a thin layer of software that creates the environment to run multiple VMs on a single host. The hypervisor manages both the VMs and shields the host from any malicious activities of the VM owners. One can customize the rented VMs according to one's requirements; however, the changes do not affect the underlying physical hardware.

The process of running multiple VMs on a physical server is commonly known as the *consolidation*. The VMs that are simultaneously running on the same physical server is known as the *co-located VMs*. In this dissertation, the interactions between the co-located VMs, their mutual performance interference, and related issues are explored.

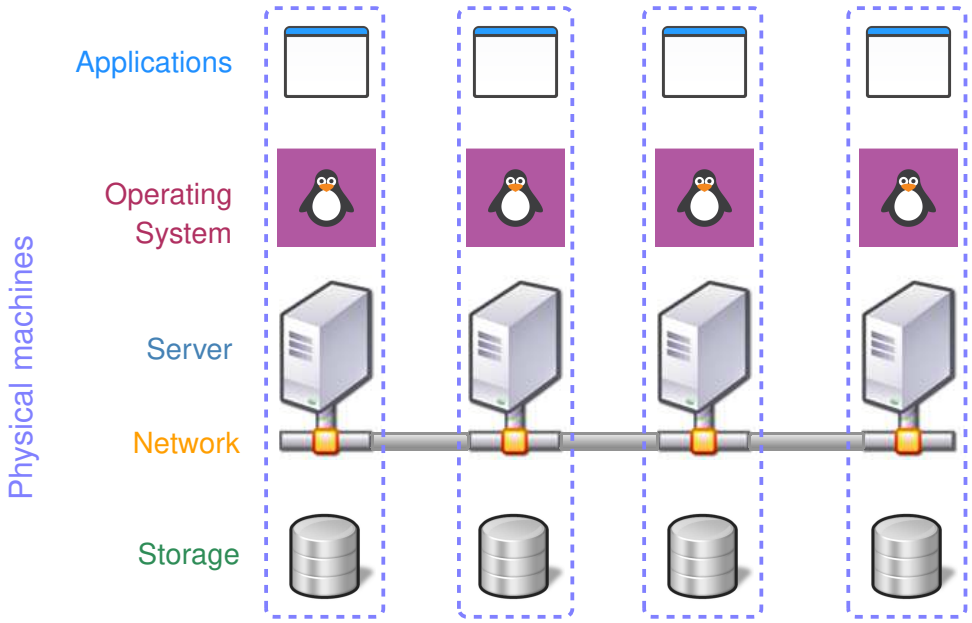
1.1 Motivation for the research

Virtualization is an essential part of modern-day data centers. Whether it is a large scale data center or just a departmental cluster, the virtualization is used everywhere. It not only enables data centers to provide essential Cloud services; rather it provides far more functionality.

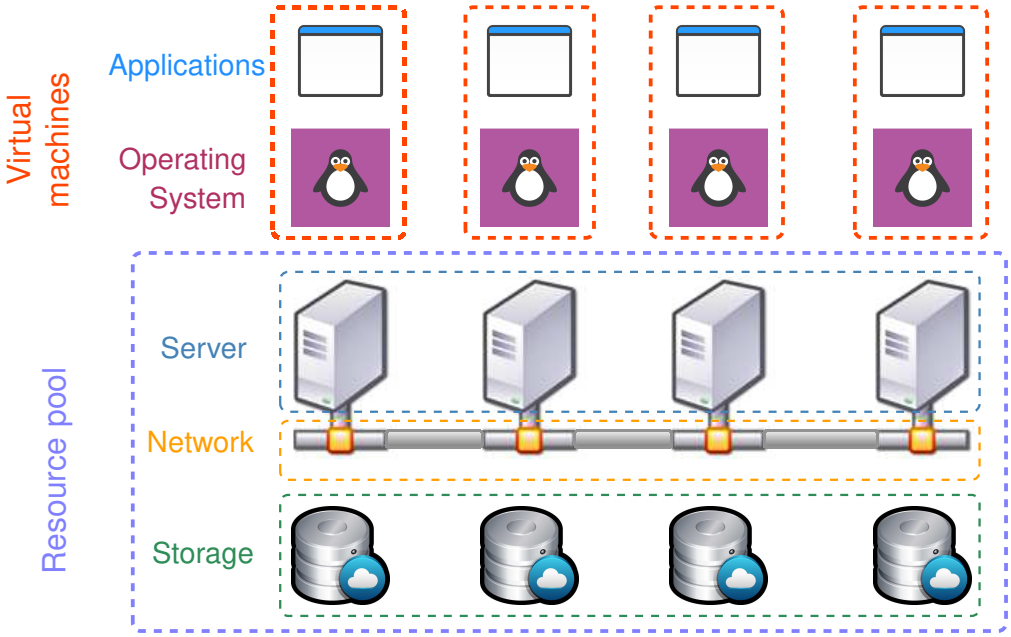
The Information Technology (IT) infrastructure has seen many transformations within the last decade. The schematic diagrams of Figure 1.1 demonstrates the difference between the tradition and modern IT infrastructure. Figure 1.1a shows the traditional IT infrastructure that was in place for a long time. After the mainframe computers were replaced by smaller and more efficient microprocessor technology in the late seventies, the traditional IT infrastructure had cemented itself into the enterprises around the globe. It mainly consisted of a cluster of physical machines connected through a relatively high-speed network.

There were several problems associated with the traditional IT infrastructure. First of all, enterprises had to go through a long and complicated procedure of procuring the hardware. That made it difficult to fulfill the sudden need for large-scale computing power. A decision about having a particular amount of computing power had to be made a long time ahead. Especially with the advent of the internet and web-based application, it became a huge inconvenience.

Furthermore, it was problematic for startup companies typically with a limited fund to own a large pool of physical resources. On top of those, the computing hardware



(a) Traditional IT infrastructure.



(b) Cloud IT infrastructure.

Figure 1.1: Schematic diagram of Information Technology (IT) infrastructure.

usually has a limit on service life. Most often, it is even more costly to administrate, maintain and upgrade the physical machines than purchasing them in the first place. As a result, operating and maintaining IT infrastructure was an expensive business throughout the eighties and nineties.

Although, with traditional IT infrastructure, enterprises have all the computing power at their disposal it was highly inefficient to own or operate. Often, the requirement for computing power in an enterprise is fluctuating, and that can lead to reduced efficiency and waste of energy.

Enterprises have been looking for a viable alternative to the traditional IT infrastructure, and that came in the form of Cloud. The new infrastructure is shown in Figure 1.1b. In this case, enterprises do not have to purchase any physical hardware, and computing power can be acquired on demand. The new IT infrastructure is seen as a pool of resources that can be rented and managed virtually.

Users can rent VMs according to their needs and use the resources as long as required. Thus, saving users from having to spend a fortune on the physical hardware. Also, Cloud makes it possible to rent computing resources on demand even for the enterprises on budget. Cloud provides a better mechanism for fault tolerance and high availability of services. Furthermore, it saves users from hardware maintenance and upgrading cost. Thus, the concept of Cloud is fundamentally different from the traditional one.

The secret behind creating a resource pool is virtualization. All the services of modern Cloud are directly dependent on virtualization technology, which makes it possible to divide physical resource into virtual units. The Cloud as we know it today would have never been possible without the help of modern virtualization technology.

From the earliest days of computing till today the primary purpose of the virtualization remains the same, resource sharing [13–22]. However, modern virtualization technology does much more than that. The high availability of Cloud services and fault tolerance mechanism of data centers are all dependent on the virtualization [23, 23–30]. Modern data centers would not be able to function the way it does today without virtualization.

Virtualization is widely used on newer clusters for various advantages. That means applications deployed in Clouds and clusters are mostly running on one or more VMs. The VMs have fundamental differences with the physical machines. One physical machine does not interfere with the performance of another one. However, simultaneously

running VMs do interfere with each other.

The co-located VMs share physical resources of the server with each other [31, 32]. The sharing creates resource contention for the VMs resulting in the performance variation. Measuring or predicting this performance variation is not a straightforward process. Various resources like CPU, memory and I/O are virtualized in different ways. Different virtualized resource behave differently during consolidation, and their effects on the consolidation are not the same [33].

Consolidation is used in all modern servers; the resource utilization depends on how many VMs can be run on a server without creating too much resource contention [32]. As data centers are getting bigger and consuming more energy the resource utilization efficiency is becoming a more pressing issue. For increasing the resource utilization efficiency of the servers, it is essential to study the reaction of the consolidated VMs.

What is more, existing scheduling algorithms are designed for physical machines [34]. As more and more servers are getting virtualized; it is essential to study the performance variation of the consolidated server to determine how to make the scheduling algorithms more efficient in the virtualized environment.

1.2 Contributions

Virtualization is essential for providing Cloud services. This dissertation deals with analyzing and improving the performance of consolidated VMs. Many important issues like resource usages and energy efficiency depend on the consolidated VMs performance. The primary research contributions of this dissertation are summarized below.

1. All experiments are conducted on real virtualized systems. Results from three well known hypervisors are used throughout the dissertation; they are *VMware ESXi* [35–37], *Citrix XenServer*, and *Xen* [38, 39]. Multiple VMs are set up on each hypervisor and experiments are done with real workloads. The experimental data are collected from both the VMs and hypervisors for analysis.

2. Real system data is used throughout the dissertation, and no simulation is used in any experiment. There are fields where simulation provides excellent results; however, virtualization system research is not one of them.

In each chapter of the dissertation, different experiments have been designed and carried out with consolidated virtual machines to collect system data. For example in chapter 4 experiments are conducted to collected CPU, memory, and I/O usages data of the VMs from the hardware. Based on the analysis of this data further experiments have been designed for chapter 5.

In chapter 5, the task execution finish times collected from the *Operating System* (OS) is used analysis. Similarly, in chapter 7 experiments are designed to collect system events and interrupt data from the hardware. Thus, all the chapters are logically connected, and analysis of experimental system data of one chapter is the basis for the experiments of the subsequent chapters.

3. Experiments are conducted with the resource consumption and performance variation of VMs. It is known that the consolidated VMs of a server interfere with the performance of each other [31, 32, 40–42]. Simultaneously running VMs of a server compete with each other for three primary computing resources; CPU, memory, and I/O.

   Furthermore, different VMs use the resources differently; hence, their contribution to resource contention is different. Experiments have been done with various resource usages pattern and number of consolidated VMs. The experimental results demonstrate how the VM performance is affected by different types of resource contention.

4. A benchmarking technique is presented to analyze the VM consolidation performance, called the *Incremental Consolidation Benchmark Method* (ICBM) [43]. In this method, the numbers of various resource intensive VMs are increased systematically. In turn, it increases the resource contention in the system and the execution finish times of tasks start to rise.

   As the increasing number of VMs compete for the resources, some VMs are deprived of the resources and can not continue to execute properly. It has consequences on the tasks running on the affected VMs. As a result, the tasks on the VMs take longer to finish execution. Therefore, in the ICBM, the number of consolidated VMs are systematically manipulated to observe their effect on the overall performance.

5. In this dissertation, the task execution time is used as a performance metric. The experimental results show that the task execution time variation can be a good indicator of the consolidation performance. The resource usage intensity varies from task to task.

   First of all, several tasks are chosen for the experiments whose resource intensities are known; then, they are combined to create combinational workloads. The tasks are combined in several patterns to create several workloads. The workloads are run on three hypervisors to collect the data about task execution time variation. The collected data is then compared and analyzed. The experimental results show that different resource intensive tasks react differently to the VM consolidation.

6. To efficiently experiments with the virtualized system, a VM workflow scheduling framework is developed [44]. It is an application entirely written in Java. It can schedule parallel tasks and workflow on multiple VMs. Also, once the task execution is finished, the application can retrieve the data for later analysis.

   The scheduler can simultaneously connect to multiple hypervisors and run workloads concurrently. The workloads are defined in a separate file, which defines which task to run at which VM and in what pattern. Executing the tasks in specific patterns is the key to analyze the execution time variation of consolidated VMs.

   The framework is built with the objective to apply the ICBM on multiple hypervisors simultaneously. A server can run many VMs at a time, and it is hard to manage all the execute tasks manually. The new scheduler can execute workloads with a large number of VMs automatically.

7. Experiments have been conducted with various workload and hypervisors. The results show that the proposed methodology, ICBM is a powerful tool for analyzing resource contention of VMs [45]. Furthermore, experiments have been conducted with a real-world parallel workflow to test the effectiveness of ICBM.

   Data collected through the ICBM can also be used to predict the task execution time variation due to resource contention of VMs. First, a set of workloads is run and their execution finish times are profiled. Then, the profiled-data is used

to build a prediction model, which is then used to predict the execution time variation of another set of workloads.

The profiled-data can be used with any machine learning technique to build a prediction model. In this dissertation, two techniques are used to build models, the *Least Square Regression* (LSR) and *Artificial Neural Network* (ANN). The accuracies of the prediction models from both the techniques are also presented and compared.

1.3 Structure of the dissertation

This chapter is the first chapter of the dissertation; it is the introduction. The rest of the chapters are organized as described below.

- Chapter 2 discusses the background of the virtualization. It describes the origin and development of virtualization technology over the decades. Also, the classification of the virtualization technology is given. There exist many classifications of virtualization, it is important to know how they are different and which ones are important for the Cloud and data centers.

  The chapter also discusses how the various computing resources can be virtualized. The benefits of using virtualization in the Cloud and data centers are discussed. Basics of Grid computing and how Cloud emerged from Grid technology are discussed. Furthermore, a classification of various types of Cloud also provided.

- Chapter 3 starts the experiments with consolidated VMs. The chapter discusses the importance of the research. In the chapter, experiments have been conducted with nine different benchmark suites; they are also described in the chapter. The benchmarks can be divided into four categories based on their resource usage intensities.

  The first three categories are CPU, memory, and I/O intensive benchmarks. The fourth set of benchmarks are multiple resources intensive, that is they use several types of resources at the same time. Those benchmarks are concurrently executed in consolidated VMs. In this chapter, experiments are conducted on the Xen hypervisor. The resource usage data of the consolidated VMs are collected

after the experiments have finished, they show how the resource usage pattern of VMs changes with the changing number of consolidated VMs. The results of this chapter also helped to design experiments for the rest of the chapters of the dissertation.

- Chapter 4 introduces and discusses the new consolidation benchmarking technique, called the ICBM [43]. The chapter describes how the ICBM is applied to the VMs of a hypervisor. In this case, Xen hypervisor is used to run the VMs. The ICBM can be divided into several steps; those steps are described in the chapter.

  For the experimental purposes, the co-located VMs are divided into two categories, target and co-located VMs. The target VM is monitored for performance variation. On the other hand, the co-located VMs are used to create resource contention. The number of co-located VMs are increased in the system that is what creates the resource contention.

  The various workloads are run on the co-located VMs and execution time variation of the target VM is collected. The profiled data are used to train LSR models. The models are then used to predict the execution time variation of the target VMs. During training and testing, separate workloads are used in the co-located VMs. Thus, the combination of co-located VMs used for the testing is different from those used for the training.

  The execution time variation of the target VM depends on the resource contention created due to the co-located VMs. Thus, a different combination of co-located VMs affects the target VM execution time differently. Data is collected for a selected combination of co-located VMs, and the model is trained. The experimental results show that the LSR model can predict the execution time variation of the target VM for other combinations of co-located VMs.

  This chapter contains material that was published as [43]. The author of this dissertation has done all the works related to publication. The author has designed and conducted the experiments, collected and analyzed the data, and wrote the draft for the paper. Since the publication, the author has also added more experiential results and data to the chapter.

- Chapter 5 presents a framework for scheduling and profiling the parallel workflows on VMs. Also, the concept of ICBM is extended to the parallel workflows. In the previous chapter, the ICBM is introduced for analyzing the performance of a group of VMs. The ICBM runs a group of tasks in the VMs according to some predefined patterns. Those patterns are stored in a file beforehand; the scheduler uses the pattern to run the tasks on VMs and collect their performance variation data.

  This chapter extends the concept of ICBM to the parallel workflows. A parallel workflow consists of many individual tasks; this chapter shows that the ICBM can be applied to those tasks, too. The framework is designed with several goals in mind those are described in this chapter. The implementation of the framework is divided into several modules, which are described in this chapter.

  Finally, the experiments are conducted with a real-world parallel workflow. The framework executed all the tasks of the workflow and collected the execution time variation data.

  This chapter was previously published as [44]. The author of this dissertation has designed and conducted all of the experiments. The author also has collected and analyzed the experimental data and compiled the draft of the paper.

- In Chapter 6, the experiments are done with the ICBM and multiple hypervisors. Three hypervisors are set up for the experiments, they are the *ESXi*, *XenServer*, and *Xen*.

  The experimental results show that the execution time variation is not dependent on the location of the VM on the server, rather it is dependent on the total number of VMs. That is the VM performance is mainly affected by the total number of VMs on the system. If the total number of VMs is increased in the system, the performance degrades, on the other hand, if the VM number is decreased the performance enhances; thus, the relationship is reciprocal.

  Various resource intensive benchmarks are run on three hypervisors and execution time variation is profiled. The number of co-located VMs are changed in the server to vary the resource contention in the system; this resource contention is responsible for the execution time variation of VMs.

Data collected from the three hypervisors have been used to build LSR models, which can predict the VM execution time variation on multiple hypervisors. Thus, showing that the LSR models can be generalized for multiple hypervisors. Furthermore, an ANN model is used to predict the execution time variation of a parallel workflow. The tasks of the workflow are profiled using ICBM, and the collected data is used to train an ANN model. Then, the ANN model is used to predict the execution time variation of the workflow tasks for a various number of co-located VMs.

This chapter contains material published in [46]. The author of this dissertation has designed all the experiments and collected the data. The author has analyzed the data and wrote the draft for the paper.

- In Chapter 7, further experiments are conducted with the ICBM. In this chapter, investigations are conducted with the effect of VM memory allocation on VM consolidation. Various tasks are used in the experiments; each task with a different memory requirement is used.

  Experiments are conducted with three hypervisors; the results show that memory allocated plays an integral part in the consolidated VMs performances. Experiments are also done with the system events, like interrupt and page-faults to find their effect on the VM consolidation performance. The system event data are collected from three hypervisors and used to build an ANN model for predicting the VM execution time variation. The model is also used to predict the execution time variation of the tasks of a workflow.

  This chapter is previously published as [45]. The author of this dissertation has designed and conducted the experiments, collected and analyzed the data, and wrote the drafts of the paper.

- Finally, Chapter 8 concludes this dissertation with a summary and closing remarks.

1.4 Research methodology

The research methodology used in this dissertation is based on the *Design Science Research* (DSR), which focuses on creating artifacts to solve a problem [47–49]. In

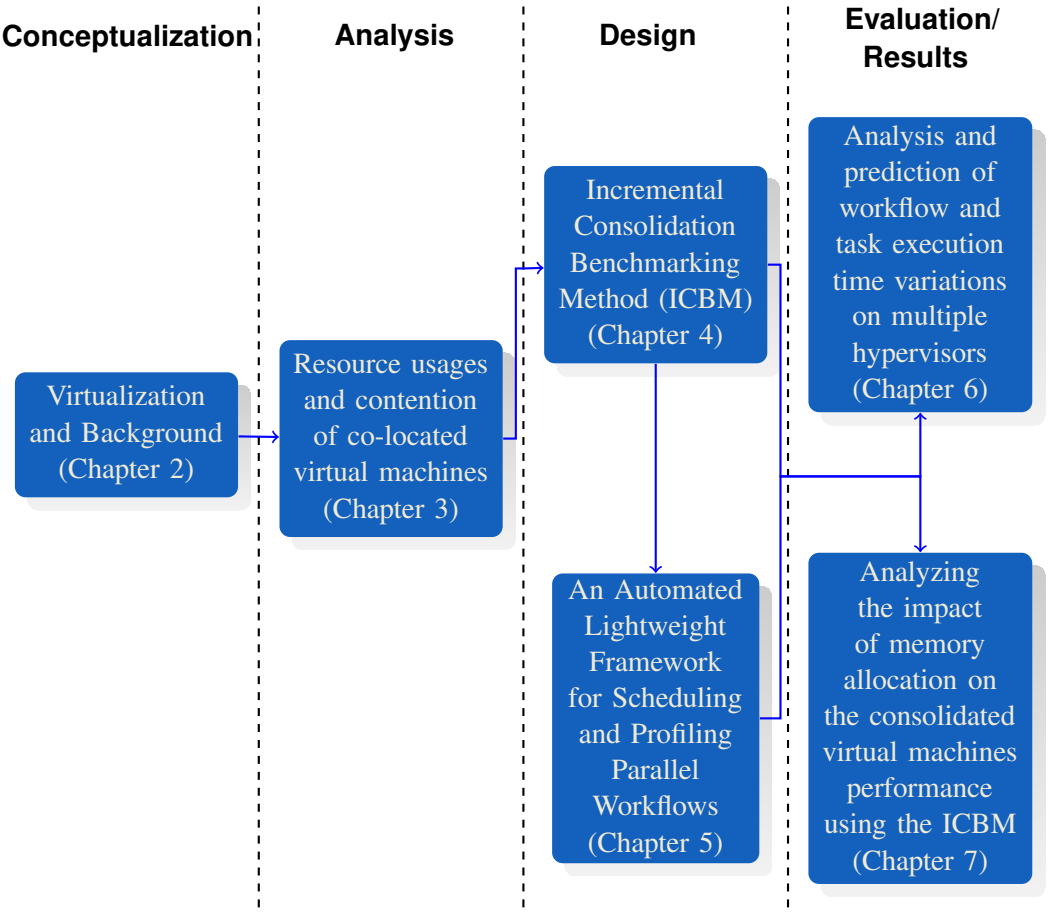


Figure 1.2: Research methodology of the dissertation.

DSR, a produced artifact can be a construct, a model, a method, or an algorithm. The methodology developed in Chapter 4 (ICBM) can be considered as an artifact for this dissertation.

Design research can be applied in many fields of research including algorithms and computer interface design [50]. It consists of formulating a design or a method, conducting experiments and evaluating results [51].

In this dissertation, the research process followed is by based on the guidelines proposed by Henver [51]. The design process of this dissertation is divided into four phases, they are *Conceptualization*, *Analysis*, *Design*, and E*valuation/Results*. Figure 1.2 shows how the chapters of the dissertation are divided into four DSR phases.

- The first phase is the conceptualization. Chapter 2 introduces virtualization to readers and demonstrates the importance of virtualization concerning modern Cloud. Different aspects of virtualization are introduced, and their prominence

is discussed briefly. Discussion of Chapter 2 is essential for understanding how the experiments are conceptualized in this dissertation.

- The next phase is analysis. In this phase, experiments are performed to collect performance data from consolidated VMs. Analysis of data collected from Chapter 3 played an essential part in designing the experiments for the rest of the dissertation. The experiments in this section give a valuable insight into how the performance of various resource intensive VMs is affected by the consolidation.

- In the third phase, a methodology is designed to systematically inspect the effect of VM consolidation on the server performance. It is called the *Incremental Consolidation Benchmarking Method* (ICBM) and described in Chapter 4.

  After reviewing the methodology of Chapter 4, a need was felt to design a new framework that can apply the steps of methodology in a large scale experimental setup. Therefore, a framework is designed and implemented in Chapter 5 to perform experiments with a large number of VMs. Thus, in this phase, a methodology and a framework are designed to conduct experiments with VM consolidation.

- In the fourth phase, the effectiveness of the presented method and the framework are tested on various hypervisors. Chapters 6 presents the experimental results on three different hypervisors. Data collected during the experiments are also used to build prediction models using machine learning methods.

  In Chapter 7, ICBM is used to conduct experiments with VM memory allocation and consolidation. Thus, the effectiveness of the presented methodology in the dissertation is evaluated in this phase.

Now that, the structure and research mythology of the dissertation are discussed, it is time to introduce the basic concepts of virtualization. In the next chapter, the importance of virtualization and Cloud computing is discussed. The next chapter is also crucial for understanding how the problem of consolidated VMs performance interference is conceptualized and experiments are conducted in this dissertation.

Chapter 2

Virtualization and Background

"I was born not knowing and have had only a little time to change that here and there."

— Richard Phillips Feynman* (1918-1988)

2.1 Introduction

Virtualization is a cunning technology, which has roots back to the early days of mainframe computers [13–22], The mainframe are the predecessor of the modern *High-Performance Computers* (HPC). Virtualization has a fascinating origin and history. This chapter discusses the beginning and development of the virtualization technology. Also, the application of virtualization in the context of modern Cloud computing is discussed. Categorization of the Cloud and virtualization technology are described as well.

2.2 Virtualization

Discussion of this chapter begins with virtualization, which allows multiple *Operating Systems* (OS) to run on a single physical machine [52]. There is no standard definition of virtualization; it is described differently by different researchers.

*Image source: `http://www.nndb.com/people/584/000026506/`

One of the earliest definition of virtualization was like this "*to present the illusion of many smaller virtual machines (VMs), each running a separate operating system instance*" [53]. Later, VMware defined virtualization as follows "*The term virtualization broadly describes the separation of a resource or request for a service from the underlying physical delivery of that service*" [54]. In 2007, IBM defined virtualization as "*virtualization is a technique for hiding the physical characteristics of computing resources from the way in which other systems, applications or end users interact with those resources*" [55].

The definition of virtualization has not been updated or standardized in years. In this dissertation, the virtualization is defined as follows:

Definition 2.1 *Virtualization is the process of creating an isolated execution environment that is either vastly or entirely different from the underlying (physical) environment.*

Where, "*execution environment*" refers to a group of resources and settings that enable an application or a set of applications to execute properly. Virtualization of resources is most often achieved with the helped of a thin layer of software or firmware that is layered on top of the physical hardware directly or another software layer.

The thin layer of software, which enables the virtualization process is known as the *hypervisor*. VMware defines hypervisor as follows "*A thin layer of software that generally provides virtual partitioning capabilities which run directly on hardware, but underneath higher-level virtualization services. Sometimes referred to as a "bare metal" approach*" [54].

2.2.1 Origin of hypervisor

It might be surprising to some readers to learn that the term "hypervisor" is older than the term "Operating System" [56, 57]. Prior to *IBM System/360* there [13–16, 58] was no "operating system" available for cluster computers. All the systems used to have a "*supervisory program*" or "*supervisor*" program to control the execution flow of other routines, work scheduling, input/output (I/O) operations, error actions and to perform other similar actions. In modern computers, similar actions are performed by the *kernel* [57, 59, 60]. Even today, the *kernel mode* of modern processors is also called the *supervisor mode*.

The *IBM 7080* was introduced in 1961 [61, 62], it was the part of hugely successful *IBM 700* series of computes. The *IBM 701* was the first commercially available scientific computer and the large-scale electronic computer that was produced for the general market [61]. All previous large-scale computing systems were designed and produced for a company or organization specifically.

The IBM 7080 become hugely popular, and various organizations widely used it. Later, when the IBM System/360 was released in mid-sixties, many people were still using the code developed for IBM 7080 and some other older machines. The IBM System/360 model 65 had an emulator to run the codes of legacy systems like that of IBM 7080. However, the only problem was that the system could run in only one mode at a time, either in IBM System/360 or IBM 7080 mode.

That encouraged the engineers in IBM to introduced an ingenious solution to run the IBM System/360 system in both modes at the same time. According to the plan, the memory space of the system was divided into two parts, the lower and upper half. Either part of the memory was able to run in IBM System/360 or IBM 7080 mode. However, the IBM System/360 model 65 had only one set of registers; thus, the register values needed to be swapped between the modes. A small code was added to facilitate the register value swapping process. The added code was referred to as the "hypervisor" by the IBM engineers.

The term hypervisor is a variant of the term supervisor. The 'supervisor' is a program used in early generation IBM clusters to control other programs and interact with hardware. The word "*hyper-*" is a stronger variant of word "*super-*". In the context of early cluster computing systems, it had been used to indicate that the hypervisor is the "supervisor of supervisor", a unique application that has control over the supervisor. Both words have the same meaning; "*super-*" is a Latin term meaning "above", while "*hyper-*" is an Ancient Greek term meaning "above".

Not to mention the virtualization technology even predates the modern microprocessor technology, which became hugely popular in the early seventies. The first commercially available microprocessor, the Intel 4004 was introduced in 1971 [63, 64]. The microprocessor technology was set in motion by Intel, a different company with different goals. Thus, virtualization has close ties to the early generation of mainframe computers that existed before even the microprocessors came into existence.

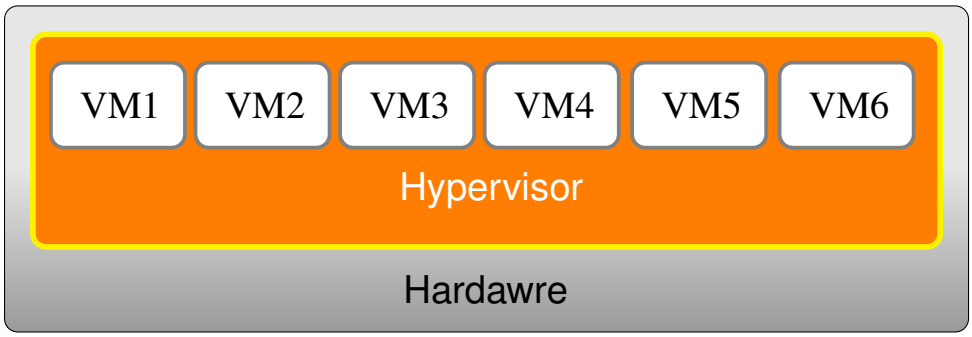
Although modern hypervisors are much more involved in design and implementation, they still bear the same name as that of their ancestors. Modern hypervisor is

also known as the *Virtual Machine Monitor* (VMM) [65–68]. Hypervisors play an essential role throughout this dissertation. All the experiments here are designed to be conducted directly on hypervisors.

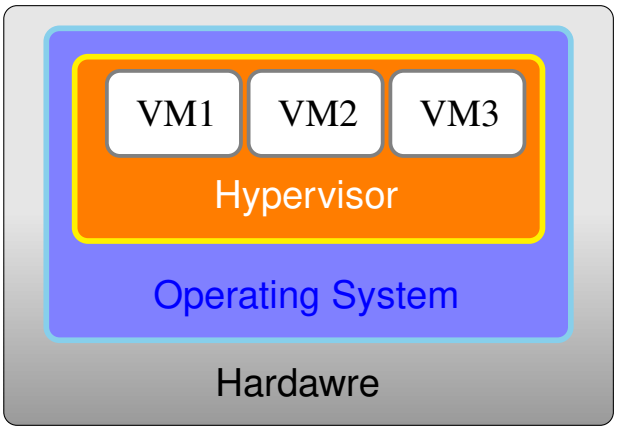
Several well-known hypervisors have been used in the experiments and data collected from the hypervisors are also used for analysis. Various types of experiments are conducted in the different chapters. More discussion about those experimental processes can be found in the respective chapters.

2.3 Types of hypervisor

Hypervisors can be categorized into two main types based on their location specific to the OS and hardware. They are **type-1**, and **type-2** hypervisors [69]. A type-1 or bare-metal hypervisor is installed on the hardware and controls all the hardware resources



(a) Type-1 / Bare-metal hypervisor



(b) Type-2 / Hosted hypervisor

Figure 2.1: Types of hypervisor.

directly. On the other hand, a type-2 or hosted hypervisor is installed on an OS and is dependent on the OS for accessing the resources. Both types have advantages and disadvantages; they are used for different purposes. Two types of hypervisors are shown in Figure 2.1 and discussed next.

2.3.1 Type-1 / Bare-metal hypervisor

A type-1 hypervisor is the most efficient for concurrently running multiple VMs on a single server. It is installed directly on the physical hardware, and it controls all aspects of the VMs. A type-1 hypervisor may be equipped with CPU schedulers, resource distributor, and other components to share the resources among the VMs efficiently. In this case, the hypervisor takes full control of the hardware and uses the entire physical resources to run VMs concurrently. It is well-suited for Cloud and data centers.

A type-1 hypervisor may have built-in one or more CPU schedulers that can dynamically share CPU resources among the VMs. Similarly, the hypervisor distributes memory and I/O resources among the VMs. A type-1 hypervisor is typically optimized for running multiple VMs on a server. For the optimization process to work correctly, the system must be booted in hypervisor supported mode. In data centers and clusters, it is widely used. Examples of type-1 hypervisor includes *Xen* [38, 39], *XenServer*, *VMware ESX* [35–37], *KVM* [70–73], and *Hyper-V* server [74, 75].

2.3.2 Type-2 / Hosted hypervisor

The type-2 hypervisor is installed on top of the OS, and the system is not required to boot into the hypervisor mode. To the host OS, the hypervisor appears just like another application. In this case, the hypervisor does not access the hardware directly, and resource sharing among the concurrently running VMs is far less efficiently compared to that of type-1. It is usually not deployed for Cloud or used in data centers. It is mainly used on desktops computers to run a second or even a third operating system. In this way, the user can use two or more OS without rebooting the system. Examples include, *Oracle virtual box* [76] and *VMware workstation* [77, 78].

All the experiments of this dissertation are conducted with type-1 hypervisor as it is predominantly used in data centers. Cloud systems are usually built upon type-1 hypervisors. That is why type-1 hypervisors are chosen for the experiments.

2.4 Virtualization and x86 Architecture

The x86 family of processors is predominantly used for both personal and commercial purposes. Data centers are no different, most of the processors used for providing Cloud services are from x86 family. Therefore, all experiments in this dissertation are conducted with x86 compatible processors. Next, the virtualization process of the x86 processor family is discussed.

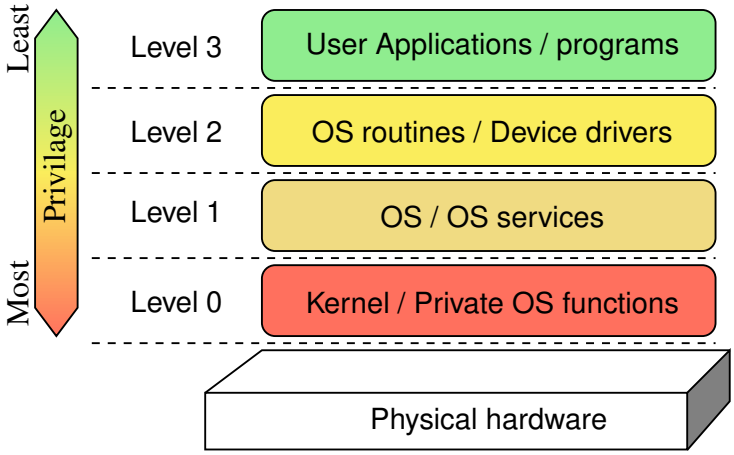
In the modern generation of processors, a hierarchical protection domain or protection ring system is used to protect data and processes from faults and malicious instructions [79]. Originally, x86 architecture was designed long before the modern virtualization hype and was never intended to be used with virtualization technology [80–83]. Hence, the x86 processor design had no hardware support for virtualization at all. When the virtualization became prevalent in the last decade or two, the software engineers and developers had to come up with an ingenious solution to virtualize the x86 architecture.

The x86 processors have four privilege rings or protection domains, which are numbered from 0 to 3 [84]. Ring-0 is the most privileged domain while the ring-3 is the least privileged one. Figure 2.2a shows how the rings are usually employed when no virtualization process is involved.

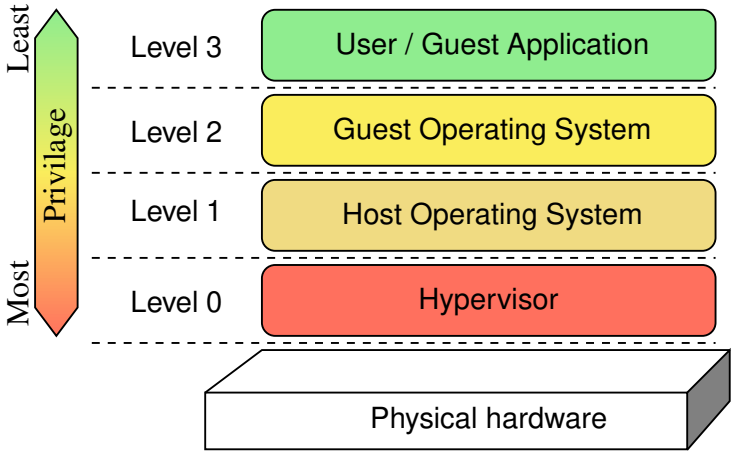
There is no hard and fast rule to use the protection rings. Two possible schemes for those rings are shown in Figure 2.2a. In the figure, Ring-0 is the highest privileged level, which either occupied by the kernel or a set of special OS functions. This ring is used for providing the core functionalities of OS.

Next, ring-1 hosts the rest of OS or OS functions that are less important than the kernel functions. A modern operating system is a massive piece of software, and not all functionalities are as important as the kernel functions. The less critical services are sometimes hosted in ring-1 to reduce operational overhead.

If a system resource is restricted to a lower number ring, then all the processes running on the higher numbered rings must send an access request to the lower level ring process through a trap or a fault. A system fault or trap initiates a context-switch process, and the system enters into the kernel mode to provide the service. If multiple processes on higher level rings send requests simultaneously, then the kernel decides which one gets the access first. The context-switching process has high overhead, and OS designers usually try to reduce the number of such overhead as much as possible.



(a) x86 protection rings as used by operating systems.



(b) Type-1 virtualization with x86 protection rings.

Figure 2.2: Hierarchical protection domains/rings of x86 architecture and virtualization.

The philosophy behind using ring-1 is that if some part of the OS can be placed on a higher level ring, then it can save the system from having to enter the kernel mode at least some of the time.

Next, ring-2 hosts the device drivers or OS routines that provide such services. The device drivers directly communicate with OS to operate various devices. Lastly, user applications are placed in level 3, the lease privileged domain. Thus, a user-level application is entirely dependent on lower level processes for providing system services.

When the concept of privileged rings was instantiated in the early seventies, system faults or traps event were costly for processors to handle [79]. Modern processors are

much faster compared to previous generations of processors and have special mechanisms to serve multiple traps or faults more efficiently. Nowadays, most kernels can perfectly operate without utilizing all four privilege rings. In fact, most modern kernels utilize only two privileged levels, 0 and 3 [85].

Next, Figure 2.2b shows how a type-1 hypervisor utilizes the rings on the server. The type-1 hypervisor is widely used in data centers and the most efficient way to run multiple VMs on a server concurrently. Throughout this dissertation, the type-1 hypervisor is used for conducting experiments. As the figure shows, type-1 hypervisor must be placed in ring-0 as it requires the highest privilege in the system. The host operating system (of the server) is placed in the ring-1; thus, having the second highest privilege. Hypervisor still has higher precedence than that of the host operating system and ultimately controls all the system resources.

The guest operating system (on VM) is placed on the next available ring, and it has lower privilege than both the guest operating system and hypervisor. Finally, applications on VMs are run in level 3, which is the least privileged of all the levels.

As mentioned above, a system does not need to use all four rings to operate, and a non-virtualized system can operate perfectly on two or three privilege levels. However, when virtualization is implemented the guest OS, host OS, and hypervisors are generally placed on the rings as shown in Figure 2.2b.

After the advent of Cloud computing and reincarnation of virtualization technology, the x86 architecture has been modified to accommodate an extra privilege level [81–83]. All modern x86 processors (produced within last decade or so) have a new privilege level, which is commonly referred to as “Ring -1”. In hardware-assisted virtualization, the hypervisor utilizes the new ring to operate at the most privileged level. The extra ring makes it possible for hardware-assisted hypervisors to run unmodified operating systems on the VMs.

2.5 Categorization of virtualization methods

In an x86 system, virtualization can be implemented at different levels and ways to serve different functional purposes. Various researchers have classified virtualization technology differently; however, there is no consensus among the researchers. In this dissertation, virtualization methods of x86 processors are divided into six categories

based on the purpose of their use. In this dissertation, the categorization is made depending on at what level the virtualization is applied and what type of service it has to offer to the system.

The virtualization technology is not monolithic rather it is a highly dynamic one. It has a different meaning to different people, and it can be used in a wide range of scenarios. A categorization of virtualization is necessary to know which category is essential for data centers and Cloud.

In this dissertation, virtualization is divided into six categories according to levels and types of services they offer. The six categories are as follows:

1. *Emulation*,
2. *Full virtualization*,
3. *Para-virtualization*,
4. *Hardware assisted virtualization*,
5. *OS-level virtualization*, and
6. *Application level virtualization*.

Table 2.1 compares the important characteristics of all six virtualization categories. The categories of virtualization are briefly discussed below.

A full discussion about the categories of virtualization is out of scope for this dissertation. The categories are introduced here to make it clear to users that the virtualization technology has many applications and functions. In the context of this dissertation, it is necessary to know what type of virtualization is essential for data centers, so that experiments can be designed and conducted accordingly.

2.5.1 Emulation

In this case, the hypervisor emulates all the components of the hardware. Every hardware component is emulated for the VM, and any OS can be ported to run on the VM without significant modification. Emulation is used for running OS with one system architecture, on a machine with another system architecture.

The instruction set of the guest system can be completely incompatible with that of the host. It is the most costly form of virtualization as it requires to translate all the

Table 2.1: Comparison of virtualization categories.

Virtualization category	Operation overhead	VM dependent on	Hypervisor operating level	Consolidation of many VMs	Examples
Emulation	High	Nothing	Usually over OS	Infeasible	QEMU, Bochs, VirtualPC
Full	Medium	Host H/D Architecture	Usually over OS	Feasible	VMWare Workstation, Parallels
Para	Medium to low	Host H/D Architecture	Below OS	Most feasible	Xen, Citrix XenServer, VMWare ESX Server
Hardware assisted	Medium to low	Special host H/D support	Usually below OS	Most feasible	Intel VT-x, AMD-V
OS-level	Usually low	Shared host OS libraries	Closely connected with OS	Only for the same OS version	Solaris Containers, BSD Jails, Docker
Application level	Usually low	Shared host application libraries	Over OS	Only for the same application	JVM

guest instructions into native instructions. The hypervisor is responsible for the binary translation of all guest OS instructions to make them compatible with host hardware.

Emulation is mainly utilized for the development of new systems and debugging. With this type of virtualization, an OS can be ported to any machine even with entirely different system architecture. *QEMU* is a popular example [86]. Other examples include *Bochs* and *VirtualPC*.

2.5.2 Full virtualization

Full virtualization is possible if the architectures of the physical host and virtual machine are the same. It uses just "enough" virtualization so that unmodified OS can run on the VM. In this kind of virtualization, hardware architecture swap is not possible for the OS. The IBM VM family fall into this category. The VM-CMS is a family of virtual machine operating systems developed for IBM mainframe computers [87–98]. They are the ancestors of the modern hypervisors, both sharing many characteristics.

The *Conversational Monitor System* or CMS is one of the earliest examples of software that acted as a hypervisor. Many of the design principles of modern hypervisors are invented initially for the VM-CMS family. Two modern examples of full virtualizations are the *VMWare Workstation* [77, 78] and *Parallels*.

2.5.3 Para-virtualization

Para-virtualization is by far the most common form of virtualization deployed today. In this case, the hardware simulation is not used for VMs; however, the guest OS inside the VM has to be modified. The guest OS uses *hypercalls* to interrupt the hypervisor. In this model the guest OS is trapped by the application then the hypervisor is trapped by the OS. The application uses the *syscall* to interrupt the OS, then, in turn, OS uses the hypercall to interrupt the hypervisor.

The guest OS needs to be modified and can not access the underlying hardware. An application can send a service request to the guest OS through syscall. Instead of servicing the interrupt by itself the modified OS interrupts the hypervisor through the hypercall. In this case, OS must be aware of uses the hypervisor API and the guest OS is modified to accomplish this.

In modern data centers and clusters, this type of virtualization is widely used. Examples include Xen [38, 39, 53], XenServer [99] and VMWare ESX Server [35–37].

2.5.4 Hardware virtualization

Hardware virtualization technology [100] transfers some of the hypervisor responsibilities to the hardware. One of the problems with the para-virtualization is that the OS needs to be modified. The modification is only possible for open source OS. Proprietary software like the Windows is not open source and this is a problem for the

para-virtualization technique. To overcome such problem, the hardware manufacturers have introduced additional features to the hardware to assist virtualization. In this case, the hardware is aware of the hypervisor and interrupts from the guest OS are transferred to the hypervisor automatically.

Initially, the x86 family of processors was not designed for virtualization, and it became a significant drawback after the advent of Cloud. The hardware assistance virtualization was introduced to overcome this issue. In this case, the hypervisor gets direct support from the hardware. Both modified and unmodified OS can be run on the hardware-assisted virtualization. Examples include the *Intel Hardware-assisted Virtualization* (Intel VT-x) [82] and *AMD-V* [101].

2.5.5 OS-level virtualization

In this case, the primary focus of the virtualization is in the OS. A single instance of OS is divided into multiple partitions. Each partition acts as a fully functioning OS. This case, only one kernel is active, and it provides services to all the OS partitions. Nonetheless, all the partitions are isolated from each other. Applications running on one partition is unaware of the other partitions.

In this case, guest OS is the same as the host OS, and only appears to be isolated. Any other type or version of OS cannot be run on the guest. Examples include *Solaris Containers* [102], *BSD Jails*, *Linux Vserver* [103] and *Docker* [104].

2.5.6 Application level virtualization

This type of virtualization is applied at the application level. Each application gets a full execution environment, and they do not share system variables. Each application controls own separate copy of the components like own registry files and global objects. The application level virtualization allows applications to avoid deadlocks and conflicts over the resources.

Multiple application level VMs can be run on one host OS. Multiple copies of the application and runtime environment can be made and run in parallel. The main advantage is that the virtualization overhead is low. The tasks run on the application level VM, and the VM communicates with the OS. The *Java Virtual Machine* (JVM) [105, 106] is an example of application level VM.

In data centers, mostly para-virtualization and hardware-assisted virtualization are used. Both techniques are the most efficient for running multiple VMs concurrently on a server. In this dissertation, experiments have been done with hypervisors that employ both para-virtualization and hardware-assisted virtualization technology.

2.6 Virtualization of different resources

Computing systems usually require three types of resources to operate, they are a) CPU, b) memory, and c) I/O. A virtualized system is no different. Three primary computing resources need to be virtualized for VMs to operate appropriately. Those three resources are fundamentally different and need to be virtualized in different ways [107]. Their virtualization processes are described below.

2.6.1 CPU / Processor virtualization

The hypervisor distributes CPU resource among the VMs almost as the same way the OS allocates CPU cycles among the processes. When the VMs are created, each VM is assigned one or more *Virtual CPU* (vCPU). The hypervisor allocates time-slices to each vCPU; essentially it is just CPU time sharing among the VMs. Each VM is allowed to run for an assigned time slot, and the VM runs on the CPU for the allocated period. When the period is over, the VM is stopped, and the registers and CPU states are saved. Then, the registers and CPU states are swapped with another VM, and the new VM is allowed to run for the allocated period [108].

The hypervisor may use various algorithms to share CPU among the VMs. CPU virtualization is very efficient and has low virtualization overhead.

2.6.2 Memory Virtualization

Memory virtualization technique is different from than that of CPU. The hypervisor allocates sperate memory address regions for each VM. A guest OS can perform operations only on the memory sectors allocated to it by the hypervisor. Thus, the hypervisor enforces memory isolation for each VM. It is critical that one VM does not have access to the memory region occupied by other VMs for the safe and secure task execution environment.

A VM is not aware of the memory sectors occupied by other VMs; only the hypervisor has that information. A guest OS can manage the page tables already allocated to it by the hypervisor; the guest OS distributes the memory among the local processes during the runtime. However, creating a new page table for VMs is a troublesome process. It requires modification of the physical memory locations.

To create a new page table, the VM first interrupt the hypervisor. The global memory space is divided, distributed, and managed by the hypervisor. A VM is not aware of the divisions, and any attempt to modify outside memory regions could easily result in conflict with the memory spaces of other VMs. Hypervisor manages all the VMs and knows the global address of the memory regions allocated to each VM. Once the hypervisor receives a request, it creates a new page table and returns the address to the VM.

A two-level page table system is used for the VMs [109, 110]. The guest OS uses a page table to map VM virtual memory pages to VM physical memory pages. The hypervisor uses a shadow page table to map VM memory pages to actual memory pages on the server. As the second level of memory mapping is controlled by the hypervisor; therefore, the VMs cannot access the actual physical memory.

Using a two-level page system; the hypervisor ensures that every VM can access own memory pages and do not interfere with the pages of other VMs. Since a two-level page system is used and hypervisor needs to be interrupted for creating and updating a shadow page table; creating or updating a page table has high overhead.

Applications run on the guest OS, and page fault can occur at any time. Whenever the guest OS detect any page fault, it initiates a sequence of events. First, the page table is changed to read the only mode; then the hypervisor is notified through an interrupt. In the para-virtualized system, the interruption is done through hypercalls. The hypervisor makes changes to the shadow page table and informs the VM through hypervisor callback interrupt.

The hypervisor ensures the VM security in this way; however, with the high overhead of manipulating two-level page tables. Thus, both the VM memory allocation and task memory requirement play a crucial part in the VM performance.

2.6.3 I/O virtualization

For I/O devices like the network interface card and disk, virtual interfaces are needed to be created. Multiple virtual interfaces are multiplexed onto a physical interface. Each VM has a disk partition; however, they are not allowed to access the physical disk directly. Hypervisor translates the I/O requests of the VMs to actual disk block addresses [111].

The actual mapping process depends on the hypervisor type. Type-2 hypervisor sends down the requests to host OS device drivers. The host OS treats the requests in the same way as it treats requests from other processes. However, type-1 hypervisor uses split drivers usually in conjunction with a special VM.

In the split driver technique two types of drivers are used, backend and frontend driver. The front-end driver resides in the VM and communicates with the processes of the VM. The front-end driver sends the data to the back-end driver, which resides either inside the hypervisor or on a special VM. The back-end driver communicates with the hardware to process I/O requests.

Use of virtual device offers extra security; however, at the cost of extra overhead [33]. The requests have to go through two different driver levels and translation phases; thus, adding overhead to the I/O processing. As a result, both the memory and I/O virtualization incurs more overhead compared to CPU virtualization.

All three types of resource virtualization play an essential part in Cloud operations and performance. In this dissertation, experiments have been performed with all three types of resource virtualization.

2.7 Data centers and the rise of virtual machines

In data centers and Cloud mainly type-1 hypervisor is used. A type-1 hypervisor is more efficient for simultaneously running multiple VMs on a server. In this case, the hypervisor takes full control of the hardware and use the resources efficiently. Benefits of using virtualization are many folds [5–12]. Many works have listed various benefits of virtualization; however, some of the listings are overlapping. In this dissertation the following six benefits of virtualization are identified:

1. Security / VM isolation

2. Increase resource utilization / reducing energy cost
3. High availability of services / Fault tolerance
4. Legacy application and system
5. Maintenance
6. Pay-as-you-go model

Next, each category of the benefit of using virtualization in Cloud is discussed in brief.

2.7.1 Security / VM isolation

Security is one of the primary reasons for using virtualization in data centers [5]. A user can have administrative control over the rented VMs and install any application. However, the user does not have access to the underlying physical hardware. The data center owner can have full control of the hardware and create VMs for the users.

Virtualization allows the VM owners to use their rented VMs without compromising the security of the physical server. In this way, virtualization ensures security for the data center owner without restricting the administrative privileges of the VM owners. The hypervisor restricts VMs from accessing the physical hardware; thus, any change made by the VM do not affect the hardware or other VMs. The hypervisor multiplexes the physical resources such as CPU, memory, and I/O in such a way that each VM is isolated from other VMs. Each VM has control over own execution environment. Thus, VM owners have the full flexibility to run any application on their rented VMs.

2.7.2 Increase resource utilization / reducing energy cost

Virtualization can increase resource utilization and reduce energy cost through consolidation [23, 23–27]. Consolidation is the process of running multiple VMs on a same physical server. Running multiple VMs on a server increases resource utilization of the physical server. In turn, that means less physical server have to be active at any given time. Thus, virtualization can help to reduce energy cost.

2.7.3 High availability of services / Fault tolerance

Virtualization helps to increase the availability of services. The key to increasing the availability of services is running multiple copies of VMs [28–30]. The high availability schemes simultaneously execute two or more copies of the same VM on different physical servers. One of the servers run the primary VM while rest of servers run the secondary VMs. The service is provided from the primary VM, while the other VMs are regularly updated with the primary VM data.

In the case, the primary VM goes down a secondary VMs is chosen to continue providing the services. The service requests are automatically diverted to the selected secondary VM. How much of the primary VM state can be recovered depends on the recovery algorithm and how many VMs are simultaneously running. Running the same task on multiple VMs in multiple locations is in the heart of the VM fault tolerance schemes.

2.7.4 Legacy application and system

There are many commercial applications still in production that was designed decades ago for the legacy hardware. Many enterprizes continue to use legacy applications and value their legacy systems highly [112–116]. All hardware has a lifetime, and they phase out eventually. Enterprises often choose to continue using the legacy applications long after the life cycle of the hardware is over. When vendors stop manufacturing legacy hardware buying new hardware becomes impossible for the legacy software owners.

Replacing legacy hardware or application is often expensive. All the source codes need to rewritten for new compatible hardware. Furthermore, all databases and files need to be moved to the new format. Thus, moving codes and data to a new system can be costly and time-consuming. One of the cost-effective solutions is to run the application on VMs that emulate the legacy hardware. The VMs are configured to emulate legacy hardware entirely; thus it can save the cost of replacing legacy hardware.

2.7.5 Maintenance

In data centers, often a server runs for an extended period. Occasionally, the physical servers face problems and need to be shut down for maintenance. In such cases, the

VM of the host needs to be moved to another server. Live migration feature of the virtualization enables VMs to migrate while they are executing tasks [117–122].

A physical server starts showing signs of fatigue usually after long hours of operation. What is more, an increase in the temperature of a part of the data center can affect the performance of the physical servers [123, 124]. The servers may also need to be shut down for other reasons like a hardware replacement or software update. In such cases, the VMs running on the servers need to be relocated to other physical servers. Live migration gives the option to relocate the VMs without interrupting the services.

The VMs can provide services to the Cloud users during the live migration process [125–127]. The actual downtime is so minimized that the VM can continue to provide services without interruption [110]. Once the migration is completed the VMs are completely moved to another part of the data center, and the old server is now free for maintenance work. Thus, using virtualization technology, it is possible to carry out maintenance tasks without interrupting the Cloud services. It is a significant advantage for the businesses.

2.7.6 Pay-as-you-go model

One of the attractive features of the public Cloud is the pay-as-you-go model for renting out the VMs [128, 129]. A Cloud user can rent as many VMs as the user want and use them as long as required. Instead of acquiring hardware a user can rent VMs almost instantly. The pay-as-you-go model is beneficial for small and medium business, which often face the surge in online user activities. VMs can be set up in minutes; on the other hand, the setting up hardware usually takes much time.

Furthermore, businesses can stop renting the VMs at any stage if the extra computing power is no longer required. On the other hand, it is not possible to get rid of unnecessary physical hardware easily. The model saves many start-up businesses from initially spending much money on the computing hardware. Businesses also save money by not having to pay for server maintenance and management costs.

The pay-as-you-go models are the most cost-effective option for many businesses. Thus, the virtualization played a significant role in the commercial success of the Cloud.

2.8 Grid Computing

Any discussion about Cloud computing should start with Grid computing. Grid computing is the predecessor of the Cloud computing. The term Grid computing was coined in the late 1990s, is borrowed from the concept of the electrical power grid. The idea behind Grid computing was to connect many computers to the internet to harness their computing power [130].

In the 1990s the internet was getting faster and individual computers were getting cheaper; sales of the personal computing was booming. The concept behind Gird is to connect multiple computers to create a global resource space for scientific computing. In this case, data and resources are distributed and shared among many institutes and departments.

The grid can be identified with three characteristics; decentralized resource control, standardization, and nontrivial qualities of service [131]. The decentralized resource control means Grid resources can span multiple administrative domains. The standardization refers to the protocols and interfaces that are used to share the Grid resources. Lastly, the non-trivial qualities of service mean parameters such as latency, throughput, and reliability can be uncertain at times.

The Grid was initially designed for running scientific applications and did not achieve commercial success for several reasons. The distributed nature of the Grid is not suitable for many organizations; efficient task scheduling also proved to be troublesome for the distributed resources. The decentralized nature of Grid also makes the coordination and security issues challenging to handle. On top of this, various organizations introduced separate standards for the Grid middleware that are not compatible with each other. As a result, Grid did not achieve commercial success.

Over the years Gird architectures become too complicated to maintain let alone of adding new functionality. Ultimately, Gird was a concept that looked good only on paper and had limited practical applications. The popularity of Grid declined rapidly after the introduction of Cloud. The Cloud was designed to avoid the architectural pitfalls that are associated with traditional Grid computing.

2.9 Cloud

There is no accepted definition of the Cloud; different organizations define Cloud in different ways. In this section, the definition provided by different leading organizations is discussed first. Then a working definition for Cloud is given that is suitable for this dissertation.

VMware defines Cloud computing as "*Cloud computing delivers convenient, on-demand access to shared pools of data, applications, and hardware. The cloud computing paradigmmade possible by sophisticated automation, provisioning, and virtualization technologiesdiffers dramatically from todays IT model*" [132].

On the other hand, Intel defines Cloud as "*a computing paradigm where services and data reside in shared resources in scalable data centers, and those services and data are accessible by any authenticated device over the Internet*" [133].

The US *National Institute of Standards and Technology* (NIST) defines Cloud computing as follows "*Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*" [2].

The *International Business Machines Corporation* (IBM) defines "*a Cloud is a pool of virtualized computer resources*" [131]. For IBM the virtualization plays an integral part for the Cloud. IBM specifies two essential characteristics of the Cloud. The first is the dynamic scale-in and scale-out capability. The Cloud allows fast provisioning and de-provisioning of resources for applications. The second is facilitating the load-balancing and reallocation of resources. Both of the features are achieved through virtualization.

It is possible to identify other characteristics of Cloud, too [134]. For example the Cloud is also characterized as "*component-based application construction*" [135]. In the beginning, Cloud was not thought of as a new computing paradigm; rather, it was believed to be a combination of some new technologies and the Grid. Notably, the idea of utility computing and virtualized data center are what made the Cloud different from the Grid.

A lot of technical concepts has changed over the years, and new technologies have been added to the Cloud; however, the core concepts remain the same. In this dissertation the core concept of Cloud is defined in the following way:

Definition 2.2 *Cloud is the concept of using computing resources as a utility.*

This definition is straightforward and precise for works done in this dissertation. The benefit of using virtualization is increased security and better resources management. As a result, the scope of deploying applications is much greater in Cloud compared to Grid. Initially, Cloud was used for deploying interactive web applications and three-tier architectures [136]. However, nowadays the scope of the Cloud has increased significantly as scientific and parallel applications are being deployed in the Cloud.

2.9.1 Cloud verses Grid

There are several differences between the Cloud and Grid, they are summarized below. The virtualization is essential for the Cloud; however, for Gird it is optional [137]. Although nowadays virtualization is widely used in Grid. Another difference that in Grid the applications are run in the batch mode. On the other hand, Cloud mainly run interactive applications [136]. Grid applications can be deployed locally; however, for Cloud, all the application must be deployed in Cloud [135].

In Grid, an application accesses the resource through the middleware, which manages the resources. In Cloud, the applications use standard web protocol, which makes the resource access easier [136]. The Grid is organized virtually; all the distributed resources are arranged in a virtual domain. On the other hand, all the Cloud resources reside in the data center; hence, the renounces are arranged physically. It is a significant advantage as the enterprises and institutes can control the physical resources.

The grid does not have any efficient business model; the entire Grid system is a share-based economy. On the other hand, Cloud uses a utility model; the user has to pay based on per unit of resource usage [138]. For Grid there is no *Service Level Agreement* (SLA) enforcement; however, Cloud enforces SLA strictly.

For Grid there is no centralized control; all resources are distributed and belongs to different owners. In contrast, Cloud is centrally managed. All physical resources reside in the data center and managed by the organization [138].

The Grid is a highly open system and vulnerable to malicious attacks. The Cloud is far less open and provides an excellent level of security. Another factor is that Grid is hard to operate; as there is no standard *Application Programming Interface,*(API) most setup has to be done manually. Cloud, on the other hand, provides standard API and almost everything is automatic.

Lastly, the initial moving cost is higher for Cloud compared to Grid. Initially, to deploy an application in the Cloud necessary number of VMs have to be rented, and all the data needs to be transferred to the Cloud. The Cloud providers charge users for both the amount of data transfer and storage [139]. It may require a substantial amount of initial investment. On the other hand, Grid is a share-based economy. Furthermore, there is no need to transfer all the distributed data to a central location. Thus, the initial setup cost for Grid is nominal. However, recall that Cloud offers an excellent level of SLA and security while the Grid does not.

2.10 Taxonomy of Cloud

In the previous sections, the basics of Cloud and virtualization have been discussed. In this section, the taxonomy of the Cloud will be discussed in brief, this discussion is necessary for understating the dissertation. However, a full discussion of the Cloud taxonomy is out of scope for this dissertation. The Cloud taxonomy of this dissertation is shown in Figure 2.3.

Researchers have identified many characteristics of Cloud; however, not everyone agrees with everyone else [140, 141]. However, the essential characteristics of the Cloud can be identified as follows:

1. Virtualization
2. Pay-as-you-go model
3. Scalability-on-demand
4. Fault-tolerance
5. Quality-of-Service

Those essential characteristics are described next. **Virtualization** is an essential characteristic shared by all the Cloud. All Cloud deployment and services employ virtualization. All the services and benefits, which are offered by the Cloud are depended on virtualization. Modern Cloud as we know it today would have never been possible without virtualization.

Pay-as-you-go model is the primary factor behind the commercial success of the Cloud. It is the most interesting aspect of Cloud for the users. A user can rent any

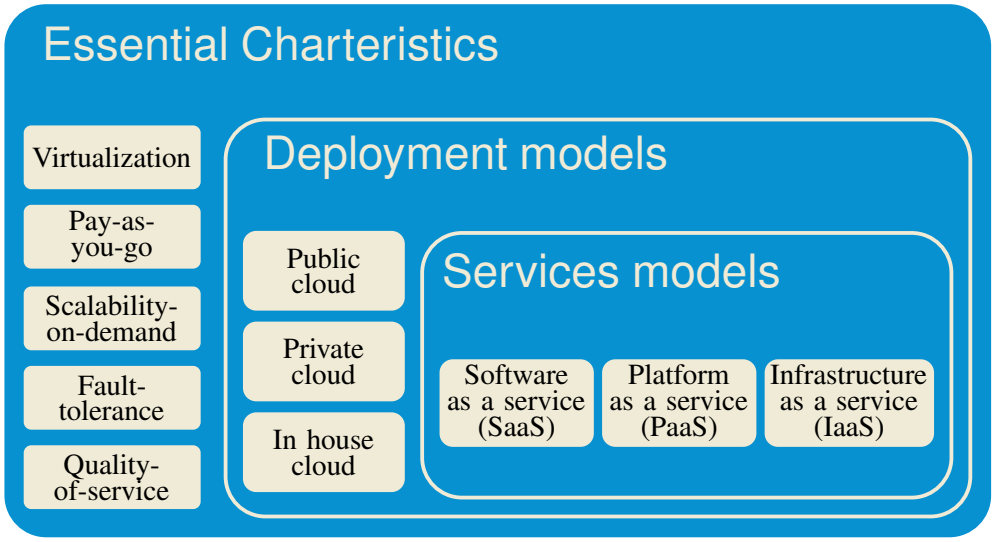


Figure 2.3: Taxonomy of Cloud services [1, 2].

amount of resources at any time. The user only has to pay for the rented resources, and there are no other costs involved. There is no infrastructure procurement, management or update cost involved with the Cloud. **Scalability-on-demand** is another feature common to every Cloud. Cloud users can rent resources on demand. Most cloud providers charge separately for each resource consumption. Depending on the computational it is possible for users to rent new resource ar anytime. Moat cloud system allows users to scale up resources both vertically and horizontally.

Fault-tolerance all Cloud systems have elaborate fault-tolerance and disaster recovery plan. Those plans are also linked to the high availability of services. However, the scope of those plans differ from provider to provider; it cloud be a simple VM duplication plan to a highly sophisticated fault-tolerance algorithm. All Cloud systems provide some scheme and measure to ensure the **Quality-of-services**. Measures and schemes may vary from provider to provider and detail about a provider's scheme can be found in the contract agreement.

Now that the essential characteristics of the Cloud are discussed, next the Cloud classification and services models are going to be addressed. There are three main deployment and three services models for the Cloud. The three deployment models are: *a*) Public Cloud, *b*) Private Cloud, and *c*) In-house Cloud. On the other hand, the three services models are: *a*) Infrastructure as a Service (IaaS), *b*) Platform as a Service (PaaS), and *c*) Software as a Service (SaaS). Those models are discussed in the

following sections.

2.11 Classification of Cloud deployment models

In this dissertation, the categorization of Cloud services and deployment models have been used to reflect our current understanding of the technology. Cloud has come a long way within a few years, and the classification is done based on the recent developments scenarios.

Cloud is a centralized system; all physical resources are kept in a data center. Several models have been proposed for Cloud services and resource deployment. There are several taxonomies proposed for Cloud services, platforms, Infrastructure-level, Interoperability and so on. Unfortunately, those works are almost a decade old [134, 142–145]. The Cloud has evolved over the years, and our understanding of the technology has developed. While some of the concepts remain the same, some other concepts have changed.

In 2011, NIST categorized the Cloud deployment models as private, community, public, and hybrid [2]. Those were based on the earlier understanding of the Cloud deployment strategies. It was prevalent among the Cloud deployment models for several years. However, with time the understanding of deployment models have been changed.

In this dissertation, Cloud is divided into three categories based on deployment models; they are *public*, *private* and *in-house* Cloud. Table 2.2 provides a comparison of the three deployment models. This classification is made by analyzing the current Cloud deployment strategies of organizations. Those deployment models are discussed in the following sections. Originally this dissertation was submitted with this classification in September 2017.

In January 2018, VMware released a new classified of the Cloud deployment model [146]. This classification contains three models; hosting, private, and public. Community and hybrid, two deployment models, considered earlier in the 2011 NIST classification have vanished from the 2018 VMware classification.

The classification presented in this dissertation is similar to that offered by VMware. All three deployment model concepts are very similar to the three models proposed by VMware. The only difference is that in this dissertation one of the categories is named

the ‘in-house’, whereas in VMware technical report the same category is being called the ‘hosting’. Other than that, both categories embody the same concept.

Thus, the deployment models released by a leading virtualization enterprize few months after the original submission of this dissertation agrees with the new deployment models presented in this dissertation. The three deployment models are discussed next.

Table 2.2: Comparison of characteristics of Cloud deployment models [3, 4].

Characteristics \ Types	Public	Private	In-house
Dedicated	Multiple customers	Single customer	
Infrastructure managed	Third party provider	Customer or Third party provider	Customer
Infrastructure located	Off-premise	On-premise or Off-premise	On-premise
Software update cycle	Third party provider	Customer or Third party provider	Customer
Modification of services	Not allowed by customer	Allowed by customer	
Service configuration	Highly standardized	Mostly customizable	Fully customizable
Application security requirement	Usually low	Medium to high	High
Application resource requirement	Usually High	High to moderate	Moderate to low
Access policy	Un-trusted	Trusted	
Fault tolerance and recovery scheme	Third party provider	Custom scheme possible	
Application performance tuning	Third party provider	Customer or Third party provider	Customer

2.11.1 Public cloud

Public Cloud is the most common form of Cloud available today; it offers VM services to all users [29]. A user can rent computing, storage, and data transfer services from the Cloud provider [139]. The VMs are rented out in the pay-as-you-go model; a user can rent a unit of resource for any duration.

In a public Cloud, VMs from multiple tenants usually share the same physical server. Public Cloud users usually have fewer security requirements, and the resource units are relatively cheaper. Users can easily rent VMs through web portals and use them.

Most deployed applications are interactive and have low latency and throughput requirements, like the Web and game servers. Deployment of scientific application is less common in public Cloud. However, nowadays public Clouds has started offering specialized services for high resource intensive applications. Some examples of public clouds include the *Amazon Elastic Cloud* (EC2), *Blue Cloud* by IBM, *Sun Cloud*, *Google AppEngine* and *Windows Azure* [147].

2.11.2 Private cloud

Private Cloud applies strict security rules and regulations [2, 29]. Extra care is taken to ensure the quality of services like, latency, and throughput. Primary users are the enterprises and departments that require greater computing power to run scientific applications. Such an application requires greater computing power compared to the applications on public Cloud. However, they have a shorter lifespan compared to the applications of the public Cloud.

For example, a company may need to simulate massive chemical reactions for a new product [148, 149]. Similarly, a biology department may require working with a significant amount of biological data [150]. Such applications need high-performance computing power. The resource requirement for parallel tasks is different from that of the ordinary web servers. Furthermore, there can be sensitive data that requires more security compared to that offered by the public Cloud. The scientific data are confidential, and it needs to be secured from unauthorized access.

The private Cloud offers a higher level of security and SLA for sensitive data and applications. For example, they make sure that VMs of only one user resides on a server and VMs of no other users are on that server. Private Clouds are usually smaller

in size and renting VMs not as easy or cheap as it is for the public Cloud. However, some public Clouds also offers the same functionality at a higher price.

2.11.3 In-house cloud

The in-house Cloud refers to a system used by an organization for internal use only. Banks and other financial institutions usually require in-house Cloud; it is used for processing mainly financial transactions. Such transactions take a short time to execute; however, requires highly secured execution environment [151–153]. Financial transactions face a high risk of malicious attacks; hence, never deployed in public or private Cloud. Also, both the public and private Cloud requires data to be transferred to the Cloud file system that is also a security risk for the financial data.

Financial transactions are small in size and do not require high-performance computing. The primary objective is to process the requests securely and as quickly as possible. In-house Clouds are smaller in size and systems are built with extra security features.

In this dissertation, experiments are done with hypervisors and VMs. All Cloud deployment models are dependent on hypervisors and VMs; The experiments of this are designed to work with all types of Cloud deployments.

2.12 Classification of Cloud service models

Cloud is dependent on virtualization for providing services. Based on how a VM is configured to provide service the Cloud can be divided into three categories as shown in Figure 2.4. The three categories are [145, 154]:

1. *Infrastructure as a service* (IaaS),
2. *Platform as a Service* (PaaS), and
3. *Software as a service* (SaaS).

All three categories of Cloud services are discussed in this section. A full discussion about the services is out of the scope of this dissertation. They are introduced here to explain how those services relate to the works of this dissertation.

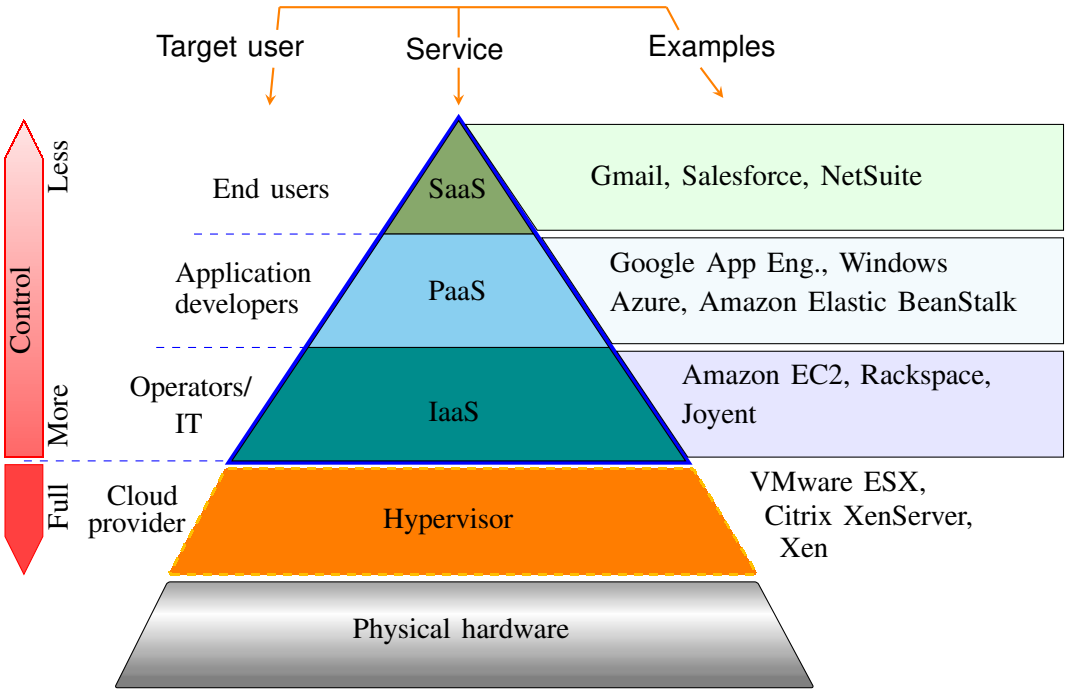


Figure 2.4: Classification of Cloud service models.

All of this services are implemented on top of physical hardware and hypervisors. All the experiments in this dissertation are conducted on hypervisors running on physical hardware. Hence, the experimental findings and results are equally applicable to all service models.

2.12.1 Infrastructure as a service (IaaS)

Iaas is the lowest level of service provided by any Cloud. In this case, rented VM can be configured to perform any task. Each VM is allocated with virtual resources like virtual CPU, memory and I/O. The VMs are highly configurable and the VM owner can install any application [128, 155–158].

A user can rent as many VMs as required, configure them and deploy the application. In this case, the virtual hardware is rented out, and the user can choose a guest OS to install. The user manages the OS and has the flexibility to run any application. Example of Iaas includes Amazon EC2 [159].

2.12.2 Platform as a Service (PaaS)

The Paas provides service a level higher than that of Iaas. In the case of Paas, the platform is managed by the Cloud provider. It is not possible for the Cloud user to install the OS of own choice [160–165]. The VMs are reconfigured for a particular development environment; thus, Paas is mainly used to develop and deploy applications on the Cloud. Examples of Paas includes Windows Azure [147] and Google App Engine [166].

2.12.3 Software as a service (SaaS)

In the case of Saas, the application layer is also managed by the Cloud provider in addition to the virtual hardware and platform. The cloud user can only interact with the applications. The applications are developed and deployed before hand; Saas is basically just a production environment [167–173].

Compared to previous service levels the Saas is more familiar to the users. Example of Saas includes the Google Apps, and Microsoft Office 365. Users can either install an app on the local device or access it through a web portal. The locally installed application is just a frontend that interacts with the Cloud to provide services.

2.12.4 Physical hardware and hypervisor

Strictly speaking, physical hardware and hypervisor are not the part of Cloud service hierarchy. Nonetheless, they are introduced here to demonstrate their position in reference to the hierarchy. The hardware and hypervisor sit at the bottom of the stack, and all three services are developed on top of them. Thus, they play a vital part in providing all three types of Cloud services.

Physical hardware can be any server or cluster computing system. In data centers, generally, a type-1 hypervisor is directly installed over the hardware. Then, all types of Cloud service models are then implemented on the top.

In this dissertation, all the experiments are conducted with the physical hardware and type-1 hypervisor directly. Thus, the experimental results and finding have equal impact on all types of services mentioned above.

2.13 Conclusion

This chapter discusses the concepts regarding virtualization. Importance and benefits of virtualization in Cloud are also explained. Virtualization has a long history, and many pieces of research have been done work in this field. Virtualization is a necessary feature of the Cloud, many services of Cloud dependents on it.

A full discussion about history, types, or service models built on the top of virtualization are out of scope for this dissertation. Those topics are introduced here to give readers an idea about at what level the experiments are conducted and how they impact various service models. From the discussion above, readers will have a better understanding of how the experiments are designed and what aspects are taken into consideration.

Different resources like, CPU, memory and I/O are virtualized differently. A data center can house thousands of virtualized servers. Resource utilization and energy efficiency are important issues for the servers. The improvement of resource utilization can save energy; therefore, it is an essential area of research.

Now, that the basics of Cloud are discussed; the next chapter begins the experiment with VMs. All experiments are done on the real virtualized system. An application may use three basic resource types; CPU, memory, and I/O. The application can also use a combination of resources, like CPU and memory combination, or memory and I/O combination. Performance of the consolidated VMs is affected by the resource consumption. In the next chapter, experiments are conducted with the performance and resource usage of the consolidated VMs.

Chapter 3

Resource usages and contention of co-located virtual machines

"It is by logic that we prove, but by intuition that we discover.
To know how to criticize is good, to know how to create is better"

— Jules Henri Poincaré* (1854–1912)

3.1 Introduction

Server virtualization is an essential feature of modern data centers. It is a cunning technology that provides applications with an alternate view of the execution environment and resources that are independent of the underlying system and architecture. Such abstraction is often necessary to run untrusted software over the internet and on different system architecture. The modern data centers and virtualization are dependent on each other. The virtualization is consistently used in the data centers to run various types of applications on high-end servers. Virtualization technique can be implemented on different computing systems in various ways.

In high-end servers, the virtualization layer usually resides over the physical hardware. In this case, the guest *Operating Systems* (OS) run on top of the virtualization layer and interact with a *hypervisor* rather than directly with the actual physical hardware. Virtualization mainly provides two advantages. Firstly it allows software to run

*Image source: `https://commons.wikimedia.org/wiki/File:Young_Poincare.jpg`

on a system for which the software was not designed and secondly it provides an easy way to run an untrusted application on a remote machine. Both of the abilities are crucial for deploying application over the internet. For above reasons, the virtualization has become a popular choice for data centers all over the world.

The history behind the inception of Cloud is fascinating. As the computing hardware has become ever cheaper during the late nineties, various enterprises started building up a vast cache of commodity hardware primarily to meet their computational requirements. However, soon it was realized that most data centers have a low level of utilization [174], it can be as low as 5% and hardly ever crosses 20%. Low utilization directly translates into both high operational cost and energy bill, which are real concerns for any data center owner [175].

On the other hand, the rapid growth of the internet provided an opportunity to both small and medium scale internet based service providers to acquire additional resources on a short-term basis. These businesses often face a sudden deluge of traffic, which may last only for a short duration of time. Such requests cannot be served sustainably with the physical resources available to most small and medium scale businesses. On the other hand, purchasing a large cache of physical hardware for occasional service requirements is not economical. Therefore, an alternative to conventional computing resource acquiring process was necessary, and out of this mixture of drastically different economic demands, the idea of Cloud was conceived to help the operators on both sides of the spectrum.

The Cloud is a technology that is born out of necessity; it is the concept of using computing power as a utility. The Cloud has already been proven as one of the most successful innovations of the past decade, and constant efforts are being made to make it more accessible for high-performance computing (HPC) [8]. As already mentioned that a fundamental concept of Cloud is to rent computing power as a utility, not the physical hardware. It is a revolutionary idea in the field of computing, and this would not have been possible without virtualization. The Cloud allows users to deploy their applications over the internet, while the actual physical machines in the data center remain out of reach for the Cloud user.

The Cloud offers users the ability to rent any amount of resources within a short period without prior reservation, run any application on them and subsequently release them without any strings attached. In the heart of the whole process is the virtualization technology, which allows the physical resources to be divided into virtual units

and rented over the internet. Deployment of virtualization makes sure that the Cloud users can have the full control over the rented *Virtual Machines* (VMs), while Cloud providers have the total control over the physical hardware. However, the consequence of running multiple VMs on a physical machine is resource contention, which leads to performance degradation. Although, unexpected performance degradation may be acceptable for applications like web services; however, may not be adequate for running parallel applications. This performance variance is influenced by a lot of factors, including resource contention among the running VMs, the architecture of the hypervisor itself and I/O requests handling by the virtualization system.

The scheduling is an integral part of executing parallel applications. To fully utilize the inherent parallelism of an application first it is broken into some smaller tasks and then scheduled for execution. During the scheduling process, each task is assigned a start-time, finish-time, and processor to execute on. Once the application is broken down into some smaller tasks, it can be represented by a task graph for convenience. A crucial part of scheduling is to estimate the execution start time and finish time of each task. That is because before a child task can start execution all the parent tasks of it must finish executions. As mentioned earlier, that on VMs tasks often faces unpredictable performance degradation; as a result, the finish time of tasks cannot be estimated accurately during the scheduling process.

Since the delay in the execution of a task of the task graph can have a cascading effect, this can make the whole scheduling scheme inefficient [176]. Therefore, to effectively schedule a parallel application on a virtualized system analyzing the performance impact of the co-located VMs is necessary [177–186]. Performance prediction models for VMs can be useful for improving energy efficiency as well [187–193]. Efficient use of energy has become a burning issue for the data centers today [194–197]. A computing system can achieve the highest efficiency only when all the hardware components are fully active, and the system is running on full load. Techniques such as temporarily shutting down of one or more cores of the processor or some blocks of memory have been shown to be not energy efficient [198]. The virtualization can be an effective way of reducing energy consumption through consolidation of VMs.

As mentioned above the servers of many data center remain under-utilized at most of the times [174]. One way to increase utilization is to consolidate more VMs on the same server. Being able to run more VMs on the same server means increased server utilization for the same amount of physical resources. On the other hand, consolidating

too many VMs on the same physical server will degrade the performance of VMs. A model that can predict the performance penalty for consolidation will help to choose the optimal number of VMs to be consolidated [26, 182, 191, 199]. The model will indicate how much performance will be lost due to the consolidation of a certain number of VMs on a server.

In this scenario, the physical system remains fully active while the consolidated VMs number may be varied to achieve different levels of performance-versus-utilization trade-offs. Obliviously, more consolidated VMs on less number of running physical servers means more performance degradation, while running additional physical servers means better performance, however, at the cost of extra energy consumption. Predicting such trade-off behavior with a model can help the system administrator to take better decisions.

To build a performance model for co-located VMs some factors need to be considered [200]. First, the relationship between the resource consumption of various VMs is not linear. Second, many hardware parameters and counters have to be taken into account for this problem, and that increases the dimensionality of the problem. Third, to build such a model various benchmarks are required, that can put stress on different systems resources both individually and collectively. Fourthly, the experiments have to be done with a variable number of simultaneously running VMs. As the number of co-located VMs on a server may change at any time, therefore, their effect on the performance needs to be examined.

It should be noted that although modern data centers are made up of heterogeneous hardware, however, it does not necessarily mean that each machine configuration is different from every other machine. Usually, for a data center, large pieces of hardware is purchased at a time in batches, and each batch usually consists of machines of similar configuration.

This chapter begins experiments with consolidated VMs and tasks. Various benchmark suits are used to collect resource usage data from various VM combinations. Separate benchmarks are used to put stress on separate computing resources. The resource consumption data from multiple VMs are collected.

A set of benchmarks are used to measure various VM resource usages like CPU, memory, disk I/O. Also, an *Online Transaction Processing* (OLTP) benchmark is used to put stress on all three resources simultaneously. The benchmarks are run on VMs

in different combinations, and each combination is a pattern. The finding of this chapter will be used in the following chapters for analyzing and building VM performance models. Next, the objectives of the works are described with an introduction to the virtualization technology. Afterward, the experimental setting and results are discussed, followed by the chapter conclusion.

3.2 Motivation and background

The objective of this work is to identify the effect of resource contention among co-located VMs. The virtualization of data-center provides some unique benefits like consolidation and live migration [125–127]. Analysis of VM resource contention will help to increase resource utilization through VM consolidation and migrations. The resource consumption pattern of VMs changes depending on the consolidation environment. Here, a virtualized server is used to investigate the VMs resource consumption pattern changes due to the change of co-located VMs number.

Initially, some benchmarks are run individually on one VM of a server and resource usages traces are collected. Then, a set of two VMs is placed on the server, and the resource usage pattern for two simultaneously running VMs are recorded. This process is repeated for sets of four, eight and fourteen co-located VMs. The objective is to observe the changes in resource usages pattern due to the change of the quantity of simultaneously running VMs.

These experimental stages are set up initially to observe how VMs react to consolidation. This chapter describes what benchmarks are used in the experiments and types of data are collected. The experiments are primarily done to collect resource usages data and identify how the resource usages pattern varies. In later chapters, it is described how the resource usages pattern can be analyzed and used for predicting VM consolidation performances.

Linux scripts are used here to run nine different benchmarks on different sets of consolidated VMs; each set is having a different number of simultaneously running VMs. While the benchmarks were running, tools like Top and Iostat [201] are used to collect CPU, memory, and disk I/O utilization from all VMs and host. The tools collect a wide variety of system data; however, the collected data is raw and do not have any uniform value range. Thus, the data requires further processing. At first, the collected

data is organized and refined, then, it is formatted and scaled for visualization. Once the refining and formatting process is done separate data files are produced for each set of co-located VMs. Also, sperate sets of files are also generated for the host machine data.

Next section gives an overview of the virtualization process used in the Cloud and data centers.

3.2.1 Application of virtualization in cloud

The virtualization is an absolute necessity for modern data centers especially after the advent of Cloud computing. There exist many different types of virtualization technique today, however, for data centers the most popular virtualization technology is the type-1 virtualization. In type-1 virtualization, the virtualization layer is placed above the physical hardware and controls the hardware directly, unlike a non-virtualized system where the operating system controls the hardware. One of the primary purposes of the virtualization layer is to provide an alternate view of the physical resources to the VMs above it. Such abstraction is necessary for providing Cloud services.

The commercial Cloud lets the users deploy their application over the internet without much administrative supervision. Such deployments of the software raise serious security concerns. To run applications on the Cloud, most users require administrative privileges and a typical data center may have thousands of such users logged on at any given time. That is why, it is not realistic for the administrators of data centers to keep track of all administrative operations of the users, this is where virtualization is so useful.

The data centers rent out VMs, which has no access to the physical resources. Nonetheless, the VMs appear as fully functional self-contained autonomous units of execution to the Cloud user. A user can have administrative privilege on the rented VM allowing them to install and run any application. On the other hand, virtualization helps to protect the physical machines from the security risks. Thus, VMs allow users to install and run any application freely and at the same time it reliefs data center administrators from having to monitor every activity in those VMs. VMs can be created within a minute with couples of commands and once rented out then it is up to the user to decide how the VM is going to be operated.

Once the user has finished running the applications the user can release the VM immediately. After the release, the VM can be either recycled or destroyed automatically by the Cloud management system without any human involvement. Thus, creating and destroying VMs on demand without the need for any human interaction saves many person-hours for the data center administrators.

The benefits of virtualization are not only limited to the public Cloud; it is also useful for private Cloud as well. A simple scenario would be like this where several teams are simultaneously developing different parts of a large-scale data analysis software for multi-core clusters. In this case, it is not possible for one team to have the whole machine for itself since such hardware is expensive and an organization may only have one such cluster. Such software may take some time to compile and in an Agile [202] software development environment all the teams need to compile their parts of software regularly. Obliviously with only one piece of hardware, it is not possible for multiple teams to compile simultaneously.

One solution is to virtualize the cluster and provide each team with own global space. Once all the teams have done compiling their parts of the software, then complete software can be compiled together to ensure the integrity of the software. In this way, all teams can independently compile their parts of software before compiling altogether with other teams. Thus, making sure that their parts compile correctly. It could save much time that would be otherwise wasted debugging the software.

A wide variety of operating systems is available today. Over the years many derivatives of Unix have been introduced like the *Oracle Solaris*, *Berkeley Software Distribution* (BSD) [203] and *IBM AIX* [204]. In the case of Cloud, different users may have different operating system requirement and it is not always possible for a data center to maintain every type of OS for users. The VM can help in this situation too as it lets the users use the OS of their choice.

Another area where virtualization can be very useful in data centers is the fault tolerance. Servers in a data center may start to show signs of fatigue for various reasons, including extended hours of operation. The increase of temperature in a part of the data center also enhances the probability of machine failure in that area of the data center [123, 205]. It is indeed not possible to move around physical servers in a data center. In such cases, the one solution is to shut down the physical servers on the particular part of the data center to avoid a possible catastrophic failure; however, this would lead to online service interruptions. The hypervisor can also allow VMs to be

migrated while they are being executed, through a process known as the live migration. The live migration makes it possible to move the VMs of a physical server to other physical servers, which are located in a low-risk part of the data center.

The live migration also is necessary for other features like consolidation and energy efficiency. The virtualization allows public Clouds to have elaborated fault tolerance mechanism through live migration and replication of VMs; those are what makes public the Cloud enable to offer a high degree of availability of services to the users. Thus, virtualization is a useful technology.

The history of virtualization technology is almost as old as the modern computers. Some may be fascinated to know that the virtualization technology was first developed during the late sixties; thus, predating the microprocessor technology itself. Virtualization was originally introduced by IBM for their mainframe computers with the motivations similar to as it is today [13–22]. Mainframes were hugely expensive pieces of machinery; therefore, investors wanted the best possible utilization out of them. As a result, the virtualization introduced to the mainframes so that the batch tasks can be consolidated. Over the years the demand for mainframe fall as the market for x86 machines grew. Initially, the x86 platform was not designed for virtualization. Nonetheless, during the late nineties, the x86 servers became much powerful and widely used that virtualizing them became a realistic option. Despite similarities of goal, the modern virtualization architecture is significantly different from that of the mainframe. Today data centers are dominated by type-1, para-virtualization technology, which is relatively low overhead.

The motivations for using VMs in a data center are summarized below.

- Virtualization can increase resource sharing and utilization through VM consolidation.
- Allows alien applications to run without the supervision from data center administrators.
- Allows implementing sophisticated fault tolerance mechanism for data centers through the live migration and replication of VMs.
- Virtualization reduces the system maintenance and administration cost for the data center owners.

From the above discussion, it is clear that virtualization is very relevant to the data centers of today. The potentials of virtualization are not still fully utilized in the area of high-performance computing. The next section discusses the main drawbacks of using virtualization in the field of high-performance computing.

3.2.2 Scheduling of parallel application and virtualization

To run a parallel application on VMs; the parallel application is first broken down into a set of smaller tasks. These tasks can be executed parallel; however, the dependencies among those tasks have to be taken into account. The task dependencies mean that the output of one task can be the input of one or more other tasks. A *Task Graph* is a directed acyclic graph, which presents the tasks as vertices and dependencies among them as edges. The task graphs have prime importance is modeling and analysis of parallel applications and architectures [206–209]. The task graph is also known as the *Activity Network* and based on the concept of individual task [210].

The concept of a *task* has not been formally defined; however, it refers to a set of instructions that can be thought of a unit of work and performed sequentially on a single processor using a fixed amount of resources [206]. In virtualized systems, a fixed amount of physical resources like CPU, memory and I/O can also be assigned to tasks like in a non-virtualized system. However, it is known that the performance of VMs does not always correspond to the assigned amount of resources [31, 40–42]. Therefore, to efficiently execute tasks on a virtualized system it is necessary to have a performance prediction model for VMs [32].

In the theory of activity network, the parallel applications or programs are considered as complex activities, which consists of several individual tasks, and a precedence relationship interconnects them. The precedence relationship is also known as constraints specify that specific tasks have to wait for some other task to finish, former is known as the *child task* while later is known as the *parent task*.

The relationship among all the tasks of complex activities can be presented in a directed graph called, the task graph. During scheduling process each task is mapped to an individual processor; hence, the tasks graphs are essential for any scheduling algorithm [209, 211–214]. Before a child task can be scheduled for the execution; the execution finish times of the parent tasks must be known. As mentioned earlier that in virtualized systems applications often suffer from an unexpected variance of

performance; therefore, without taking the variation into account, it is not possible to accurately estimate finish time of tasks for scheduling. The ability to predict how the performance of applications running on co-located VMs varies will help to build efficient task schedules.

The model can be used to predict how the performance of individual VMs would be affected if they are consolidated together on a single physical machine. Different tasks use different resources in different quantities, and when two are placed together, each will influence the performance of others. A performance model for the virtualized systems will help to choose a group of tasks that will perform better when they are placed together on a physical server.

The next section discusses the existing VM consolidation benchmarks.

3.3 Consolidation benchmarks

There exists several consolidation benchmarks for virtualized servers like, *vConsolidate* [215] and *vmMark* [216]. Both of the above benchmarks use the concept of tiling, which is just a simple approach to identify the average overall impact of consolidation only. The strategy used in those consolidation benchmarks is fundamentally different from that used in this chapter, they will be further discussed below.

3.3.1 Concept of tiles in the consolidation benchmarks

Both benchmarks use combinations of different applications on VMs to build a structure resembling *tiles*. Each VM runs an application that is representative of a type of modern-day server workload, for example, mail server, database server, web server, and Java server. Each VM runs only one type of workload. During the experiments, the total number of transactions successfully commit by each VM is recorded. Afterward, a weighted average is calculated from all the VMs data. Different benchmarks use slightly different formulas to calculate the overall consolidation marks; thus, marks obtained from them are not the same [215, 216].

In contrast, this work focuses on examining the resource consumption of various computing resources; thus, giving a more in-depth view of the system under consolidation. It will help to understand the overall impact of virtualization on the physical server and individual VMs. Once the resource requirement of a VM is analyzed, this

can be used to explain the performance of the consolidated VMs. The experimental results will help data center owners and system administrators to take a more informed decision about the server consolidation and live migration.

3.3.2 Scoring methodology in the consolidation benchmarks

Above benchmarks provide some points or marks to characterize the average performance of a virtualized system. However, those are not very useful for building a performance model for the resource contention of co-located VMs. Any performance prediction model requires detail information about the system performance. More details about the scoring system are given next.

One VMmark tile consists of six different VMs running six different workloads; they are the Mail server, Java server, Standby server, Web server, Database server, File server. Different metrics are collected from the different workload. For example, metric collect from the Mail server is the actions/minute. Metric collect from the Web server is the accesses/second. Metric collected from the File server is the MB/second. Thus different workload provides different metrics. Since those metrics have different measurement units, VMmark normalizes the individual scores and provides an overall score [216].

An example of VMmark system score on the Hewlett Packard Enterprise (HPE) ProLiant DL380 Gen10 Server can be found in [217]. This system hardware employs 4 sockets, 64 cores, and 128 hardware threads. VMware ESXi server version 6.0.0 is used in the experiments. Total of 22 tiles are run on the system, and VMmark V2.5.2 Score is 25.86. The VMware keeps the record of VMmark results submitted by various enterprises, further examples of VMmark scores can be found in [218]. The vConsolidate also follows the similar methodology to perform the tests, however, uses a different formula to aggregate the results.

In the scores of the above benchmarks, there is no indication of how a particular resource intensive VM affects the server performance. For example, how a CPU-intensive VM effects the performance or memory-intensive VM affecting the performance. Therefore, the results are not categorized resource-wise.

Another problem is that VMmark runs all the tests for a fixed number of tiles only. The VMmark runs the tiles and collects workload data at regular intervals. Tests on the server can be run for three or more hours at a time. At first, VMmark waits for VMs to

reach a steady state. Once the steady state is reached, experiments are divided into 40-minute sections. In each section, data is collected with 60 seconds interval. After each 40-minute section, the results for the tiles are calculated. At least three sections of tests are run, and the median score of each section for each tile is collected for calculation.

During the test duration, the number of titles is not changed. In a virtualized production server, the number of VMs can change at any time. Moreover, new VM can be a CPU or memory intensive one. Recall that a VMmark tile is just a combination of six VM workloads. In the real world, the number of VMs may change in any randomly order not necessarily according to the arrangement of VMmark tile. Thus, VMmark is not equipped to handle the changing state of VM consolidation. Therefore, a separate profiling tool is necessary to collect such detailed data from the VMs.

In this dissertation, those issues are taken into consideration while performing experiments with consolidated VMs. In this chapters, it is also investigated how new issues may arise with the changing dynamic of the experiments. In subsequent chapters, the new challenges are analyzed, and methods are proposed to overcome those issues. To conduct the experiments with consolidated VMs performance a set of benchmarks need to be selected. Next section describes the benchmarks used in the experiments of this and subsequent chapters.

3.4 Benchmarks used

Benchmarks selection plays an important part in system performance analysis. For any system, three basic types of system resources can be identified, they are CPU, memory, and disk I/O. No single benchmark can represent all three resources; hence, several benchmarks are used. The benchmarks employed in this work are listed in the table 3.1, they are also described below.

These benchmarks play a vital role in the experiments conducted in this and subsequent chapters of the dissertation. In this chapter, the benchmarks are used to conduct experiments with VM consolidation performance. In the subsequent chapters the benchmarks are used to create workload combinations; details are given in the respective chapters. Sections 3.5 and 3.6.1 provides further information about how the benchmarks are arraigned in experiments and data is collected from the benchmarks.

After describing the benchmarks and experimental setup, the experimental results

for benchmarks are discussed in the section 3.6.1. Changes in resource usages patterns of each essential resource like CPU, memory and I/O are discussed separately. To compare and summarize the performances of various benchmarks the arithmetic mean of resource usages of all the benchmarks are calculated and shown in Section 3.9.

3.4.1 CPU intensive benchmarks

Two CPU-intensive benchmarks have been used in the experiments. They are the *Nbench* and *Unixbench*. The benchmarks are described next.

3.4.1.1 Nbench

The Nbench [219] is a CPU benchmark suite consisting of various types of benchmarks to test all the vital functions of a processing unit. It was initially introduced in mid-1990's by researchers of once popular the *BYTE magazine*. Subsequently, it has been updated several times to make it more compatible with modern CPUs. The benchmark suite is written in C and was initially intended for Windows machines. However, later Linux compliant versions were introduced; hence the benchmark is portable between systems.

The benchmark suite uses a comprehensive set of real-world algorithms to make the tests as realistic as possible. These tests are called, *Numeric sort*, *String sort*,

Resource intensity	Benchmark
CPU	Nbench
	Unixbench
Memory	Cachebench
	STREAM
I/O	Dbench
	Filebench
	IOzone
Other	LMbench
	Sysbench

Table 3.1: Resource intensities of the used benchmarks.

Bitfield, *Emulated floating-point*, *Fourier coefficients*, *Assignment algorithm*, *Huffman compression*, *IDEA encryption*, *Neural Net* and *LU Decomposition*.

In the numeric sort test, several arrays of 32-bit integers are sorted to test the integer handling capability of the system. In the string sort test, several arrays of strings of arbitrary lengths are sorted. A variety of bit manipulation functions is executed in the bit-field test. A small software package is used to stress the floating-point unit of the system in the emulated floating-point test. In the Fourier coefficients test, numerical analysis is done to calculate a series approximation of a waveform.

Next test, the assignment problem is one of the fundamental combinatorial optimization problems in the operational research. Here, the objective is to find a maximum matching for a weighted bipartite graph. The Huffman compression test uses the Huffman algorithm to test the performance of a system for compressing both text and graphics.

Next, the IDEA encryption test is for evaluating the efficiency of a system using the IDEA algorithm to cipher and then decipher blocks of data. The neural net test consists of a small self-sufficient back-propagation network simulator to perform tests. The last test is the LU Decomposition it is for testing the performance of the system for solving a set of linear equations. Thus, the Nbench benchmarks consist of a comprehensive set of test for a VM.

3.4.1.2 Unixbench

The Unixbench [220] performs multiple tests on the system. The resource usage patterns of various tests of the Unixbench are designed to resemble that of commonly used commercial applications. The Unixbench has been used to analyze the performance of VMs in previous works [221]. The tests that are included in Unixbench are described next.

The Dhrystone [222] and Whetstone [223] are two well-known synthetic benchmarks that are part of the Unixbench suite and used for measuring the CPU performance. Synthetic benchmarks are carefully designed programs based on statistical analysis of various parts of widely used applications. The objective here is to mimic the operations of certain parts of real-world applications that are responsible for most of the resource consumption by the application. It is motivated by the fact that during execution applications usually spend the most time on certain parts of the code, and

these parts have huge influences on the overall performance of the application.

The Whetstone measures the speed and efficiency of some floating point operations that are common in scientific applications. It employs a wide variety of C library functions like *sin*, *cos*, *sqrt*, *exp*, and *log* along with other system related functions like conditional branches and procedure calls. On the other hand, the Dhrystone focuses on integer and array access efficiency of the system. Design of these tests is influenced by the design techniques of software such as the compiler, code optimization, and wait-states.

Next, the *execl throughput* test measures the execl call execution efficiency of the system. The execl() is an essential function that replaces the currently executing process with a new one. It is an essential member of the exec family of functions. Next, the *file copy* test is for testing how fast chunks of data can be copied. For this purpose read, write and copy operations are performed on buffers of different sizes. Next, in the *pipe throughput* test, a process is repeatedly used to write and read on 512 kb buffers as many times as possible over the pipe, which is a simple mechanism for interprocess communication. Next, in the *pipe-based context switching* test two processes interchange an integer through a bi-directional pipe, and size of the integer is increased in each successive step.

The *process creation* test measures how many times a process can fork and reap child processes. Creating and reaping child processes is a standard feature of applications. Next, in the test called *shell scripts*, it is tested how many copies of a shell script the system can run concurrently and how quickly. Next, in the *system call overhead* test, the getpid() system call is repeatedly issued by a simple program. Each system call causes the operating system to enter into kernel mode and subsequently depart, purpose of the test is to check how fast the system can enter into and exit from kernel mode. Besides these, the Unixbench also performs both 2D and 3D graphical tests for the graphics driver of the system.

3.4.2 Memory intensive benchmarks

Two memory intensive benchmarks are used in the experiments; they are the *Cachebench* and *Stream*. Both of the benchmarks are described next.

3.4.2.1 Cachebench

The Cachebench [224] is a benchmark for testing the memory system and cache; it performs eight different tests to determine the bandwidth of the system memory hierarchy. The memory is an essential part of any computing system; the VM performance depends in many ways on the available memory. The memory system of all modern servers has a hierarchical structure for faster execution of the code. In the hierarchical memory system, the data is stored on various levels of cache depending on the spatial and temporal properties of the data.

The Cachebench run eight different tests to determine the overall efficiency of the cache levels. The first three tests are the read, write and modification tests for the cache. In those tests, data of different length are randomly written, read and modified systematically to measure the efficiency of the memory system. Next, three tests are conducted that are similar to previous three tests; however, they use a specially modified chunk of codes, which are not affected by the optimization method of any compiler. These codes are part of the commercial software that is known not to get changed by well-known code optimizers. That is why the scores from those tests remain almost the same over different systems even though different optimization technique may have been used.

The last two tests are involved with using well known C library functions *memset()* and *memcpy()*. C applications frequently use these two functions to both access and modify the memory. Therefore, their scores are good indicators of the overall performance of an application on the system. The Cachebench was chosen because all eight tests together provide a relatively balanced picture of the performance of memory and cache hierarchy.

3.4.2.2 Stream

The *Stream* [225, 226] is a light-weight synthetic benchmark for measuring the memory bandwidth of the system. It consists of four basic operations that are performed on vectors [227]. The first test is called the COPY, where the elements of a vector are systematically copied from one to another vector of the same size, this test is designed to mimic the operation of a generic garbage collector. The tests are done on pre-allocated large size vector of double-value elements residing in main memory. Next test is the SCALE, where a large scalar value is multiplied with all the elements of a vector.

The third test is SUM, where all the elements of two vectors are added. In the last test contents of one vector is multiplied by a constant value and added to the contains another vector, the operation is known as the TRIAD. The Stream is designed for the analysis of the frequently applied memory operations; hence, it is useful for VM memory usage analysis.

3.4.3 Disk I/O intensive benchmarks

The I/O is an essential resource for a system. Three I/O intensive benchmarks have been used in the experiments, they are discussed below. The I/O intensive VMs usually show a higher amount of variation compared to CPU intensive VMs. Three benchmarks have been used for the disk I/O as they can influence the VM performance significantly. Furthermore, the performances of virtual I/O devices get easily affected by the presence of other VMs. Thus, the VM I/O usage is a significant factor for the system performance.

3.4.3.1 Dbench

The *Dbench* [228] is a filesystem benchmark, it can quickly generate a significant amount of load to put a filesystem under stress. It generates the I/O workload for the file system from a pre-configured "*loadfile*", which can be designed by one according to one's requirements. Specialized loadfiles are available for downloaded.

The primary purpose of generating workloads is to find out the relationship between the amount of I/O stress and performance loss of the system. Analyzing the I/O characteristics of a VM is vital because the impact of consolidation is most profound on the I/O intensive VMs. This capability of putting the I/O system under stress according to a predefined loadfile makes this benchmark significantly useful. According to the requirements, it is possible to build custom loadfiles to match the I/O usage pattern of a parallel application. The loadfile makes it possible to perform the same test as many times as necessary repeatedly.

3.4.3.2 Filebench

The filebench [229] is another filesystem and storage benchmark used in the experiments. A wide variety of workloads can be generated by using the Filebench along

with *Workload Model Language*, which is a synthetic application model description language used to reconstruct the I/O usage footprint of any application. Two predefined workloads are also available for download; they are called the *file-micro* and *file-macro*. These two workloads are designed by analyzing various commercial software.

The file-macro workload has been designed to test the overall file system performance of a server. It includes the small and large size database servers, multithreaded web and proxy servers, mail and file servers. On the other hand, file-micro uses both random, and sequential writes, and reads operations to test the file system performance at the micro level. Various combinations of reads and writes operations are performed along with other operations like the file create and delete to test the basic response time of the file system. The experts in the respective fields have created both of those workloads. The workload files are useful for putting the file system under a considerable amount of stress.

3.4.3.3 Iozone

The *Iozone* [230] is another file system benchmark that can be used for the analysis of the I/O system. In the Cloud environment, different VMs may run applications with different I/O requirements at different times. Hence, for a particular application, the VM may show good performance while for applications it may. The Iozone is designed to explore the I/O performance of a system.

The Iozone contains some microbenchmarks and specialized test for the I/O system. It performs a mix of operations to measure the bandwidth of the I/O system. In the tests file sectors are read and write in both forward and backward direction in some predefined patterns. The patterns include sequential, random, and stride.

The two specialized tests of IOzone are the *Mmap* and *Async I/O*. The Mmap test uses Unix system call *mmap()* to map disk spaces to the memory address space of the user. Then, data stored in the mapped portion of memory are written to a disk file. The test is mainly focused on evaluating the performance of the mmap in a system. The memory mapped files are different from regular files in semantics. They treat data like chunks of memory mimicking the behavior of well-known Cloud applications, like the *MapReduce* [231].

Next, the Async I/O test measures the performance of *POSIX async I/O* operations

in the system using asynchronous I/O interfaces like, $aio\_write()$, $aio\_read()$ and $aio\_error()$. Such asynchronous I/O operations are frequently used in applications; hence, the Iozone is helpful for measuring the I/O performance of applications.

3.4.4 Other benchmarks

Some benchmarks can perform multiple resource-intensive tests. Two such benchmarks are discussed here; they are the *LMbench* and *Sysbench*.

3.4.4.1 LMbench

The LMbench [232] is a system benchmark suite consisting of a set of micro-benchmarks to measure the system performance. The micro-benchmarks have two categories; they are the bandwidth and latency tests.

The purpose of bandwidth tests is to measure how fast data can be moved around on the system. Performance of any application greatly depends on the data movement efficiency of the system. The performance of both the system overhead and memory clock cycles are tested. On the other hand, the latency tests measure the time taken by the system to carry out commands. Time taken by the system to execute the control messages is essential for the response time of an application. The micro-benchmarks of the LMbench are designed to measure the response time performance.

Another design goal of the LMbench was to make the benchmark as portable as possible. It is entirely written in ANSI-C and uses POSIX interfaces so that it can be used on a broad range of OS.

3.4.4.2 Sysbench

The sysbench [233] is a popular system benchmarking suite with various types of tests programs. It is a multi-purpose benchmark that can be used to measure the performance of individual system resources like CPU, memory, and disk I/O and complex processes like multi-thread operations, and Online Transaction Processing (OLTP) [233]. In the OLTP test, a MySQL database is used. The OLTP is an important part of the web applications, which are commonly deployed on the Cloud. The ability of the MySQL server to successfully process transactions in a given amount of time is calculated. The transactions are themselves small in size; to complete them

the database access is required, and the database access operations consume multiple system resources. The goal of the OLTP test is to put pressure on the MySQL database server, which in turn puts pressure on multiple system resources like, CPU, memory, and disk. The benchmark is very suitable for reproducing the resource usage footprint of most database applications.

Two observations play essential parts in performance benchmark design. The first is that applications usually spend most of the time executing a small segment of code [234]. The second is that the application performance depends on some factors like the system call handling efficiency of a particular machine [222]. Those two factors significantly affect the performance of an application. The benchmarks are designed based on statistical analysis of popular applications.

There is no universal consensus on the two issues; therefore, statistical analysis from different angles have given rise to different benchmark suites. No single benchmark is sufficient enough to provide the complete picture of the VM performance interference. The goal of this work is to analyze the performance of co-located VMs, and different benchmarks suites are used for this purpose. Each benchmark has some unique features and representative of a particular type of application. In the experiments, workloads are build by combining those benchmarks.

3.5 Data collection process

The previous section describes the benchmarks used in the experiments in this chapter. This section describes how the data is collected from the VMs during the experiments. Linux distributions are accompanied with some built-in tool to gather data about the running processes, these tools along with some other utilities are used. The resource usage data is collected from all the VMs and physical host.

Figure 3.1 shows the diagram of the experiment setup and data collection process. The middle section of the figure shows that several hosts are running different numbers of consolidated VMs. In a data center, different hosts may concurrently run different numbers of VMs at different times. The experimental setup replicates this situation by running different numbers of VMs on different hosts.

A virtualized host is capable of running several VMs concurrently. Figure 3.1 shows the VMs that are running on the host with solid lines. Empty VM slots on the

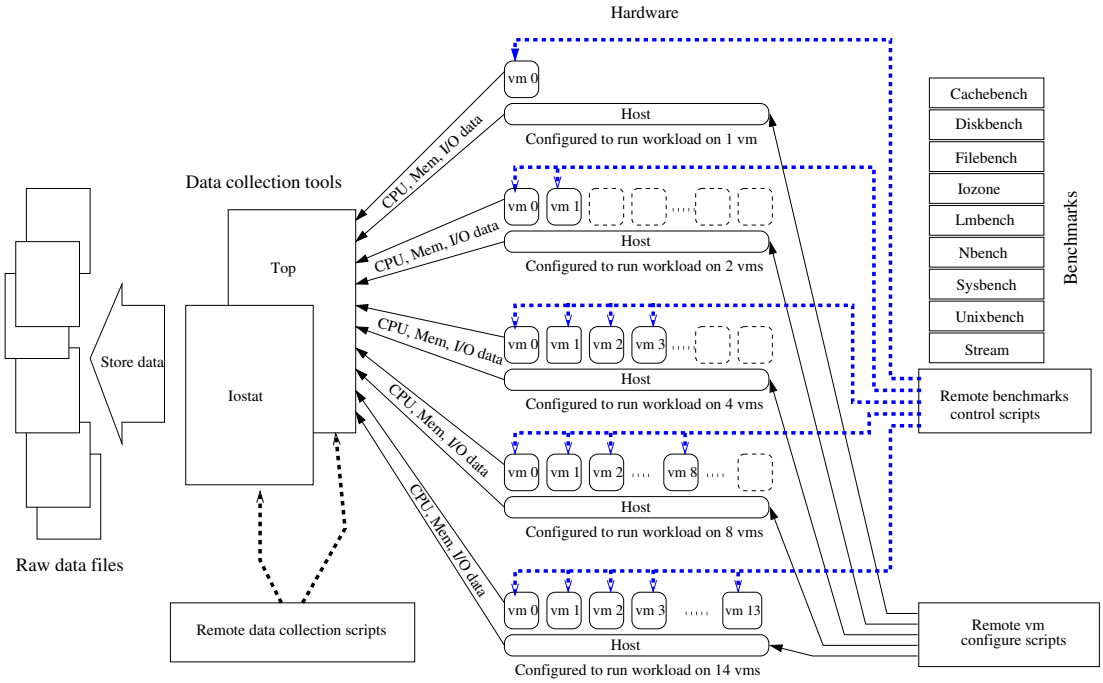


Figure 3.1: Data collection process from the co-located VMs.

server are shown with dashed lines; the empty slots are shown to indicate that the host is capable of consolidating more VMs.

The right side of Figure 3.1 shows the scripts are used to control the benchmarks and VMs. During the stages of experiments, several benchmarks need to be run. Also, the number of concurrently running VMs need to change at each stage. The shell scripts on the right are used to control the benchmarks and VMs. The scripts themselves run on a separate machine and connects to the host through SSH connection.

On the left side of Figure 3.1 several other scripts are running to collect resource usages data from the hosts. The scripts run tools to collect information about CPU, memory, and I/O usages. These scripts are also run on a separate machine and store all data on the remote machine. Data from different tools have the different format, and they are stored in files for later analysis.

Next, Figure 3.2 shows how the usage data collected from the physical host and each VM on the host. As mentioned before, that three resources usages are monitored, they are CPU, memory, and disk I/O.

The Linux *top* command is used for collecting data of CPU and memory usage data. For the CPU various data are gathered from all VMs and host. Those include the

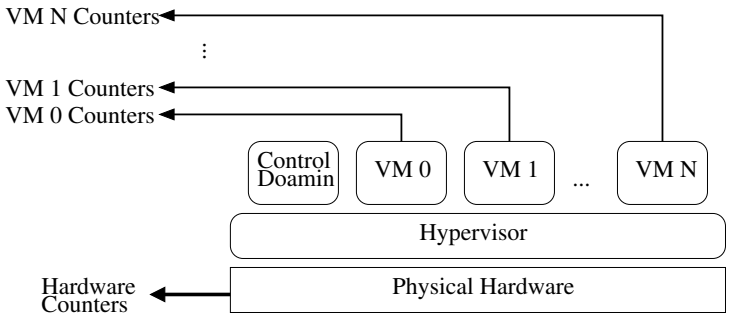


Figure 3.2: System data collected from the co-located VMs and host.

percentage of CPU usage by user and system level processes, system idle time and I/O-wait percentages. The top command provides information about used memory pages; however, they are not in the same format as the CPU data. Therefore, memory usage data need to be refined and scaled.

The *Iostat* [201] is used to measure the disk I/O activity of the VMs and physical host. The Iostat provides various data about device utilization, including the number of kilobytes written and read from the device per second. The data provided by the Iostat also need to be scaled and formatted. The system parameters collected with the above tools are summarized in Table 3.2.

The experiments were conducted with different numbers of simultaneously running VMs. First data is collected by running each benchmark separately on a single VM on the host; resource usages trace for each benchmark is collected separately. Afterward, VMs are run in groups of two, four, eight and fourteen co-located VMs. This process is repeated for all nine benchmarks.

Shell scripts were used to run the experiments from a remote machine over the *Secure Shell (SSH)*. The shell scripting has been used for two reasons. The first is shell scripts can efficiently run benchmarks and collect data from remote machines. The second is that the scripts can be easily created and manipulated. The shell scripting is a powerful way to run processes and collect information about the running process from the system. For each benchmark, a separate script is written to run the experiments and collect information. All the collected data are filtered and scaled using another shell script for analysis.

Resource		Field	Description
CPU	1	*%us*	Percentage of the CPU for user processes
	2	*%sy*	Percentage of the CPU for system processes
	3	*%id*	Percentage of the CPU idle
	4	*%wa*	Percentage of the CPU processes waiting for I/O operations
Memory	5	mem	Current physical memory (RAM) in use
Disk I/O	6	rkB/s	Number of kilobytes read per second
	7	wkB/s	Number of kilobytes written per second

Table 3.2: System parameters profiled from the VMs.

3.6 Experimental setup

All the experiments are done on a Dell XPS-8500 system with an Intel i7-3770 processor with 16 GB of RAM. The i7-3770 processor consists of four cores having the clock speed of 3.4 GHz each. Each core, in turn, contains two threads, that is eight hardware threads in total.

The Fedora 17 is installed as the host operating system, and Xen 4.2 is installed on top of it as the hypervisor. The guest operating systems were all Fedora 16 with 1 GB of RAM and 30 GB of disk space each. Disk spaces for all the guest VMs are allocated through the logical volume manager (LVM). The Xen supports both file or LVM based VMs; nonetheless, LVM based system was chosen because it shows better performance and flexibility compared to file-based VMs.

As mentioned earlier the physical host has 16 GB of RAM and VMs were configured to have 1 GB of RAM. It was noticed that under such circumstances the highest number of VMs that the physical machine can run successfully is fourteen. The system becomes irresponsive when attempts were made to run any higher number of VMs simultaneously. Next section presents the results of the experiments.

3.6.1 The experimental results

The experiments were conducted with the nine benchmarks, which are discussed in the above section. These experiments are divided into two categories.

The first set of experiments is done with a single VM. In this case, a benchmark is

run on a VM, and various resource usage data is collected from both the VM and host. The resource usage data collected are for CPU, memory, and Disk I/O.

In the second set of experiments, the number of simultaneously running VMs is increased in the system. The objective is to observe how the resource usage pattern of VMs change due to consolidation. This set of experiments start with one single VM, and the number of VMs is increased in later stages; five different sets of simultaneously running VMs are used here.

During the experiments, data are collected at one-second intervals. Each data file contains up to $66,000$ rows of data, each line corresponding to one second of data. A file can be up to 250 MB in size; each contains over 18 hours of physical and several VM resource usage and hardware counter trace data. For each benchmark, both the host and VM resource usage traces are collected for analysis.

The experimental results are discussed in the following sections; primarily two sets of experimental results are presented. In Section 3.7, three types of resource usages of individual tasks are presented. The three types of resources are CPU, memory, and I/O. Individual tasks are run on the VMs, and their resource usages data is collected from both the VMs and physical host. The objective, in this case, is to examine the resources usages patterns of individual benchmarks.

Next, in Section 3.8, it is examined how the resource utilization patterns change due to consolidation. In this case, the number of co-located VMs are increased in the host; as a result, the resource usages patterns of benchmarks are changed. In this section, the utilization pattern of each resource is checked separately. During experiments, resource usages data is collected from both the VM and host, which are presented in the section.

3.7 Resource usages pattern of individual benchmarks

The first set of results show the individual resource usage of the nine benchmarks. Figures 3.5, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 3.10, and 3.11 show the data for this set of experiments. The benchmarks are separately run on individual VMs.

The objective of the experiments is to observe the resource usages pattern of various the benchmarks. In this case, the host is running only one VM and data are collected from both the VM and physical host. From the collected experimental data

the graphs are produced using GNUPLOT [235–237]. Throughout this dissertation, GNUPLOT scripts are used for graph generation and data visualization.

In each figure, the x-axis shows the execution time of the benchmarks in seconds. On the other hand, the y-axis represents the resource utilization within the range 0 to 1. The current resource utilization is calculated as the percentage of maximum resource usage. First, the resource utilization data from all the profiling tools are collected. Then, for each resource maximum resource usage is identified and current resource usage value is calculated as the percentage of maximum value.

All the figures of this section, Y-axis values has the range from 0 to 1. A 0 (Zero) value on the Y-axis indicates zero resource utilization, while 1 (One) represents the maximal (100%) utilization. All other values on the Y-axis fall in-between 0 and 1. The values are transformed in this way for continence of represent and comparison.

Figures 3.5 to 3.11 show that all benchmarks have different usages pattern for three resources; CPU, memory, and disk I/O. No two benchmarks produce identical resource usage traces. What is more, for the same benchmark the resource usages traces of the physical host and VMs are different. No workload was run on the physical host during the experiments; all benchmarks are run on VMs.

VMs can not directly access the physical disk; it is the duty of the hypervisor to carry out such tasks of accessing. The hypervisor does all the actual disk operations on behalf of VMs. To perform disk operations; the hypervisor needs to enter and exit the kernel mood several times; this results in increased CPU usage of the host.

From Figures 3.5 to 3.11, resource usages patterns of both the VM and host are shown. In each figure, the top graph shows the three resource usage pattern of the VM while the bottom graph shows the pattern for the host. It can be seen that resource usage patterns of the VMs and physical host are identical in some cases and not identical in other cases.

For example, Figure 3.5a shows the CPU usage of the Cachebench in black. This particular CPU usage pattern is created by running the Cachebench on the VM. Figure 3.5b shows the CPU usage pattern host in black, also. The host CPU usage pattern is due to CPU overhead caused by running the VM, and in this case, it is low. Figure 3.5a also shows the I/O usage pattern of the Cachebench in red.

Figure 3.5b shows that the I/O usage overhead for the host in red. In this case, the I/O usage overhead is significantly higher. Recall from the previous chapter (Section 2.6) that the three resources are virtualized differently, and virtualization overhead

can be significantly higher for memory or I/O intensive tasks. This disparity between the host and VM resource usages are further discussed in the following sections.

Each figure shows two graphs for a benchmark; one for the VM data the other one is for physical host. For example, Figure 3.3a shows the resource usage data collected for the Nbench from the VM. Figure 3.3b shows the resource usage data from the host for the same benchmark. Similarly, in the above figures, the resources usage of VMs and physical host are shown together.

3.7.1 Resource utilization patterns of CPU-intensive benchmarks

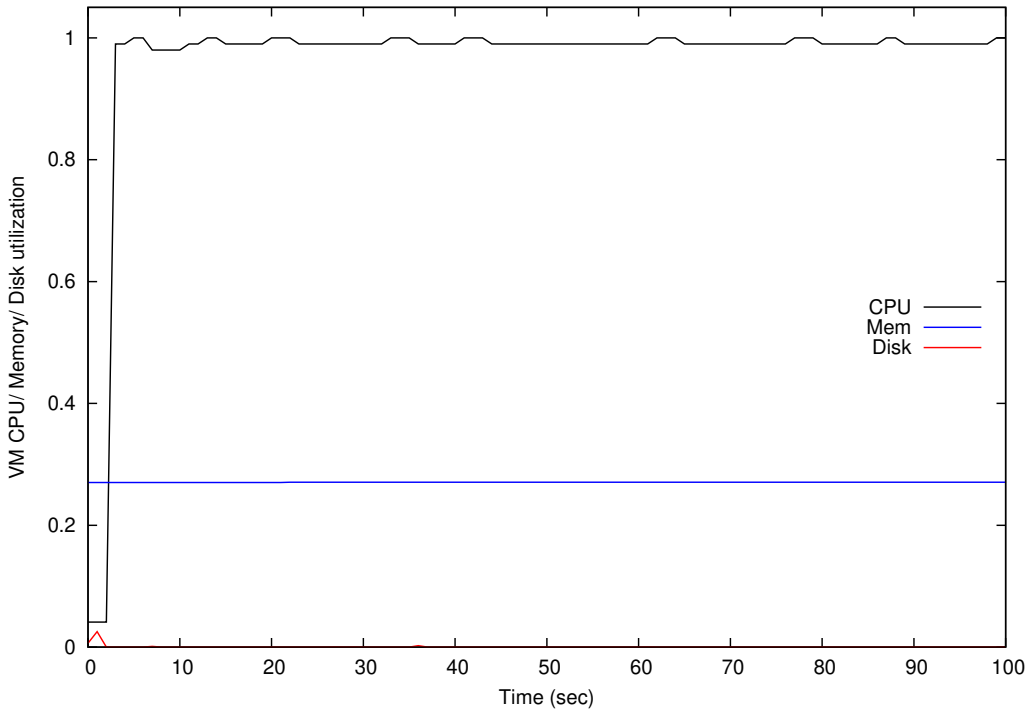
Figures 3.3 and 3.4 show the resource usage patterns of two CPU intensive benchmarks; the Nbench and Unixbench. In each graph, the three resource usage are shown in three different colors. In both cases, the VM CPU utilization is high; however, the host CPU usages are much lower. This difference is because of how the CPU resource is virtualized.

Figures 3.3b and 3.4b show the CPU usages pattern of the physical host. As discussed in the previous chapter (Section 2.6), the virtualization of the CPU resource is very efficient. In above figures, the VMs have a lot more CPU utilization compared to that of the physical host.

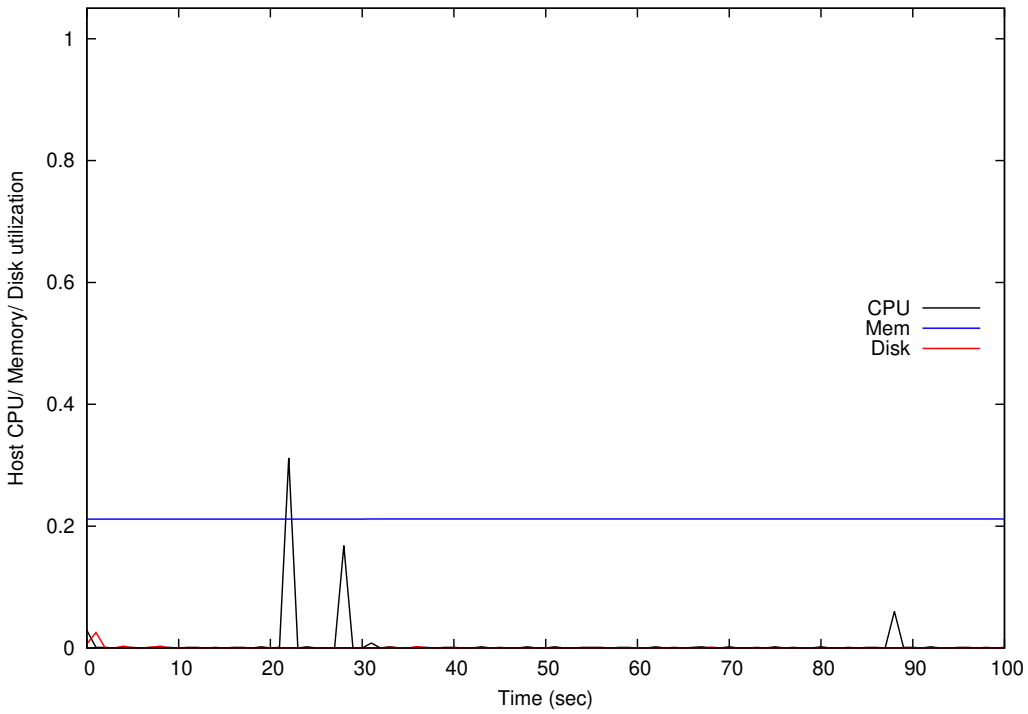
A virtual CPU can carry on execution quite smoothly except for a few privileged instructions; hence, not much overhead is induced on the physical host. When the workload is mainly comprised of CPU workload, the VM can continue execution without too much involvement from the hypervisor. That is why the host CPU utilization is different and do not reflect the CPU usage of the VM. Therefore, CPU usage data collected from the VM is more accurate compared to that collected from the host.

The resource usage data is collected through the Linux performance measurement tools. Performance measurement tools read various system counters and attribute the resource usage to a VM or the host.

In Figures 3.3 and 3.4, the actual workloads are run on VMs, and the performance profiling tool attributed a high CPU usage to the VM. On the other hand, the CPU usage of the physical server is due to the disk I/O operations performed on behalf of VMs and domain switching operations.

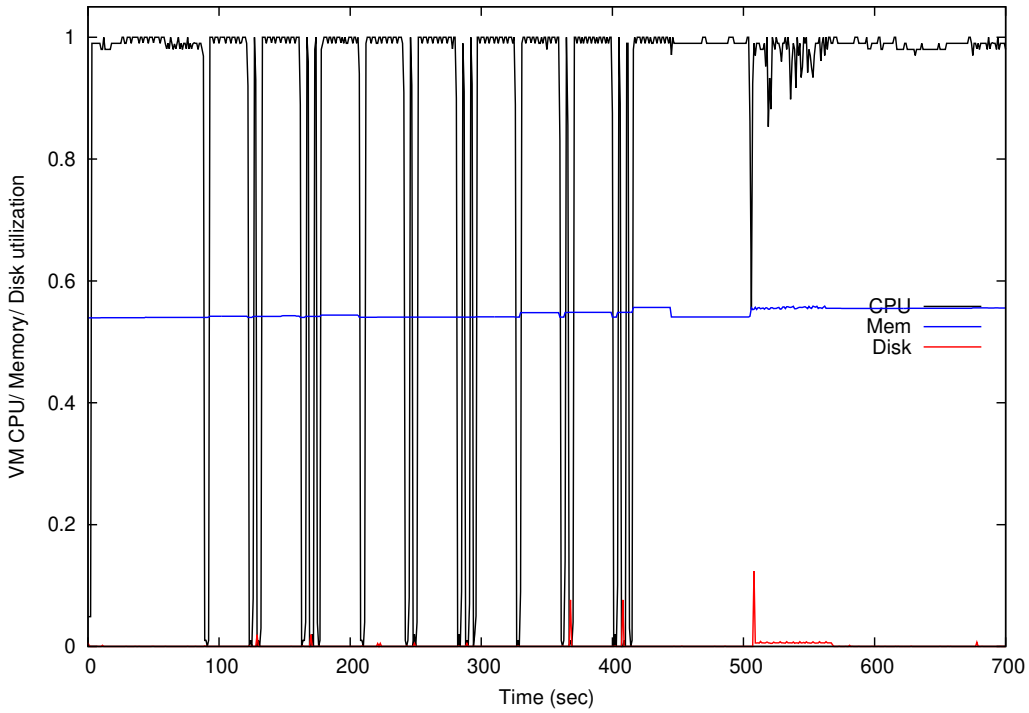


(a) VM data.

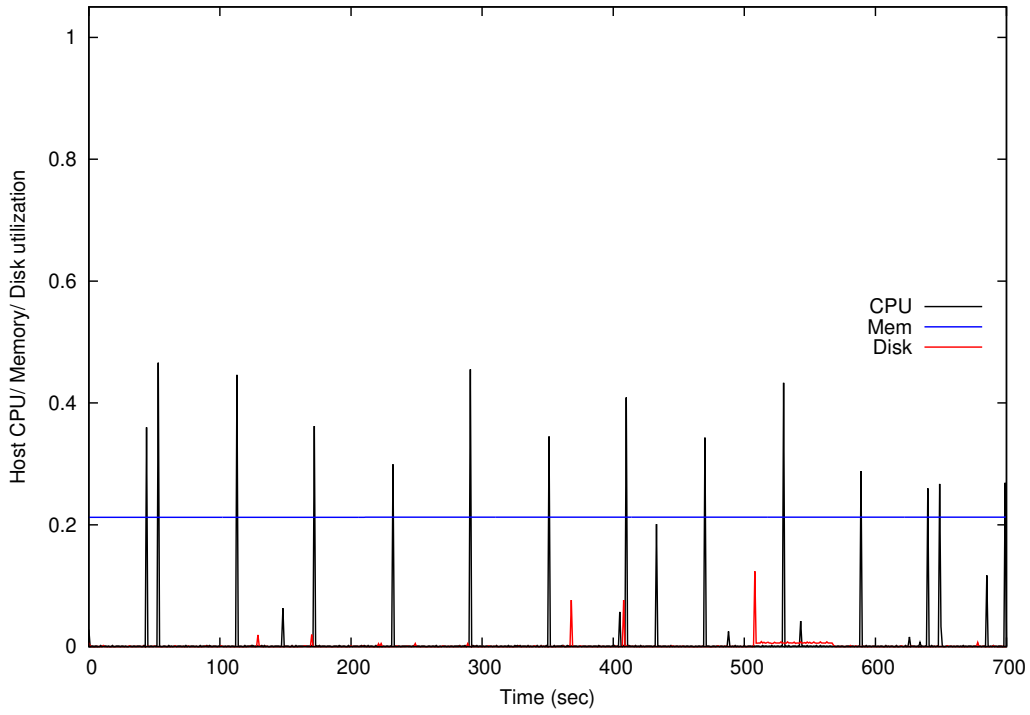


(b) Physical host data.

Figure 3.3: Resource usages pattern of a CPU intensive benchmark (1): the Nbench.



(a) VM data.



(b) Physical host data.

Figure 3.4: Resource usages pattern of a CPU intensive benchmark (2): the Unixbench.

3.7.2 Resource utilization patterns of Memory-intensive benchmarks

In this section, resource utilization patterns of two memory-intensive tasks are presented and discussed. The two memory intensive benchmarks are Cachebench, and Stream; their resource usages patterns are shown in Figures 3.5, and 3.6, respectively. Usually memory-intensive and I/O-intensive VMs show much more resource usage variance compared to the CPU intensive VMs.

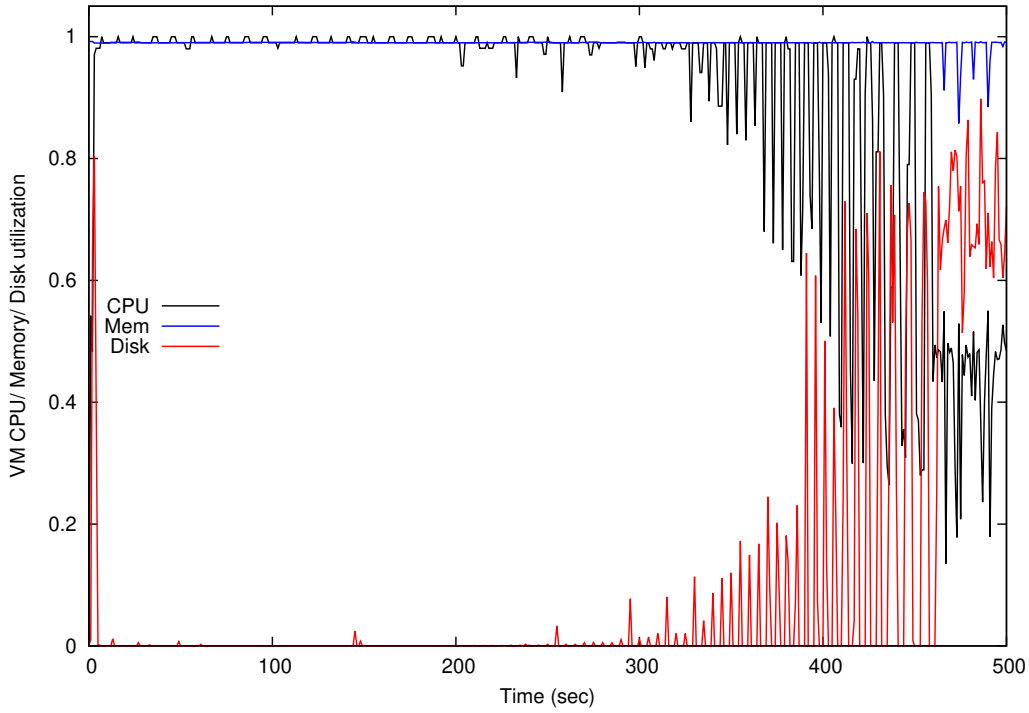
As before, the first graph of each figure shows the resources usages pattern of the VM. Figures 3.5a, and 3.6a show resource usage patterns of the Cachebench, and stream benchmarks on VMs, respectively. On the other hand, Figures 3.5b, and 3.6b show the host resource usage for the same benchmarks, respectively. In this case, the benchmarks are being run VMs; however, VMs need resources from the physical host to execute. Thus, the resource usages pattern on the host are created by the running VMs and they are shown on Figures 3.5b, and 3.6b.

Figure 3.5a shows that the VM memory usages of Ionone increase over time. However, Figure 3.5b shows that during the same period host memory usage do not show much variation. This lack of variation happens because of the hierarchical nature of system memory.

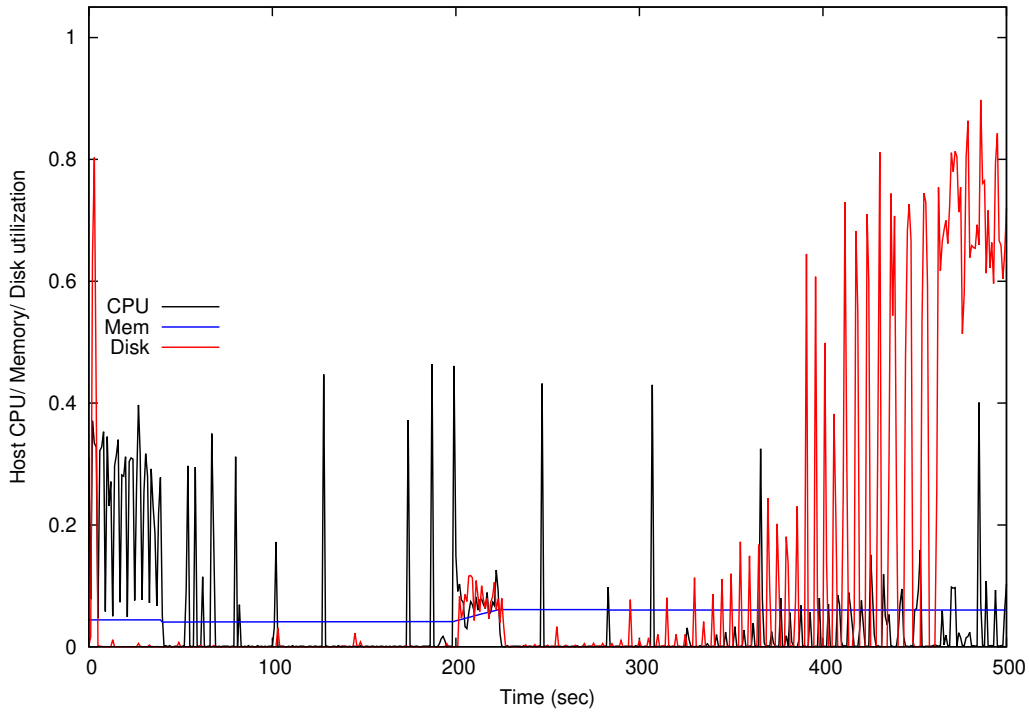
The VM keeps track of own memory; however, the hypervisor does not keep detail record about how the tasks of a VM. There is a temporal and spatial correlation among the memory data pages, and it is not easy for the physical host to acquire or release memory instantly in a virtualized system.

The hypervisor adds an extra level of indirection, which makes it more difficult for the physical host to determine which memory block is in use currently and which are not by the VM. The guest OS of the VM controls that information. All these factors subsequently make it difficult for the host to keep track of the VM memory usages. A VM is in much more informed position to know which pages are going to be accessed by the guest operating system shortly.

Thus, memory usage data collected from the VM are more accurate compared to the host data. Next, it is going to be investigated how the I/O resource usages vary between the VM and host.

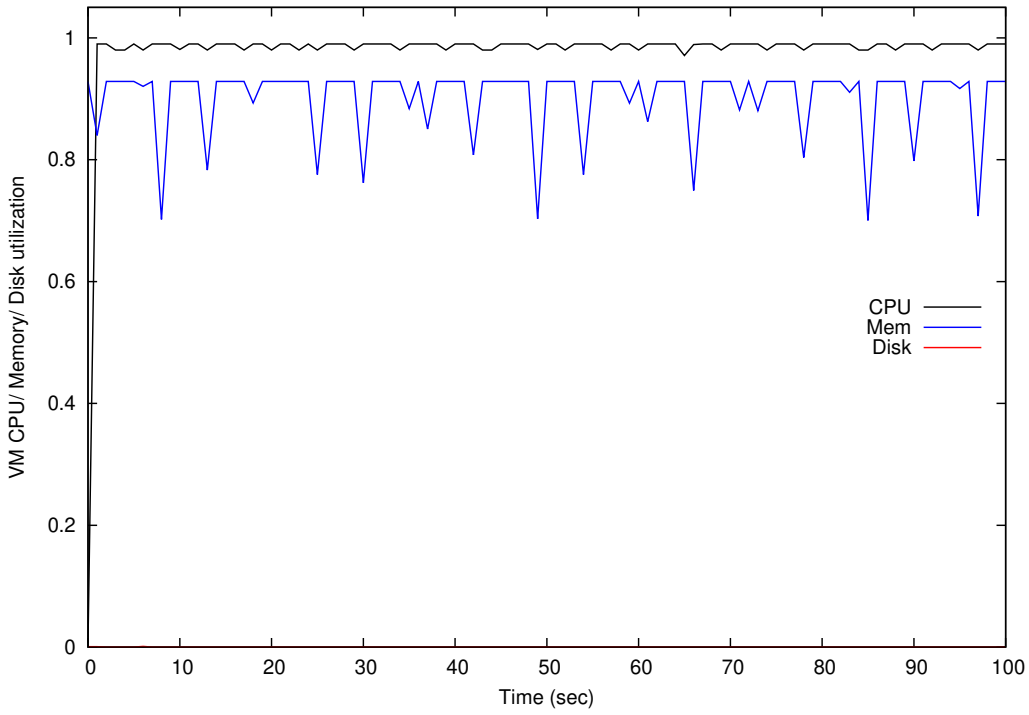


(a) VM data.

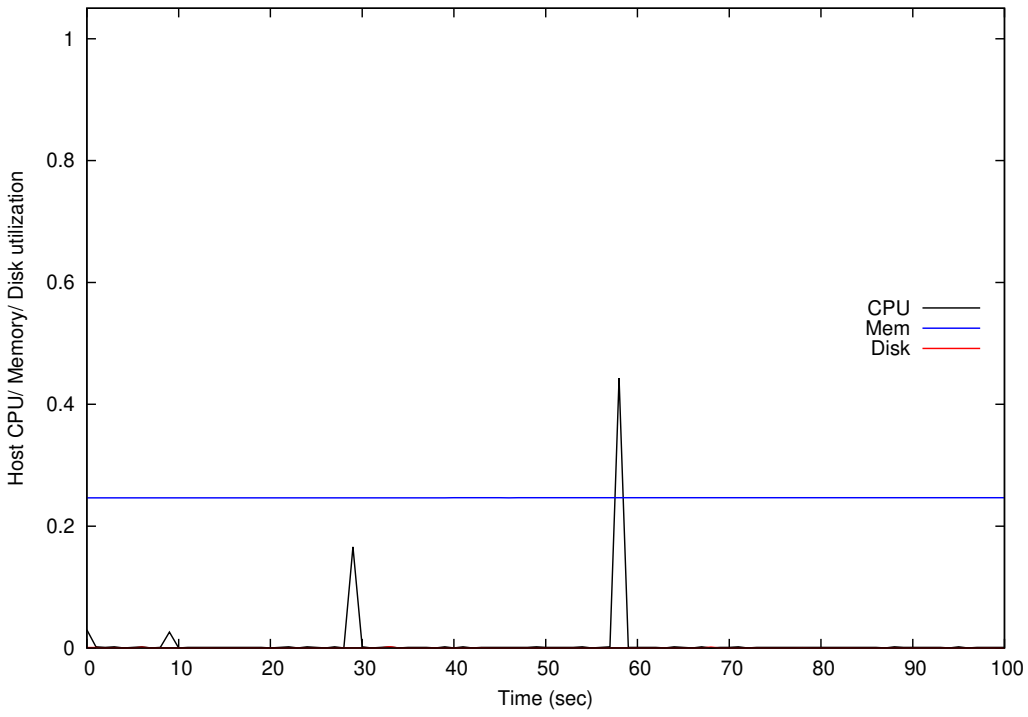


(b) Physical host data.

Figure 3.5: Resource usages pattern of a memory intensive benchmark (1): the Cachebench.



(a) VM data.



(b) Physical host data.

Figure 3.6: Resource usages pattern of a memory intensive benchmark (2): the Stream.

3.7.3 Resource utilization patterns of I/O and multiple intensive benchmarks

In this section, resource usages pattern of three I/O-intensive and two multiple resource intensive tasks are discussed. The three I/O-intensive benchmarks are Diskbench, Filebench, and IOzone; their resource usages patterns are shown in Figures 3.7, 3.8, and 3.9, respectively. On the other hand, two multiple resource intensive benchmarks are LMbench, and sysbenchOLTP; their resource usages patterns are shown in Figures 3.10, and 3.11, respectively.

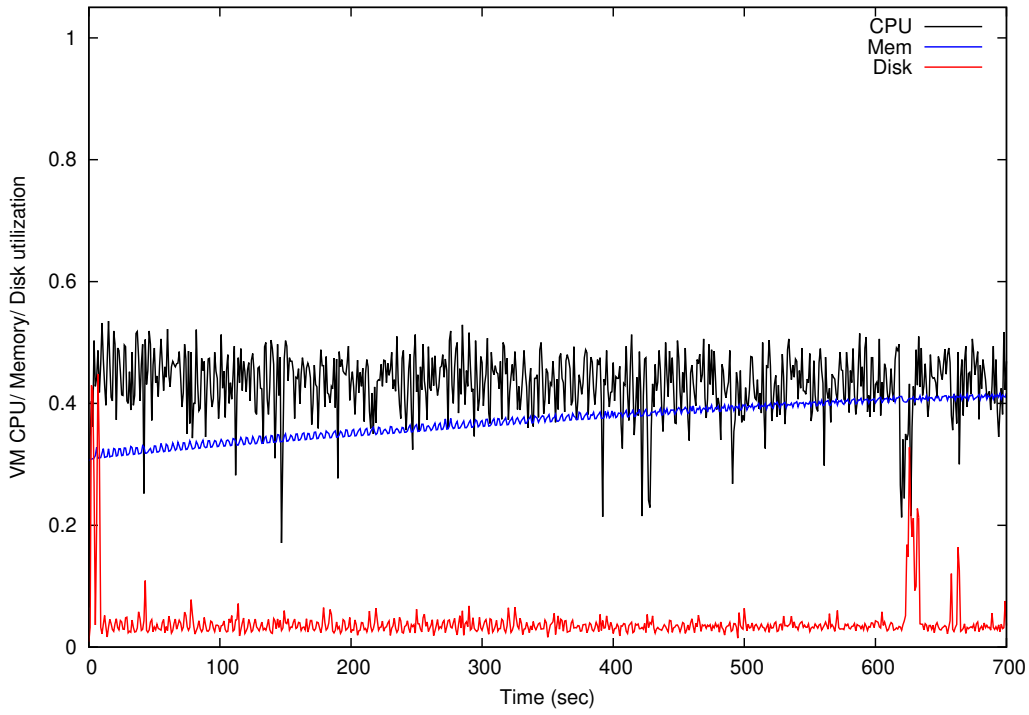
The multiple resource intensive benchmark suites have tests that can put pressure on multiple resources. Their tests are designed such a way that they can put pressure on all three resources; CPU, memory, and I/O. However, it is also possible to pass special parameters to those multiple benchmarks so that they run only one type of resource intensive tests.

In each figure, the disk I/O usages pattern of the physical host almost similar to that of the VM. Recall, that it was not the case for the CPU and memory the resource usage patterns of the VM and host. It can be seen that compared to CPU and memory virtualization the I/O virtualization have more effect on the host resources; reasons for this are discussed next.

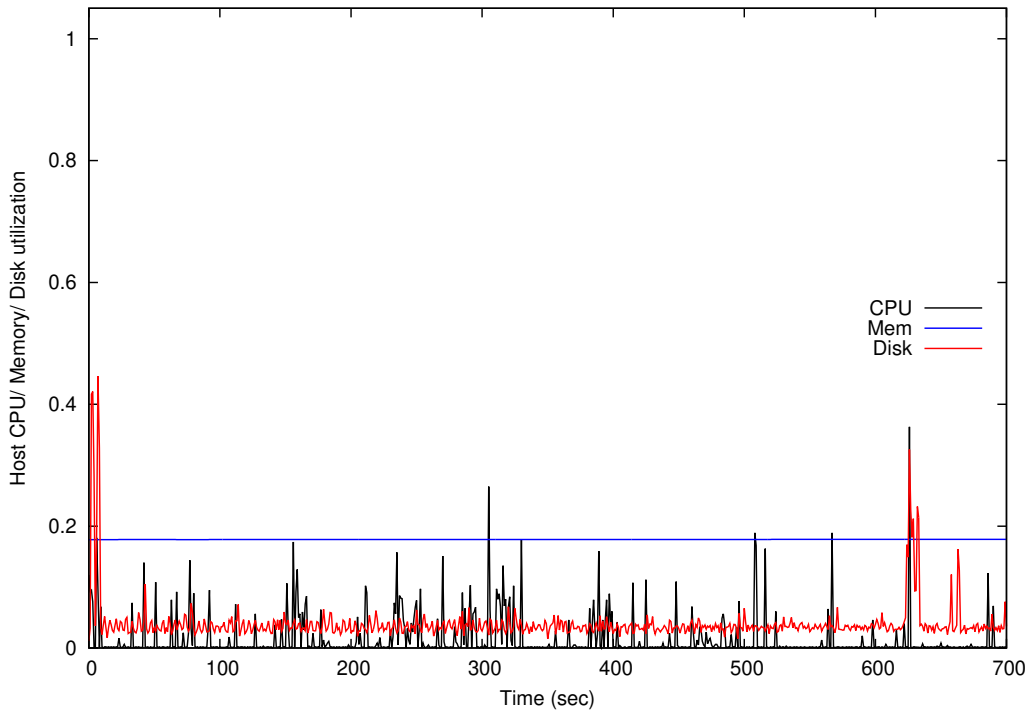
The non-privileged VMs cannot access the I/O devices directly. The I/O request inside a VM generates an exception and causes the system to enter into the *kernel mode*. In a para-virtualized system, like the Xen, the VMs need the help of the hypervisor to perform I/O operations. Interrupt handling is a complicated process.

First, the I/O data is copied to a specific location of the memory and the control is transferred to the hypervisor from the guest OS of VM. To accomplish this, either the guest OS of VM or the physical hardware must be modified. In any case, the hypervisor hides the absolute memory location from the VM and transfers the data to a special privileged domain, which does the actual I/O processing. Then, the new data is copied back to the same selected memory region, again.

Xen uses a double-ended circular queue to pass all the control information [53, 69]. Ultimately the privileged domain does all the real I/O processing to physical disk; that is why the I/O usages pattern of the host is similar to that of the VM. The I/O operation of VMs is a complicated process which involves multiple context switching and transferring control back and forth to the privileged domain. Thus, the I/O operations

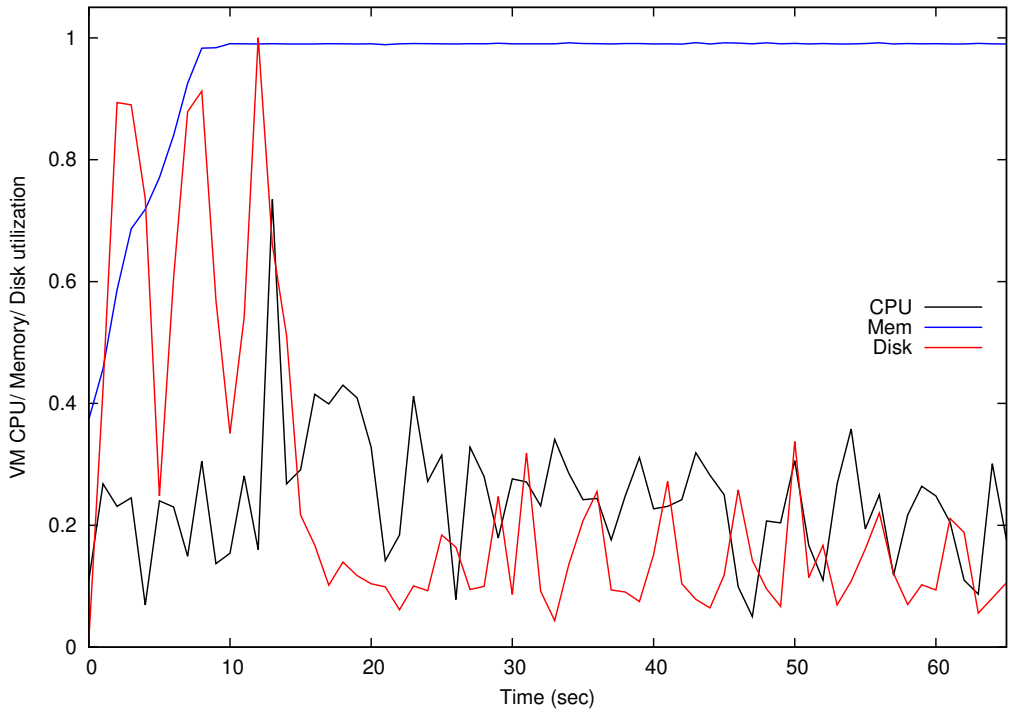


(a) VM data.

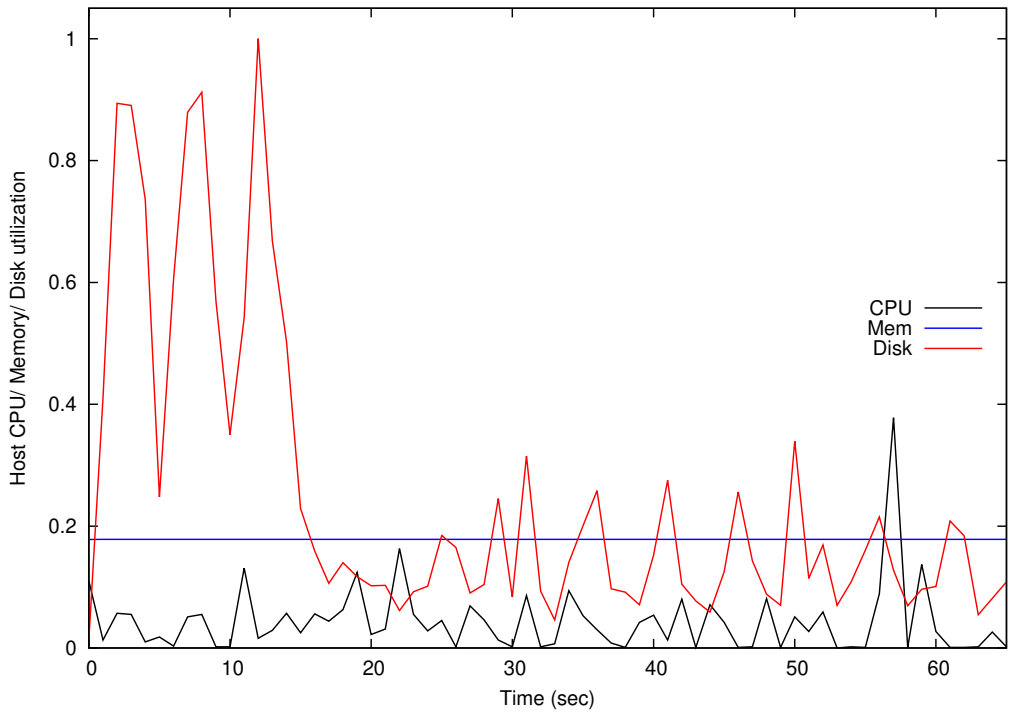


(b) Physical host data.

Figure 3.7: Resource usages pattern of a I/O intensive benchmark (1): the Dbench.

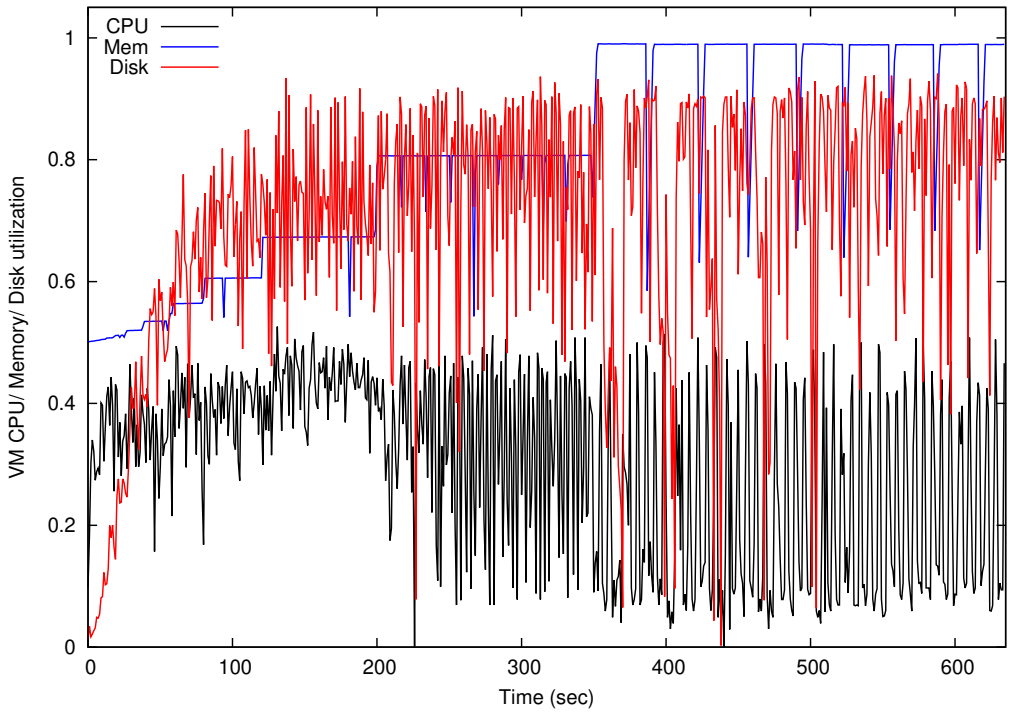


(a) VM data.

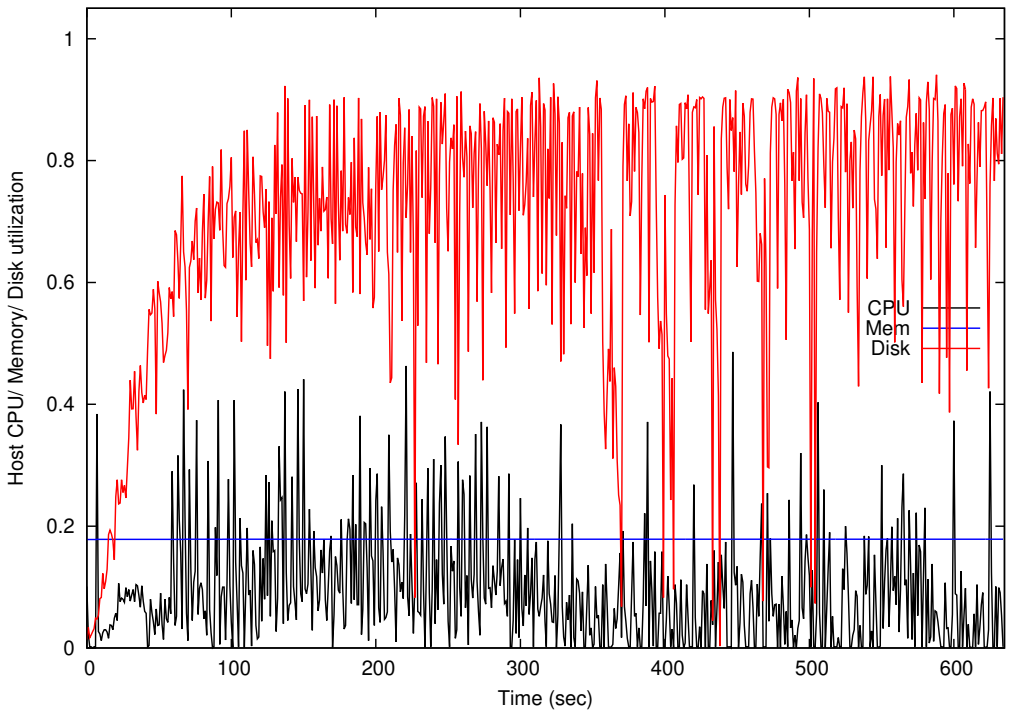


(b) Physical host data.

Figure 3.8: Resource usages pattern of a I/O intensive benchmark (2): the Filebench.

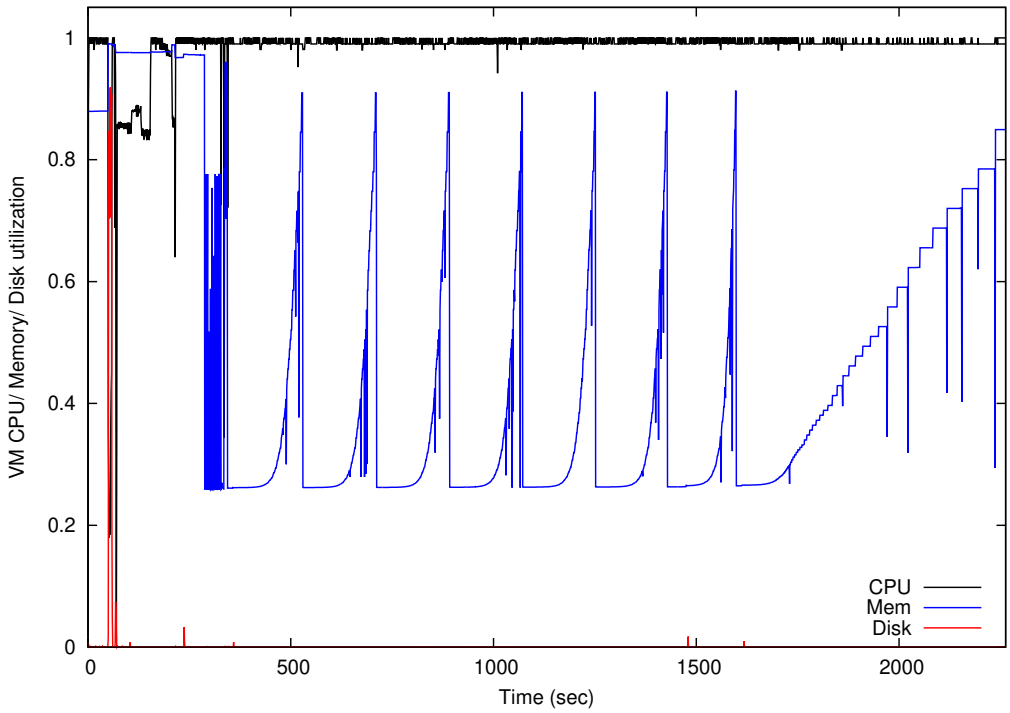


(a) VM data.

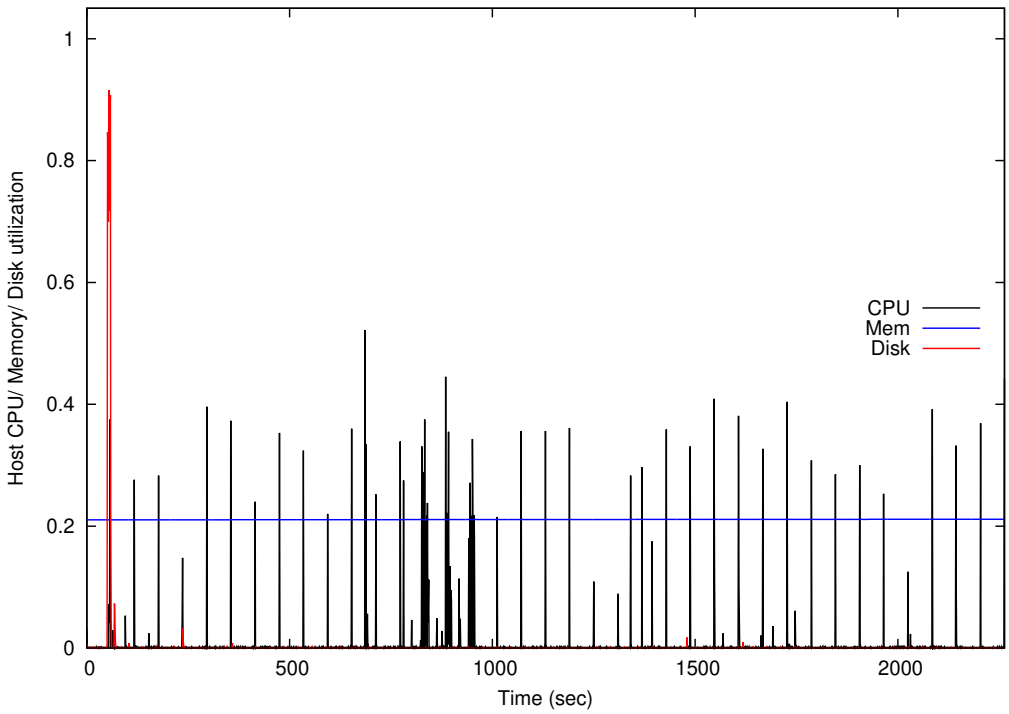


(b) Physical host data.

Figure 3.9: Resource usages pattern of a I/O intensive benchmark (3): the Iozone.

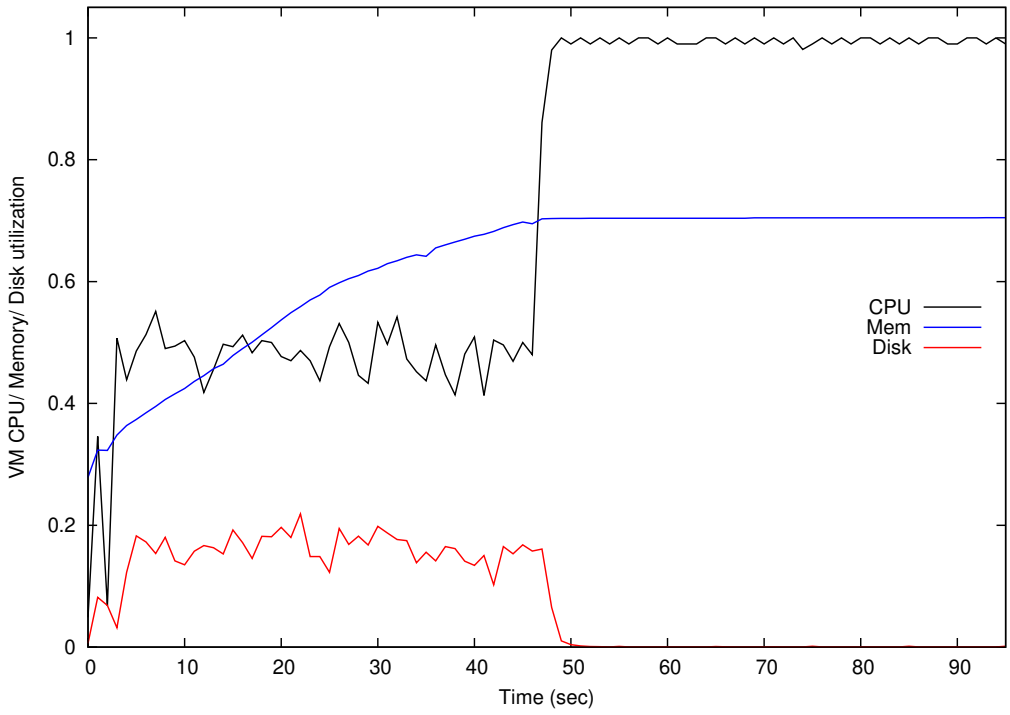


(a) VM data.

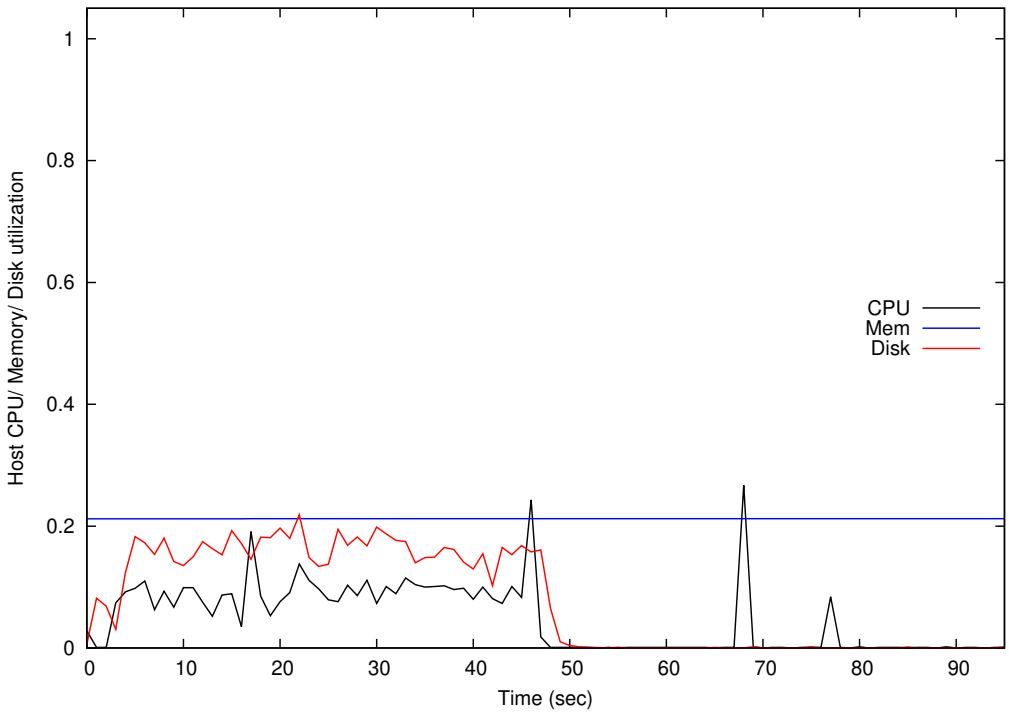


(b) Physical host data.

Figure 3.10: Resource usages pattern of the LMbench.



(a) VM data.



(b) Physical host data.

Figure 3.11: Resource usages pattern of the Sysbench OLTP test.

increase the CPU usage of the physical host, too.

For example, Figure 3.9b shows the I/O usages of the physical host for IOzone. Figure 3.9a shows the I/O usages pattern of the VM for the same benchmark. It can be seen that those two patterns are very similar. As the hypervisor is involved with the entire I/O processing of the VM; their I/O processing patterns are similar. Furthermore, Figure 3.9b shows that the CPU usages of IOzone physical host are higher compared to that of other benchmarks discussed before. I/O processing typically requires a large number of context-switching, this is what causes the higher CPU usages pattern for the physical host.

The resource usage data from this section shows that different benchmarks produce different resource usage patterns. Again, for each benchmark, the resource usage pattern of the VM and the physical host is not the same. In the next section, different combinations of the simultaneously running VMs are used to observe their resource usages pattern changes.

3.8 Resource usage pattern changes due to VM consolidation

This section gives the result for the second set of experiments. Here, more than one instance of the benchmark is run on the host at a time. In the previous section, the resource usage patterns of nine benchmarks are presented. This section investigates how the resource usage patterns on VMs change due to consolidation.

In this set of experiments, the benchmarks of the previous section are run simultaneously to observe the resource usage pattern changes. Total five sets of VMs are used for experiments; each set contains a different number of VMs. The experimental results demonstrate that different resources react differently to the VM consolidation. The number of VMs used in the five sets are one, two, four, eight and fourteen, respectively.

In data centers, servers usually host VMs of some predefined configurations; this is done to make the administrative tasks more manageable. If different VMs are created with arbitrary memory sizes, the summation of their memories may not be equal to the physical memory of the host, and some part of the physical memory will remain unused. Thus, uneven VM memory allocation can create problems; that is why in

Cloud the VMs are rented out in a selected set of memory configurations.

Next, the resource usage pattern changes of the benchmarks are discussed separately. As the number of co-located VMs changed in the system the resource usages pattern are affected. Each resource like CPU, memory and I/O shows different patterns of changes. That is why the effect of each resource is discussed separately. The discussion starts with the pattern changes in CPU resource utilization.

3.8.1 Effect of VM consolidation on CPU usages

Figures 3.12, 3.13, 3.14, 3.15 and 3.16 show the CPU usages change of the Unixbench, Cachebench, Dbench, LMbench, and Sysbench OLTP benchmarks, respectively. Recall that, the Unixbench is a CPU intensive benchmark, while the Cachebench is a memory intensive benchmark. The Dbench is an I/O intensive benchmark. The LMbench is multiple resource-intensive. Similarly, Sysbench OLTP test is also multiple resource-intensive.

Figure 3.12a shows the VM CPU usage pattern changes of the Unixbench. The Y-axis shows the time, while the X-axis shows the CPU utilization. Five CPU utilization data sets are plotted in Figure 3.12a, one for each VM set mentioned above. For example, *1vm* means only one instance of the Unixbench was run on a VM in the system. In this case, no other VM run any task. Similarly, *2vm* represents CPU usages data when two instances of the Unixbench are simultaneously run on two VMs of the server.

Next, *4vm* represents the CPU usages when instances of the Unixbench was running on the host. Figure 3.12a shows that up to eight simultaneously running VMs the CPU utilization is quite good for the Unixbench. However, for fourteen simultaneously running VMs (*14vm*) the CPU utilization drops significantly.

Next, Figure 3.12b shows the host CPU utilization for five VM sets. It can be seen that the host CPU utilization level is much lower compared to that of the VMs. Thus, in the case of Unixbench, the vCPU efficiently share and utilize hardware threads until eight VMs are running. However, with fourteen VMs the utilization degrades rapidly.

Next, Figure 3.13 shows the CPU usage of both the VM and physical host for the Cachebench. The graphs show how the CPU execution pattern changes with the changing number of simultaneously running VMs. In this case, all VMs run the Cachebench, a memory-intensive benchmark.

Figure 3.13a show the change of CPU resource usage pattern of VM due to the

changing number of co-located VMs. While a single VM is running (shown in black), the peak CPU usage is between 5 and 10 seconds; rest of the time CPU usage is average. However, when multiple running VMs are running the system the resources usages pattern changes into a few isolated peaks and low utilization regions.

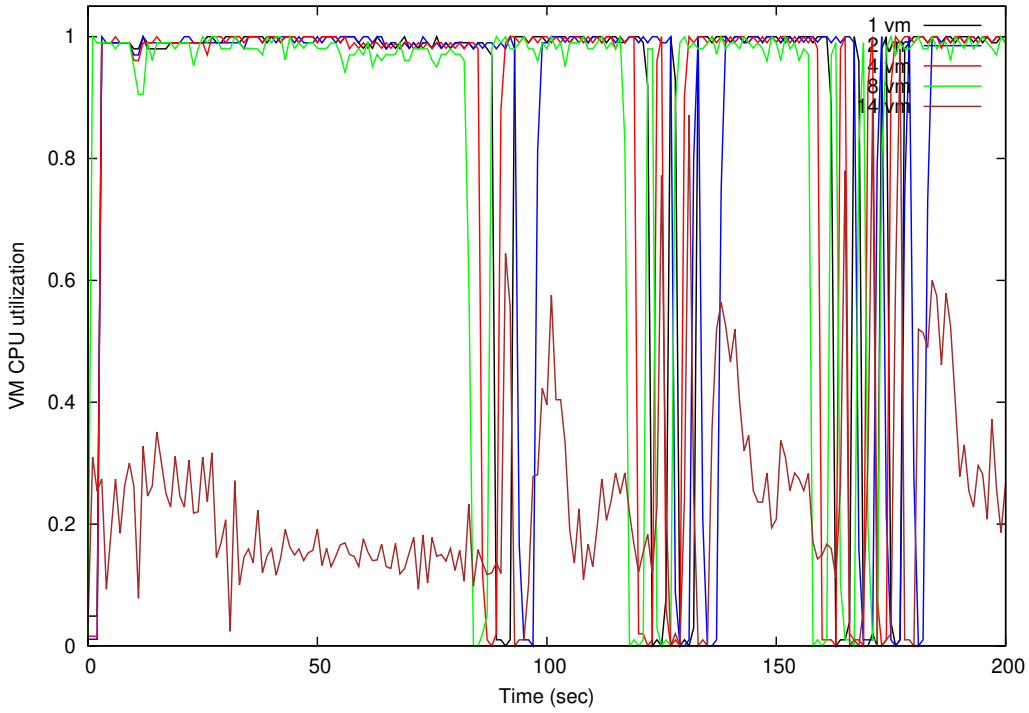
The reason is that there is only one physical memory hierarchy and multiple VMs are trying to access it. Thus, each VM has to wait for its turn, resulting in isolated peak CPU usage. When a VM is in wait-state, obviously the VM CPU utilization is low. It can be seen that for a single running VM the CPU usage is more or less average; however, when multiple VMs are running the CPU usage of them turns into segments of peak and low utilization regions.

Next, Figure 3.13b shows the CPU usage of the physical host during the experiment with Cachebench. It can be seen that CPU overhead of physical host is comparability small for one VM in the system and does not increase much for a higher number of simultaneously running VMs.

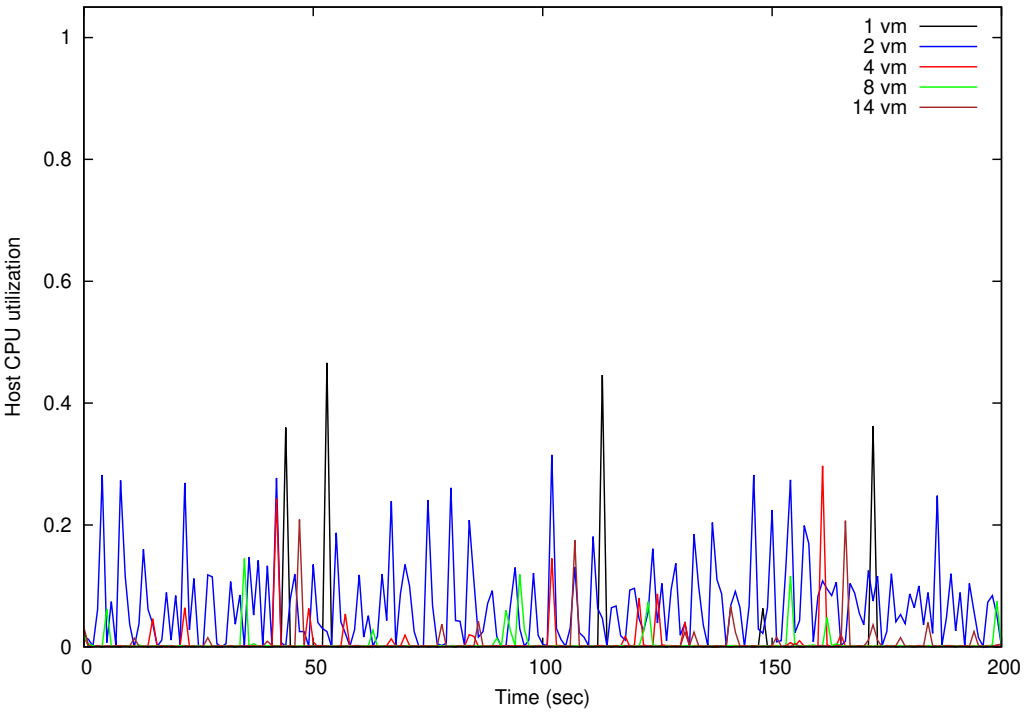
Next, Figure 3.14a shows the CPU usage pattern of the Dbench for five sets of co-located VMs. For one running VM, the CPU utilization is nearly 50%; this CPU usage can be attributed to the disk access operations by the hypervisor [33]. As the number of simultaneously running VMs are increased one can see a gradual decrease in CPU usage; that is because multiple VMs are competing for the disk system and increasing the resource contention. Thus, running too many VM can result in the overall decrease of I/O processing ability of the hypervisor.

Next, Figure 3.15a shows the CPU usage of the LMbench for the different number of simultaneously running VMs. The CPU utilization for two and four simultaneously running VM are high. However, up to eight VMs, the CPU utilization pattern remains almost the same. On the other hand, for fourteen simultaneously running VMs, the CPU usage pattern shows significant change.

As described in the experimental setup, the server has one Intel i7-3770 processor, which has four physical cores and eight hardware threads. Figure 3.15a shows that up to eight simultaneously running VMs the CPU utilization is high. As eight VMs are using eight hardware threads; one VM is using one hardware thread. The co-located VMs are sharing logical CPU efficiently, even though two VMs shares each physical core. However, when fourteen VMs are simultaneously running each hardware thread is shared by two VMs, and this leads to a dramatic reduction in CPU usage. The resource contention has a tremendous effect in this case.

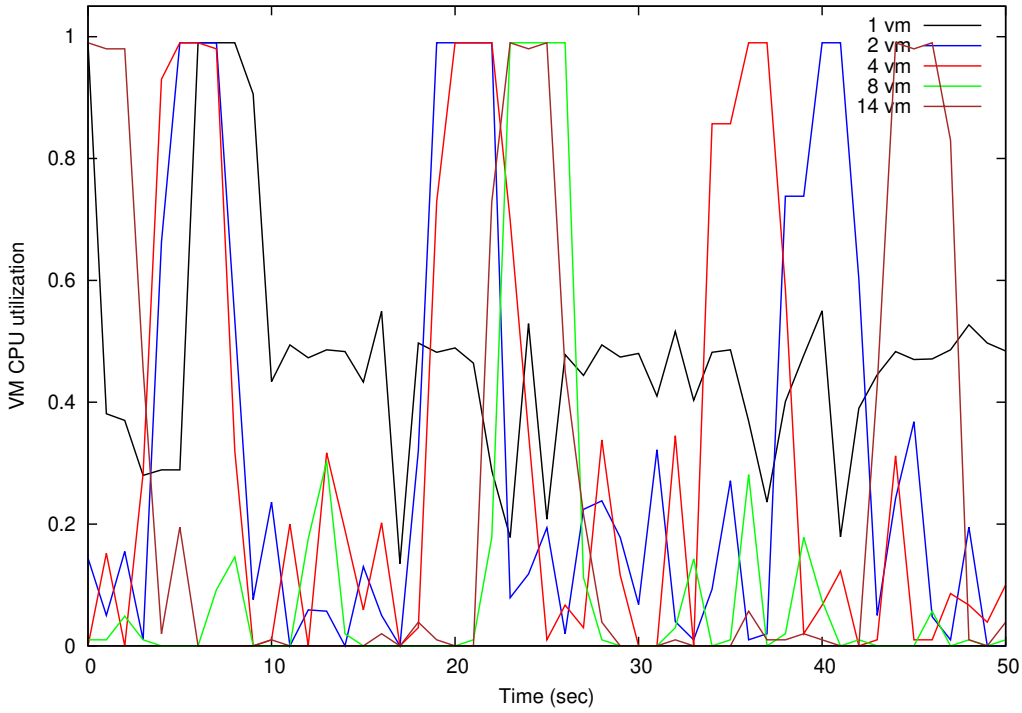


(a) VM data.

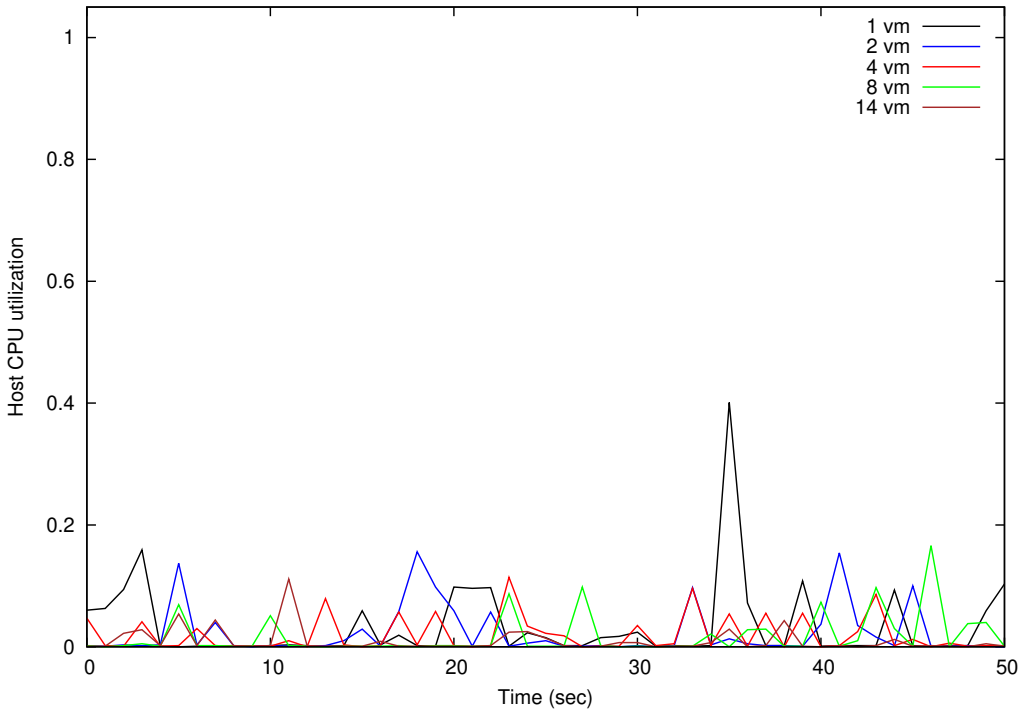


(b) Physical host data.

Figure 3.12: Unixbench: CPU usages pattern changes due to various number of co-located VMs.

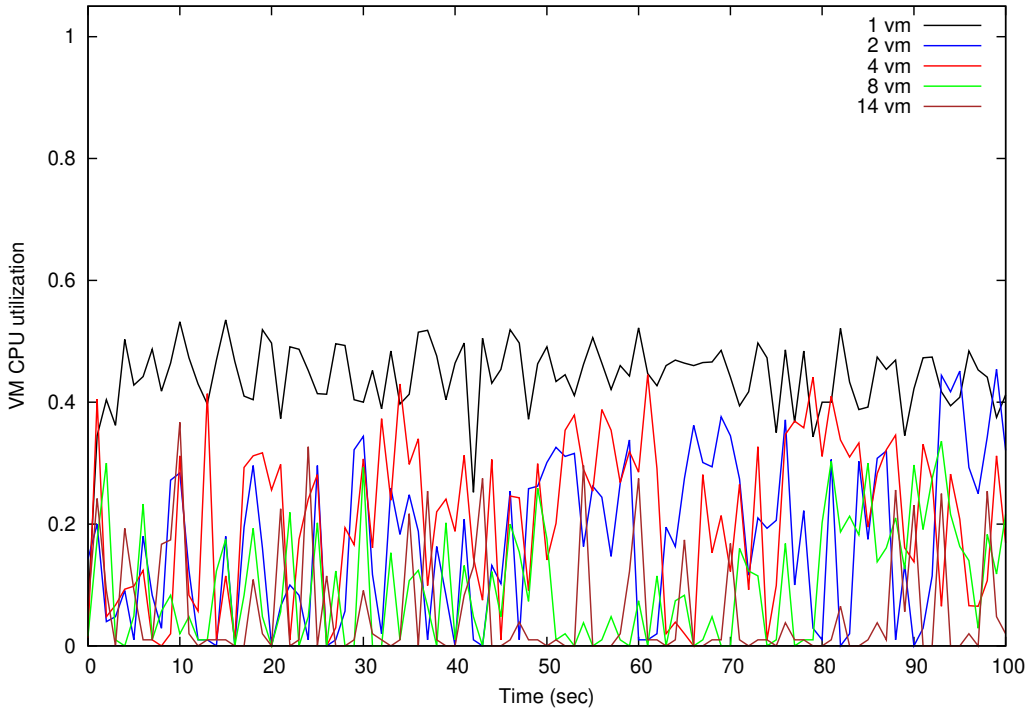


(a) VM data.

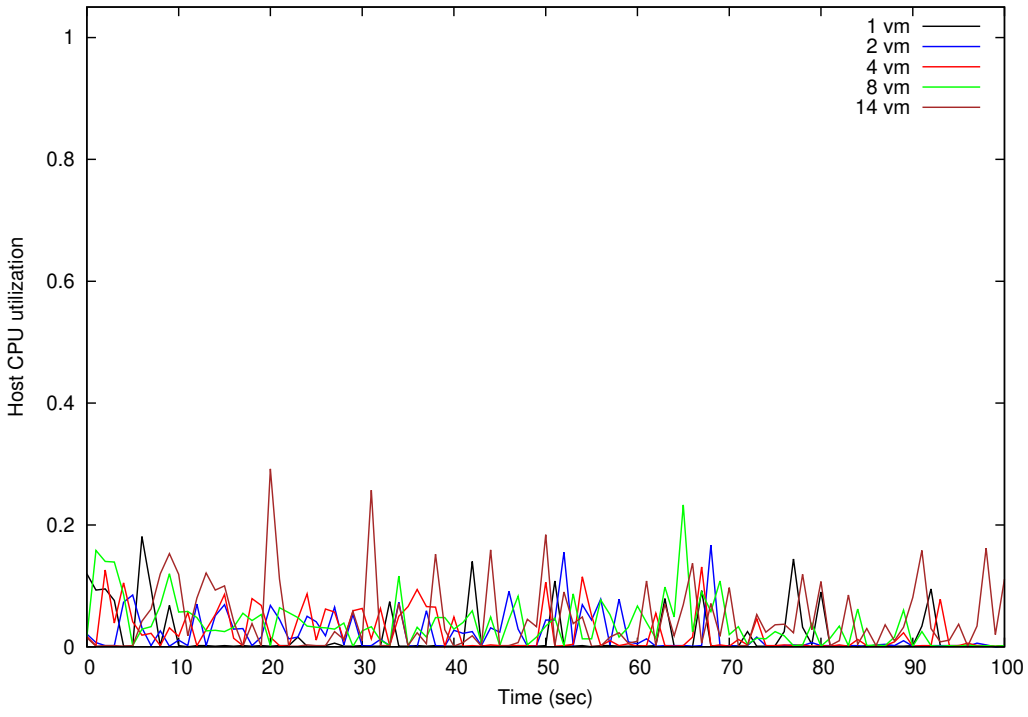


(b) Physical host data.

Figure 3.13: Cachebench: CPU usages pattern changes due to various number of co-located VMs.



(a) VM data.



(b) Physical host data.

Figure 3.14: Dbench: CPU usages pattern changes due to various number of co-located VMs.

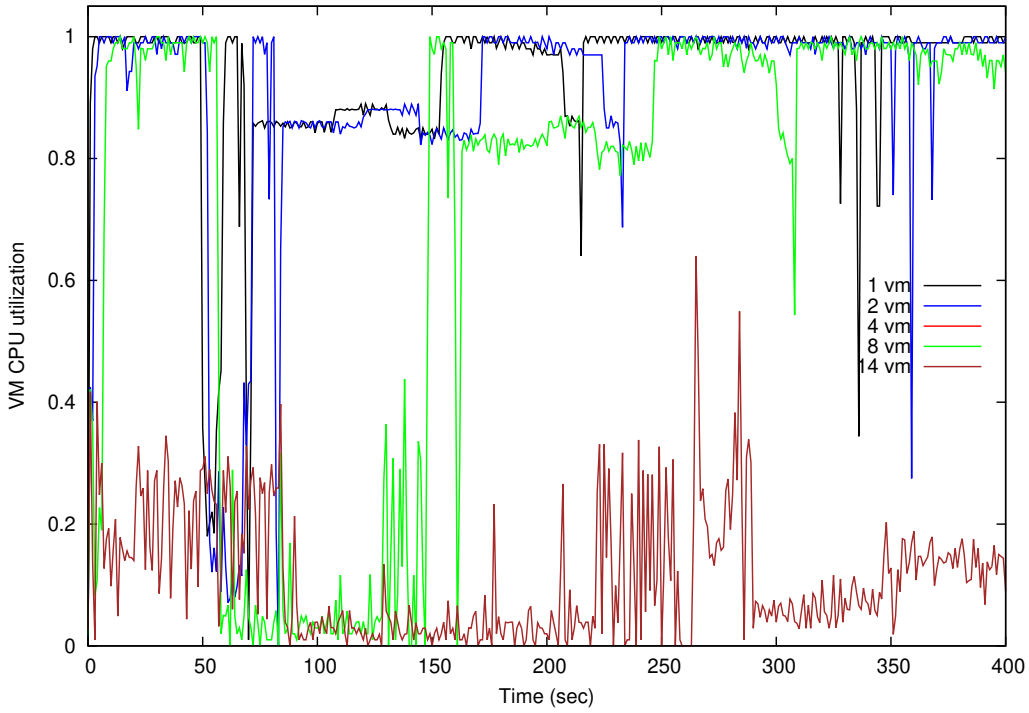
Next, Figure 3.15b shows that the physical host CPU overhead is increased for both eight and fourteen VMs sets compared to the lesser number of VMs. As the number of simultaneously running VMs are increasing the hypervisor has to handle more context switching and other interrupts. That is why the CPU usage overhead is increased in the system. The results show that an increasing number of consolidating VMs increases overhead in the physical host. As a result, the overall server resource utilization can decrease.

Figure 3.16a shows the CPU utilization of the Sysbench OLTP benchmark for various numbers of simultaneously running VMs. The OLTP benchmark carries out a complex set of operations involves CPU and I/O usage. Recall, in Figure 3.11a the resource usage pattern of OLTP is shown, the pattern has two parts. The OLTP benchmark reads data from the files in the first part; then, they are processed in the second part. In the first part, the benchmark has almost 50% CPU usage; however, lesser I/O usage. In the second part, CPU utilization is nearly 100%, while the I/O usage is negligible.

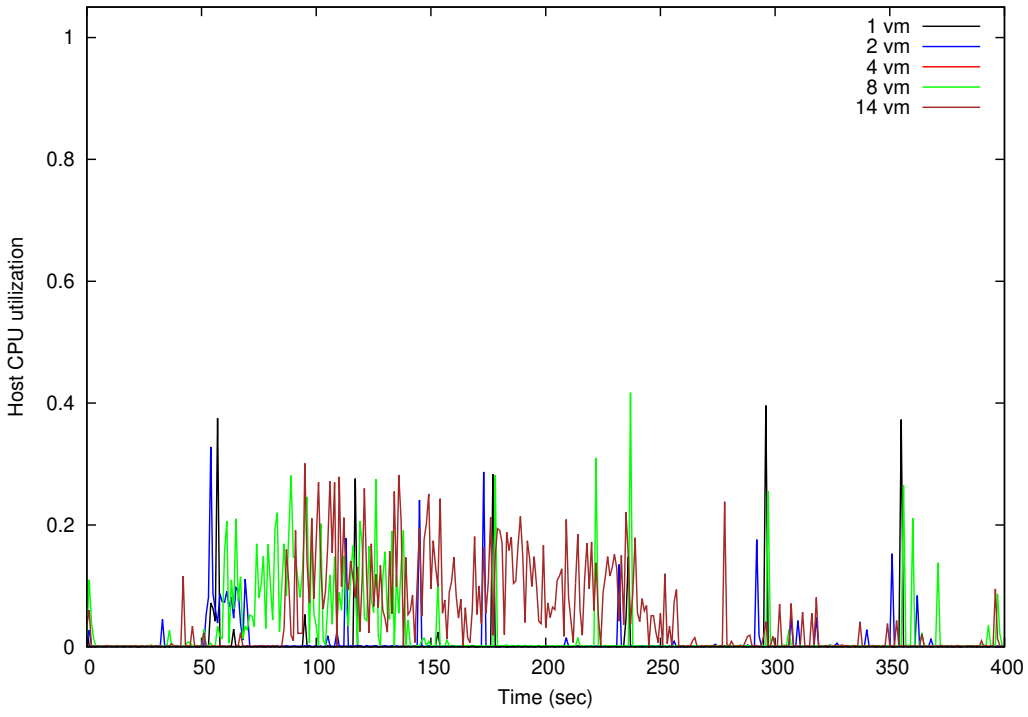
Figure 3.16a shows that once the number of simultaneously running VMs is increased in the system, the changes occur in both parts of the resource usage pattern. For single running VM (*1vm*), the first part of CPU operations finished around 45-second mark, followed by a peak in CPU usage between 50 to 100 seconds. However, when two VMs are simultaneously running, the CPU operations take longer to complete. The first part now finishes around the 55-second mark; therefore, this part of execution time is almost 10-second longer now. The second part does not start immediately; it starts around 95 seconds leaving a gap of almost 40 seconds in-between. Thus, the resource usage pattern changed significantly.

For other VM sets similar pattern changes can be seen; the CPU operation completion time is getting longer, and access to the disk is also getting delayed. The results show that the execution finish times of the benchmarks are extending due to consolidation. This finding is significant. In the next chapter, this observation is used to derive a new methodology for VM consolidation benchmarking.

Next section discusses the changes in memory usages pattern due to consolidation.

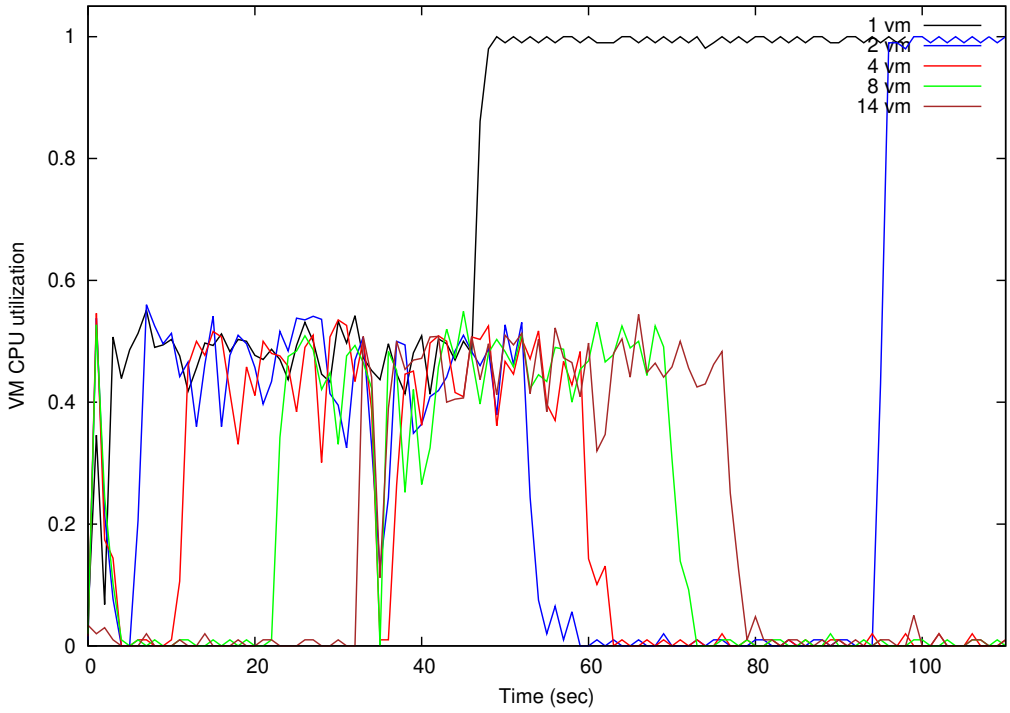


(a) VM data.

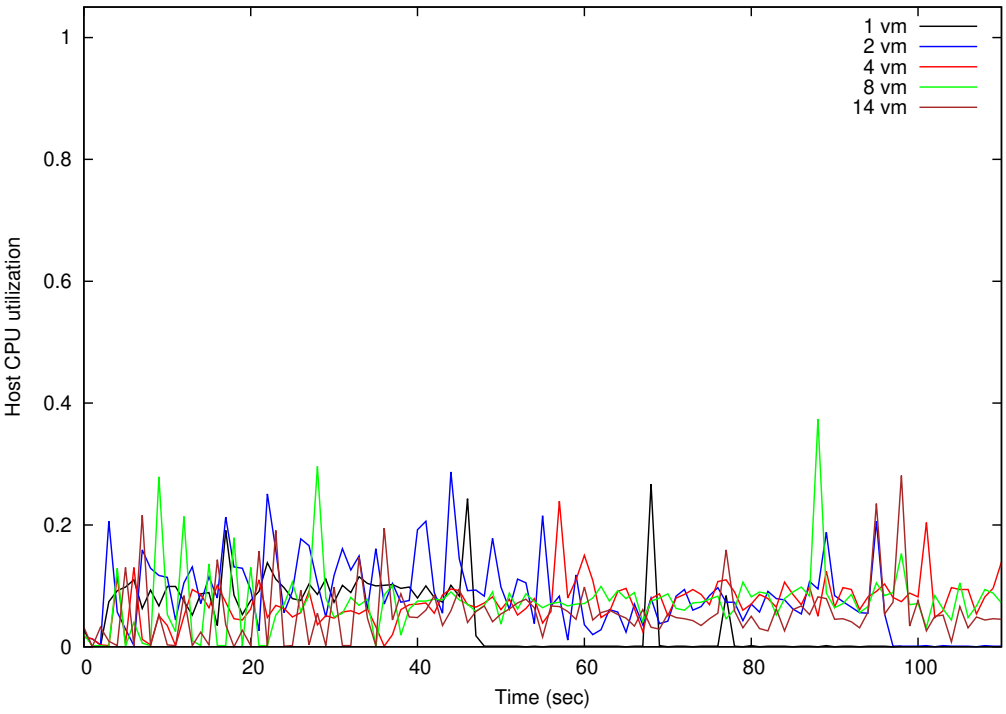


(b) Physical host data.

Figure 3.15: LMbench: CPU usages pattern changes due to various number of co-located VMs.



(a) VM data.



(b) Physical host data.

Figure 3.16: Sysbench: OLTP CPU usages pattern changes due to various number of co-located VMs.

3.8.2 Effect of VM consolidation on memory usages

Figures 3.17, 3.18, 3.19, and 3.20 show the changes in memory usages pattern of the Stream, Filebench, Iozone, and LMbench, respectively. Each figure has two graphs, the first graph shows the memory usage data of the VM, while the second graph shows the memory usage data of the host. Those usages data is obtained by running VMs in stages, where each stage has a different number of simultaneously running VMs on the server just as described in the above sections.

Figure 3.17 shows the memory utilization changes of the Stream due to different numbers of simultaneously running VMs. Recall that, it is a memory-intensive benchmark.

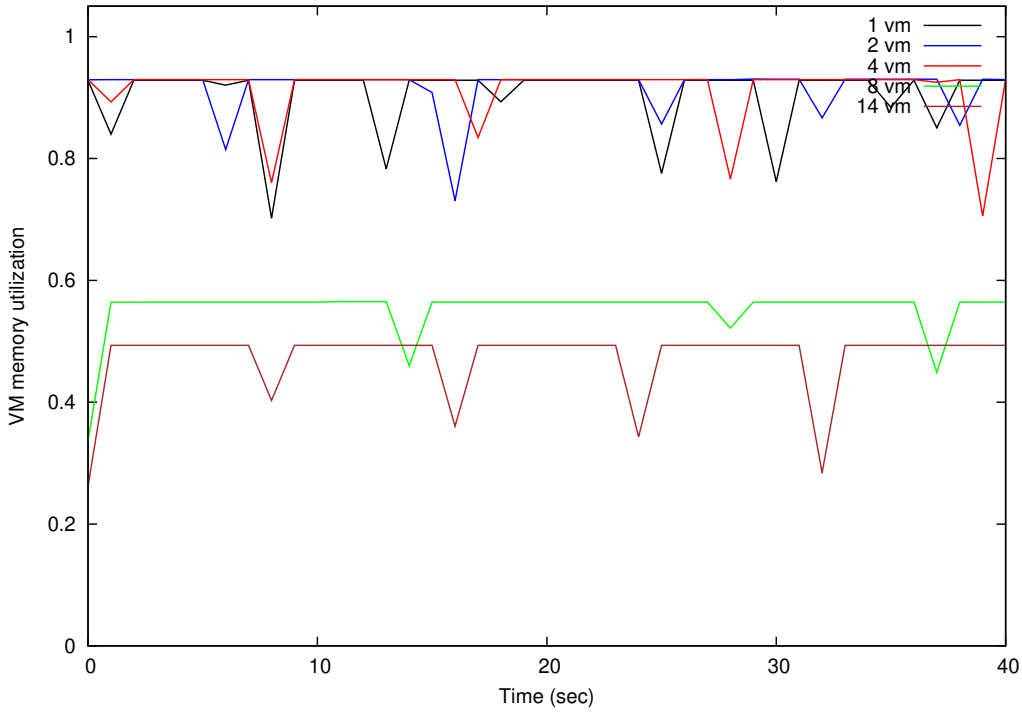
Figure 3.17a shows that for four co-located VMs the memory utilization is reasonably high. However, For fourteen simultaneously running VMs, the memory utilization is almost 50% of that of a single VM. Although, the system has 16 GB of memory and each VM has been allocated a 1 GB of memory in the physical host; the performance degradation still happens due to resource contention.

Various parts of the shared memory system like the memory address bus and space have to be multiplexed among the VMs. Therefore, an increasing number of co-located VM has adverse effects on the resource utilization.

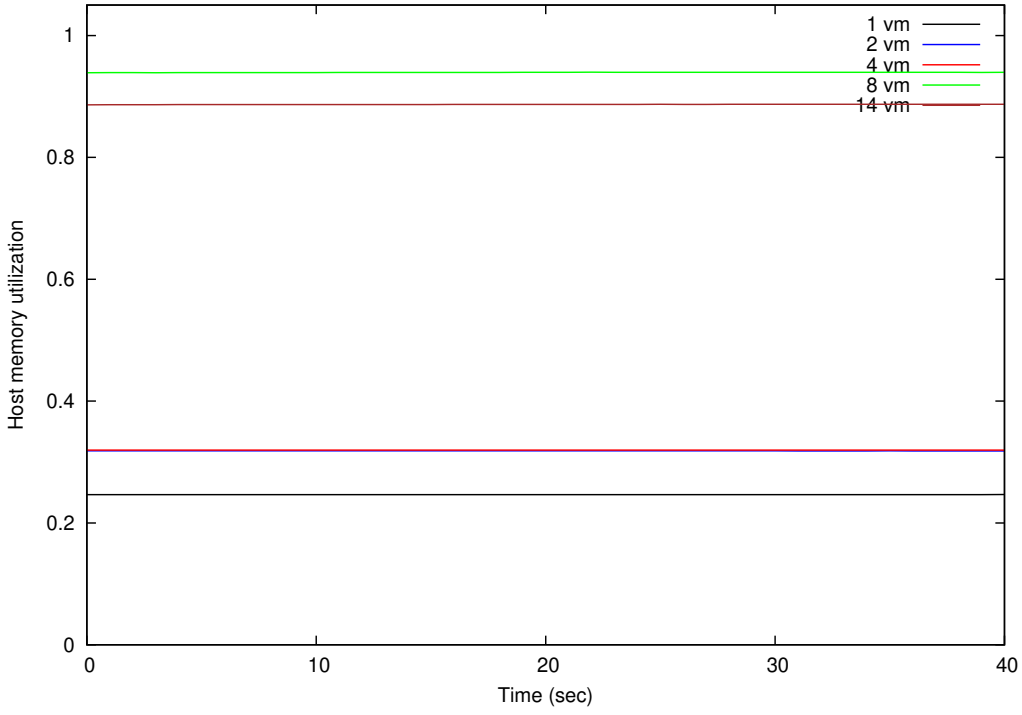
Next, Figure 3.18 shows the changes in memory usage of the Filebench due to VMs consolidation. Figure 3.18a shows that for one or two simultaneously running VMs the VM memory utilization reaches peak value quickly. Thus, indicates a low level of resource contention among the VMs. However, as the number of co-located VMs are increased the VM memory utilization takes longer to reach a high utilization level.

As mentioned before, although separate memory locations are allocated for the VMs still there is resource contention among them. Because of excessive resource contention among too many co-located VMs the memory utilization increase is slowed down compared to fewer VMs number. Thus, the too much resource contention can be detrimental to the system performance.

Figure 3.18b shows the memory usage pattern changes of the Filebench due to previous five sets of VMs. The physical host memory usages show almost no variance due to the different number of simultaneously running VMs. As it is explained above, the virtualization adds an extra level of indirection in memory hierarchy leaving the

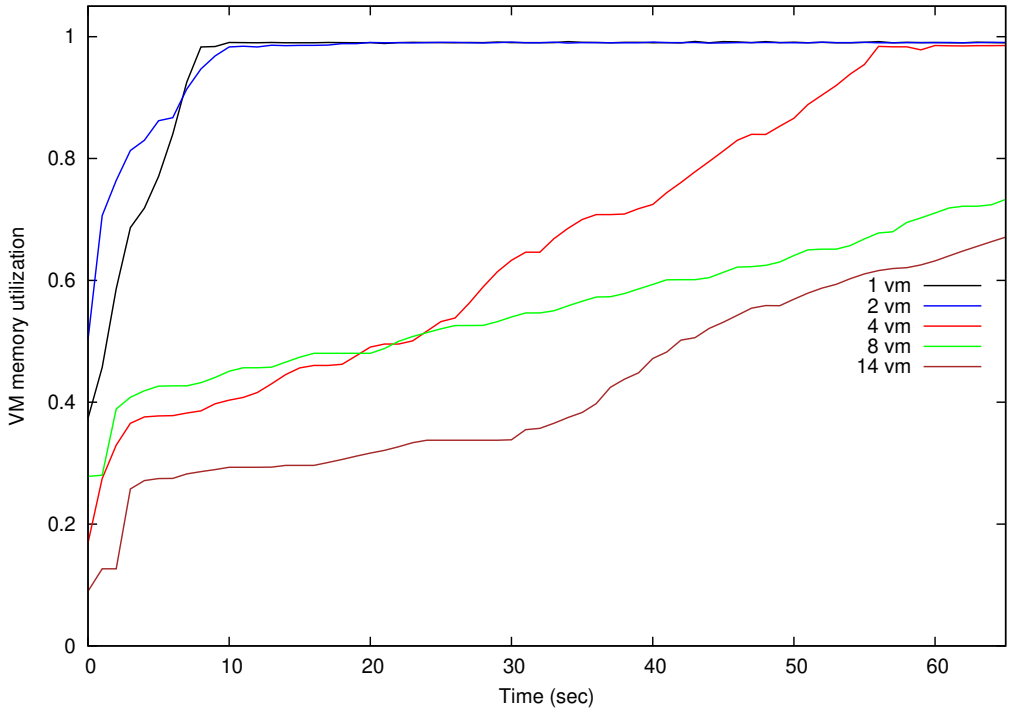


(a) VM data.

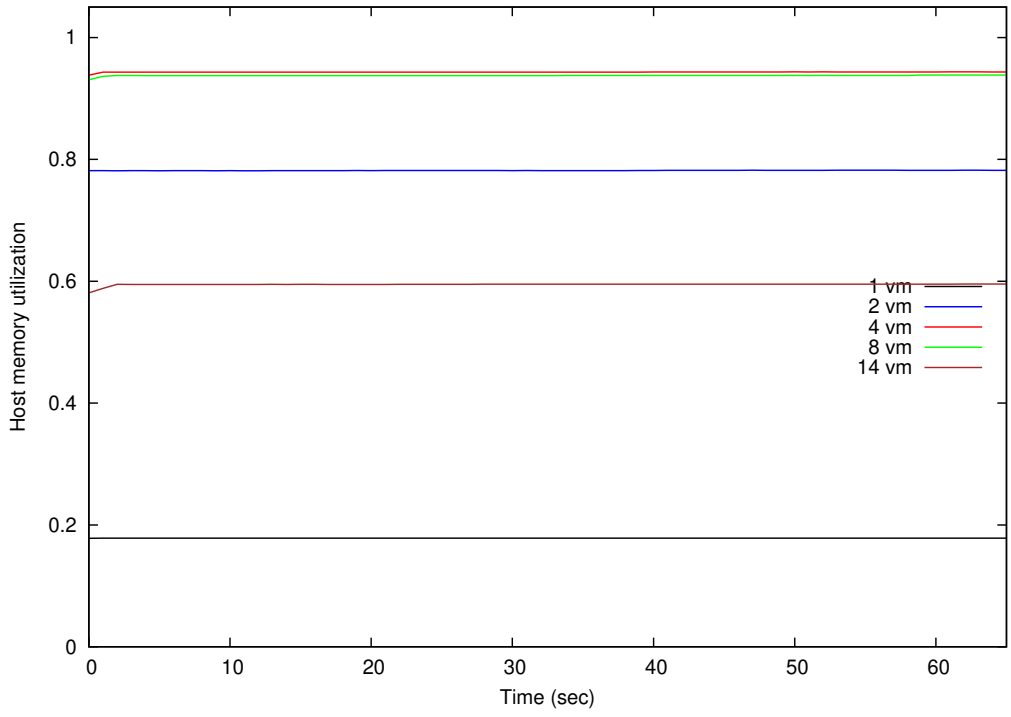


(b) Physical host data.

Figure 3.17: Stream: memory usages pattern changes due to different number of co-located VMs.

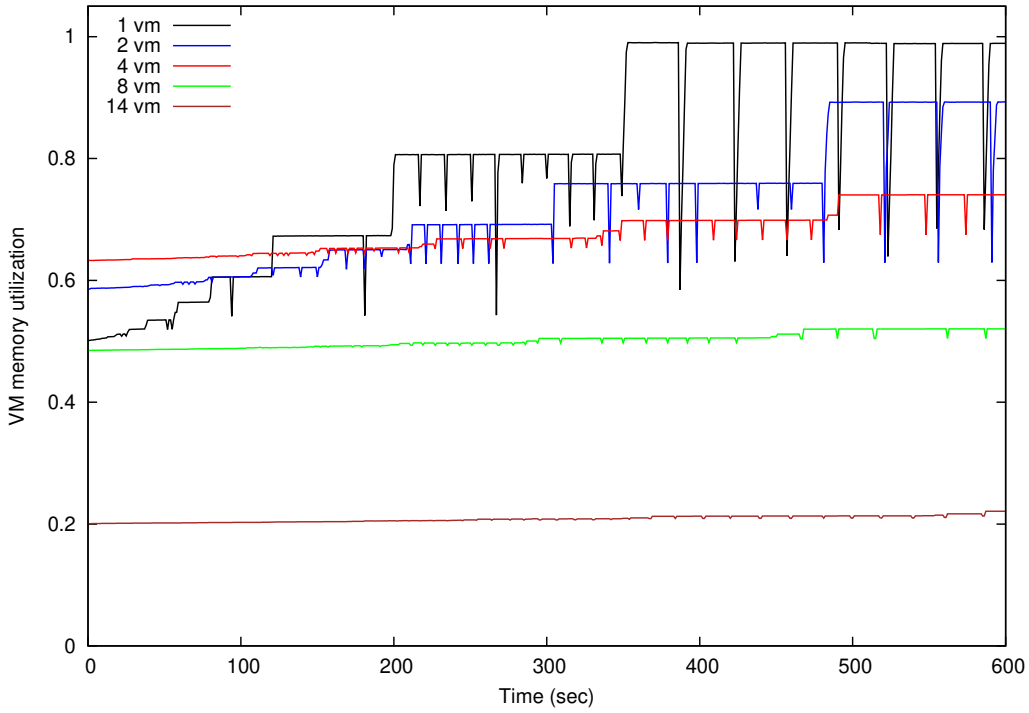


(a) VM data.

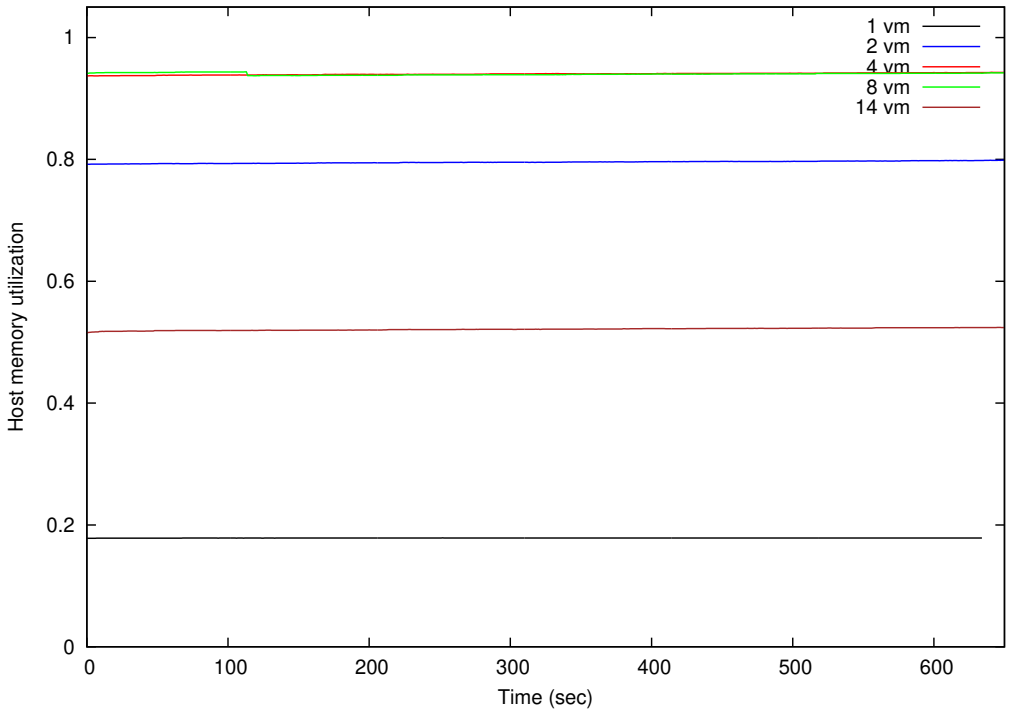


(b) Physical host data.

Figure 3.18: Filebench: memory usages pattern changes due to different number of co-located VMs.

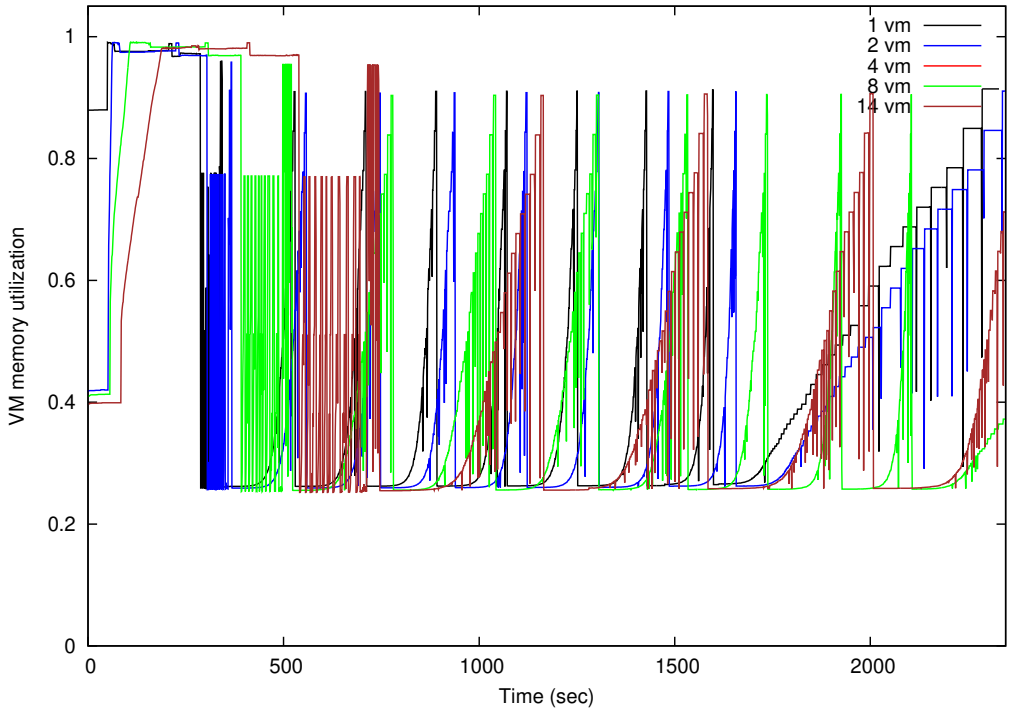


(a) VM data.

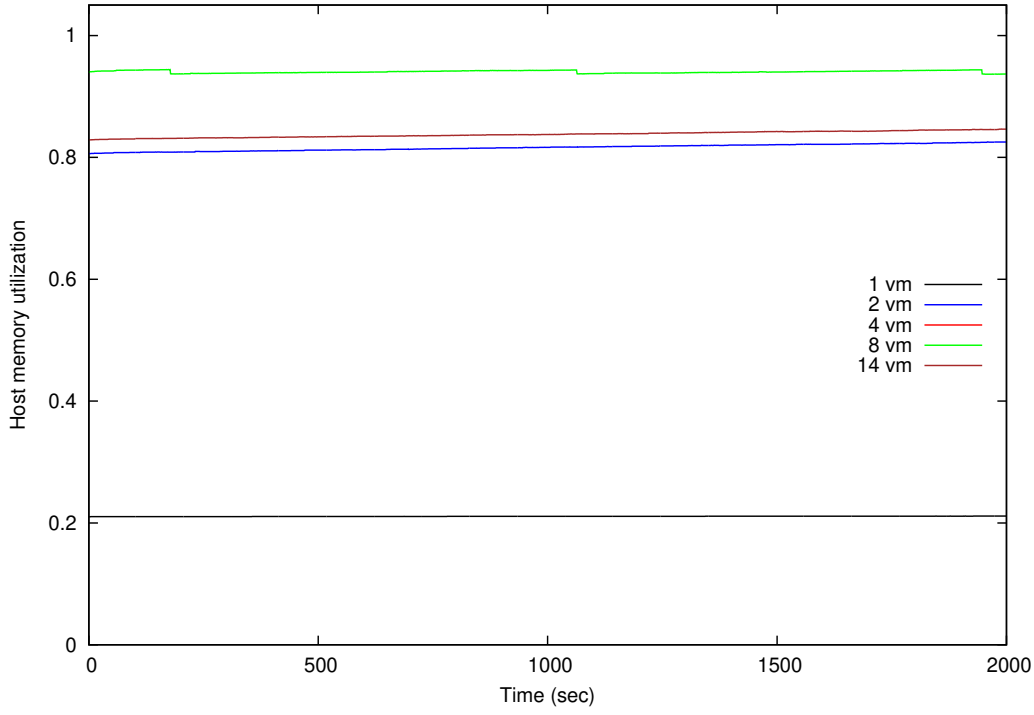


(b) Physical host data.

Figure 3.19: Iozone: memory usages pattern changes due to different number of co-located VMs.



(a) VM data.



(b) Physical host data.

Figure 3.20: LMbench: memory usages pattern changes due to different number of co-located VMs.

physical host almost in the dark about the real-time memory usage of a VM. Therefore, the physical memory utilization data of the host do not indicate any change in the utilization of VM memory.

Next, Figures 3.19 and 3.20 show the memory utilization changes of Iozone and LMbench for different sets of co-located VMs. Recall that, both are multi-resource intensive VMs.

Figure 3.19a shows the memory utilization of a VM running the Iozone benchmark. With only a single running VM the physical host the memory utilization gradually increases. On the other hand, with two or four running VMs the resource contention stops the utilization from reaching a high level. As the number of VMs increase in the system, the utilization pattern becomes rather flat.

Figure 3.20a shows that memory utilization pattern of the LMbench on a VM. The memory utilization pattern of LMbench is different from that of Iozone. For a single VM, the resource usage of the running benchmark consists of alternate levels of utilization; high and low. However, the resource utilization becomes more flat with the increase in the number of VMs just like in the previous cases.

Furthermore, as the number of VMs increase, the execution time tends to stretch longer. For example, when one VM is running on the host, between 250 and 300 seconds, a high level of memory utilization is observed. As the number of co-located VMs increase, it takes longer to reach a high level of utilization. For eight co-located VMs it takes more than double time to execute, and the memory usages pattern becomes spread between 375 to 500 seconds. It can be seen that for fourteen simultaneously running VMs it take even longer.

The experimental results show that the tasks are taking longer to execute as the number of simultaneously running VMs are increased in the server. This observation encouraged the author to do further experiments with VM consolidation and task execution time in the later chapters. Next, the system I/O utilization pattern changes are discussed.

3.8.3 Effect of VM consolidation on disk usages

Figures 3.21, 3.22, and 3.23 show how disk utilization of co-located VMs are affected for the Filebench, Iozone, and LMbench, respectively. The graphs of the figures show the I/O utilization of the benchmarks for five sets of co-located VMs.

Figures 3.21b, 3.22b and 3.23b show the I/O utilization of the physical host. The figures show that the physical host shows more I/O utilization compared to the figures discussed in the previous section. As already explained, that each I/O request of the VM is ultimately processed by the hypervisor; this results in high I/O utilization for the physical host.

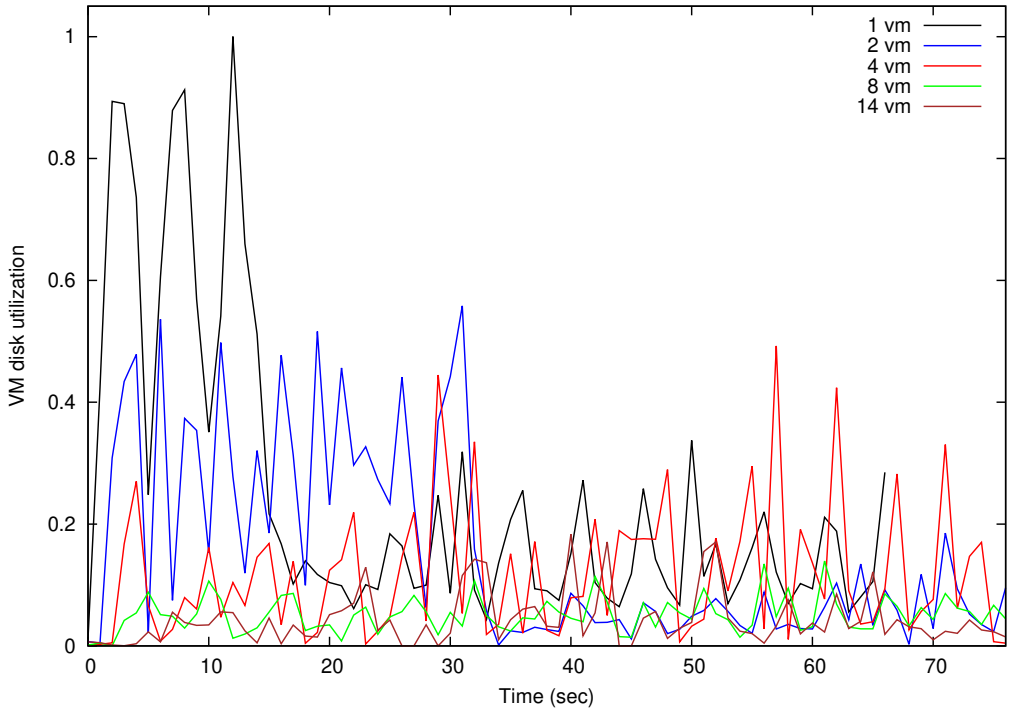
In each of the three figures, while only one VM is running on the host, the I/O usages pattern of the VM and physical host are identical. VMs can not process I/O requests by themselves. Whenever a VM needs to perform an I/O operation, it informs the hypervisor through interrupts; the hypervisor then performs the I/O on behalf of the VM. That is why in above three figures the I/O usage of the VM and host are almost identical for one running VM.

Figure 3.21a shows the I/O utilization pattern changes of the Filebench for five VM sets. For a single running VM, a high utilization can be seen between 0 to 15 seconds. On the other hand, for two co-located VMs the utilization is spread-out between 0 to 30 seconds; it is almost double the amount of time taken for a single VM. It can be seen that as the number of VMs increase in the system the I/O usage pattern spread-out more. Next, Figure 3.21b shows that the cumulative host I/O utilization increase with the increase in the number of VM. The host I/O utilization reaches the peak between four and eight number of co-located VM; however, shows a drastic reduction of overall utilization for fourteen simultaneously running VMs.

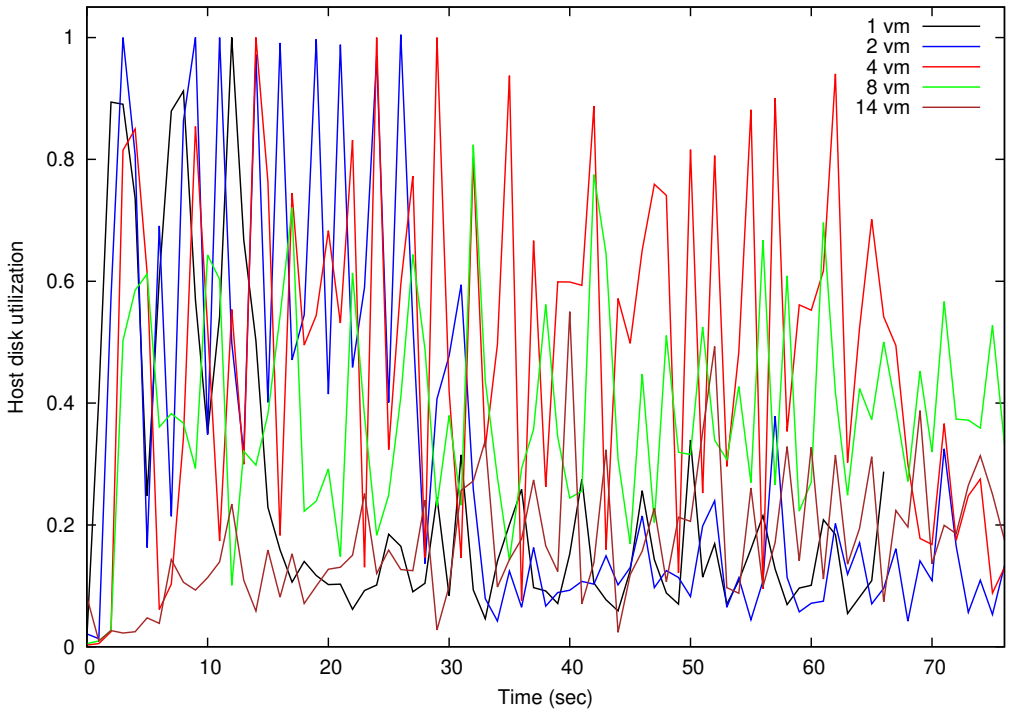
Next, Figure 3.22a shows the I/O utilization pattern changes of the Iozone. The Iozone shows a different pattern compared to the Filebench discussed above. For example, the host cumulative I/O utilization increase of the previous example is not present in the case of Iozone.

Figure 3.22b shows that Iozone running on a single VM is causing high I/O utilization for the VM. However, when the VM number is increased in the system, the result is not a cumulative higher utilization rather a massive loss of utilization. As the number of co-located VMs increase in the system the physical host I/O utilization gets worse. It shows that the resource contention has a more significant effect on I/O utilization compared to other resources.

The last example of this section shows the I/O utilization of the LMbench for different VM sets. Figure 3.23a shows that the LMbench has much lower I/O utilization compared to above two benchmarks. Figure 3.23b demonstrates that the cumulative physical host I/O utilization is getting higher with the increase in the number of VM.

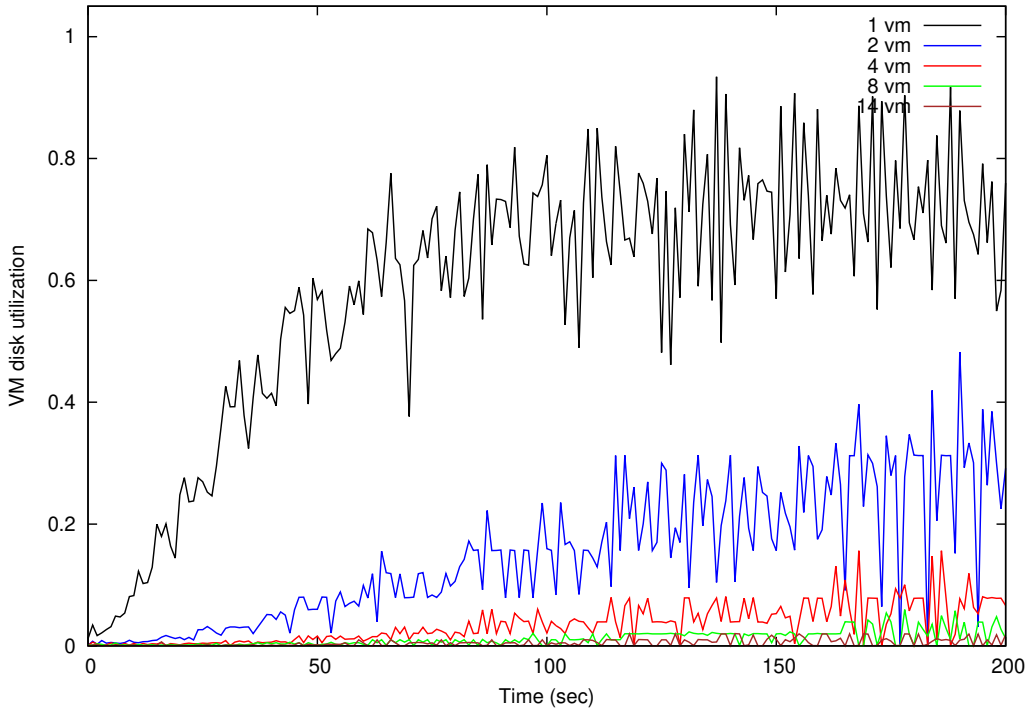


(a) VM data.

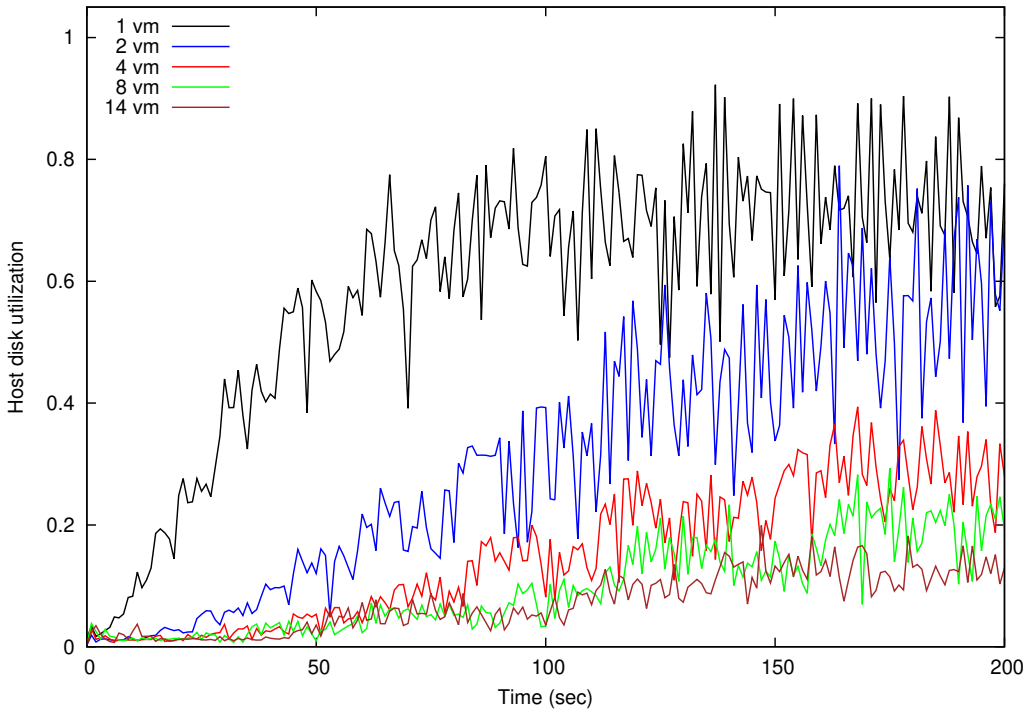


(b) Physical host data.

Figure 3.21: Filebench: disk I/O usages pattern changes due to different number of co-located VMs.

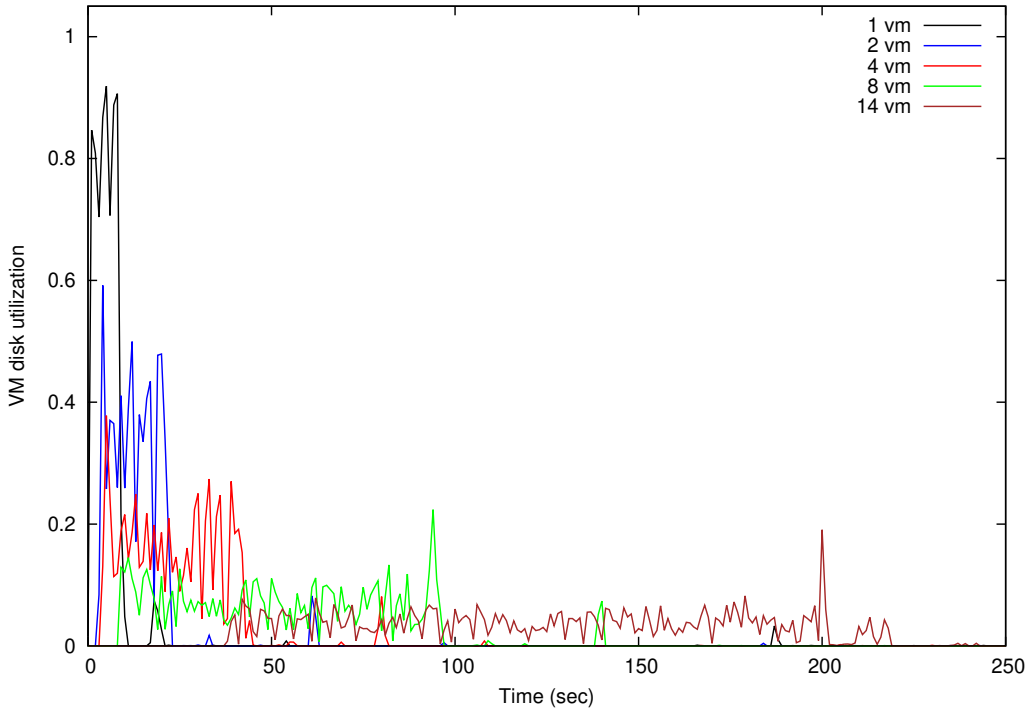


(a) VM data.

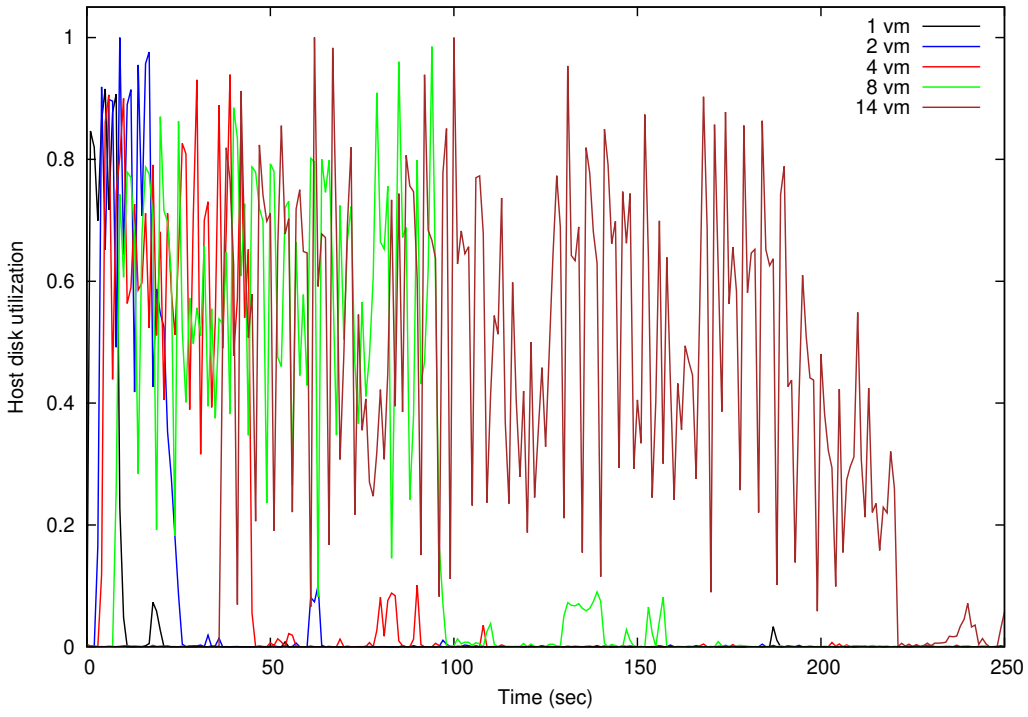


(b) Physical host data.

Figure 3.22: Iozone: disk I/O usages pattern changes due to different number of co-located VMs.



(a) VM data.



(b) Physical host data.

Figure 3.23: LMbench: disk I/O usages pattern changes due to different number of co-located VMs.

Therefore, consolidating the VMs with moderate or low I/O utilization can increase the overall host I/O utilization.

Recall that, Figure 3.22a showed that each instance of Iozone has higher I/O utilization. When multiple instances of Iozone are simultaneously run, it results in degradation of I/O performance. On the other hand, Figure 3.23a shows that a single LMbench instance has much lower I/O utilization. When benchmarks with little I/O utilization are consolidated, the cumulative host I/O utilization increases.

3.9 Comparing the arithmetic mean of resource usages of the benchmarks

Figures 3.24, 3.25 and 3.26 the arithmetic mean of the resource usages of all benchmarks discussed in above sections. In the graphs of the above sections, the X-axis shows the time in seconds, while the Y-axis shows the resource utilization. From those graphs, the arithmetic mean of the utilization for each benchmark is calculated and shown in the three figures of this section.

The arithmetic means are grouped according to the resource utilization types. For example, Figure 3.24 shows the arithmetic mean of CPU utilization of the nine benchmarks. As mentioned above, the VMs are grouped in five sets and each set contains a different number of VMs. All the VMs of a set are run simultaneously, and resource utilization data are collected. Then, the arithmetic mean of resource usage for each set of VMs is calculated separately.

Recall that, Figure 3.13a showed five CPU usages pattern of the Cachebench for five sets of simultaneously running VMs. The sets contain one, two, four, eight, and fourteen VMs, respectively. For each set, two hundred seconds of execution time is shown. Along the X-axis, data is plotted with the one-second interval. The Y-axis shows the CPU utilization for each second. From the data of Figure 3.13a five arithmetic means are calculated for five sets of VMs.

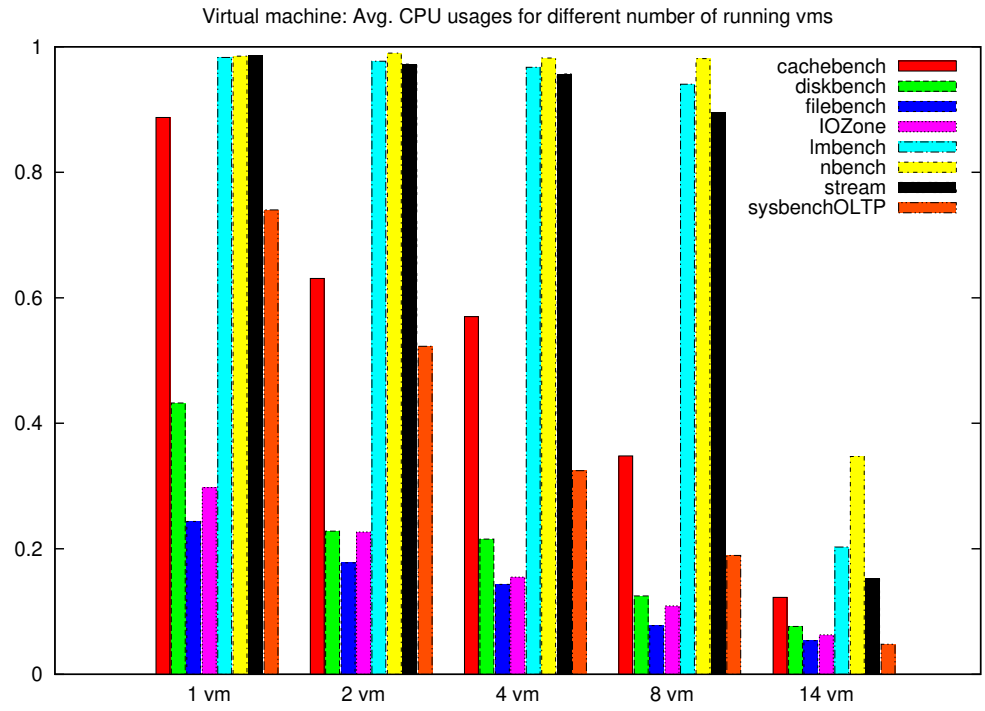
Now Figure 3.24a shows the five arithmetic means for the Cachebench. The X-axis shows how many VMs were running on the host, while the Y-axis shows the arithmetic mean of the CPU utilization. In this way, arithmetic means of CPU utilization of eight benchmarks are calculated. The arithmetic means of all eight benchmarks are shown in different colors in Figure 3.24a.

Next, Figure 3.24b shows the arithmetic mean of the host CPU utilization of the same benchmarks. They are calculated in the same way; however, host CPU utilization data is used instead of VM data. In this way, the arithmetic mean of memory, and I/O utilization of all the eight benchmarks are calculated, and they are shown in Figures 3.25, and 3.26, respectively. In each figure, the arithmetic mean of resource utilization for each benchmark is shown with a different colored bar.

3.9.1 The arithmetic mean of CPU usage data

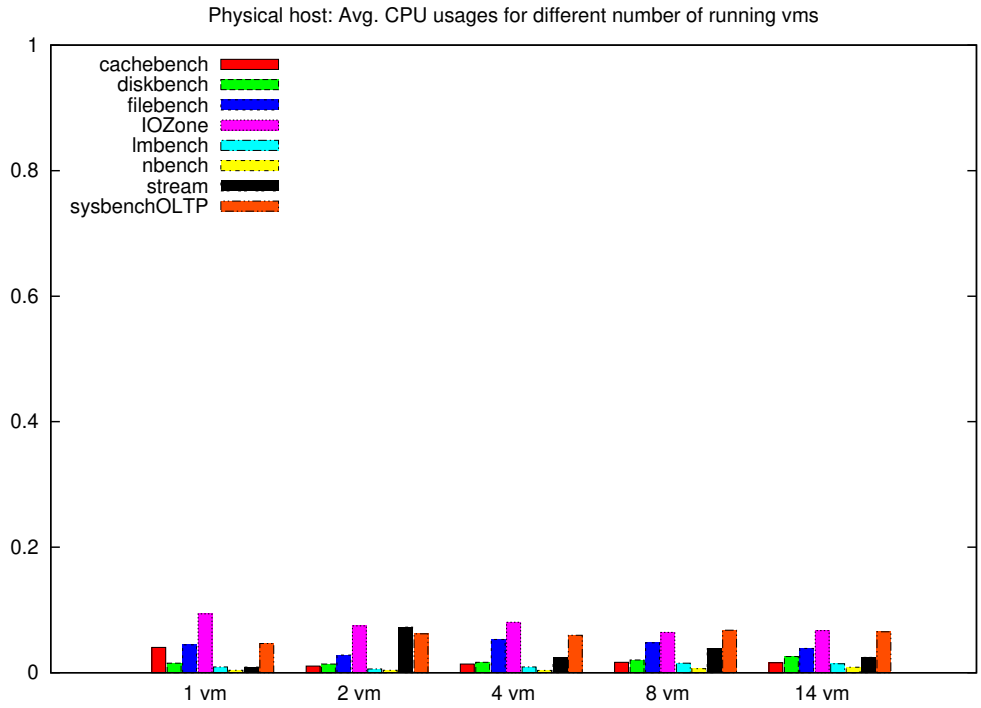
Figure 3.24a shows that the LMbench, Nbench, and Stream have a high arithmetic mean of CPU utilization for single running VM. Their arithmetic means of CPU utilization remain high until eight simultaneously running co-located VMs. However, the arithmetic mean is much lower for fourteen VMs, indicating a high level of resource contention among VMs.

The Nbench is a CPU intensive benchmark and has the highest arithmetic mean of CPU utilization among all the benchmarks. On the other hand, other benchmarks



(a) VM data.

Figure 3.24: Arithmetic mean of CPU usages for various number of co-located VM.



(b) Physical host data.

Figure 3.24: Arithmetic mean of CPU usages for various number of co-located VM (Continued).

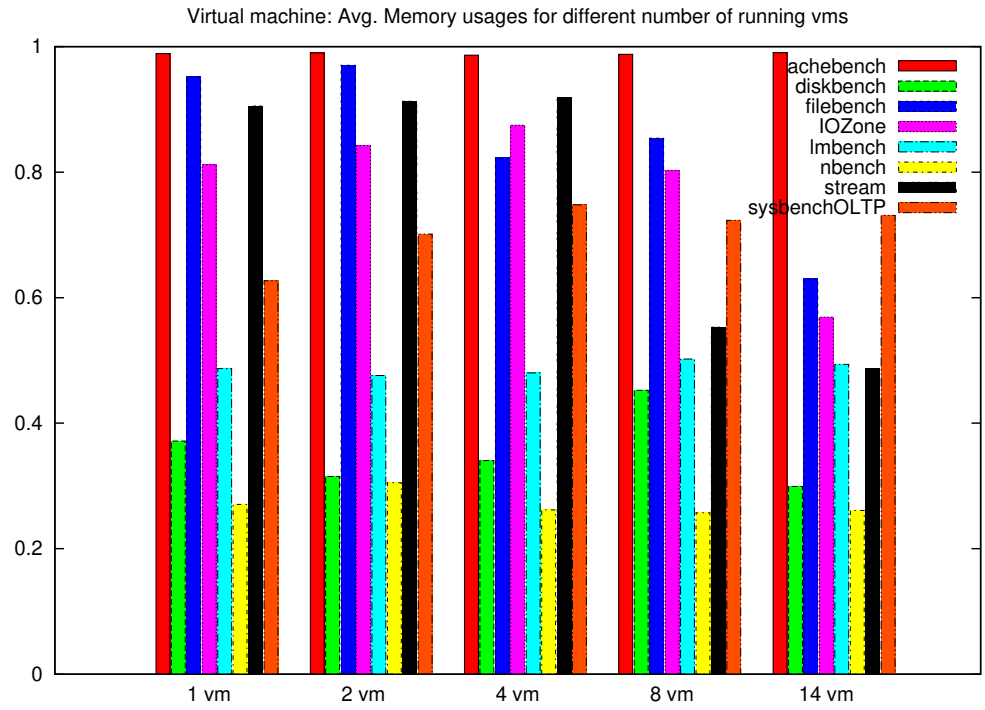
have a decreasing value of the arithmetic mean of CPU utilization with the increasing number of co-located VM.

Next, Figure 3.24b shows the arithmetic mean of CPU utilization of the host for eight benchmarks. The physical host has a low arithmetic mean of CPU utilization in all cases. The host CPU utilization is mainly due to context switching, and system interrupts.

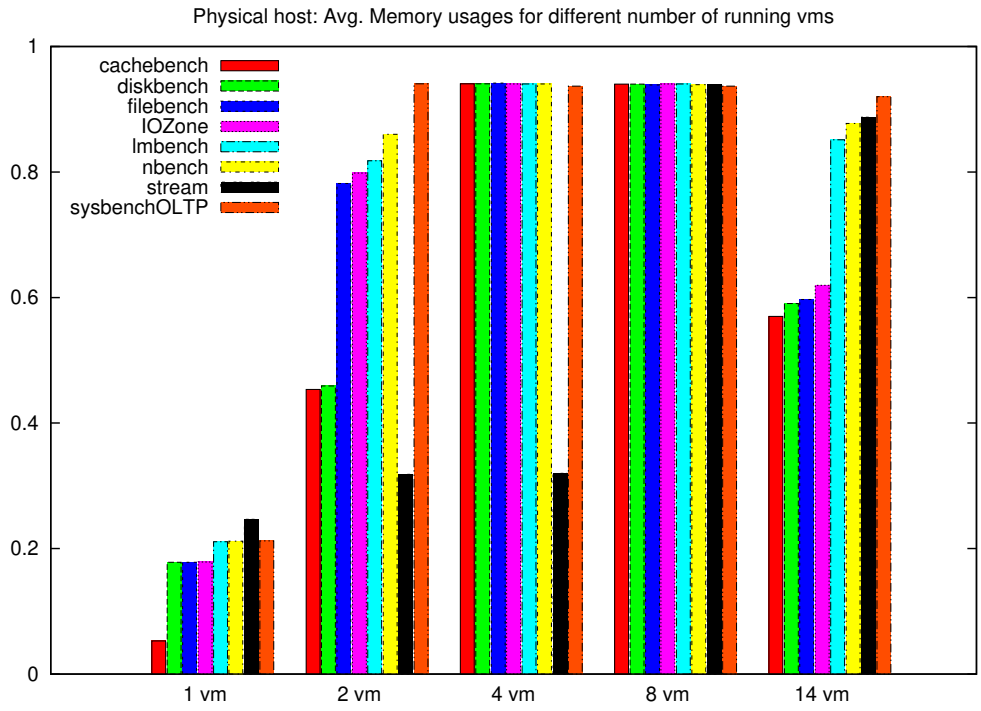
3.9.2 The arithmetic mean of memory usage data

Next, Figure 3.25a shows the arithmetic mean of memory utilization by eight benchmarks on VMs. The arithmetic mean of memory utilization of VMs does not change much with the change of the number of co-located VMs.

Next, Figure 3.25b shows that the physical host memory utilization does not change with increasing VM number either.



(a) VM data.



(b) Physical host data.

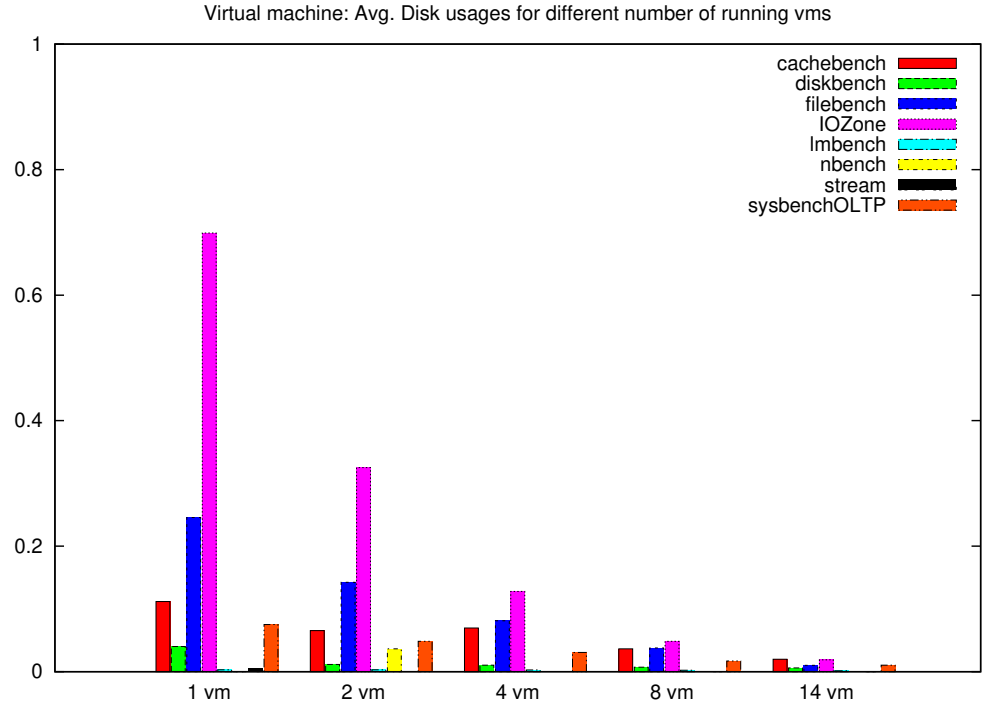
Figure 3.25: Arithmetic mean of memory usages for various number of co-located VM.

3.9.3 The arithmetic mean of I/O usage data

Figure 3.26 shows the arithmetic mean of I/O utilization for both VM and physical host. Here, the Iozone which has about 75% I/O utilization for a single running VM. However, it shows severe degradation of I/O utilization with the increase of co-located VM number. Both for the VM and physical host the arithmetic mean of I/O utilization decreases with the increase of co-located VM number. For eight or more co-located VMs the arithmetic mean of I/O utilization of VM falls under 10%.

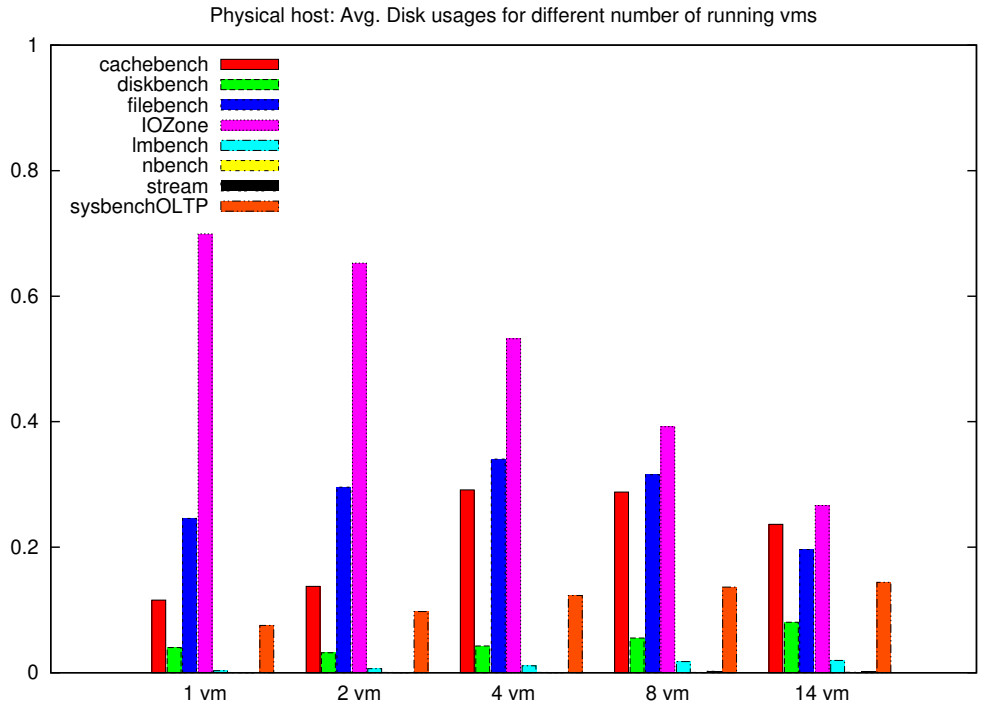
On the other hand, both the Cachebench and Filebench have a relatively higher arithmetic mean of physical host utilization for increasing number of co-located VMs. Initially, with one running VMs, both benchmarks have a moderate level of I/O utilization. The arithmetic mean of I/O utilization of both benchmarks are less than 30% when running on a single VM.

Next, Figure 3.26b shows that with increasing number of VMs the arithmetic mean of physical host I/O utilization remains under 40%. These results illustrate that the arithmetic-mean of I/O utilization is low for an increasing number of consolidated



(a) VM data.

Figure 3.26: Arithmetic mean of disk usages for various number of co-located VM.



(b) Physical host data.

Figure 3.26: Arithmetic mean of disk usages for various number of co-located VM (Continued).

VMs.

3.10 Conclusion

The performance interference of co-located VMs is an essential issue for the VM consolidation and migration. The resource usage efficiency is another important factor for the data centers. In this chapter, resource usages changes of VM due to consolidation is investigated. Several sets of benchmarks are run on the VMs, and their resource usage traces are collected. Results show that each benchmark has unique resource usage signature and behave differently to the VM consolidation.

From the experimental results above three observations are made:

a) Consolidated VMs interfere with the performance of each other, and the nature of the interference depends on the number of simultaneously running VMs. It can be seen that the resource usage pattern of the VMs are changing as the number of VMs are increasing in the system;

b) The resource intensities of the VMs play an essential part in the performance interference. The performance degradations of the same VM for different resources, like CPU, memory, or I/O are different. In other words, the reaction of each resource towards the consolidation and resource contention is different.

c) Under consolidation tasks take longer to finish execution on VMs, the resource contention is mainly responsible for this delay. The experimental results of above sections show that there is a relationship between the consolidated VMs and the task execution finish time; this encouraged the author to do more experiments with task execution time variation in the later chapters.

Observations and findings of this chapter are used in the next chapter to set up new experiments with consolidation. Especially the effect of consolidation on the task execution time variation of VMs is critically examined.

Experimental setup and results of this section are used as guidelines to perform large-scale experiments in successive chapters. Data collection technique is changed, and various statistical analysis is presented.

In the next chapter, a methodology is presented to profile the execution time variation of consolidated VMs. Experimental results of this chapter helped to design the new methodology and set up experimental stages.

In this chapter, Linux shell scripts are used to run benchmarks and control VMs. Advantages and limitations of using scripts to run experiments were apparent to the author. As a result, the need for designation a new framework became clear. The experience of data and VM controlling process during the experiment of this section had helped to design a new framework for conducting experiments with VMs. The new framework is presented and discussed in details in Chapter 5.

Thus, the experiments conducted in this chapter paved the way to conduct more experiments with VM consolidation and performance in successive chapters of the dissertation.

Chapter 4

Incremental Consolidation Benchmarking Method (ICBM)

"It is strange that only extraordinary men make the discoveries, which later appear so easy and simple."

— Georg Christoph Lichtenberg* (1742–1799)

4.1 Introduction **

The previous chapter (Chapter 2) describes the importance of virtualization for both the Cloud and data centers. Virtualization enables data centers to provide Cloud services cost-effectively and reliably. VM consolidation helps to reduce operating cost and increase resource utilization. On the other hand, consolidation has effects on the VM performance [187, 238–241].

VM *consolidation* is done to increase the resource utilization of the virtualized servers. However, it imposes a performance penalty, which manifests itself through the execution time variation of co-located VMs [242–244]. This performance variation occurs because of resource contention among the VMs. It is an obstacle to efficiently scheduling parallel applications on virtualized systems; the reasons are stated below:

*Image source: `https://www.babelio.com/auteur/Georg-Christoph-Lichtenberg/118291`

**This chapter contains materials that were published as [43]. All the contents of this chapter are entirely original work of the author. The author has designed and conducted the experiments. The author also collected the data and analyzed the results to prepare the draft for the publication.

(a) The variation depends on the server load and resource contention among the VMs. The same task may take a different amount of time to be completed on different VMs decreasing the efficiency of the virtualized server;

(b) To schedule a task of any parallel application determining the execution finish times of all parents is necessary. It becomes difficult due to the execution time variation. Thus, it is an important issue to address.

Most of the previous works in this area fall into two broad categories. The first one, is to explore the cost-performance models of parallel applications on Clouds [158, 245–256]. The other one, is virtualized server consolidation benchmarking [216, 257–262].

One can quickly identify few shortcomings of those works:

(i) They do not explore the resource contention and performance variation of co-located VMs explicitly;

(ii) Experiments have been done, mainly with parallel applications [263]. Those have a complex internal structure of their own represented by a task graph. Virtualization technique involves many layers of abstraction and hardware indirection, and it is not easy to build a performance model. Furthermore, complex internal structures of a parallel application can make it difficult to capture the relationship among the co-located VMs accurately;

(iii) They do not provide the option to control usages of different computing resources either individually or granularly during experiments. During experiments, such abilities are highly desirable;

(iv) Consolidation benchmarks are designed to provide an overall average point of some combination of tests not details about different levels of consolidation;

(v) The consolidation benchmarks are hugely dependent on vendors, and their applicability for comparing different virtualization technologies are not well defined. For example, the *VMmark* benchmark is designed for VMware ESX servers;

(vi) Most works use sophisticated mathematical optimization tools, which have high overhead. The performance modeling greatly depends on system configuration,

and changes to system configuration may require model retraining, which in turn becomes a hugely time-consuming process due to high overhead;

(vii) Many works deal with a theoretically derived model of the Cloud and simulation is used for verification. There is no well-accepted method to reflect the effect of performance variation due to virtualization in simulation; hence, often simulation does not portray the system behavior faithfully.

Chapter 3 discusses the importance of ensuring the quality of service for Cloud providers. Moreover, the energy consumption is becoming a significant issue for the data centers. It is essential to analyze the performance of consolidated VMs to optimize the server resource usage. Performance analysis can help the system administrators to decide how many VMs to be consolidated on a server with an acceptable amount of performance loss [187, 188, 264, 265].

This chapter introduces task *Execution Time Variation* (ETV) as a performance metric for consolidated VMs [266]. A methodology for profiling the task execution time is also introduced, called the *Incremental Consolidation Benchmarking Method* (ICBM). Key design features of the ICBM are discussed next:

1) Combinations of various syntactic benchmarks suites are used in the experiments. Manipulation of different types of syntactic benchmarks gives the ability to manipulate basic resources of the server. Careful manipulation of benchmarks on the VMs makes it possible to control resources like CPU, memory and I/O both individually and collectively. This resource controlling makes it possible to analyze the effect of consolidation on each resource type more discretely than the previous works;

2) The ICBM is a method to profile task ETV for a set of VMs for different types of resource contention. It is not dependent on virtualization technology or system configuration. It is a methodology that can be applied to any system, making it suitable to compare a wide range of systems;

3) The task ETV prediction models for combinations of resources have been built from profiled VM data. Separately collected task ETV data due to primary resources, like CPU, memory and I/O, have been used to predict task ETV for the combination of resources, like CPU-Memory, CPU-I/O, and Memory-I/O.

The prediction results have a reasonable level of accuracy demonstrating that profiling for every combination of resource load is not necessary. It can save a significant amount of time while profiling consolidated VMs;

4) All experiments have been done on actual virtualized servers. All results presented here are real system data. The results show that the ICBM can predict the task ETV of real virtualized systems quite accurately;

The ICBM is a methodology to profile and predict performances of various types and numbers of consolidated VMs. In this chapter high lights various issues involved with experimenting with consolidated VMs and proposes a methodology to overcome them. Experimental results show the methodology can be successfully applied to real systems;

5) The ICBM uses task execution time as a metric. During experiments, task execution time variation of the VMs are recorded. However, from the collected execution time performance variation is calculated in percentage. It is possible to calculate the performance variations of other system using the ICBM, and that can make it possible to compare the impact of consolidation on different systems.

Furthermore, prediction models are trained with the help of performance variation curve data. The performance variation graph, which represents performance improvement or degradation of the consolidated VMs. The ICBM is not reliant on absolute system value rather it depends on relative performance gain or loss of consolidated VMs. Those percentage data from one system can be compared with system easily.

Furthermore, in subsequent chapters, the performance profiling and prediction techniques are further refined. Those improvements make the ICBM more suitable to be applied to various types of systems.

6) Prediction models have been built using the *Least Square Regression* (LSR), which has low overhead. Use of LSR makes the training and prediction process much faster. Often, changes in system configuration may require retraining of models, in such cases a low overhead process can save much time;

7) Analysis of profiled data reveals some interesting patterns. For example, it shows

that certain types of resource combinations can cause the task execution time of consolidated VMs to degrade more rapidly than the others. This rapid execution time degradation indicates that resource contention is one of the main factors, behind the average utilization of virtualized servers being so low.

The syntactic benchmarks suites play an essential part in the design of the ICBM. They are essential tools for server performance analysis, and many studies have been done on their design. These benchmarks are the result of long-time analysis of commercially successful applications. They are inspired by an interesting phenomenon that, the applications spend both 80% of their time and resources, executing just 20% of the code [234].The syntactic benchmark suites are carefully crafted to mimic such essential parts rather than running the entire application. Each benchmark suite consists of several individual applications. These applications are grouped in a well-thought-out pattern.

The benchmarks suites are used here to get a finer control on the individual server resource types. Without using these benchmark suites, such controlling of resources is not possible. Experimental results show that the benchmark suites can cause a significant amount of resource contention and ETV on VMs. Thus, the benchmark suites can be a set of powerful tools for studying the performance variation of virtualized servers. The experiments that are conducted from several angles and they provide some interesting results.

Rest of the chapter is structured as follows. Section 4.2 discusses the importance of VMs in data centers, resource utilization, and energy efficiency. Section 4.3 introduces the task execution time as a performance metric. The section demonstrates the rationale behind using the task execution time as a metric. Section 4.4 discusses the ICBM in detail. In this chapter, the consolidated VMs are divided into two categories for performance measuring. Section 4.5 shows the task ETV results for the first category, the target VM. Section 4.6 shows the variations for the second category, the co-located VMs. Section 4.7 shows the prediction model was trained and the prediction results. It shows the parameters and coefficients of the trained LSR model. Finally, Section 4.10 concludes the chapter.

4.2 Data centers and VMs

The primary technology enabling the Cloud and modern data centers is the virtualization. The Cloud and data centers can be thought of ecosystems, where VMs go through the all stage of life cycle. VMs are created, operated, maintained, migrated, and eventually destroyed. All the activities of the Cloud revolve around the virtualization technology.

The Cloud users rent VMs to deploy their applications. As the Cloud is being used for new fields of application, like big data analysis; the demand for the Cloud is ever growing. The increasing demands for the Cloud services are naturally making the data centers bigger and bigger. The data centers are warehouses full of hundreds of thousands of virtualized server to provide all necessary services.

When scores of servers are involved, the efficiency and energy consumption became a significant concern. Improvement of server consolidation efficiency can help to save energy usage [244].

4.2.1 Consolidation and the energy cost

In data centers several VMs are simultaneously run on a server to increase the resource utilization and reduce operational cost; the process is known as the VM consolidation [244]. Since the consolidated VMs share the server resources, they create performance interference for each other. Thus, running too many VMs on a server can severely degrade their performances [193].

On the other hand, running too few VMs reduces the resource utilization. Studies have shown that the data center resource utilization is still remarkably low [267]. As the data centers are getting bigger, the energy efficiency is becoming a more significant issue. In 2005, the worldwide combined data center power demand was equivalent to that of the output of seventeen 1000 MW power plants. What is more, the average data center electricity use grew at a rate of 16.7% in 2005 [268]. Improving the resource utilization of servers can help to reduce the data center energy costs.

4.2.2 Performance variation of consolidated VMs

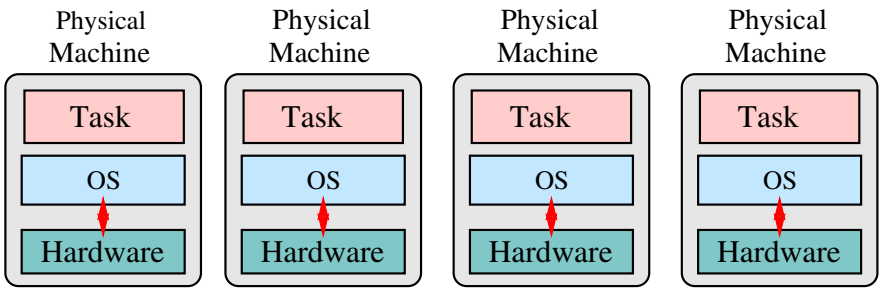
Understanding the causes of the VM performance variations on consolidated servers is essential. The VMs behave differently than physical machines. For example, the

system events handling is more expensive for VMs compared to physical machines. A typical physical machine has one *Operating System* (OS); thus, all interrupts are handled by the OS. On the other hand, in a virtualized server all VM interrupts must go through the hypervisor. Presence of hypervisor, makes the system interrupt even more complicated and expensive to handle.

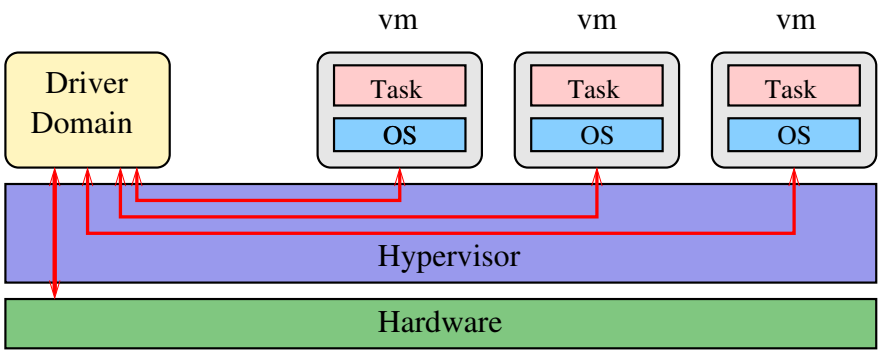
Figure 4.1a depicts a collection of physical machines each with own hardware, OS, and task. Each gray block is a separate physical machine, each of which has separate physical resources and one OS. The OS directly communicates with the hardware, whenever the running task requires to access physical resources. The task running on the physical machine uses system interrupts to draw the attention of the OS. The interrupts are costly events regarding both the time and complexity.

In the case of VMs, the situation is different. Usually, some VMs reside on a physical machine and share resources as shown in Figure 4.1b. Here, a hypervisor runs on top of the hardware and VMs are placed on top of the hypervisor. The VMs do not have access to the underlying hardware because of security reasons [38].

To access the hardware, the VM must first interrupt the hypervisor. To allow this



(a) Four separate physical machines.



(b) Four VMs sharing a physical server.

Figure 4.1: Two clusters of physical servers and virtual machines.

to happen either the guest OS (inside the VM) has to be modified or such support has to be incorporated in the hardware. Once the hypervisor receives the interrupt, then it proceeds forward with communicating with the hardware. The data from the hardware is also relayed back to the original VM in the same manner.

The hypervisors use one special VM, which is often called the driver domain to communicate with the hardware. The special VM contains all the device drivers to communicate with the hardware. Since the other VMs cannot communicate with the hardware directly, the driver domain is required to do the communication on their behalf. The driver domain is kept separate from the hypervisor to keep the hypervisor lightweight and secured. Otherwise, a bug in a device driver may compromise the entire hypervisor.

All the VMs running on the physical machine has to access and share hardware through a complicated path that follows through the hypervisor and driver domain. Thus, the cost of system interrupt and accessing physical resources are considerably higher for the VMs.

The consolidated VMs of Figure 4.1b interfere with the performance of each other, what types of tasks are running on the determines the nature and magnitude of the interference. On the other hand, the physical machines of Figure 4.1a are independent; the tasks running on them do not interfere with each other. Thus, in the performance point of view VMs are different from physical machines. Experiments with the resource contention of the consolidated VMs can help to improve the performance of the VMs involved.

4.3 Task execution time variation (ETV) as a performance metric

There has been much work done with the performance metric. System counters are predominantly used for system performance evaluation [260, 269, 270]. At the beginning of the dissertation, at Chapter 3 experiments have been done with system parameters also. Those experiments gave some insight into how the system parameter values change with the number of consolidated virtual machines. It was observed that as the number of VMs increase in the system the task completion time also start to vary. In this Chapter further excrements have been done with task execution finish times.

This chapter uses the task execution time variation (ETV) as a performance metric for the consolidated VMs. Previously, Section 4.2.2 discussed how the consolidated VMs interfere with the performance of each other. This section will explain how the execution times of the tasks on consolidated VMs changes depending on how many and what types of tasks are running on the server. It will also be shown that the task ETV can be used as a useful performance metric.

In the next few sections, the motivations for using the task ETV as a metric for the consolidated VMs is explained. The relationship between the consolidated VMs and ETV is demonstrated with a set of experiments in the next section. A host can have many VMs, and they can be arranged in many different ways, and it is necessary to examine how the VM arrangements affect the performance. It is important to remember that a fixed number of VMs can be arranged in a large number of ways. Now the question is if only the location of a VM changes in the server then would it affect the overall server performance or not? In the next section, experiments are designed to examine those situations.

4.3.1 Combination of VMs

Several VMs may be consolidated on a server at a time. The objective of this section is to investigate whether the location of the VMs on the server influences the performance or not. The problem is explained in an example next.

Assume one server has the physical resources to run 16 VMs and currently only 5 VMs are running. The 5 VMs may be placed in any location on the server. Placement of the VMs can change for several reasons. It is possible that some of the VMs have finished their tasks leaving some vacant places in-between the running VMs. Another scenario would be a VM is added to the server and it is placed in a random location. In this way a server may end up with the same number of VMs; however, they are located in different places. In either case, the server has the same number of VMs; however, the combination is different.

If all combinations of the five VMS are to be considered, then there would be 4368 combinations in total. Because there are C_5^{16} or 4368 ways to select five locations for VMs from sixteen possible available locations on the server. Now the question is, do all of those combinations have the same effect on the server performance or not? Profiling VMs with a series of micro-benchmarks is already a time-consuming

process [271]. Furthermore, profiling for all of the combinations of VMs would require even more time.

The experiments in this section aim to show that different combination of the same number of VMs has the same overall effect on the consolidation. In this section, experiments are conducted with different combinations of VMs, and they show almost similar results. Since all of the VM combinations show almost similar effect; profiling the performance of all combinations of VM is not necessary. Thus, much VM profiling time can be saved. Next, the experimental setup for this section is described.

A consolidated server can have many different combinations of co-located VMs, the number of consolidated VMs can change at any time. Figure 4.2 depicts a consolidated hypervisor with ten different VM sets. Each row independently represents a set of VM combination at a particular time. Thus, ten rows represent ten different combinations or sets.

In each set, on one VM the Filebench [229] is executed and the completion time is recorded. Each line of Figure 4.2 represents a different combination of VMs. The top line shows that one VM is running the Filebench (shown in yellow), while other VMs are idle (shown in white). The Filebench is run eight times in eight randomly selected locations and the arithmetic mean of all the execution times is recorded. That is the

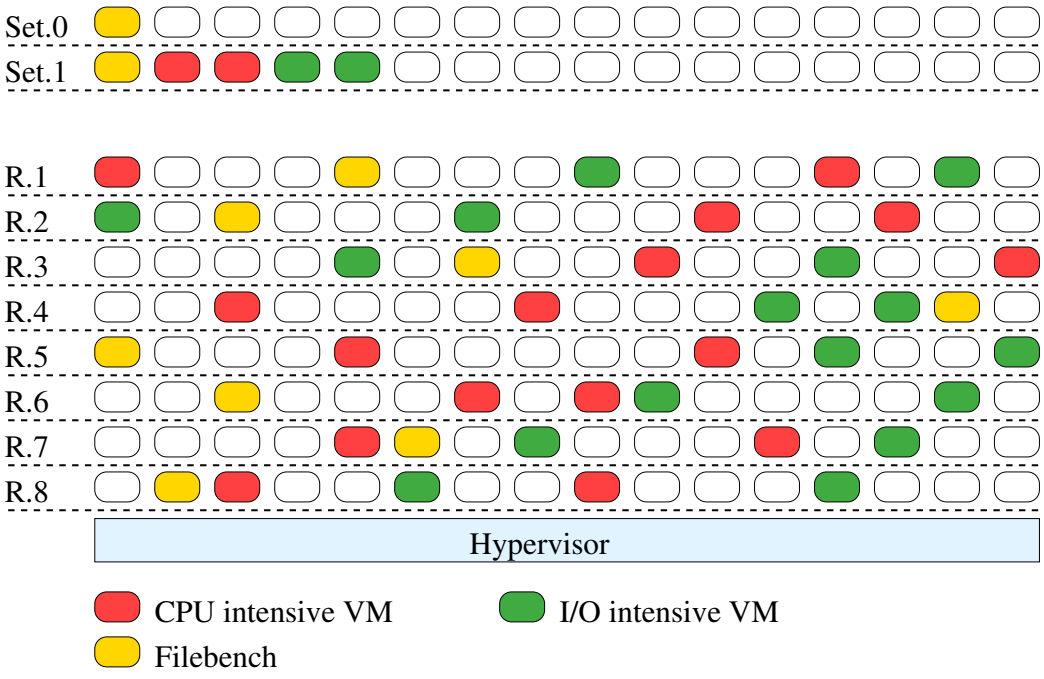


Figure 4.2: Combinations and rearrangements of the same five VMs.

setting of the first line is repeated eight times and each time a random VM was chosen to run the Filebench.

Next, the second line depicts that two CPU (shown in red) and two I/O-intensive (shown in green) VMs are added to the system. In this case, one VM runs the Filebench as before, and another four co-located VMs are also run simultaneously. The rest of the eleven VMs do not run any task (shown in white). Eight different random combinations of those five VMs are run and the arithmetic mean of the execution time of Filebench is calculated. The process of running those random combinations are further explained next.

In the second row, there are sixteen VMs in total, and only five are busy. Those five VMs are all placed at the left end of the row. However, it is possible to place those five VMs anywhere on the server. Like the eight random combinations of the VMs are shown in the next eight lines ($R.1$-$R.8$) of Figure 4.2. Although they are in random locations, each line contains exactly five VMs.

The experimental results of this section show that all random combinations of the same number of VMs suffer similar performance degradation. In other words, the results here show that the VM performance variation does not depend on locations of the VMs on the server rather it depends on the total number of VMs and their types. For example, the execution times of the eight random rows ($R.1$-$R.8$) of Figure 4.2 are 20.32, 19.5, 18.51, 20.22, 20.23, 19.74, 20.48, and 19.51 minutes, respectively. Their arithmetic mean is 19.985 minute.

The results show that the locations of the VMs do not influence the performance. Rather the total number of VMs and their resource intensities that are responsible for the task ETV. Thus, performance interference of the second VMs set (second line) of Figure 4.2 is the same as that of the eight random sets ($R.1$-$R.8$).

Above result has important significance for the rest of the experiments. It means that for a fixed group of VMs, there is no need to examine their performance interference for all possible VM combinations on the server. Experimental results show that the performance data collected for a combination is equivalent to all other combinations.

In the above experiment, only a set of 5 VMs is used. However, those experiments can be done with any number of VMs. In the next section, experiments are done with different sets of VMs with different number of VMs. Each time the similar results are observed.

In this section experiments are done with a fixed number of VMs, in the next section experiments are done by changing the number of VMs. That is in the next section, during experiments the total number of VMs is also changed. Again, for each of those set of VMs different combinations are used in the experiments.

4.3.2 Task performance variation and numbers of co-located VMs

The previous section demonstrates that if the numbers of co-located VMs is constant on a server; then shuffling of their locations do not affect the consolidation performance. This section deals with the question of what would happen if the total number of co-located VMs are changed.

In a data center, the number of simultaneously running VMs can change at any time. It is a troublesome process to profile task execution times for a vast number of combinations. Fortunately, the observation from the previous section can help us to overcome this problem. Results obtained in the previous section show that for a given number of VMs even if the VM combinations are changed their cumulative resource contention on the server remains the same.

In this section different experiments are done with different sets of VMs, where each set contains a different number of VMs. Figure 4.3 shows the experiential set up for this section. It differs from the experimental set up of the previous section (Figure 4.2), where all experiments were with a set of 5 VMs.

In this section, the VM number is increased at each stage. Figure 4.3 shows that several sets of CPU and I/O intensive VMs are run on the server. Each line represents a stage and set of VMs. The VM sets are arranged in increasing order, to record their cumulative effect on VM performance in increasing order. Figure 4.3 has nine stages, representing nine different sets of VMs. Each set has a different number of VMs. For example, first set in the top line has only one running VM. The second line has a set of three VMs. Next line has a set of five VMs. Thus, in each successive stage from top to bottom, the number of VMs is increased by two.

In stage 1 (top line of Figure 4.3), the Filebench is run on one VM. There are sixteen VMs in the system, and the Filebench can be run on any of those sixteen VMs. As shown in the second column of Table 4.1 there are C_1^{16} or 16 ways select a VM location for the Filebench to run. Eight random locations are selected, and the Filebench is run eight times, each time in a different location. Therefore, it is the same

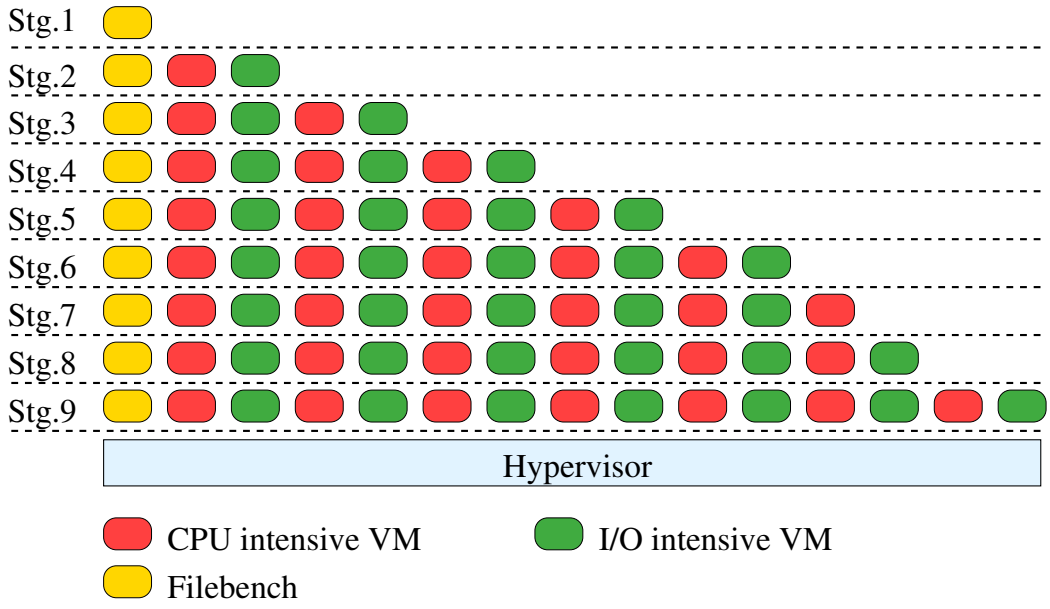


Figure 4.3: Incrementing VM number at each stage of experiment.

task to run eight different VM location on the server. The objective is to find how the performances of those eight runs differ from each other. The collected execution time data is shown in Figure 4.4.

The cluster of the first eight values on Figure 4.4 shows the execution finish times for eight different runs. In the graph, the Y-axis represents the execution finish time of the Filebench, while the X-axis indicates the number of co-located VMs running on the server beside the Filebench. For first eight runs, no other VMs are running beside the one running the Filebench. Hence, in the Y-axis the value is 0, indicating there are zero co-located VMs are running on the server. The X-axis shows that the first eight execution time values are close to each other. Those values are further are discussed below.

The Y-axis of Figure 4.4 shows execution finish times of the Filebench. First eight values of the graph show the execution finish time when the Filebench is run alone in the server. That means no-other co-located VMs are running on the server. Each time the Filebench is run on a VM located in a different location on the server. The first run takes 13.76 minutes to finish. Then the Filebench is moved to a VM in a different location and run again. The second run takes 14.27 minutes to finish. Similarly, the Filebench is run inside a VM six more time, and each time the VM moved to a different location. Execution finish times of those six runs are 13.92, 14.34, 13.02, 13.44, 13.99,

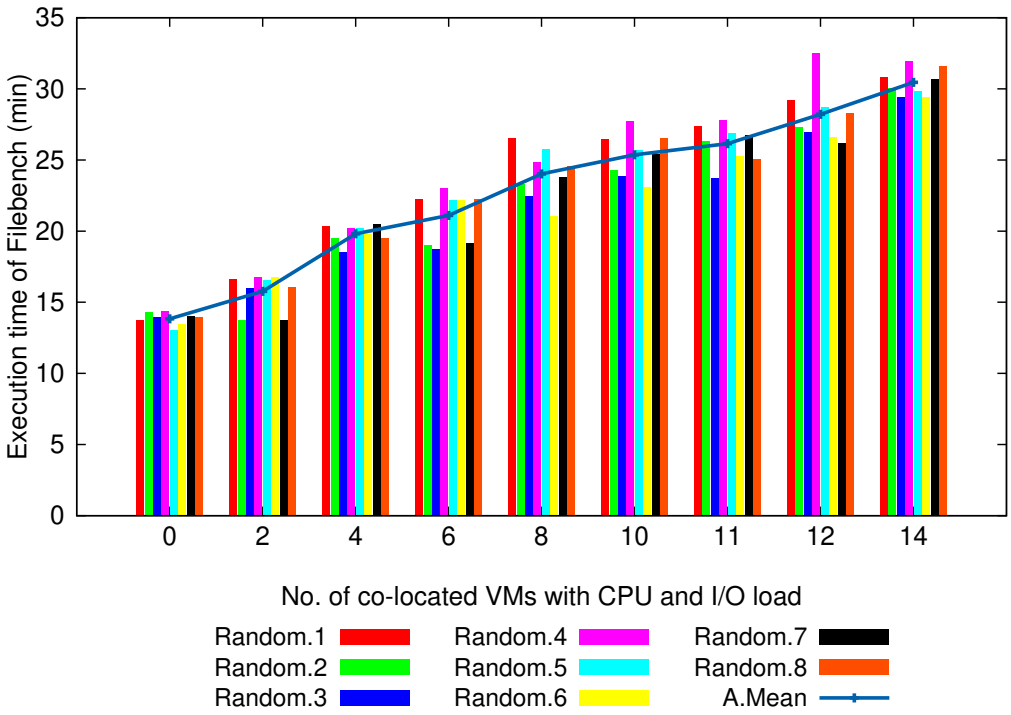


Figure 4.4: Execution time of increasing number of co-located VMs in Figure 4.3.

13.91 minutes, respectively.

Even though, the Filebench was run eight different location of the server their respective finish times are close to each other. The arithmetic mean of those eight-run times is 13.83125 minutes and *Standard Deviation* (SD) is only 0.4320859. A small value of SD indicates that the magnitude of variation among the original values is relatively small. The arithmetic mean and SD of those eight values are shown in the second row of the Table 4.1.

Table 4.1 shows the arithmetic mean and SD for each cluster of values in Figure 4.4. There are nine clusters of values in the Figure 4.4 they are presented in nine rows of Table 4.1. For example, the second row of Table 4.1 shows the arithmetic mean of the first eight execution times; it is 13.8312 minute. The SD of those eight runs is shown in the next column. The SD of those eight values is only 0.4320, which is only 3.12% of the absolute value of their arithmetic mean. Hence, those execution time values have very slight variation among themselves.

Next, stage 2 of the Figure 4.3 indicates that one CPU intensive and one I/O intensive VMs have been added to the server. Now, one VM run the Filebench as before and two more VMs run CPU-memory load combination. For those three VMs randomly

Table 4.1: Task execution time data for increasing number of co-located VMs in Figure 4.4.

No. of co-located VMs	Possible combinations (C_i^{16})	Arithmetic Mean of Filebench Exe. time (minute)	Standard deviation (SD)	Exe. Time increase compared to Stage 1 (%)	Exe. Time increase in consecutive Stages (%)
0 (0 CPU+0 I/O)	16	13.8312	0.4320	-	-
2 (1 CPU+1 I/O)	560	15.7637	1.2848	13.9452	13.9452
4 (2 CPU+2 I/O)	4368	19.8137	0.6474	43.2269	25.6979
6 (3 CPU+3 I/O)	11440	21.1000	1.7791	52.5536	06.5118
8 (4 CPU+4 I/O)	11440	24.0250	1.7798	73.7014	13.8625
10 (5 CPU+5 I/O)	4368	25.3662	1.5521	83.3969	05.5816
11 (5 CPU+6 I/O)	1820	26.1412	1.3670	89.0016	03.0560
12 (6 CPU+6 I/O)	560	28.2075	2.0380	103.9375	07.9043
14 (7 CPU+7 I/O)	120	30.4587	0.9555	120.2173	07.9500

eight different combinations are chosen to run on the server. After each run, the tasks are moved to VMs located in different locations and run again. Thus, those three VMs are run in eight different locations on the server. Execution times of those eight VM combinations are shown in the second cluster of values in Figure 4.4. The third row of Table 4.1 shows the arithmetic mean and SD for these eight execution times.

As shown in the table, there are C_3^{15} or 560 possible ways to place those three VMs on the sixteen available slots in the server. Figure 4.4 again shows that those execution times are close to each other. As shown in the third row of Table 4.1, the SD of all eight runs are only 1.2848, and the arithmetic mean is 15.7637 minutes. The SD of those eight values is only 8.15% to their arithmetic mean. Thus, statistically, variation among those eight values is relatively small. In other words, although the VMs are rearranged eight times; however, their execution times do not deviate much from their

mean value.

The execution time increase of the Filebench from the first to the second stage is also calculated. The arithmetic mean of execution times of eight runs of Filebench at stage 1 is 13.8312 minutes. It is shown in the second row of Table 4.1. At stage 2, two co-located VMs are added to the system. The execution time of Filebench in the presence of two additional VMs are also recorded. They are run in eight different combinations and the arithmetic mean of their execution times is 15.7637 minutes. Thus, the execution time of the Filebench is increased by 13.9452%, as calculated below.

$$(15.7637 - 13.8312)/13.8312) \times 100 = 13.9452\%$$

Those additional VMs have caused the execution time of the Filebench to be increased by 13.9452% from the initial value.

This value is shown in the third row of Table 4.1. The execution time of the Filebench increased by 13.9452% in stage 2 compared to that of stage 1. Recall that the arithmetic mean of execution times of stage 1 is shown in the second row of the table. The arithmetic mean of execution times of stage 2 is shown in the third row. Similarly, the percentage of execution time increases for other stages are calculated and shown in the fifth column of Table 4.1.

Next, at stage 3 five VMs are run on the system, one VM running the Filebench along with four other co-located VMs. Thus, stage 3 has two extra co-located VMs compared to the stage 2. Stage 3 is shown in the third line of Figure 4.3, it shows all five VMs. In this case, also, all VMs are run eight times on eight combinations of locations. After each run, the five VMs are moved to other random locations of the server.

The fourth row of Table 4.1 shows the execution time data of stage 3. The arithmetic mean of those eight-run times is 19.8137 minute, and SD is 0.6474. Here, SD is only 3.26% of the absolute value of the arithmetic mean of the execution times. Thus, statistically, the execution time variation among the eight runs is quite small. Thus, changing locations of VMs do not have much effect on execution times as long as the total number of VMs remain the same.

Furthermore, as shown in the fifth column of the Table 4.1, the execution time on stage 3 has increased by 43.2269% compared to that of stage 1. Recall that at stage 1,

the Filebench is run alone without any co-located VMs. Data for stage 1 is shown in the second row of the table.

Also, the sixth column shows that the execution time increase is 25.6979% compared to the stage 2. Recall that only 2 co-located VMs are used at stage 2 and data are shown in the third row of the table. Thus, this execution time increase is due to running two extra co-located VMs on the server.

In the same way, the number of VMs is increased by two at each stage. Figure 4.3 shows the all nine stages of the experimental setup. At each stage, VMs are run in eight different combinations. Then, Figure 4.4 shows the execution times of each of those nine stages for eight combinations. In Figure 4.4, nine cluster of values represent nine experimental stages.

The execution time values are grouped (as clusters in the Figure 4.4) according to the number of co-located VMs running on the system. In X-axis shows the number of co-located VMs. Value of 0 means there was no other VM on the system other than the Filebench. Value of 2 means two other VMs are running on the system besides the Filebench. The number of co-located VMs increased by 2 at each stage. Thus, in the final stage, 14 co-located VMs are running on the system.

Each cluster of Figure 4.4 has eight different execution time values for eight VM combinations. The values are clustered together based on the number of co-located VMs. For example, the first cluster of eight values is for running Filebench in eight different locations on the server. Here, no other co-located VMs were run in the system; hence, X-axis value for this cluster of values is 0.

The second cluster of Figure 4.4 groups together eight execution time values. In each of the eight cases, the Filebench is run along with two co-located VMs. Hence, for this cluster of values, the X-axis tick value is 2.

In this way, two extra co-located VMs are added to the system. Thus, in the final stage of Figure 4.3 (stage 9), seven CPU intensive VMs and seven I/O intensive co-located VMs are run on the server. The execution times of all nine stages are shown on Figure 4.4.

Recall that in each stage, eight random combinations of VMs were run. The execution time of Filebench for all of the setting is shown in Figure 4.4. In the figure, eight execution times for each set of VMs are shown as histograms in a cluster. Execution times from a different set of VMs are shown in different clusters. In Figure 4.4, there are nine such clusters for nine set of VMs. The arithmetic means of eight execution

time values of each cluster are also calculated. Those nine values of arithmetic means are shown as a line graph in the figure as well. Arithmetic mean and SD for each cluster of values is calculated separately and shown in Table 4.1.

Table 4.1 shows the arithmetic mean of execution time for all nine stages of the experiment. The table shows that the execution time of the Filebench increases with the increase in the number of co-located VMs. In the table, the number of co-located VMs are increased by two at each stage. However, there is an exception between the seventh and eighth rows.

The number of co-located VMs between the seventh and eighth rows differ by only one. It is done to show that if the number of co-located VMs is changed by one, then execution time difference becomes smaller. The seventh row of Table 4.1 shows the execution finish time of Filebench due to ten co-located VMs. On the other hand, the eighth rows show the execution finish time of the same Filebench due to eleven co-located VMs.

Recall that the sixth column of Table 4.1 shows the execution time difference percentage between two consecutive stages. It can be seen that execution time differences between the seventh and eighth rows are only 3.0560%. It is relatively small compared to other cases. That is why the number of VMs is increased by two in each stage instead of one. As increasing the VM number by one does not show a significant difference in execution time.

Recall, Figure 4.4 shows the execution time of Filebench for nine group of co-located VMs. Each group has the same number of co-located VMs; however, in randomly selected locations. Table 4.1 shows the arithmetic mean and SD for each of the nine groups. One row of the table presents data for one stage of the experiment.

The fourth column of Table 4.1 shows that the SD values for each stage. The SD is calculated from the eight execution times of each group. A small SD for a group of values indicate that the variation among the original values is small. The second row of the table shows that the SD of eight execution time of stage 1 is only 0.4320. The third row shows that the SD of the eight execution times of stage 2 is 1.2848. Likewise, from the fourth column, it can be seen that for each of the nine stages of the experiment the SD values are small.

Small SD for a stage, indicates that the execution times do not change much due to re-arrangement of the VM locations. Small SD values of the fourth column indicate that as long as the number of co-located VMs remains constant no matter how many

ways they rearranged or recombined the execution time varies a little. Therefore, those VMs cumulative effect on the resource contention remains the same.

The fifth column of Table 4.1 shows the increase in execution time from the initial execution time. The initial execution time means the execution time of Filebench at stage 1. Recall that at stage 1, the Filebench was run alone without any co-located VM. The co-located VMs are added in successive stages and execution time increased because of that. The fifth column indicates the execution time increase in percentage compared to the stage 1. The sixth column of Table 4.1 shows the execution time increase percentage for two consecutive stages.

The findings from the experiments in this section can be summarized as follows:

1) The execution times of VMs start to increase as the number of co-located VMs is increased in the system;

2) The variation of VM execution time is not dependent on the location of the VMs; rather it depends on the total number of VMs running on the system and their resource usage.

In this section, experiments are conducted with only one task, the Filebench. However, the result summary of this section provides a way to profile the task ETV variation due to VM consolidation. Based on the findings of this section a methodology is developed in the next section. With the help of that methodology, the experiments are repeated with a larger number of tasks and hypervisors in the following chapters.

4.4 Incremental Consolidation Benchmarking Method (ICBM)

Based on the results of the previous sections this section introduces a methodology for profiling the task ETV of consolidated servers called, the Incremental Consolidation Benchmarking Method (ICBM). Pseudocode for the methodology is shown in Figure 4.5. Parameters of this pseudocode are described in Table 4.2.

The methodology can be applied step by step to any server with virtual machines; there is no particular requirement for applying the method. The only assumption is that the server is capable of consolidating multiple VMs; as the example shown in Figure 4.6.

Table 4.2: Parameters of the pseudocode of Figure 4.5.

Symbol	Meaning
T	Is a list containing all the tasks used in the experiments.
B_{cpu}	Is a set of benchmarks used to increase CPU resource contention in the server.
B_{mem}	Is a set of benchmarks used to increase Memory resource contention in the server.
$B_{i/o}$	Is a set of benchmarks used to increase I/O resource contention in the server.
v_t	One VM designated as the target VM. During experiments, the execution time of tasks of this VMs is profiled.
v_{co}	A set of VMs designated as co-located VMs. During experiments, tasks are run on these VMs to create resource contention on the server.
Δ	Number of VMs increased at succussive stages of the experiment.
t_i	Is the i^{th} task in the list, T.
$ETV^i_{0,0}$	Execution time of the task t_i at stage 1. At this stage the task is run without any co-located VM.
k	is a variable that determines how many co-located VMs should run at each stage.
$ETV^i_{j,k}$	is the execution time of task i when it is consolidated with k number of type j co-located VMs.

Steps of the methodology (Figure 4.5) are explained with the example of Figure 4.6. Lines 1-9 of Figure 4.5 initializes the variables. Let T is a set of task and ETV of each task, $t_i \in T$ will be measured. B_{cpu}, B_{mem}, and $B_{i/o}$ are sets of CPU, memory, and I/O intensive benchmarks respectively.

One of the VMs is designated as the target vm (v_t), while rest are designated as co-located (v_{co}) vms. The v_t is used to run different tasks to record their ETV. On the

1: $T \leftarrow$ A set of tasks.
2: $B_{cpu} \leftarrow$ A set of CPU intensive benchmarks.
3: $B_{mem} \leftarrow$ A set of memory intensive benchmarks.
4: $B_{i/o} \leftarrow$ A set of I/O intensive benchmarks.
5: $v_t \leftarrow$ A target vm.
6: $v_{co} \leftarrow$ A set of co-located vms.
7: $\Delta \leftarrow$ Number of VMs increased at each stage.
8: Benchmark types $B \leftarrow \{B_{cpu}\}, \{B_{mem}\}, \{B_{i/o}\},$
9: $\{B_{cpu}, B_{mem}\}, \{B_{mem}, B_{i/o}\}, \{B_{cpu}, B_{i/o}\}.$
10: **for** Each task $t_i \in T$ **do**
11: Add v_t to the host.
12: $ETV^i_{0,0}$ = Execution of t_i on v_t alone.
13: $k = 0$
14: **for** Each benchmark type $b_j \in B$ **do**
15: **while** All the v_{co} are responding **do**
16: $k = k + \Delta$.
17: Add k number of v_{co} with benchmark of type b_j to host.
18: Run v_t simultaneously with the added v_{co}.
19: $ETV^i_{j,k}$ = Execution finish time of the v_t.
20: **end while**
21: Remove all v_{co} and free related system resources.
22: **end for**
23: **end for**

Figure 4.5: Incremental Consolidation Benchmarking Method (ICBM) pseudocode.

other hand, v_{co} have been used to create resource contention collectively. The tasks of T are run on v_t, while benchmarks of B_{cpu}, B_{mem}, and $B_{i/o}$ are run on v_{co}.

Let $t_i \in T$ be a task whose ETV due to CPU intensive co-located VMs is going to be investigated. At first, the execution finish time of t_1 is measured without any interference from co-located VMs. Lines 11-12 shows that t_i is run alone on a single VM (v_t) of the host, this corresponds to the stage 1 of Figure 4.6. The execution time of task, t_i without any interference is stored in ETV^i_0.

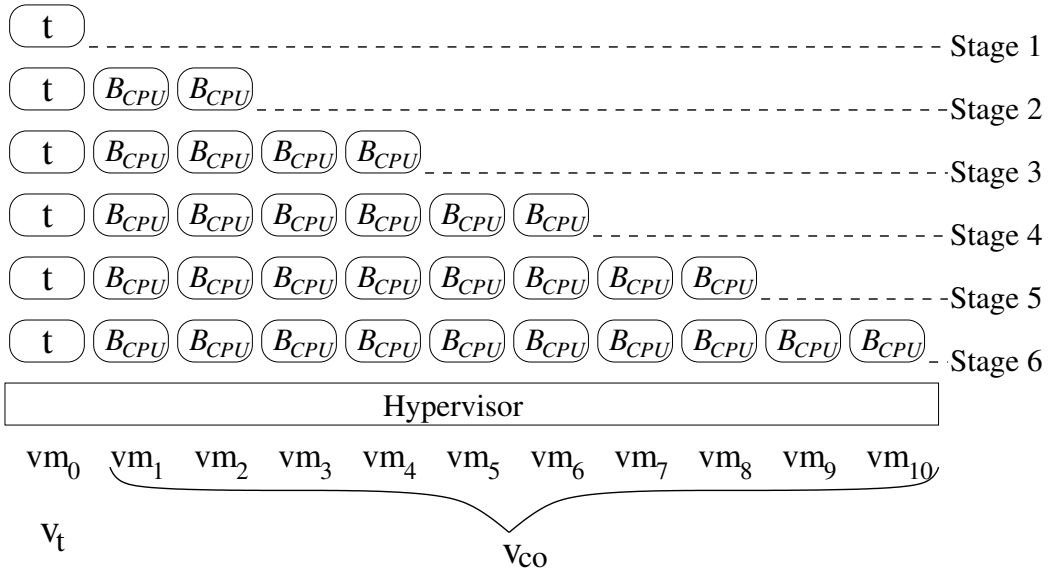


Figure 4.6: Task execution time profiling with CPU intensive benchmarks in co-located VMs.

Next, in stage 2 of Figure 4.6 the t_i is run again, this time along with two simultaneously running co-located vms (v_1 and v_2). Both new VMs, run one B_{cpu} type benchmark each, thus only increasing CPU intensive load on the system. It gives the execution time of t_i on v_t, which is now consolidated with two co-located CPU intensive VMs. At each stage, the number of co-located VMs are increased that is why a while loop is used in lines 15-20.

The while loop repeatedly runs t_i, and the number of co-located VM is increased at each stage. How many co-located VMs would be run at each stage is determined by the variable k, which is initialized to 0 at line 13 just before the loop starts. Another variable Δ determines how many new co-located VMs are to be added to the host; it is also initialized at the beginning.

At each iteration, the value of k is increased by the amount of Δ as shown in line 16. All the benchmarks, b_j used in the co-located VMs are stored in the list B. Above steps are need to be repeated for each benchmark, $b_j \in B$. Therefore, another for loop is placed in the lines 14-22. The execution time of the i^{th} task for k number of j^{th} benchmark is stored in $ETV^i_{j,k}$. As the loops iterate, all the values are updated accordingly.

In the next iteration, two more CPU intensive v_{co} are added (stage 3 of Figure 4.6),

increasing the number of CPU intensive co-located VMs to four. The execution finishes time of v_t is profiled again for this setting. This way, two new v_{co} are added at each stage until the co-located VMs stop responding.

The maximum number of VMs that can be simultaneously run on a host without making it non-responsive is dependent on the host configuration. Most hypervisors are designed to handle a large number of VMs simultaneously. For example, XenServer version 7.5 is designed to run up to 1000 VMs concurrently on a host. However, in reality, the number of concurrent VMs are limited by VM workload, system load and other environmental factors [272]. Thus, in practice, the VM number depends on the VM and host hardware configuration.

In the experimental set of above, the host has one *Intel i7-3770* processor with four cores and eight hardware threads. Assigning, a logical CPU to each VM the system could simultaneously run a maximum of fourteen such VMs, adding anymore VM would make the whole system non-responsive.

With one VM designated as v_t and two more new v_{co} added at each stage, the final stage (stage 7) of the experiment had thirteen simultaneously running VMs (one v_t along with twelve v_{co}). The VMs are created with a least possible subset of CPU resources so that CPU load can be increased granularly. However, VMs with a larger number of logical CPU can also be created depending on the host hardware configuration.

Once the task execution times are profiled with CPU intensive co-located VMs the memory intensive co-located VMs. The same steps are repeated for memory intensive v_{co}; as an example is shown in Figure 4.7. However, in this case, memory intensive benchmarks are run on co-located VMs instead of CPU intensive benchmarks.

All VMs are configured to have 1 GB of RAM. The experiment starts (stage 1 of Figure 4.7) with a single VM (v_t) running the task, whose ETV due to memory intensive co-located VMs is going to be investigated. Next, (stage 2 of Figure 4.7) v_t is run again along with two new co-located vms (v_1 and v_2), each running one memory intensive benchmark. Similarly, two more VMs (v_i and v_{i+1}) are added at each stage until the host system reaches a predetermined point of memory load.

In this case, adding a new v_{co} makes the host memory load to be increased by 1 GB. It was done so that, the host memory load can be changed granularly. The host has 16 GB of RAM. By restricting the number of VMs to thirteen at the final stage, maximum of 13 GB RAM is allocated to the co-located VMs leaving rest for the hypervisor. As

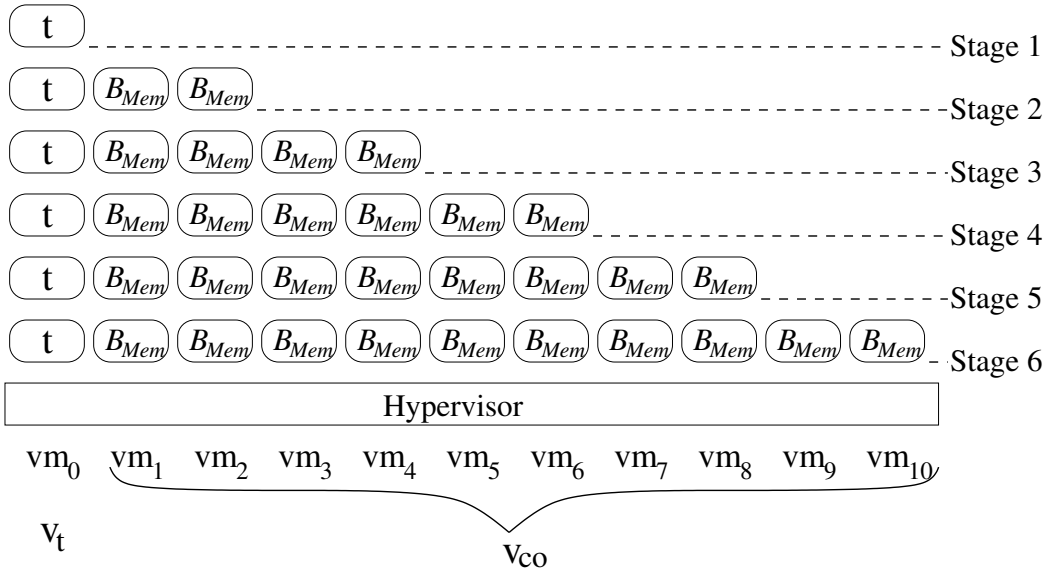


Figure 4.7: Task execution time profiling with memory intensive benchmarks in co-located VMs.

with the CPU load, this predetermined load is also not a fixed number.

Afterward, for I/O load the same steps are repeated by adding two I/O intensive benchmarks on two v_{co}, at each stage. Thus, the ETV data for three basic resource types (CPU, memory and I/O) are collected.

4.4.1 Measuring ETV for resource combinations

Next, the algorithm of Figure 4.5 repeats the same procedure for resource combinations, too. The combinations are made by choosing two resource types at a time from the previously mentioned basic three resources. The basic resources are (i) CPU, (ii) Memory, and (iii) I/O. Their combinations used here are (iv) CPU-Memory, (v) Memory-I/O and (vi) I/O-CPU. Thus, six resource types are used in the co-located VMs; three are basic, and the other three are combinational.

Experiments for the combination of loads are done the same way. That is, start with one VM (v_t), and at each stage add two co-located VMs (v_i and v_{i+1}) to increase the load. The difference is that now two new VMs run two different types of benchmarks. Both of them, together create the effect of load combinations. For example, to increase CPU-Memory load, two v_{co} are added at each stage. One (v_i) runs a CPU intensive

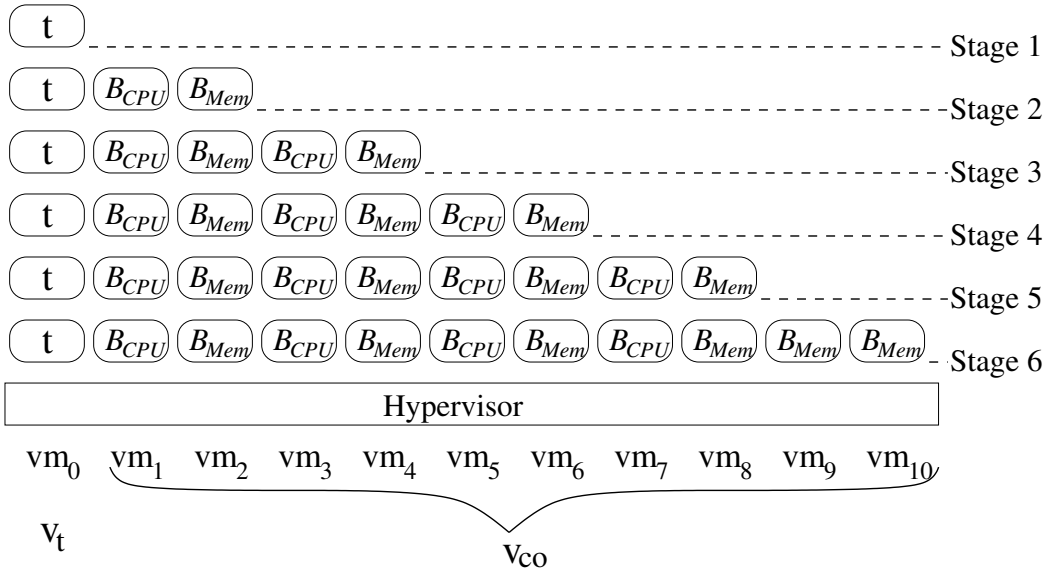


Figure 4.8: Task execution time profiling with CPU-memory resource combination in co-located VMs.

benchmark, while the other one (v_{i+1}) runs a memory intensive benchmark. An example of using resource combination in co-located VMs is given in the Figure 4.8. For the other two resource combinations, the same procedure is followed.

Above experiments demonstrate how the execution time of v_t is varied due to co-located VMs (v_{co}^i). However, there is another angle to this problem, that is how the v_{co} are collectively affecting the execution times of each other. To examine this the execution finish times of all the v_i are also recorded at each stage. Finally, the whole procedure is repeated without the v_t altogether. That is, all the experimental steps are repeated by adding only load (on v_{co}^i) at each stage. The ETV data obtained by applying the algorithm are presented in the next few sections.

4.5 Execution time variations (ETV) of the target VM (v_t)

Discussion about the ETV of the targets VM (v_t) is divided into two sections. In this section, the ETVs due to basic types of load on the co-located VMs are addressed, while in the next section ETVs due to the combination of resources are discussed.

Discussion about ETV of both basic and combinational resources are necessary.

Later in the chapter, it is described how the prediction models can be built from the collected ETV data. Finally, prediction models are presented in Section 4.9. The models are relationships between the ETV due to basic resources to that due to the combination of resources. Therefore, discussion about both basic and combinational types of resources is necessary.

The basic resource load is created by using one of the following three resource types: CPU, memory, and I/O. During this section, only one type of resource load is increased in the co-located VMs. Figures 4.9, 4.10 and 4.11 show the task ETV of v_t due to three basic types of resource loads. The three resources are being, CPU (Figure 4.9), memory (Figure 4.10) and I/O (Figure 4.11).

In each graph, the X-axis shows the total number of VMs running on the system, while the Y-axis presents the task execution completion time of v_t. The first point of the graph shows the execution time of v_t when it is run alone on the server. At this stage v_t is free from any interference from co-located VMs. As the algorithm described previously in Figure 4.5 and example given in Figure 4.6. Then, two co-located VMs (v_{co}) are added to the server at each successive stage. In the final stage, twelve v_{co} are simultaneously run beside the v_t.

The X-axis is marked with how many VMs were running at each stage. For example, at stage 1 only the v_t was running. In stage 2, one v_t and two v_{co} run on the host. Finally at stage 7, v_t is run with twelve v_{co}. Various resource intensive benchmarks are run on v_{co} to create resource contentions systematically. On the other hand, tasks are executed on the v_t to measure ETV.

The Y-axis shows the ETV of the tasks on the v_t. ETVs of seven tasks have been measured they are, Nbbench, Unixbench, Cachebench, Stream, Dbench, Filebench, and I/O zone. Among the above tasks, the Nbbench, Unixbench, and Cachebench are CPU intensive tasks. The Cachebench, and Stream intensive tasks. While, Stream, Dbench, and Filebench are I/O intensive tasks. Three types of tasks have been used because they react differently to various types of resource contention.

From left to right along the X-axis, the interference from co-located VMs increases. The first point of each graph gives the execution time of the task without any interference from co-located VMs. On the other hand, the last point gives the execution time with maximum interference from co-located VMs. Results show that different types and number of v_{co} make the task execution time to vary at a different rate.

Figure 4.9 shows that the task ETV of v_t with the increase of CPU intensive v_{co}.

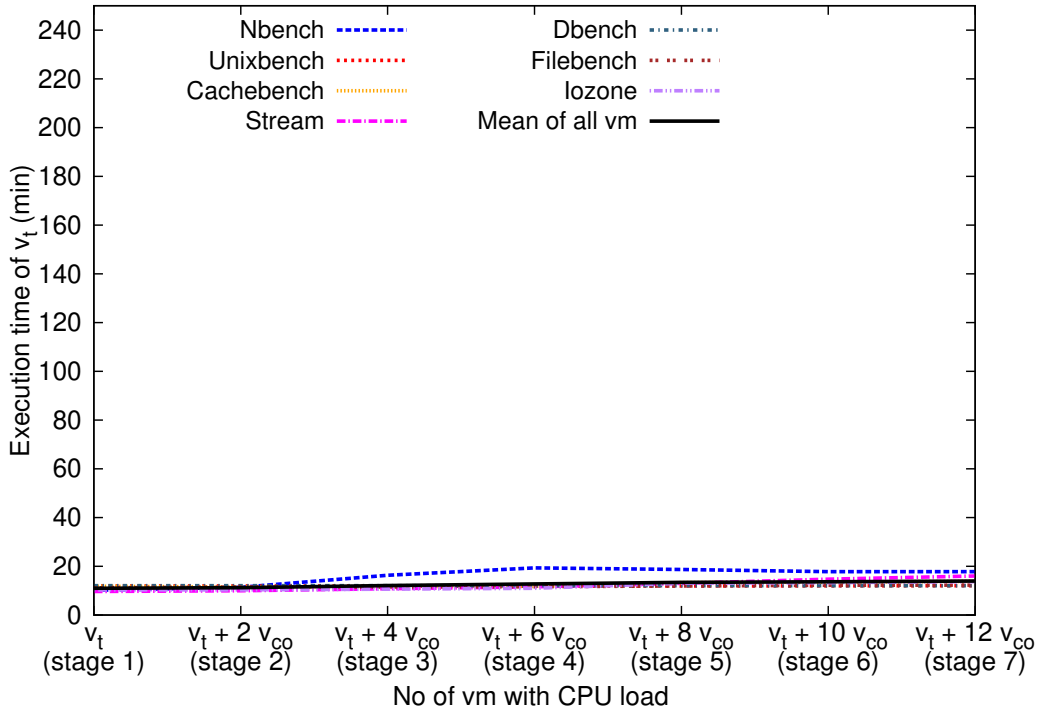


Figure 4.9: The task ETV of v_t due to CPU resource load on v_{co}.

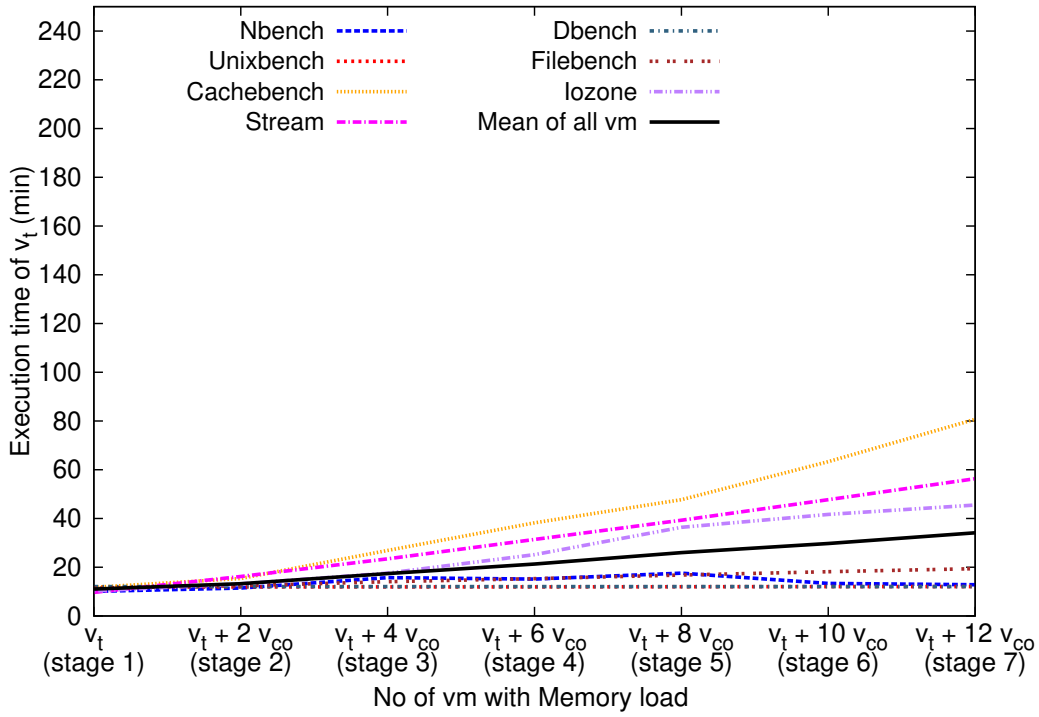


Figure 4.10: The task ETV of v_t due to Memory resource load on v_{co}.

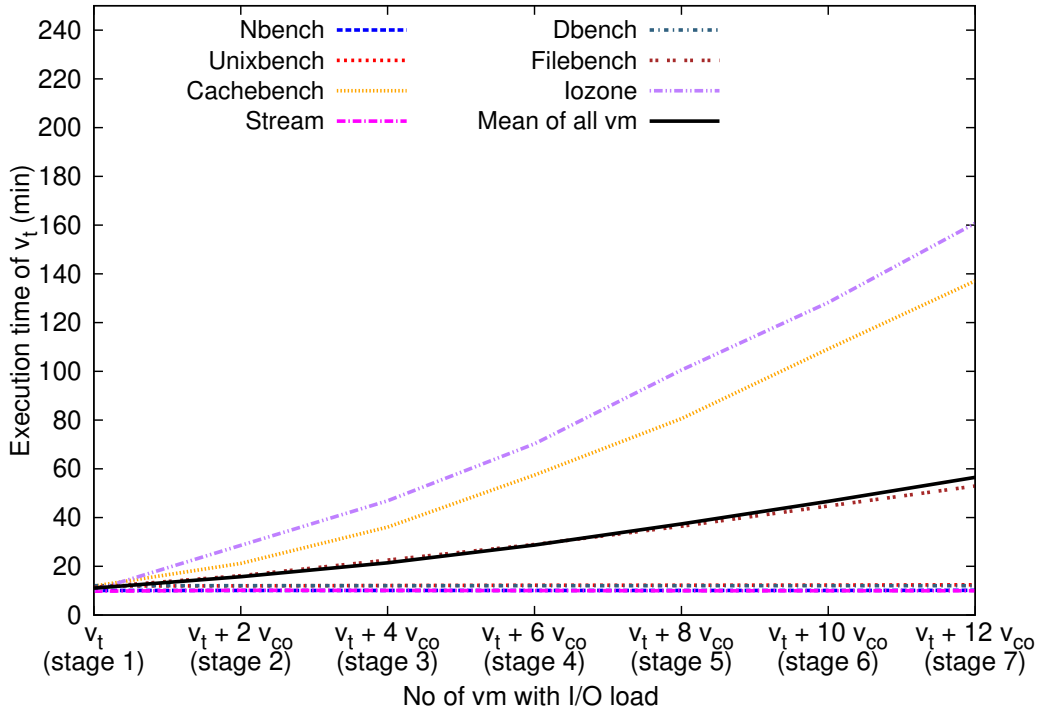


Figure 4.11: The task ETV of v_t due to I/O resource load on v_{co}.

The task ETVs due to CPU intensive resource contention is the minimum among six types of workloads.

On the other hand, for memory intensive v_{co} (Figure 4.10) the two CPU intensive tasks (Nbench and Unixbench) on v_t show less variation compared two memory intensive tasks (Cachebench and Stream) under the same situation. As an example, in isolation, the Nbench takes 10.1 minutes to execute on v_t. With other 12 memory intensive co-located VMs (v_{co}) it extends to 12.83 minutes, which is only 27.02% extension. In contrast, the Cachebench under the same setup takes 587.58% longer to execute (execution time goes from 11.76 min to 80.68 min).

Figure 4.11 shows the task ETV due to I/O intensive v_{co}. Here, CPU intensive tasks do not show much performance degradation, while both the memory and I/O intensive tasks do. For example, the Cachebench (a memory intensive task) have 1057.311% increase in execution time, while Iozone (an I/O intensive task) have 1482.93%.

4.5.1 ETV results of v_t due to combination of resources

In the previous section, the v_t task ETV data due to basic resource types have been discussed. In this section, the v_t task ETV data due to resource combination is going to be discussed. Recall that Section 4.4.1 describes the experimental stages to collect ETV data due to resource combination on co-located VMs.

Figures 4.12, 4.13 and 4.14 show the task ETV on v_t for combinational workloads on v_{co}. The workload combinations are being CPU-Memory (Figure 4.12), Memory-I/O (Figure 4.13) and CPU-I/O (Figure 4.14).

Figure 4.12 shows the ETV of seven tasks due to a mix of CPU and memory intensive v_{co}. Here, CPU intensive tasks (Nbench and Unixbench) show the least amount of degradation just as observed previously. Among the memory intensive benchmarks, the Cachebench shows the highest rate of performance degradation (542.74%).

On the other hand, for I/O intensive tasks the effect of CPU-Memory v_{co} combination is less adverse compared to memory intensive tasks. Figures 4.14 and 4.13 show the performance degradation of v_t when I/O intensive v_{co} is coupled with CPU and memory intensive v_{co}, respectively.

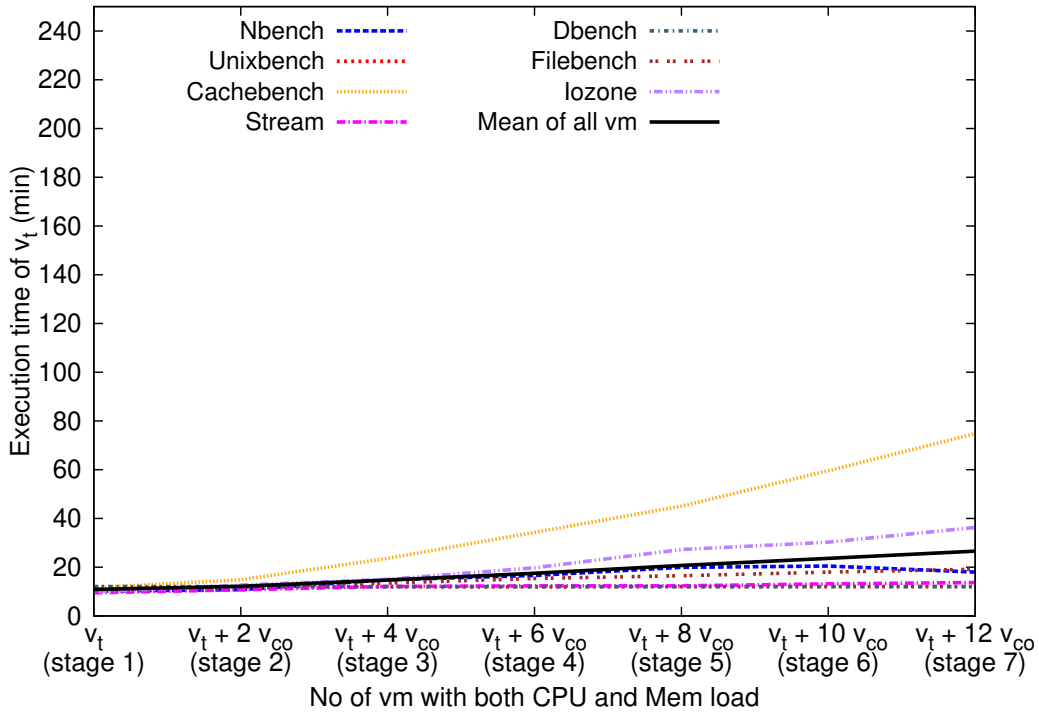


Figure 4.12: The task ETV of v_t due to CPU-Memory resource combination on v_{co}.

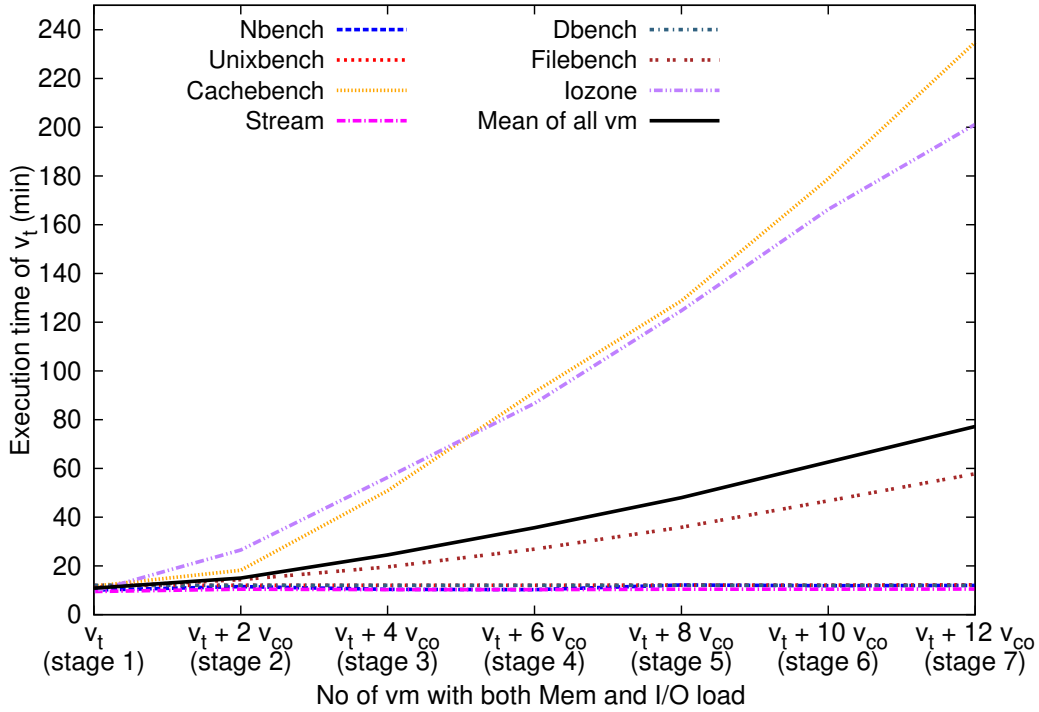


Figure 4.13: The task ETV of v_t due to Memory-I/O resource combination on v_{co}.

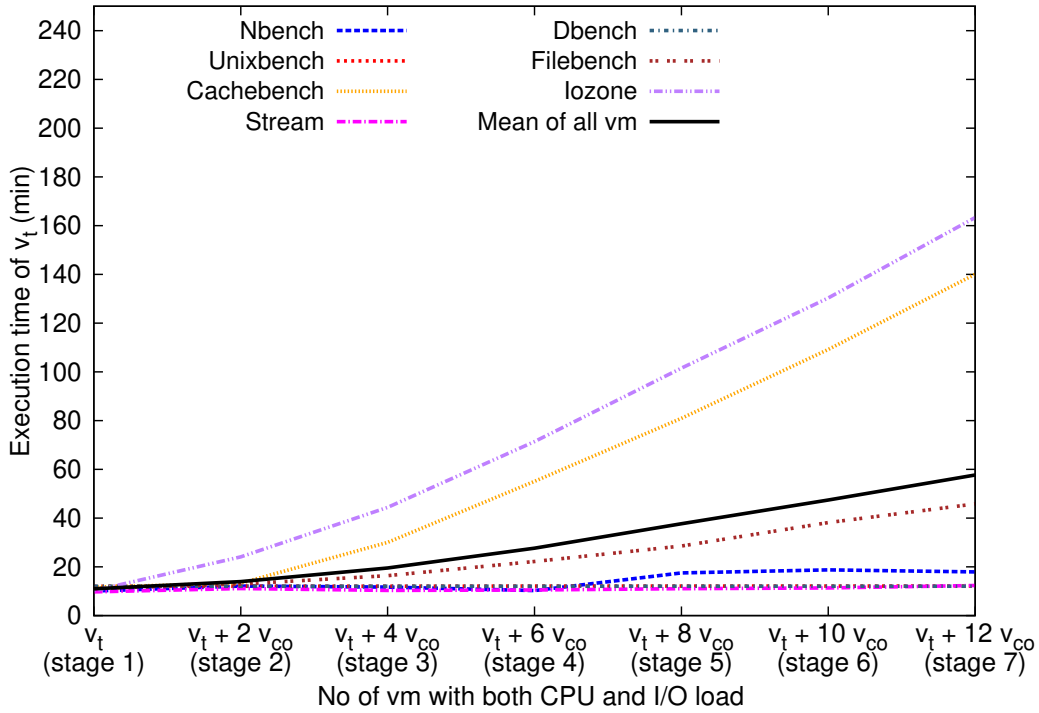


Figure 4.14: The task ETV of v_t due to CPU-I/O resource combinations on v_{co}.

In both cases, memory and I/O intensive tasks on v_t show comparatively more performance degradation. In the case of CPU-I/O load (Figure 4.14), the Cachebench and Iozone show increase of execution time by 1907.21% and 1991.67%, respectively.

Again for Memory-I/O load (Figure 4.13), the same two benchmarks show worse performance degradation (1100.09% and 1502.56%, respectively).

The graphs in this section show that the amount of performance degradation is different for different types of tasks. The amount is depended on the resource requirement of the task and co-located VMs. The SD of the execution times of tasks is also calculated and shown in Tables 4.3, 4.4, 4.5. The significance of those SD values are also discussed in the next section.

4.5.2 Standard deviation (SD) of the target VM (v_t) execution times

The Standard Deviation or SD is a measure that is used to quantify the amount of variation or dispersion of a set of data values [273]. A low SD indicates that the values are close to the arithmetic mean of the values. On the other hand, a high SD indicates that the values are spread over a broader range.

Table 4.3 shows the SD of execution times of two CPU-intensive tasks on v_t through stage 1 to stage 7. Table 4.4 shows the SD for two memory-intensive tasks, while the Table 4.5 shows the SD for two I/O-intensive tasks. Those values are calculated from the graphs in the previous section.

Recall that Section 4.5 discusses the ETV of v_t for basic resource loads. During the stages of the experiment, the number of VMs with basic loads are increased in the system. The results for three basic resources are shown in the Figures 4.9, 4.10 and 4.11.

In Section 4.5.1, the ETV due to combination of resources are considered. Again, Figures 4.12, 4.13 and 4.14 show the results for co-located VMs. As discussed in those section different types of resource intensive tasks faces different amount of performance degradation.

Above mentioned graphs show the task execution times of the target (v_t) at seven stages of the experiment. Each graph shows the ETV of the same seven tasks along the Y-axis. However, each graph has different workload on the co-located VMs along X-axis. In this section, SD of the execution times of tasks has been calculated from the data.

Table 4.3: Standard deviation (SD) of ETV of two CPU-intensive tasks on v_t.

	Task name / Workload type on co-located VMs	Nbench	Unixbench
Basic	CPU-intensive	3.66	0.05
	Memory-intensive	2.58	0.05
	I/O-intensive	0.01	0.15
Combination	CPU-Memory-intensive	4.12	0.05
	CPU-I/O-intensive	0.88	0.08
	Memory-I/O-intensive	3.82	0.06

For example, Figure 4.9 plots the execution times variation of the Cachebench for CPU intensive co-located VM workload. As discussed in the Section 4.5, the Cachebench being a memory intensive benchmark do not show much execution time variation due to CPU intensive workload. From the graph, seven execution times of the Cachebench is collected, and their SD is calculated, it is only 0.99. This value is shown in the second row of the third column of Table 4.4.

Next, recall that Figure 4.10 shows seven execution times for the Cachebench. However, this time memory intensive co-located VMs are being used, and it can be seen that the execution times variation is different. Here, also the seven execution times are collected from the graph and SD is calculated. In this case, SD of the execution times of the Cachebench is 25.28, indicating a more significant difference among the all original values. Thus, memory intensive workload has a more significant influence on the execution time of Cachebench. This value is shown in the third row of the third column of Table 4.4.

Also, recall that Figure 4.11 shows the execution time variation of the Cachebench due to I/O intensive workload. The SD of those execution time is 46.59. This value is shown in the fourth row of the third column of Table 4.4.

In Section 4.5.1, ETV of the v_t due to combinational resource load is discussed. In this case, experiments are performed using resource combinations on the co-located

Table 4.4: Standard deviation (SD) of ETV of two Memory-intensive tasks on v_t.

	Task name / Workload type on co-located VMs	Cachebench	Stream
Basic	CPU-intensive	0.99	2.42
	Memory-intensive	25.28	16.89
	I/O-intensive	46.59	0.12
Combination	CPU-Memory-intensive	23.51	1.43
	CPU-I/O-intensive	83.85	0.35
	Memory-I/O-intensive	49.40	0.82

Table 4.5: Standard deviation (SD) of ETV of two I/O-intensive tasks on v_t.

	Task name / Workload type on co-located VMs	Filebench	Iozone
Basic	CPU-intensive	0.41	1.51
	Memory-intensive	2.97	14.18
	I/O-intensive	15.16	54.70
Combination	CPU-Memory-intensive	3.06	9.89
	CPU-I/O-intensive	17.33	71.62
	Memory-I/O-intensive	13.17	56.62

VMs. For example, Figure 4.12 shows the execution time variation of the Cachebench due to CPU and memory load combination. The SD of this set of execution times is shown on the fifth row of the third column of Table 4.4.

Similarly, six SD values for the Cachebench are calculated and shown in the third column of Table 4.4. Six SD values are for six workload types on co-located VMs. The second column of the table also shows the workload types used in the co-located VMs. The fourth column of the table shows the SD values for another memory intensive task, the Stream.

Using the procedure as described above, SD for two CPU and two I/O intensive tasks are also calculated; their values are shown in Tables 4.3 and 4.5, respectively.

Recall that in Sections 4.5 and 4.5.1, the performance degradation of the v_t is expressed in terms of execution time increase percentage. Observing the ETV of tasks is one way to present the effect of VM consolidation on task performance. A higher execution time increase percentage indicate that VMs are performing worse. It is also discussed that the performance of v_t can degrade rather quickly particularly with the increase of memory and I/O intensive co-located VMs. For those two types of resource loads, the performance starts to deteriorate even when half the number of v_{co} that the physical system is capable of hosting, is running.

In this section, the task ETV results of two previous sections (4.5, and 4.5.1) are presented in terms of SD. Tables 4.3, 4.4, and 4.5 show that the memory and I/O intensive task execution times shows greater SD values indicating their execution time values are spread over a wider range. Thus, SD provides another way to measure the VM performance degradation quantitatively.

In last three sections (4.5, 4.5.1, 4.5.2), ETV data for the target VM (v_t) is presented and and their significance is discussed. In the next section, ETV data for the co-located VMs (v_{co}) are analyzed.

4.6 Execution time variations of the co-located VMs (v_{co}^{i})

In the previous section (Section 4.5), task ETV data is collected from the target VM (v_t) and presented in different figures. Figures 4.9, 4.10, and 4.11 show the ETV data due to basic resource loads. Figures 4.12, 4.13, and 4.14 show the ETV data due to combinational resource loads. During the same stages of experiments, data is

also collected from the co-located VMs (v_{co}^{i}), and those data are now presented in this section.

Recall that in Section 4.4, a methodology is presented to systematically profile the task execution time changes due to the changing number of VMs. In the experimental setup two categories of VMs are used, target VM (v_t) and co-located VM (v_{co}). Then in section 4.5, the experimental results of the target VM (v_t) are discussed. In this section, experimental results for the co-located VM (v_{co}) are discussed.

Section 4.4 describes the stages of the experiment with the consolidated VMs. During the experiments, the execution finishes time of all the co-located VMs (v_{co}) at each stage are profiled. This section discusses the EVT data of v_{co} and this data has some interesting properties. Later in the section 4.7.3 it will be shown that these properties can be useful for predicting the task ETV values of v_t.

This section presents the ETV data of co-located VMs. Figures 4.15, 4.16, 4.17, 4.18, 4.19, and 4.20 show the arithmetic mean of execution times of all v_{co}^{i} at each stage, separately.

During experiments, it was observed that the arithmetic mean of the execution time

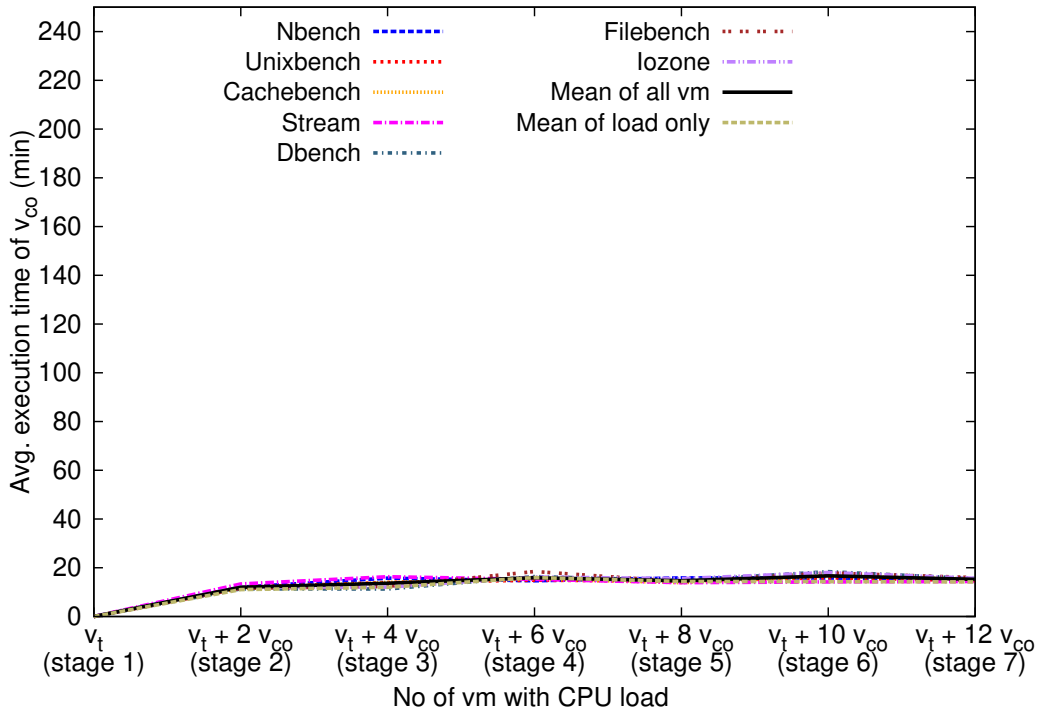


Figure 4.15: Arithmetic mean of task ETV of v_{co} with respect to CPU load changes.

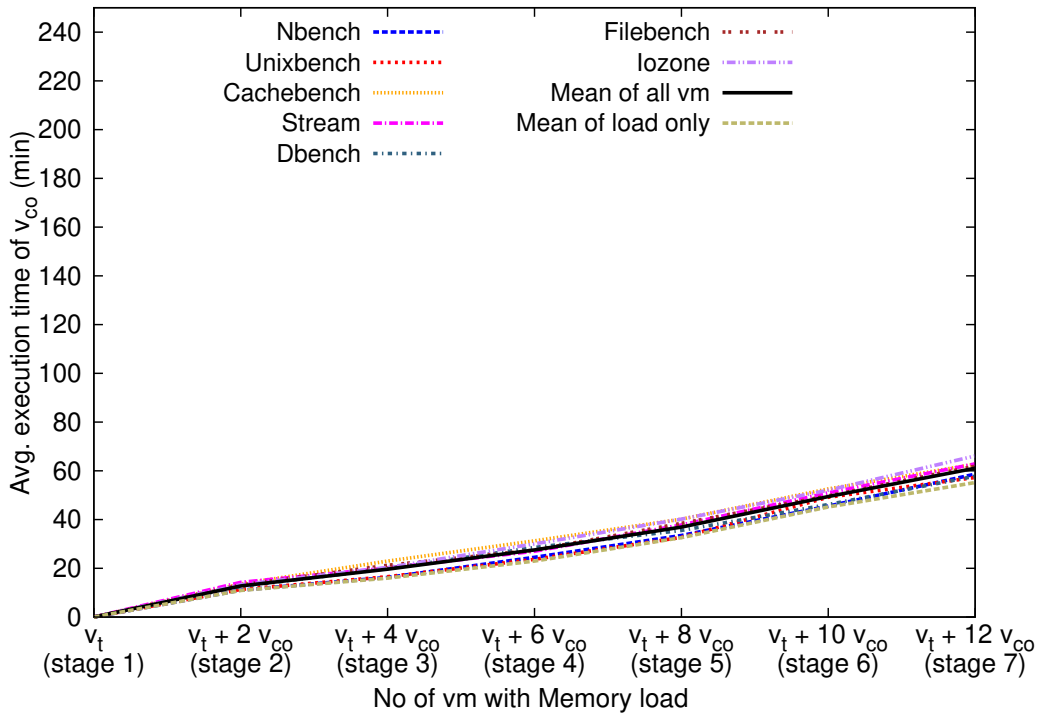


Figure 4.16: Arithmetic mean of task ETV of v_{co} with respect to Memory load changes.

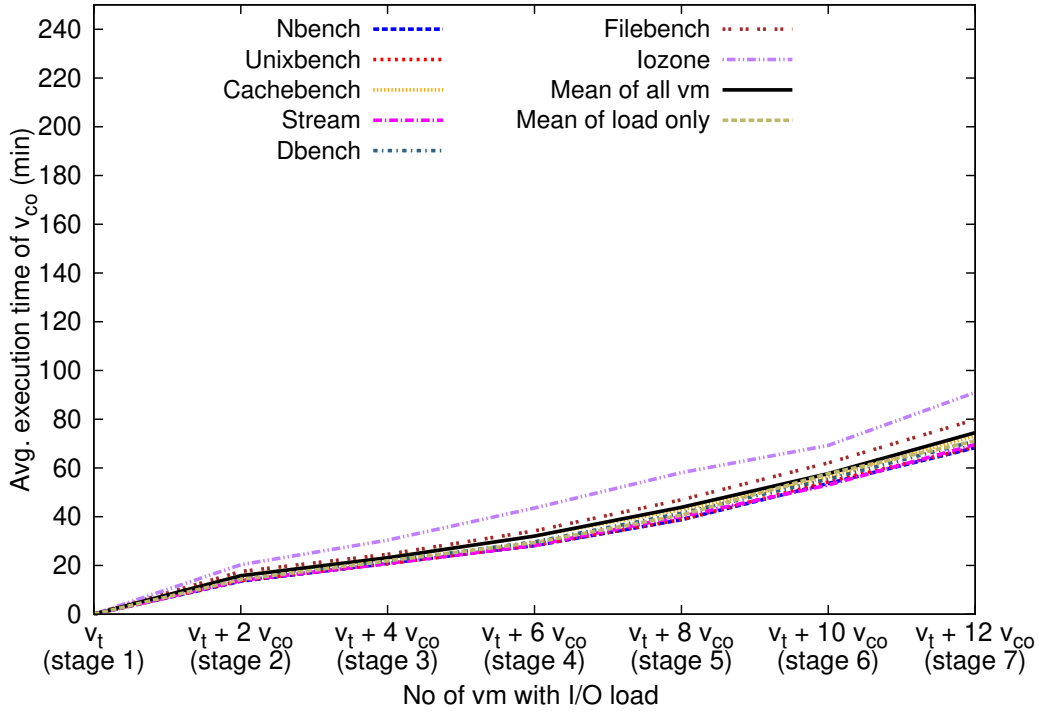


Figure 4.17: Arithmetic mean of task ETV of v_{co} with respect to I/O load changes.

of all the v_{co}^{i} follows a pattern for each resource type. At each stage, the execution finish time of each co-located VM is collected and their arithmetic mean is calculated. They show a typical pattern even though the individual v_{co}^{i} execution time may not have such characteristic indicating that resource usages of all the co-located VMs collectively shapes the performance degradation curve during consolidation.

Figures 4.15, 4.16, 4.17, 4.18, 4.19, and 4.20 show the arithmetic mean of co-located VMs at each stage. The first point of each graph is always zero as there is no v_{co}^{i} running on the host at stage 1 (see Figure 4.6). At stage 2, two v_{co} are running, therefore second point is the arithmetic mean of task execution times of two VMs (v_1, v_2). Similarly, third point is the arithmetic mean of values of four VMs (v_1, v_2, v_3 and v_4) and so on.

As described above, each graph has seven arithmetic mean variation plot of v_{co} for seven different tasks. Furthermore, the arithmetic mean of those seven values is also shown in black in each graph. Arithmetic mean data data is obtained using three basic resources workload, they are CPU (Figure 4.15), memory (Figure 4.16), and I/O (Figure 4.17). The three resource combinational workload used are CPU-Mem (Figure 4.18), Mem-IO (Figure 4.19), and CPU-IO (Figure 4.20).

The increase of different types of workloads causes the arithmetic mean of execution times to change differently. It can be seen that the task ETV is the minimum for CPU intensive load (Figure 4.15) among all three basic types of resources.

Among the three combinations of resources, the CPU-Memory (Figure 4.18) intensive v_{co} combination shows the least amount of performance degradation. On the other hand, the combination of Memory-I/O intensive v_{co} (Figure 4.19) have a debilitating effect on the system as the arithmetic-mean of execution time rises rather quickly compared to other combinations.

One interesting observation made about the v_{co} ETV data presented here is that they show similarities in the pattern unlike the v_t data. Recall that task ETV data from the target VM (v_t) is presented in Section 4.5.

Figure 4.15 shows the arithmetic mean of CPU-intensive co-located VMs for seven tasks. The figure shows that the ETV patterns of all seven graphs are similar to each other. Again, Figure 4.16 shows the arithmetic mean of v_{co} ETV for seven tasks for memory-intensive co-located VMs. Once again, the v_{co} task ETV pattern for all seven tasks are similar. Same thing can be seen in Figure 4.17 for I/O intensive co-located VMs, too. Thus, Figures 4.15, 4.16, and 4.17 show that for each of the three basic

resource intensive co-located VMs the ETV data shows similar patterns.

A similar observation can be made for ETV of co-located VMs with resource combinations, also. For example, Figure 4.18 shows the ETV of co-located VMs due to CPU-memory load combination. At each stage, the task ETV values of the co-located VMs have the matching patterns. Similar characteristics among the ETV values can be also seen for memory-I/O (Figure 4.19) and CPU-I/O (Figure 4.20) load combinations on co-located VMs.

This type of pattern regularity is not visible in all the ETV data of target VM (Section 4.5.2). Recall, Figure 4.10 shows the task ETV of the target VM due to memory intensive co-located VMs. In this case, the Cachebench shows more execution time variation compared to other tasks. On the other hand, the Nbench shows little variation.

Another example is in Figure 4.13. Here, the ETV of target VM is shown for memory and I/O load combination on the co-located VMs. In this case, benchmarks like the Cachebench and IOzone shows much more execution time variation compared to that of Nbench and Unixbench.

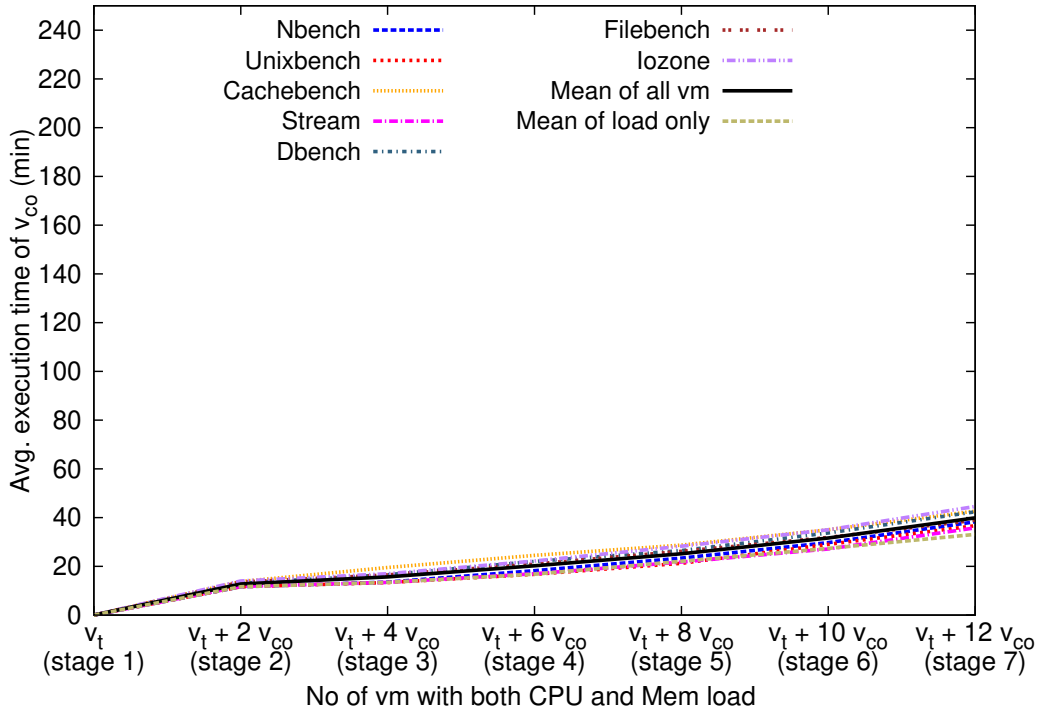


Figure 4.18: Arithmetic mean of task ETV of v_{co} with respect to CPU-Memory load combination changes.

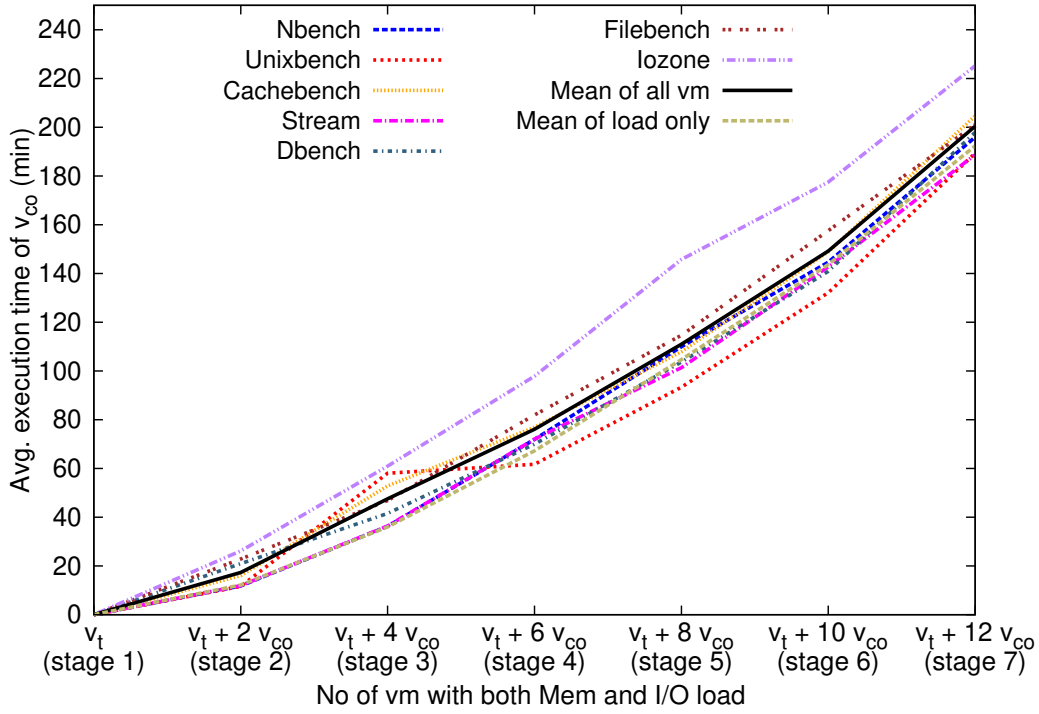


Figure 4.19: Arithmetic mean of task ETV of v_{co} with respect to Mem-I/O load combination changes.

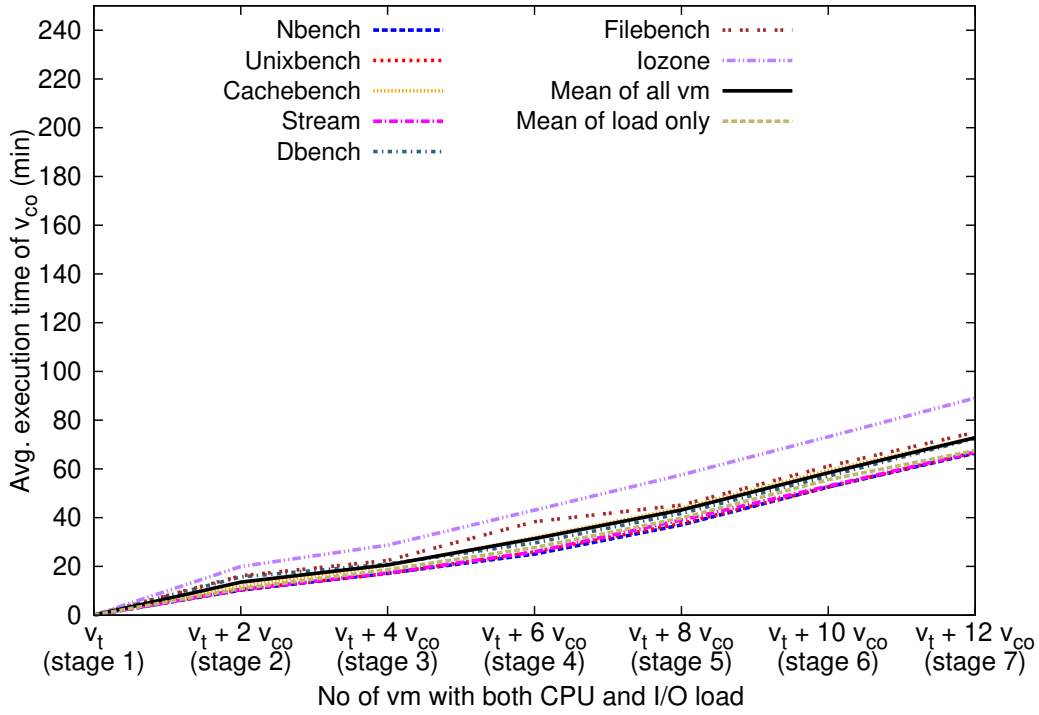


Figure 4.20: Arithmetic mean of task ETV of v_{co} with respect to CPU-I/O load combination changes.

Thus, the ETV data of the target VM do not show regularity like that is observed in the co-located VM data. It should be noted that one set of target VM data show some regularities. Figure 4.9 shows the ETV of seven tasks on v_t due to CPU intensive co-located Vms. As the figure shows, the CPU intensive workload does not cause much execution time variation on any task their ETV pattern looks similar. However, all other resource combinations the ETV pattern of target VM is different as it can be seen from Figures 4.10 to 4.14 and discussed in Section 4.5.2.

This regularity among the co-located VM data can be used as a heuristic for expected performance of consolidated VMs. In the next section, it is shown that this characteristic can be used to predict task ETV of v_t.

Each experiment has been repeated at least three times to obtain the above results. The order of adding VMs has been shuffled, to observe their effect. However, no significant difference between the results has been observed. Two VMs are added at each stage, only to conduct the experiments uniformly. In our observation, the order of adding VMs do not change the results ultimately; rather it depends on the cumulative resource usages of VMs.

4.7 Task execution time variation prediction

This section presents the prediction results for the task ETV due to the combination of resources. In the previous section, task execution time was profiled for three primary resource workloads and three combinational workloads. This section shows that task ETV for combinational workload can be predicted from the ETV due to primary resources. Predictions are separately made from the target VM (v_t) data, and co-located VM data (v_{co}) in Sections 4.7.2, and 4.7.3, respectively. In both cases, the non-linear least square regression (LSR) is used for building prediction models.

Before presenting the prediction results, it is necessary to describe how data are rearranged for training and testing prediction models. The description of the data rearrangement process and examples are given in the next section, and this is done only make the training process clear.

4.7.1 Rearrangement of ETV data for training and testing

Recall that in Figures 4.9, 4.10, 4.11, 4.12, 4.13 and 4.14 (of Section 4.5), data is grouped according to the workload types in co-located VMs. For example, in Figure 4.9 the all seven task ETV data is due to CPU workload in the co-located VMs. Similarly, in Figure 4.10 seven task data is grouped according to the memory workload and so on. It is done to demonstrate how the different types of resource workload influence the task execution time differently. This grouping is essential for studying the effect of each resource type separately.

To properly train the prediction models, the training (both input and target) data needs to be grouped task wise. It is required because during training the task ETV data due to the basic resources of a task needs to correspond to the ETV data due to the combination of resources of the very same task. Otherwise, training data set would not be consistent.

Figure 4.21 shows an example, of this rearranged input and target data set used for training. All the v_t ETV data used here are from the graphs of Section 4.5. It combines the task ETV data of the Cachebench on v_t for three basic (Figures 4.9, 4.10, and 4.11)

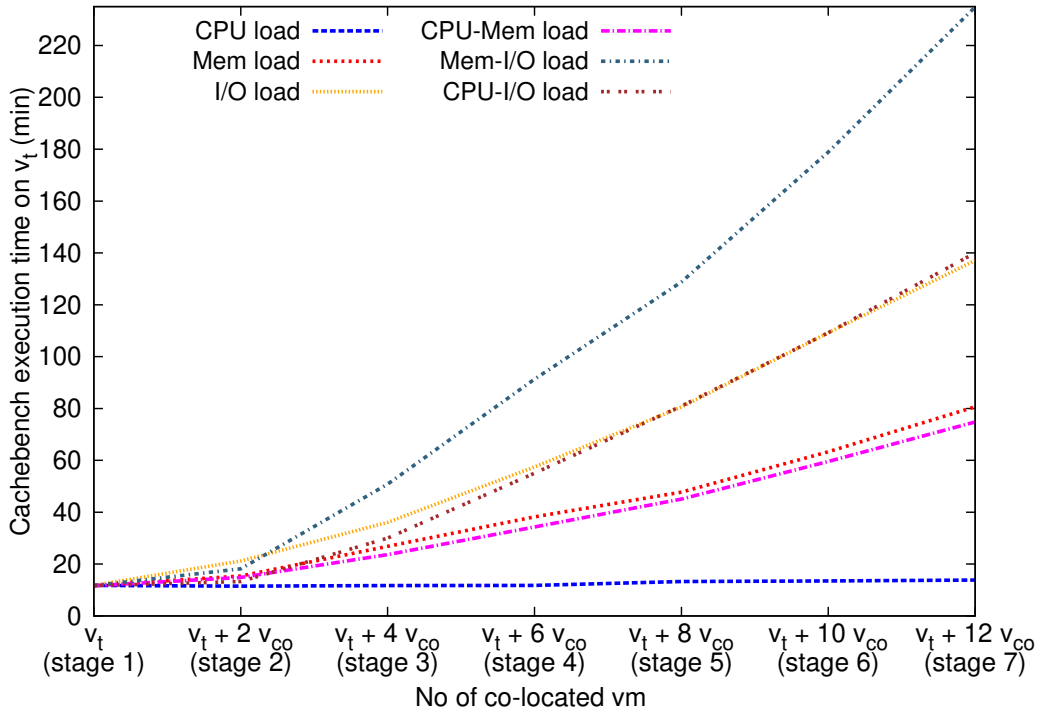


Figure 4.21: Examples of input and target v_t training data: Cachebench.

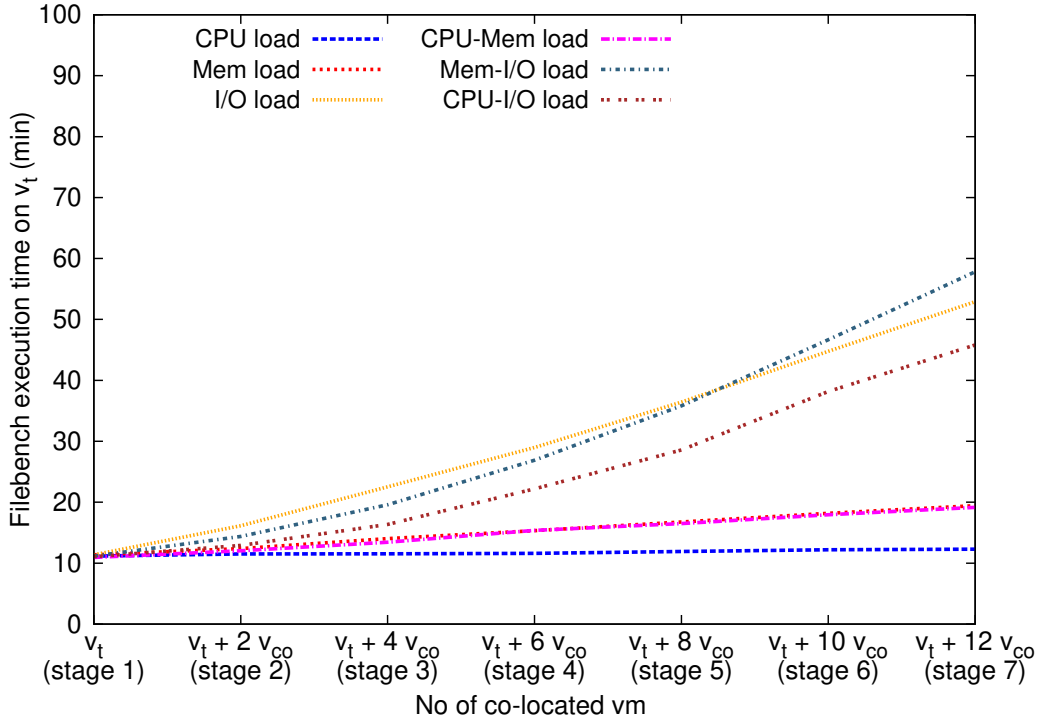


Figure 4.22: Examples of input and target data for both v_t & v_{co} testing: Filebench.

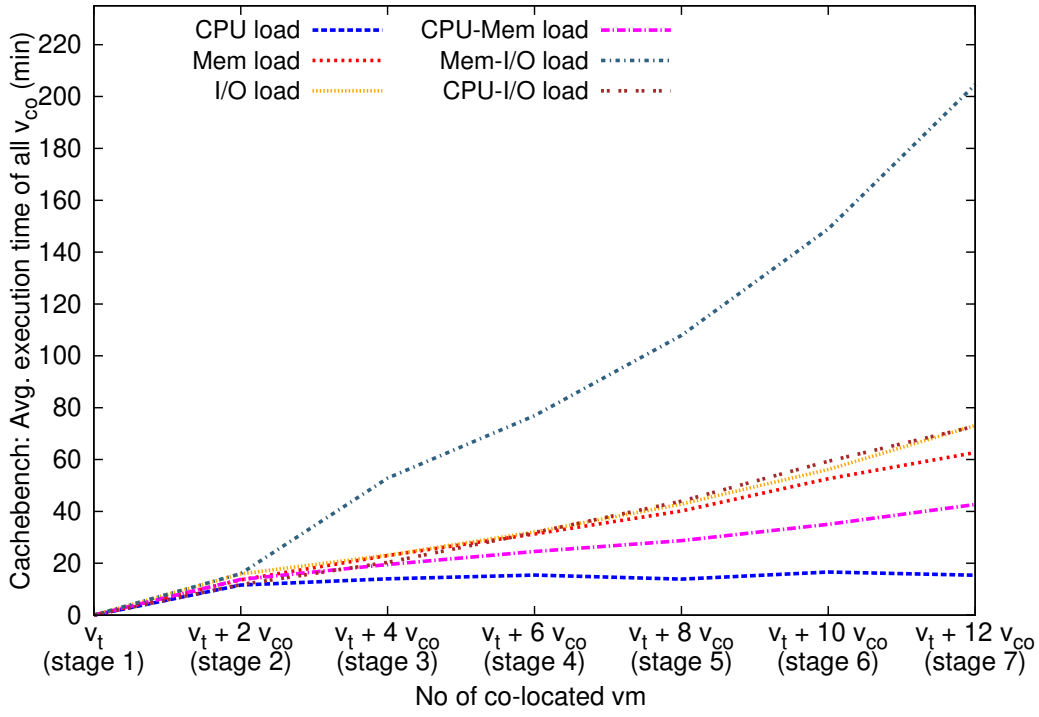


Figure 4.23: Examples of input and target v_{co} training data: Cachebench.

and three combined (Figures 4.12, 4.13 and 4.14) resource loads on v_{co}, respectively. All the v_t task ETV data are rearranged similarly.

During training, the three basic resources data (CPU, Memory and I/O) are used as inputs and three combinations of resources data (CPU-Memory, CPU-I/O, and Memory-I/O) as targets. From the training data set, three sperate models have been generated for three sets of target data (CPU-Memory, CPU-I/O, and Memory-I/O). All three targets use the same three basic resources data as inputs.

An example of test data set is shown in Figure 4.22. It combines six ETV Filebench data from Figures 4.9, 4.10, 4.11, 4.12, 4.13 and 4.14. This data set is used for testing the accuracy of the prediction models. In the same way, rest of the task ETV data are rearranged for training and testing purposes; however, they are not shown here. Prediction results results for models built with v_t data is presented in the next section (Section 4.7.2).

Next, Figure 4.23 shows how the data collected from the co-located VMs (v_{co}) are rearranged for training a separate set of prediction models. In this case, the v_{co} data from the Section 4.6 is reorganized to create test and training data sets. The results for this set of prediction models are presented later in Section 4.7.3.

4.7.2 Target VM (v_t) execution time prediction from the ETV data

Four benchmarks data have been used as tasks for training. They are the Nbench, Unixbench, Cachebench, and Stream. Two other benchmarks have been used for testing; they are the Filebench and Iozone.

All of the test benchmarks have different levels of performance degradation. Therefore, they can help to evaluate the prediction process. Training and testing have been done on different sets of data. No training data have been used for testing and vice versa.

The six graphs of Figures 4.24, 4.25, 4.26, 4.27, 4.28, and 4.29, separately show prediction results for three resources of two test tasks on v_t. Each graph contains two sets of predictions obtained from two separate data sets, which are described next.

The first set of predictions is shown in blue in the figures. In this case, the task ETV data of v_t for basic resource types have been used to predict the task ETV of v_t for the combination of resources. First, the task ETV data of v_t for CPU (Figure 4.9), Memory (Figure 4.10) and I/O (Figure 4.11) intensive v_{co} are recorded, separately.

Those are used as training inputs.

Then, three resource combinations have been used as three separate training targets. They are CPU-Memory (Figure 4.12), CPU-I/O (Figure 4.14) and Memory-I/O (Figure 4.13). From the above training data prediction models are built. The model building process and prediction results are discussed next.

The second set of predictions is shown in red in the figures. In this case, co-located VM data is used for prediction, and the process is mentioned in the later section.

Prediction results for three resources (CPU, Memory and I/O) of two test tasks (Filebench, and Iozone) are shown in Figures 4.24, 4.25, 4.26, 4.27, 4.28, and 4.29. The *Root Mean Square Error (RMSE)* for this set of predictions are shown later in Table 4.6 in Section 4.8.

4.7.3 Target VM (v_t) execution time prediction from the v_{co} data

The second set of prediction models are built by training the LSR models only with v_{co} data. The prediction results are also shown in red on the previously presented Figures 4.24, 4.25, 4.26, 4.27, 4.28, and 4.29. These prediction results demonstrate an interesting aspect of ETV data, the models generated using co-located VM (v_{co}) data

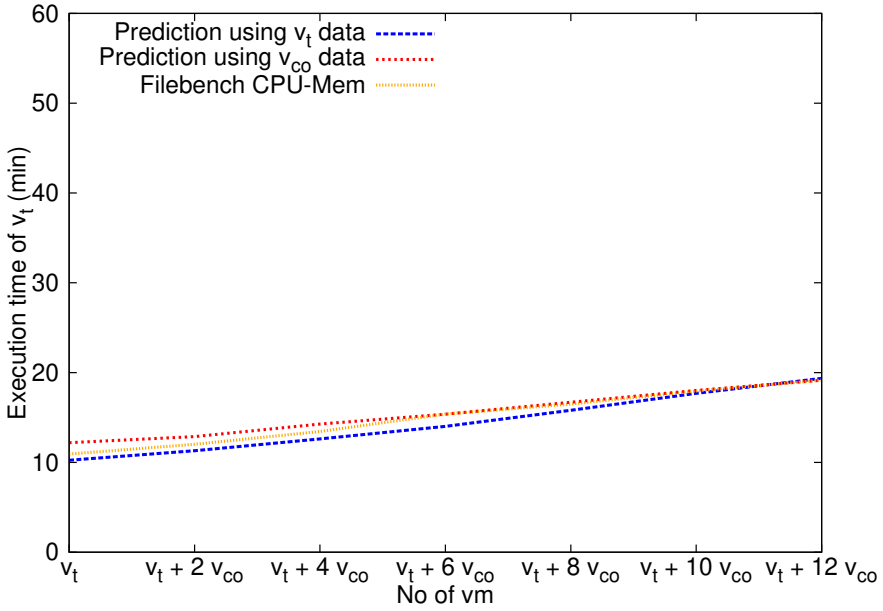


Figure 4.24: Task execution time prediction from v_t and v_{co} data for Filebench: CPU-Mem load.

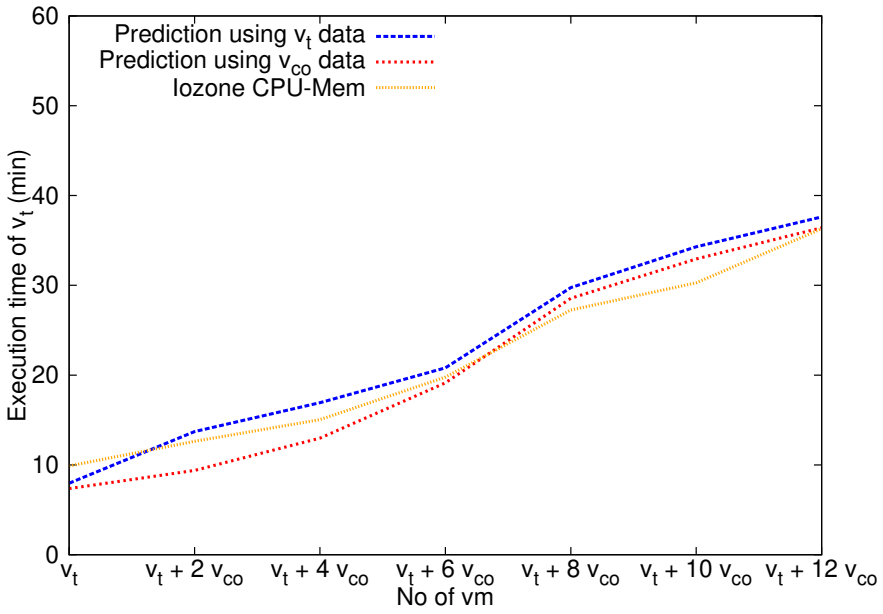


Figure 4.25: Task execution time prediction from v_t and v_{co} data for Iozone: CPU-Mem load.

can predict the ETV of target VM (v_t), too.

All the data used here to train the prediction models are from the figures of Section 4.6. During training the CPU (Figure 4.15), memory (Figure 4.16) and I/O (Figure 4.17) data of v_{co} are used as inputs, while CPU-Mem (Figure 4.18), Mem-I/O (Figure 4.19) and CPU-I/O (Figure 4.20) combination data of v_{co} are used as targets.

On the other hand, during testing, the v_t data are used as both input and target. All testing has been done with the same two tasks (Filebench, and Iozone), which are also used in the previous section for testing the prediction models. That is, in this case, models are generated using only co-located VMs data, while testing is done with target VM data.

An example of the training data used here is shown in Figure 4.23 of Section 4.7.1. It is a collection of six different Cachebench ETV data from six graphs of Figures 4.15, 4.16, 4.17, 4.18, 4.19, 4.20.

During training, the variation of the arithmetic mean of v_{co} ETV due to CPU, memory and I/O are used as inputs. The CPU-Memory, Memory-I/O and CPU-I/O combinations ETV data are used as targets. Three models have been generated in this way, with three v_{co} target data set. However, testing is done with v_t test data like, the example shown in the Figure 4.22. Recall that, it was also used for testing in the

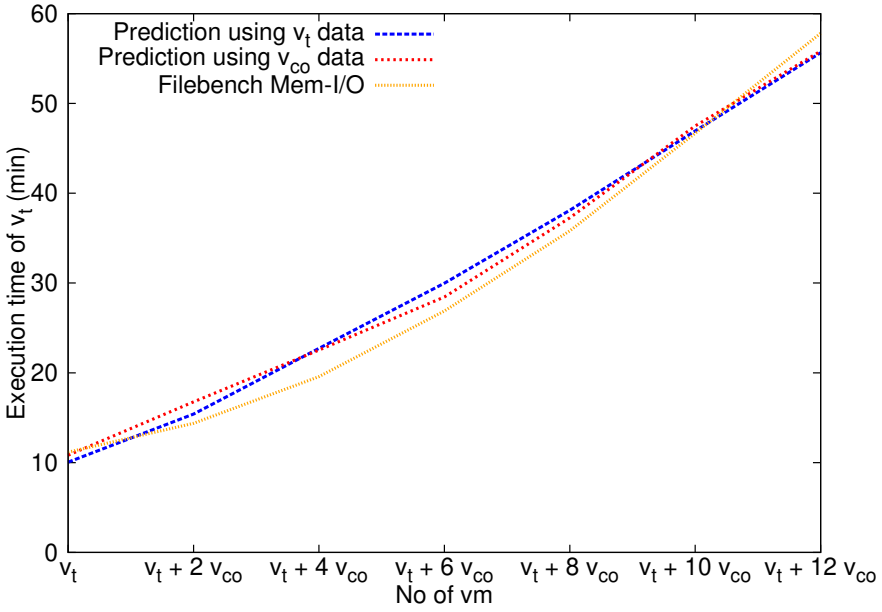


Figure 4.26: Task execution time prediction from v_t and v_{co} data for Filebench: Mem-I/O load.

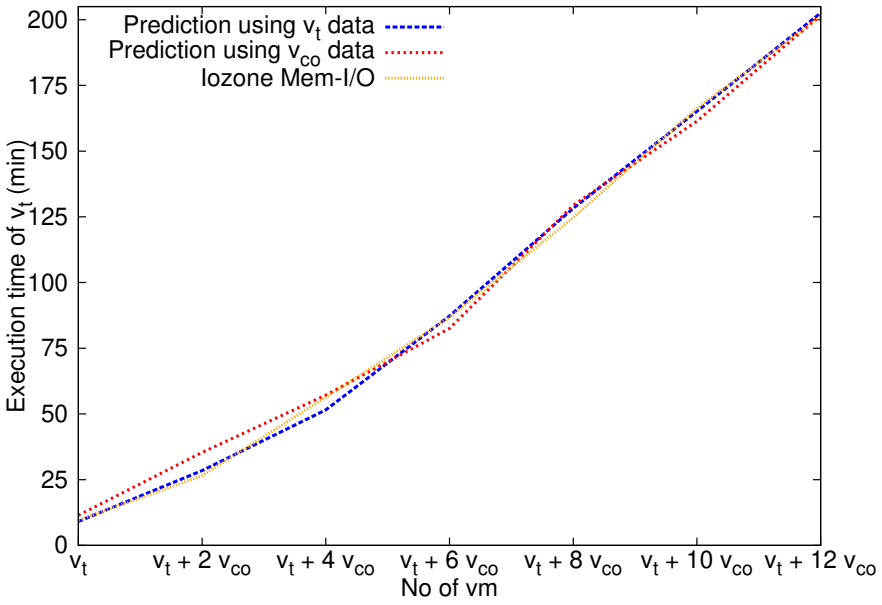


Figure 4.27: Task execution time prediction from v_t and v_{co} data for Iozone: Mem-I/O load.

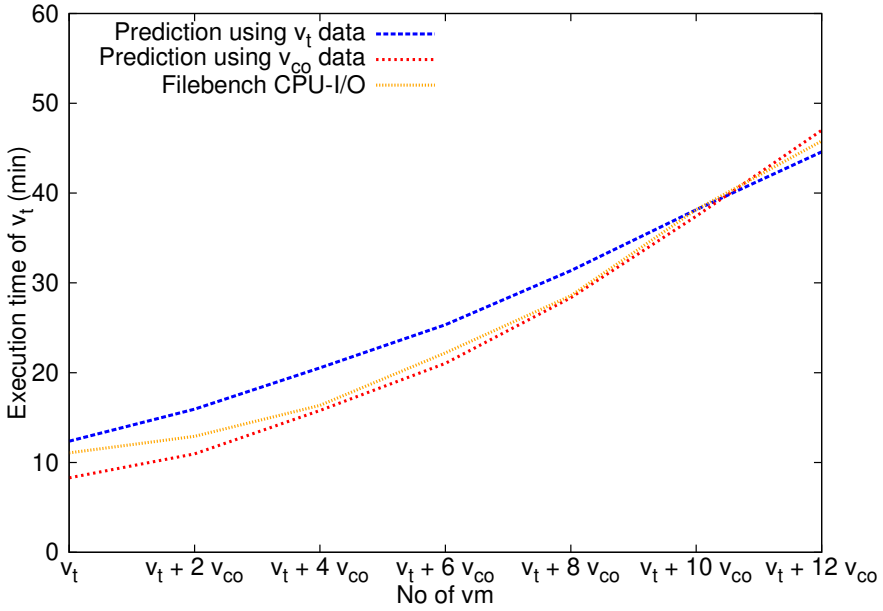


Figure 4.28: Task execution time prediction from v_t and v_{co} data for Filebench: CPU-I/O.

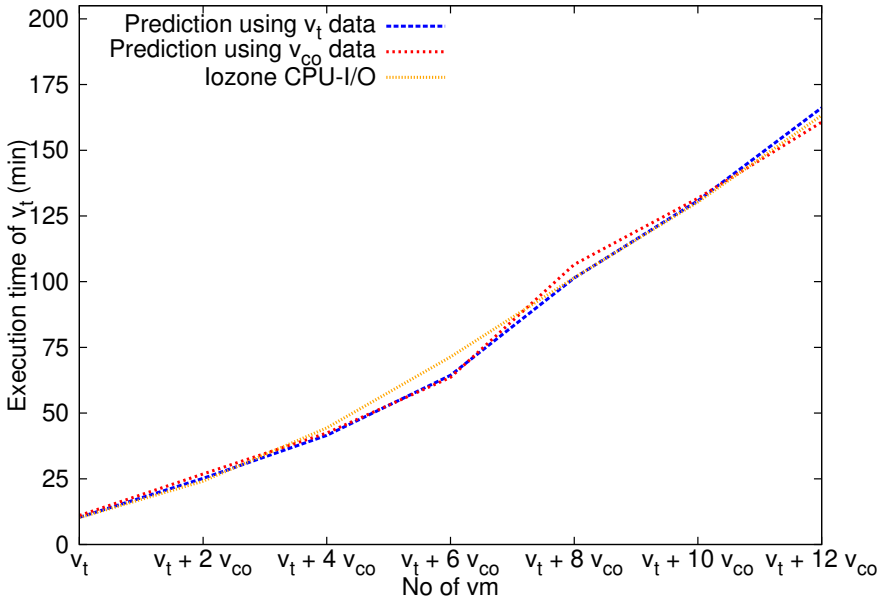


Figure 4.29: Task execution time prediction from v_t and v_{co} data for Iozone: CPU-I/O load.

previous section.

Next, in Section 4.8, RMSE for this set of prediction models is presented in Table 4.7 and discussed.

4.8 The accuracy of prediction models

The Root Mean Square Error (RMSE) is a widely used measure to prediction accuracy of a model [274–278]. It is also commonly used in regression analysis. It measures the differences between the predicted values and the observed values. RMSE calculates the differences between the values using the following formula

$$\sqrt{\overline{(f-o)^2}} \tag{4.1}$$

Where, f is the predicted or forecasted values, and o is the observed or known results. The bar above the squared difference of values signifies the arithmetic mean. Therefore, the formula can be re-written slightly differently to highlight the arithmetic mean of values in the following way [279]:

$$RMSE_{fo} = \sqrt{\frac{\sum_{i=1}^{N}(z_{f_i} - z_{o_i})^2}{N}} \tag{4.2}$$

Where z_f is the standardized forecast value and z_o is the standardized observed value. Values can be standardized using the following formula.

$$z = \frac{X - \mu}{\sigma} \tag{4.3}$$

Where, X is a set of values, μ is the arithmetic mean of the values, and σ is the standard deviation of the values.

The RMSE is the standard deviation of the *residuals*. The residual of an observed value is the difference between the observed value and the forecasted or predicted value of the quantity of interest. Residuals are a measure of how far the actual data points are from the regression line.

The formula for calculating RMSE can also be written as follows [276]:

$$RMSE = \left[n^{-1} \sum_{i=1}^{n} |e_i|^2 \right]^{1/2} \tag{4.4}$$

Where e_i is the absolute forecasting error. Again, e_i can be calculated using following formula [277]:

$$e_i = \left(y_i - f_i^{(m)} \right) \tag{4.5}$$

Where y_i is the observed value at time i. On the other hand, $f_i^{(m)}$ is the predicted value at time i from model m. From now on, the index of the prediction model will be omitted from the equations.

From the absolute forecasting error (e_i) the *Mean Absolute Error* (MAE) can be calculated as follows [277]:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |e_i| \tag{4.6}$$

Then, Mean Square Error (MSE) is calculated using the formula:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} |e_i^2| \tag{4.7}$$

Finally, RMSE is calculated as:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} |e_i^2|} \tag{4.8}$$

The RMSE of predictions for two sets of data mentioned in the previous sections are shown in Tables 4.6 and 4.7, respectively.

4.8.1 Comparison of RMSE for prediction models

Recall that Section 4.7.2 discusses the prediction results for the models built with target VM data. In this case, prediction models are trained with basic resource ETV data. Then, the model is used for predicting the ETV of the target VM due to resource

combinations. Prediction results are shown in the Figures 4.24, 4.25, 4.26, 4.27, 4.28, and 4.29. Actual and predicted ETV data are collected from those graphs and used to calculate the RMSE of predictions in Table 4.6.

Similarly, Section 4.7.3 discusses the ETV prediction results for models built from co-located VM data. The prediction results for this set of data is also shown the Figures 4.24, 4.25, 4.26, 4.27, 4.28, and 4.29. The RMSE values of Table 4.7 is calculated from this set of data. Rest of the sections discusses the significance of RMSE data in detail.

To the best knowledge of the author there exist no approximation algorithm to estimate the task ETV of co-located VMs. Therefore, there is no acceptable theoretical bound for the RMSE values [254]. In general, a lower value of RMSE indicates better prediction accuracy.

One of the observations made during the prediction process is that the number of types of resource data is used in train phase influences on the outcome. In this section, the influences of resource types and number on prediction model accuracy are discussed in detail. Table 4.6 shows the RMSE of prediction results for the models build with v_t data. Recall that, execution time prediction process using target VM (v_t) data is discussed in Section 4.7.2. A color scheme is used in columns of the table to differentiate between two types of prediction results.

In Table 4.6, six columns shows the RMSE of prediction results. The six columns are divided into three groups; each group has two columns. The three groups show the task ETV prediction for three resource combinations; CPU-memory, memory-I/O, and CPU-I/O. Again, each group has two columns of predictions; one prediction is made using three resources, and another prediction is made using two resources.

For example, both the second and third columns of Table 4.6 show the RMSE of prediction for CPU-memory resource combination. However, in two columns the prediction models are built in different ways. For the third column, the prediction model is built using data from two types of resources, which are CPU and memory. Whereas, in the second column the prediction is made using three types of resources; CPU, memory, and I/O. It can be seen that the RMSEs of prediction are better in the second column. The third column is also highlighted in gray to emphasize the difference of prediction accuracy.

Among the two sets of prediction models, the main difference is how many types of resource data are used for training. Other than this, both the second and third column

Table 4.6: Root mean square error (RMSE) for prediction using target vm (v_t) data.

Benchmarks	Resource types used to build prediction model					
	CPU-Memory model		CPU-I/O model		Mem-I/O model	
	Using all three resources (CPU, Mem, I/O)	Using only two resources (CPU, Mem)	Using all three resources (CPU, Mem, I/O)	Using only two resources (CPU, I/O)	Using all three resources (CPU, Mem, I/O)	Using only two resources (Mem, I/O)
Filebench	0.771	2.628	2.131	11.933	2.605	3.022
Iozone	2.193	10.505	2.475	21.566	3.083	3.588

of Table 4.6 shows the prediction for the same resource combination. In the second column, all three primary resources execution time data have been used. During the training phase, ETV due to CPU, memory, and I/O have been used as the model inputs, and CPU-memory ETV data is used as the target. On the other hand, the third column only CPU and memory ETV have been used as the input. The ETV due to I/O workload is not used in this column. As it is visible in the third column, the prediction accuracy is lower than that of column two.

Next, the fourth and fifth columns show the RMSE of prediction for CPU and I/O resource combination. Lastly, both the sixth and seventh column shows the RMSE of prediction for memory and I/O combination. In both cases, prediction results are differentiated using a color scheme in the columns. The results indicate that better prediction result can be achieved by using three resources instead of two. In other words, the RMSE of predictions is better in the column where three resources are used for building the models.

Above paragraphs discuss the significance of using different colors in the columns and show that the RMSE of the predictions are always better when three resource data are used to build prediction models. Next, the prediction accuracies of individual tasks

are discussed. Table 4.6 shows the prediction results for two test tasks, the Filebench, and IOzone. The third row shows the RMSE of prediction for the Filebench, while the fourth row shows RMSE for the IOzone. Again, for each task, the RMSE is lower when three resources are used for training the prediction model.

First, the prediction accuracy of the CPU-memory resource combination for the Filebench is discussed. For the Filebench, the RMSE for prediction is 2.628 when two resources are used to build the model. The RMSE of prediction is only 0.771 when all three resources are used to build the prediction model. Similarly, for other two resource combinations (CPU-I/O, and Memory-I/O) also it can be seen that RMSE values are lower when three types of resource data are used during the prediction model training.

Next, the fifth row of Table 4.6 shows the prediction accuracy data the IOzone. For IOzone, the RMSE of prediction for the model built using two resource data is 10.505. However, when three resources are used in the model, the RMSE drops to 2.193. Recall that a lower RMSE value indicates a better prediction accuracy. The same results can be observed for the RMSE values of the CPU-I/O and Memory-I/O resource combinations, too.

Next, Table 4.7 shows the prediction results for the model trained with co-located

Table 4.7: Root mean square error (RMSE) for prediction using co-located vm (v_{co}) data.

	Resource types used to build prediction model					
	CPU-Memory model		CPU-I/O model		Mem-I/O model	
Benchmarks	Using all three resources (CPU, Mem, I/O)	Using only two resources (CPU, Mem)	Using all three resources (CPU, Mem, I/O)	Using only two resources (CPU, I/O)	Using all three resources (CPU, Mem, I/O)	Using only two resources (Mem, I/O)
Filebench	0.657	4.515	1.841	10.607	1.475	47.797
Iozone	2.083	43.546	4.572	87.725	3.898	59.863

VM (v_{co}) data. In this case, prediction results are shown for the same two tasks (Filebench, and IOzone) have been used; however, v_{co} data is used instead of v_t data. Recall that, prediction model building process using co-located VM (v_{co}) is discussed in Section 4.7.3.

In Table 4.7 also, the prediction accuracy is better when all three resources data are used to generate the model rather than two. For example, both second and third columns of Table 4.7 show the RMSE of prediction for CPU and memory resource combination. It can be seen that the RMSE values of the second column are always lower than that of the third column. In other words, the prediction accuracy for the CPU-Memory load combinations is better when the model is built using CPU, memory, and I/O data rather than using only CPU and memory data. Recall that, similar patterns are also observed in the RMSE values of Table 4.6 and the significance of those values are discussed earlier in the section.

From the discussion of this section, it can be concluded that the execution time prediction accuracy for a model built using all three basic resources data is better compared to that is generated by using only two resource types data.

4.9 Process of obtaining LSR prediction model parameters

This section presents the parameters and coefficient of the prediction models. Recall that Sections 4.7.2 and 4.7.3 presents the execution time prediction results. The accuracy of the prediction is discussed in Section 4.8. Finally, in this section parametric models for the prediction are presented. Those models are shown in equations 4.10, 4.11, 4.12, 4.13, 4.14, and 4.15.

As described in the Sections 4.7.2 and 4.7.3 the models are trained with profiled v_t and v_{co} data, respectively. The profiled ETV data of VMs have been used to train LSR models. Parameters and co-efficient shown in equations 4.10, 4.11, 4.12, 4.13, 4.14, and 4.15 are collected from the trained LSR prediction models models. The equations demonstrate the relation between input and prediction data formats.

The LSR models are built using statistical packages of the R programming language [280, 281]. Specifically, the *lm()* and *Glm()* functions from the R statistical package have been used [282–284]. Those packages and functions are discussed in the

following section.

4.9.1 Least Square Regression (LSR) packages of R

The *lm()* is a useful function for carrying out regression, single stratum analysis of variance, and analysis of covariance [283]. It has the following function interface:

```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE,
   y = FALSE, qr = TRUE,
   singular.ok = TRUE, contrasts = NULL, offset, ... )
```

Parameters of the *lm()* functions are described next. Parameter **formula** is a symbolic description of the model to be fitted. Models has typical form `response ~ terms`. `response` is the numeric response vector. On the other hand, `terms` is a series of terms that specify the predictor for `response`.

Parameter **data** is an optional data frame or a list containing thy variables of the model. Parameter **subset** is an optional vector that specifies a subset of data to be used for the model fitting process. Parameter **weights** is an optional vector of weights used in the fitting process. Parameter **na.action** is a function indicating the course of action if some of the data are unavailable (NA) or missing.

Parameter **method** points to the method to be used for fitting. Parameters **model, x, y, qr** are the logicals that indicate which corresponding components of the fit are to be returned. The components represented by the logicals model, x, y, qr are the model frame, the model matrix, the response, and the QR decomposition, respectively. Parameter **singular.ok** is another logical. It contains FALSE if a singular fit is an error. Parameter **contrasts** is a set of optional values. Parameter **offset** allow to set an offset value for the predictor during the fitting process. Lastly, Parameter (**...**) are additional arguments that can be passed to low-level regression fitting functions.

The *glm()* is used to fit generalized linear models, it has the following function interface [282]:

```
Glm(formula, family = gaussian, data = list(),
    weights = NULL, subset = NULL,
    na.action = na.delete, start = NULL, offset = NULL,
```

```
control = glm.control(...), model = TRUE,
method = "glm.fit", x = FALSE, y = TRUE,
contrasts = NULL, ...)
```

Most of the arguments like **formula**, **data**, **weights**, **subset**, **na.action**, **offset**, **model**, **method**, **x**, **y**, and **contrasts** are already discussed above for the *lm()* function. In addition are there two more parameters present in *glm()* function. They are discussed below.

Parameter **family** points to an error distribution description and link function to be used in the model. **start** points to the staring values of the paraments in the linear predictor. Lastly, **control** has a list of parameters controlling the fitting process.

There is also a *print()* method that helps to print parameters and coefficients for glm models [282]. The interface for this function is given next.

```
print(x, digits=4, coefs=TRUE,
       title='General Linear Model', ...)
```

Parameter **digits** indicates the number of significant digits to be printed. **coefs** is a logical determining whether to print model coefficients or not. For `coefs = TRUE` the function prints all values from table of coefficients, standard errors, etc. **title** allows a character string to be passed to the method.

4.9.2 Formulas for lm() and glm() functions

For both *lm()* and *glm()* functions, formulas or models can be specified symbolically. Experiments were done with different terms and formulas. A grid search was conducted on the parameter space to find the best parameter values. Next, it is going to be discussed how the parameters are selected and coefficients are obtained for the equations of this section.

Different prediction models have been used for different resource combinations. It can be seen that the equations 4.10, 4.11, and 4.12 presents prediction models for CPU-memory, CPU-I/O, and memory-I/O resource combinations, respectively. For example, equation 4.10 shows the prediction results for the CPU and memory resource combination. First experiments have been done with various formulas. Formulas were

chosen based on their prediction accuracy on the test data set. For the CPU-memory resource combination following formula was chosen based on the RMSE value of prediction.

```
c_m ~ (I(c^0.7)+I(m^0.9)+I(io))^2
```

In the above formula, `c`, `m`, and `io` are three vectors containing the target VM data; they contain the ETV data due to CPU, memory, and I/O intensive co-located VMs, respectively. Those data are presented in the graphs of Figures 4.9, 4.10 and 4.11 and discussed in Section 4.5. On the other hand, `c_m` is the prediction for CPU and memory resource combination. The prediction results are previously shown in Section 4.22.

Above formula, which is compatible with the R programming language can be expanded into an equation as follows [285, 286]:

$$\begin{aligned} c\_m_i = \alpha_1 + \alpha_2 * c_i^{0.7} + \alpha_3 * m_i^{0.9} + \alpha_4 * io_i \\ + \alpha_5 * c_i^{0.7} * m_i^{0.9} + \alpha_6 * c_i^{0.7} * io_i + \alpha_7 * m_i^{0.9} * io_i \end{aligned} \tag{4.9}$$

Where, $c_i \in$ `c`, $m_i \in$ `m`, and $io_i \in$ `io` are members of the vectors containing ETV data of CPU, memory, and I/O, respectively. On the other hand, α_1, α_2, ... α_7 are model coefficients, which needs to be determined from the trained model. Lastly, $c\_m_i \in$ `c_m` is the predicted execution time for a particular CPU and memory combination.

4.9.3 Process of obtaining coefficient estimate

The function *lm()* returns an object of class "lm", which is a list containing many components like *coefficients*, *residuals*, *fitted.values*, *rank*, *weights*, *df.residual*, *terms*, etc. Once the model is fitted, function maned *summary()* can be called to obtain and print a summary and analysis of variance table of the results [287–289].

```
print(summary(fit))
```

Where `fit` is an object of lm class returned from the *lm()* function after fitting process. An output of this function call is shown in Figure 4.30.

```
Call:
lm(formula = c_m ~ (I(c^0.7) + I(m^0.9) + I(io))^2,
                                              data = data2)

Residuals:
    Min      1Q  Median      3Q     Max
-2.3308 -0.4663 -0.1265  0.2751  2.7262

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)       -21.130543   4.883250  -4.327 0.000297
                                                      ***
I(c^0.7)            4.963360   0.842037   5.894 7.51e-06
                                                      ***
I(m^0.9)            0.752094   0.336192   2.237 0.036254
                                                        *
I(io)               0.600729   0.368986   1.628 0.118425
I(c^0.7):I(m^0.9)  -0.124518   0.051755  -2.406 0.025425
                                                        *
I(c^0.7):I(io)     -0.037475   0.066595  -0.563 0.579577
I(m^0.9):I(io)      0.002292   0.001269   1.807 0.085143
                                                        .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1
                                                      ‘ ’ 1

Residual standard error: 1.114 on 21 degrees of freedom
Multiple R-squared:  0.9961,    Adjusted R-squared:
                                                    0.9949
F-statistic: 886.9 on 6 and 21 DF,  p-value: < 2.2e-16
```

Figure 4.30: An output from *lm()* function.

At the top of the result, the formula used in the function is printed. Next statistics for residuals are shown. Residuals are the difference between the actual response values and the response values from the prediction.

Next, the coefficients of the model are printed. In regression analysis, the coefficients are unknown constants that represents the *intercept* and *slope* of the linear model [287]. The estimates of coefficients of the model formula are obtained during the fitting process with the training dataset.

The first column below the 'coefficients', prints all the terms relevant to the model formula. The second column prints their estimated values. The coefficient standard errors are printed in the third column.

The coefficient *t-value* column is a measure of how many standard deviations of the coefficient estimates are further away from zero (0). Far away values from zero indicate that the corresponding coefficient represents an essential relationship in the model. Last column, the Coefficient $Pr(>|t|)$ represents the probability of observing any value equal or larger than *t*. Residual Standard Error is a measure of the quality of the regression model fitting process. Rest of the statistics show how well the model fits the actual data.

Thus, the general prediction model for the CPU-memory combination is shown in equation 4.9. After fitting the model with R, coefficient estimates are obtained from the *lm* object. Finally, all coefficients and parameters are put in the formula to obtain the Equation 4.10.

In the same way, prediction models for other resources combinations are also obtained. Equations 4.11, and 4.12 show the prediction models CPU-I/O and memory-I/O resource combinations, respectively.

4.9.4 Parametric models

In previous three sections (Sections 4.9.1, 4.9.2, and 4.9.3) process of building prediction models with R statistical packages is described. The sections describe how the formulas and coefficients can be obtained from the R statistical functions. However, the functions provide output formulas in a minimal form, which is most suitable for people with technical know-how. In this section, the formulas are expanded and presented in a more human-readable form.

Equations 4.10, 4.11, and 4.12 show the three prediction models built with target VM data as described throughout the previous sections. The terms used in the Equations 4.10, 4.11 and 4.12 are self-explanatory; nonetheless, they are listed in Table 4.8 for convenience. For example, $T^t_{cpu-mem,i}$ denotes predicted task execution time of a task on v_t, when it is consolidated with the i number of CPU and memory intensive co-located VMs. Similarly, $T^t_{cpu-io,i}$ and $T^t_{mem-io,i}$ represent the predicted task execution time for CPU-I/O and Memory-I/O load combinations, respectively.

As explained in the Section 4.7, the same input parameters have been used for

all three equations. $T^t_{cpu,i}$ represents the task execution time of v_t when the server is consolidated with the i number of CPU intensive co-located VMs. Similarly, $T^t_{mem,i}$ and $T^t_{io,i}$ represent the task execution time when the server is consolidated with the same number of memory and I/O intensive co-located VMs, respectively.

Next, Equations 4.13, 4.14, and 4.15 present the prediction models built from co-located VM data. Recall that the prediction results of models built using the co-located VM data is described in Section 4.7.3, and the process of obtaining model parameters and coefficients are discussed in Sections 4.9.1, 4.9.2, and 4.9.3 above.

Table 4.8: Parameters of the equations 4.10, 4.11, 4.12, 4.13, 4.14, and 4.15.

$T^t_{cpu-mem,i}$	Expected execution time of a task on target VM. The target VM is consolidated with total of i co-located VMs. The co-located VMs are running a CPU and memory intensive load combination.
$T^t_{cpu-io,i}$	Expected execution time of a task on target VM. The target VM is consolidated with total of i co-located VMs. The co-located VMs are running a CPU and I/O intensive load combination.
$T^t_{mem-io,i}$	Expected execution time of a task on target VM. The target VM is consolidated with total of i co-located VMs. The co-located VMs are running a memory and I/O intensive load combination.
$T^t_{cpu,i}$	Observed task execution on target VM. The target VM is consolidated with the i number of CPU-intensive co-located VMs.
$T^t_{mem,i}$	Observed task execution on target VM. The target VM is consolidated with the i number of memory-intensive co-located VMs.
$T^t_{io,i}$	Observed task execution on target VM. The target VM is consolidated with the i number of I/O-intensive co-located VMs.
$T^{co}_{cpu,i}$	Arithmetic mean of task execution times of i number of co-located VMs, all running CPU intensive benchmarks.
$T^{co}_{mem,i}$	Arithmetic mean of task execution times of i number of co-located VMs, all running memory intensive benchmarks.
$T^{co}_{io,i}$	Arithmetic mean of task execution times of i number of co-located VMs, all running I/O intensive benchmarks.

$$T^t_{cpu-mem,i} = -20.415 + 4.795 * (T^t_{cpu,i})^{0.7} + 0.752 * (T^t_{mem,i})^{0.9} + 0.579 * (T^t_{io,i}) - 0.119 * (T^t_{cpu,i})^{0.7} * (T^t_{mem,i})^{0.9} - 0.035 * (T^t_{cpu,i})^{0.7} * (T^t_{io,i}) + 0.002 * (T^t_{mem,i})^{0.9} * (T^t_{io,i}) \quad (4.10)$$

$$T^t_{cpu-io,i} = -16.393527 + 0.802 * (T^t_{cpu,i}) + 0.282 * (T^t_{mem,i}) + 1.769 * (T^t_{io,i}) - 0.020 * (T^t_{cpu,i}) * (T^t_{mem,i}) - 0.023 * (T^t_{cpu,i}) * (T^t_{io,i}) + 0.003 * (T^t_{mem,i}) * (T^t_{io,i}) \quad (4.11)$$

$$T^t_{mem-io,i} = -16.549 + 2.104 * (T^t_{cpu,i}) + 7.920 * (T^t_{mem,i})^{0.2} - 0.169 * (T^t_{io,i})^{0.96} - 0.957 * (T^t_{cpu,i}) * (T^t_{mem,i})^{0.2} + 0.031 * (T^t_{cpu,i}) * (T^t_{io,i})^{0.96} + 0.455 * (T^t_{mem,i})^{0.2} * (T^t_{io,i})^{0.96} \quad (4.12)$$

The models in Equations 4.13, 4.14, and 4.15 have the same targets as that of Equations 4.10, 4.11, and 4.12, respectively. However, inputs of the first three set of equations (4.10, 4.11, and 4.12) are different than that of second set of three equations (4.13, 4.14, and 4.15).

As explained in Section 4.7.3, in this case, the arithmetic mean of execution times

of co-located VMs has been used as inputs. For example, $T^{co}_{cpu,i}$ represents the arithmetic mean of execution times of the i number of CPU intensive co-located VMs. In this case, the arithmetic mean of execution times of co-located VMs has been used to predict the task ETV of target VM.

Coefficients for these three models are obtained using the same procedure as described earlier in the section. First, a suitable formula for the prediction models is obtained from the co-located VM data through experiments. Then, the model coefficients are estimated from the trained models. Meanings of the terms used in the equations are also explained in Table 4.8 as well.

$$T^{t}_{cpu-mem,i} = -0.009 + 0.190 * (T^{co}_{cpu,i})^{1.2} + 0.220 * (T^{co}_{mem,i})^{1.2} + 0.250 * (T^{co}_{io,i}) \tag{4.13}$$

$$T^{t}_{cpu-io,i} = -0.102 + 4.217 * (T^{co}_{cpu,i})^{3.25} - 0.034 * (T^{co}_{mem,i}) + 1.383 * (T^{co}_{io,i})^{0.7} + 0.544 * (T^{co}_{cpu,i})^{3.25} * (T^{co}_{mem,i}) - 5.394 * (T^{co}_{cpu,i})^{3.25} * (T^{co}_{io,i})^{0.7} + 0.163 * (T^{co}_{mem,i}) * (T^{co}_{io,i})^{0.7} \tag{4.14}$$

$$T^{t}_{mem-io,i} = 3.720 * (T^{co}_{cpu,i})^{0.325} + 2.376 * (T^{co}_{mem,i}) - 0.020 * (T^{co}_{io,i})^{0.05} + 1.079 * (T^{co}_{cpu,i})^{0.325} * (T^{co}_{mem,i}) - 4.394 * (T^{co}_{cpu,i})^{0.325} * (T^{co}_{io,i})^{0.05} - 3.144 * (T^{co}_{mem,i}) * (T^{co}_{io,i})^{0.05} \tag{4.15}$$

Changes in the basic configuration of VMs may require some modification of the model coefficients. The public Cloud offers VMs for renting in a certain number of predefined configurations. Therefore, some such models required in reality are limited. What is more, since LSR has low overhead building or rebuilding of a limited number of such models would not be time-consuming.

4.10 Conclusion

This chapter addresses the critical issue of profiling and predicting the performance variations of VMs due to consolidation. This chapter is built upon the works of the previous chapter. Based on the experimental results and observations of the Chapter 3, A straightforward methodology has been introduced here that is practically feasible to implement.

Work presented in this chapter is different from most complementary works on virtualization, which uses simulation to derive results. All results and data shown are collected from real system data. This work introduces a methodology to profile the task ETV of consolidated VMs, called the Incremental Consolidation Benchmarking Method (ICBM).

Several combinations of benchmark suites have been used to profile and predict the execution time variation (ETV) of tasks due to consolidation. The ICBM presented in this chapter uses a step-by-step VM profiling process. At each step, a selected set of VMs is run to create the desired amount of resource contention. The methodology introduced here for consolidated VMs is unique and useful. Experimental results from real virtualized systems show that in this way it is possible to predict the task ETV of VMs quite accurately.

The consolidated VMs are divided into two categories for the conducting the experiments. At each stage of the ICBM, there can be one target VM (v_t) and zero or more co-located VMs (v_{co}). The v_t is used to measure task execution time variation due to consolidation. On the other hand, v_{co} are used to create resource contention on the server.

First task execution times are profiled for VM consolidation and then the data this used to train a prediction model. Two sets of data are used to build two sets of prediction models; one collected from the target VM and another from the co-located

VMs. Furthermore, model training and prediction are done entirely different sets of task execution time data. Models built from both sets of data show a good accuracy of prediction. Predictions are made for three combinations of resources; they are CPU-memory, CPU-I/O, and memory-I/O.

One set of prediction results are obtained using the target VM (v_t) data. Two tasks are used to test the prediction accuracy of the models. For the Filebench, the RMSE for prediction for CPU-memory, CPU-I/O, and memory-I/O resource combinations are 0.771, 2.131, and 2.605, respectively. For the IOzone the RMSE of prediction for the same three combinations are 2.193, 2.475, and 3.083, respectively.

Another set of prediction results are obtained using the co-located VMs (v_{co}) data. Predictions are made for the three same set of resource combinations (CPU-memory, CPU-I/O, and memory-I/O). In this, case the RMSE of prediction for the Filebench are 0.657, 1.841, and 1.475, respectively. For the IOzone the RMSE values are 2.083, 4.572, and 3.898, respectively.

A lower value of RMSE generally means a better prediction accuracy. This chapter also presents parametric models for the prediction. The parameters and coefficient of the model are collected during the experiments.

This chapter provides a quantitative way to compare the mutual performance interference of co-located VMs. The results also provide some valuable inside into the nature of resource contention due to the consolidation of VMs involving various types of system resources.

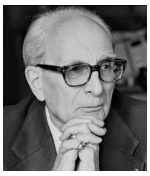
The experimental results encourage one to continue working in this direction. In the next chapters, the experiments are extended to a broader range of virtualization techniques and server configurations to derive a more generalized model.

Chapter 5

An Automated Lightweight Framework for Scheduling and Profiling Parallel Workflows

"The scientist is not a person who gives the right answers, he is one who asks the right questions."

— Claude Lévi-Strauss* (1908–2009)



5.1 Introduction

This work presents a lightweight framework for performing automated experiments with the execution time and performance variations of parallel workflows on consolidated *Virtual Machines* (VMs). As discussed in the previous chapter (Chapter 4), the task ETV due to consolidation is an obstacle to efficiently scheduling workflows on VMs. In data centers, VMs are usually consolidated to increase resource utilization. However, this causes resource contention and performance degradation among the VMs.

Previously, Chapter 4 showed the application of the *Incremental Consolidation*

*Image source: `https://media.gettyimages.com/photos/claude-levistrauss-picture-id952436610`

**The contents of this chapter were published as [44]. Author of the dissertation has conducted all the experiments, analyzed the data and prepared the drafted.

Benchmarking Method (ICBM) on the tasks running on VMs. In Chapter 4, the experiments are conducted with the *Xen* hypervisor only. In this chapter, the ICBM is extended to parallel workflows. A parallel workflow consists of multiple tasks with inter-decencies. Experiments are also extended to three hypervisors; they are *ESXi*, *XenServer*, and Xen. Furthermore, a framework is introduced to perform experiments with the consolidated VM performance. Those topics are further described throughout the chapter.

To address the VM consolidation performance issues; it is necessary to perform experiments with large sets of tasks and VMs. The interaction among the VMs is explored in many recent works to improve the performance of consolidated VMs and applications [290–296]. The proposed framework makes it easy to conduct experiments with large numbers of task execution patterns. It is capable of profiling the execution time variation of each task of a workflow.

The design principles, implementation issues, and tradeoffs of the framework are discussed in detail in this chapter. The effectiveness of the framework is demonstrated with a data-intensive scientific workflow. This workflow process the *Galactic Arecibo L-band Feed Array HI* (GALFA-HI) survey data [297] with the *Montage* toolkit [298]. With this framework, the experiments are simultaneously conducted on three different hypervisors and the execution time variation of each task is retrieved.

The three hypervisors used here are the VMware ESXi 5.5, XenServer 6.5, and Xen 4.6. This framework will enable researchers to perform large-scale experiments with the execution time variations of parallel tasks on multiple hypervisors and the Cloud.

Virtualization plays a vital role in both the data centers and Cloud. Among other advantages, it allows consolidation of VMs in data centers. The consolidation process of running multiple VMs simultaneously on the same server. It is a conventional technique to increase the resource utilization, reducing operational cost and energy consumption. However, the main drawback of consolidation is performance variation due to resource contention and interferences among the VMs.

More and more applications and workflows are being deployed on the Cloud. However, scheduling of scientific applications and workflows on the Cloud is still inefficient because of the task execution time variation. On consolidated servers, the task execution finish time may very unexpectedly; thus, it is difficult to determine which applications should be consolidated for the better performance [299, 300]. Recently, many

works have focused on this issue [292–296, 301, 302].

The works on VM consolidation, rely on the experimental results of how the consolidated applications react to the resource contention. They usually require running a vast number of experiments with various applications and workflows. This chapter proposes a framework to efficiently manage and conduct experiments with a large number of VM schedules and resource usage patterns. The VMs schedules are designed to make them as general as possible and as close to real-world workload as possible. Great care has been taken to design the experimental stage to make them cover a wide range of experimental situation. This chapter is a continuation of the work of the previous chapter (Chapter 4) and depends on the concepts and methodology introduced previously.

As discussed previously in the Section 4.3.1, using execution time as a performance metric it is possible to cover a wide range of scenarios with one combination of VMs. Furthermore, as discussed in the Section 4.3.2, a systematic increase of the number of VMs in the system can help to profile VMs for a wide variety of workload patterns.

There are many Cloud management and maintenance software available today [39, 303–313]. Although they are designed for performing maintenance, fault tolerance, and data backup services; they are not the best suited for performing experiments with task scheduling and resource usage patterns on VMs for several reasons:

a) The software stacks are primarily designed for providing the Cloud services not for performing experiments with VM workloads. For example, they have unique features for providing fault tolerance, VM replication, migration and high availability of VMs to a data center. The software stacks do not offer any built-in features for performing experiments with application scheduling patterns on the Cloud;

b) The software has many modules, and they require a lot of time and effort to master. System administrators require much experience to manage these systems efficiently. On the other hand, most researchers are concerned with a quick and easy setup of experiments. There is an open source Cloud management software. However, it takes a lot of time and effort to modify a large piece of software to implement special interfaces to conduct scientific experiments. Making such changes to a massive software stack with many modules can be a cumbersome process.

c) Cloud management software has many layers and hides many complexities of execution from the users. That convenient for the system administrators, who are primarily concerned with the maintenance. However, during the experiments measuring the performance impact of each task execution is necessary. The experiments are required to understand the VM consolidation behavior and identify any anomaly quickly.

The proposed framework is designed to bypass the interaction with management software and run task scheduling experiments directly on the VMs. The framework has a simple construct; it can execute parallel workflows on VMs of multiple servers. Some features of the framework and contribution of this chapter are briefly stated below:

i) A lightweight framework for profiling the execution time variations of parallel workflow on the Cloud is presented. It provides a simple interface for conducting experiments on VMs and scheduling parallel workflows across on multiple hypervisors. The primary objective is to provide an accessible platform to carry out experiments on the Cloud. It can be used independent of any data center management software, thus making the general experimental process easier.

ii) This framework is lightweight and easy to handle; no specialized knowledge is required.

iii) The framework allows researchers to specify a sequence of workload execution pattern for the VMs. A human-readable *workload descriptor* file stores all the task execution patterns. The exact sequence of tasks that are to be executed is defined in the file.

iv) Another feature of the framework is the *command descriptor* file. Parallel applications usually consist of several smaller tasks and a set of commands may be required to run each task. The command descriptor file contains the actual commands, and one mnemonic is issued against each set of commands. Thus, the workload descriptor file remains small, and workload patterns are easy to create and modify.

The command descriptor file also allows running complex applications like web servers or database servers like any other task. The framework scans the workload file twice. During the first scan all the mnemonics are replaced and in the

second scan actual commands are executed. Thus, adding or modifying real command sets is much easier as they are stored only in one place, in the command descriptor file;

v) The framework can simultaneously conduct experiments with workflow schedules and resource usage patterns on multiple hypervisors. It uses the *Secure Shell* (SSH) to connect to virtualized servers instead of the API. The use of SSH increases flexibility, and any hypervisors can be connected using just SSH. On the other hand, coding in multiple APIs for multiple hypervisors is a cumbersome process. The SSH gives the ability to connect to any Cloud;

vi) The framework is implemented entirely in Java and can run on any *Operating System* (OS). It can be used as a stand-alone application or plugged-in with any other Java task scheduling program. It is lightweight, completely portable and requires no installation.

To the best of the author, there is no other framework written in any language to do experiments with execution time variation of a large number of parallel workflows and VM combinations. To the best knowledge of the author, this framework is the first of its kind. There is no other framework designed and implemented before to conduct experiments with task execution time variation of a wide variety of VM consolidation.

This framework is both independent of and complementary to Cloud management software. While the management software can be used for providing Cloud services; this framework can be used to run experiments with workload patterns on the Cloud. It is a lightweight framework designed and implemented in Java to conduct experiments with virtualized systems. This framework is designed to manage large combinations of VMs and schedule often required while conducting experiments with virtualized servers.

The effectiveness of the framework is demonstrated with a real data-intensive workflow, which processes the GALFA-HI [314] survey data with the Montage toolkit [315].

The *Incremental Consolidation Benchmarking Method* (ICBM) [43] has been used to analyze the tasks of the workflow. Initially, the ICBM was introduced to analyze the execution time variations of individual tasks on VMs. In this chapter, it is extended to analyze the tasks of the scientific workflow.

The rest of the chapter is organized as follows. Section 5.2 describes the problem considered in this chapter with an example. Design goals are discussed in Section 5.4, followed by the framework design in Section 5.5. Section 5.7 discusses the workflow and benchmarks used along with experimental setup. Section 5.8 gives the results of experiments with task execution patterns on three hypervisors. Section 5.9 provides a brief overview and shortcomings of complementary works. A discussion about future work and conclusion are in Section 5.10.

5.2 Scheduling workflows and consolidation

The task execution time variation due to VM consolidation is one of the major problems for the Cloud. It can be even more troublesome for parallel applications and scientific workflows because of task dependencies.

Figure 5.1 shows an example of workflow, which processes the GALFA-HI survey data [314] using the Montage toolkit [298, 315]. It is a data-intensive workflow that creates a mosaic image of a part of the Milky Way galaxy from some data cubes. The survey employs the 1,000-foot (305-meter) radio telescope of the Arecibo Observatory in Puerto Rico. The radio telescope was the largest single-aperture telescope from its completion in 1963 until 2016. For the survey, first, the sky is divided into grids. Then, the radio telescope makes detail scans of the sky and provides data in the form of radio wave. Thus, separate data cubes are generated from the separate parts of the sky. The data generated in this way needs to be compiled and merged to create a visible mosaic image of the sky. The GALFA-HI workflow presents the process of compile-and-create the mosaic image.

The data cubes are released at regular intervals as a part of the ongoing survey. Such workflows are a widely used in the field of astronomy. The workflow itself has 16 tasks (t_1 to t_16) on 8 levels (l_1 to l_8). Figure 5.2 shows one schedule of these tasks on a set of co-located VMs.

In Figure 5.2, the tasks of the GALFA-HI workflow (Figure 5.1) are scheduled on the VMs of a single server. Here, the server has eight simultaneously running VMs. As the tasks of the workflow have internal dependencies, they need to be scheduled hierarchically. The tasks that can be run simultaneously are grouped in one level. The tasks of the level below are dependent on tasks of the immediately upper level.

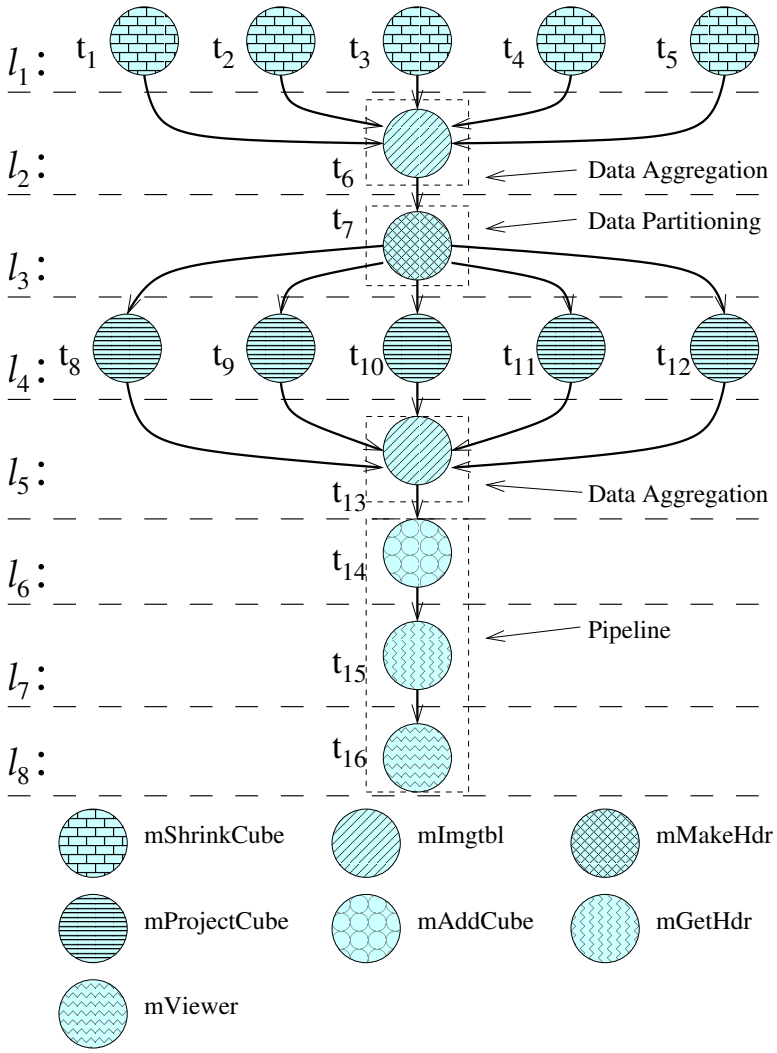


Figure 5.1: A workflow: GALFA-HI data processing with the Montage toolkit.

Figure 5.2 depicts that the tasks are being executed level by level on the VMs of a single server. There are VMs of three colors on the server. Light blue VMs are where the tasks of GALFA-HI are being executed. In a consolidated server, tasks from other applications are also being executed; they are shown in red. Finally, white VMs represent empty VMs, where no tasks are being run at present.

The tasks on additional VMs (shown in red) are responsible for creating resource contention and performance degradation of tasks of GALFA-HI workflow. In this case, performance deterioration of a task can have a cascading effect on the other tasks of the workflow because of the task dependencies [176]. Furthermore, the performance of tasks of the critical path would directly affect the makespan.

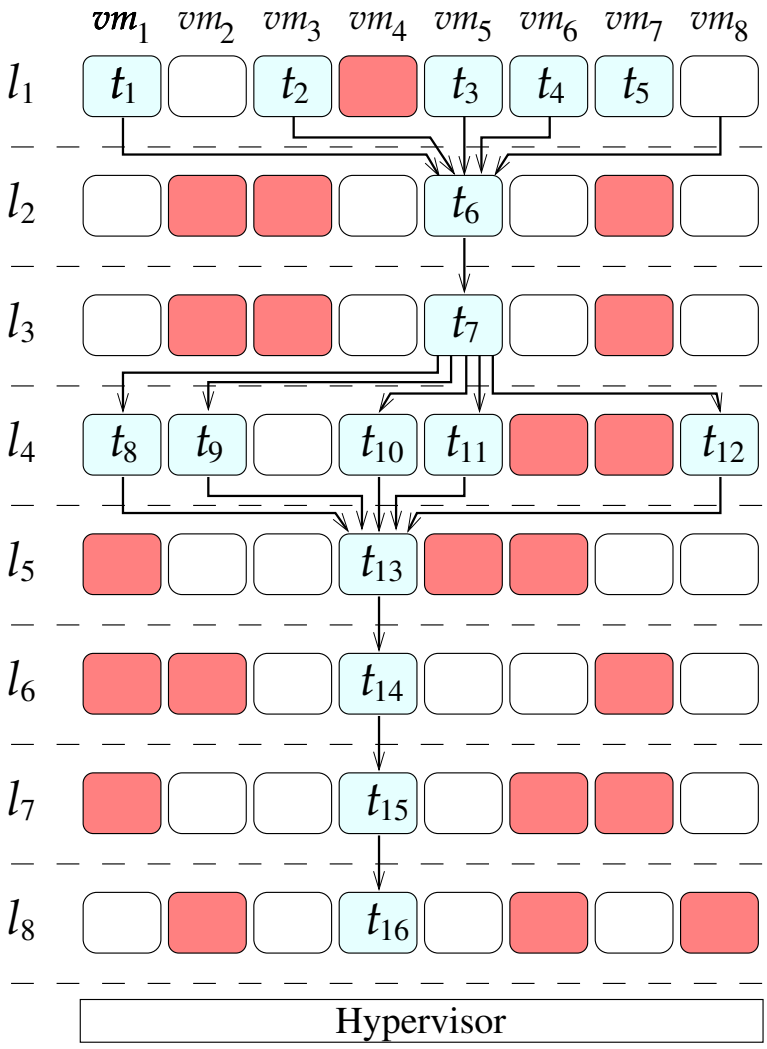


Figure 5.2: Scheduling GALFA-HI workflow on VMs.

To efficiently schedule the workflows on the Cloud; it is necessary to take the execution time variations into account [316]. Presently, there is no well-accepted theoretical solution for this issue. Therefore, most works rely on heuristics [292–296].

To design heuristic algorithm based solutions; a significant amount of experimental data is required. Figure 5.2 shows one schedule on co-located VMs. Now by changing the number of VMs running other tasks (shown in red), it is possible to create other schedules. This framework makes it easier to carry out experiments with a score of such VM schedules and also can retrieve the data. The obtained data can help to design better heuristics algorithms for VM consolidation. A unique way to profile the task execution time variation is presented in [43], called the ICBM. This work further

expands the ICBM to schedule scientific workflows on the VMs.

5.3 ICBM for workflow

Initially, the ICBM was introduced to retrieve the execution time variations of VMs on consolidated servers [43]. This work shows that the concept of ICBM can be applied to parallel workflows, too. The concept of ICBM involves increasing the resource usage of a virtualized server systematically to cause execution time variations on VMs. A parallel workflow can have more than one task. It means that for a parallel workflow the same resource usage pattern has to be applied to each of the tasks, separately. This process is described next.

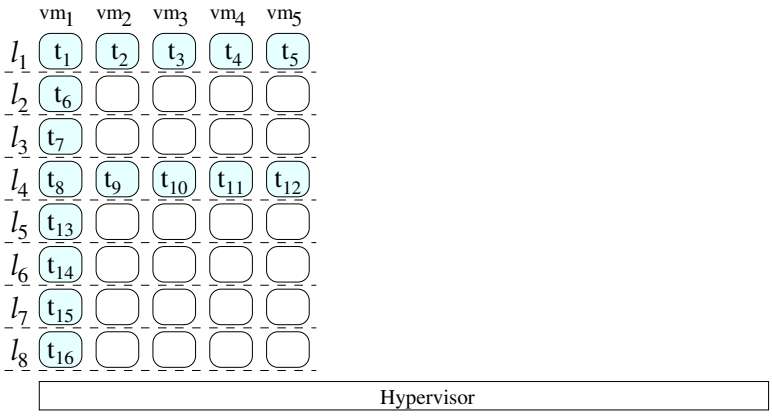
Figure 5.3 shows the steps of ICBM for applying a CPU-intensive resource usages pattern on the GALFA-HI workflow. Initially, only tasks of the workflow are run on the server. As it is shown in Figure 5.3a, at this stage tasks from no other applications are run on the server. Thus, the execution finish times of tasks of the workflow are obtained without interferences from VMs belonging to other tasks. Afterward, the workflow is rerun.

In the next stage, two additional CPU-intensive tasks are executed at each level of execution. It is referred to as stage 2 and shown in Figure 5.3b. At stage 3, four additional CPU-intensive VMs are being run along with the workflow (Figure 5.3c). Thus, the workflow is repeatedly executed, and CPU-intensive VMs are increased systematically.

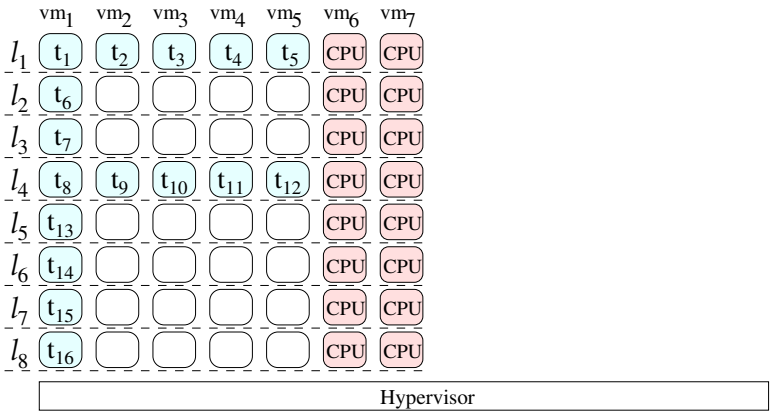
This process is repeated until all VMs of the server are utilized, and that is the final stage of the experiment. Figure 5.3d shows the final stage for this particular server configuration. This server can accommodate a maximum of 13 VMs, and all of them have been used in the final stage. Tasks of the workflow are occupying five VMs, while the remaining eight are CPU-intensive VMs.

The ICBM divides experiments into stages so that the tasks of a workflow suffer the least amount of interference at stage 1 (Figure 5.3a), while they face the most CPU-intensive resource usage contention at the final stage (Figure 5.3d). Then, the entire procedure is repeated for another resource intensive VMs, like memory (Figure 5.4a) and I/O (Figure 5.4b) intensive VMs.

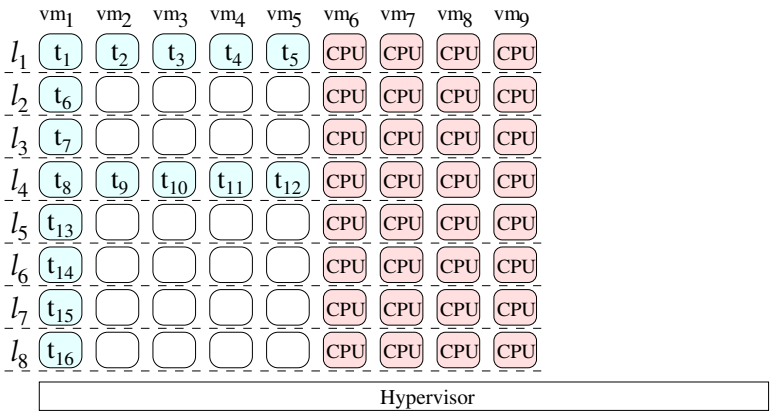
Afterward, the steps are repeated for combinations of resources, too. One example



(a) CPU resource usages pattern: Stage 1.

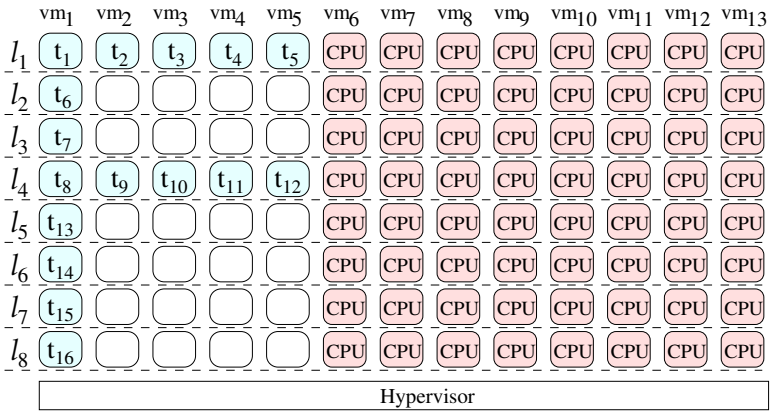


(b) CPU resource usages pattern: Stage 2.



(c) CPU resource usages pattern: Stage 3.

Figure 5.3: Applying CPU-intensive resource usages pattern on GALFA-HI workflow.



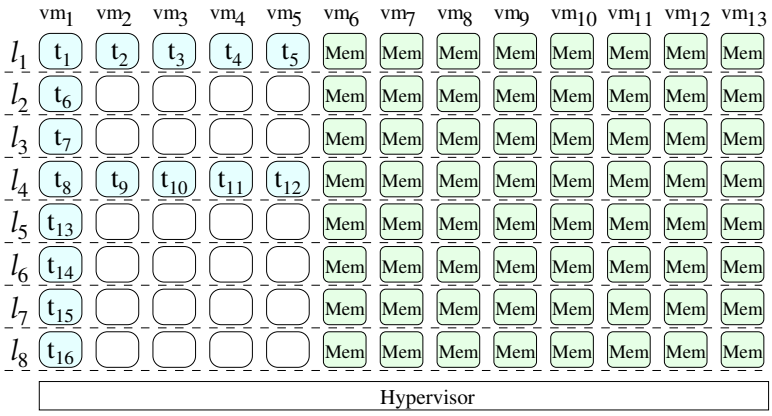
(d) CPU resource usages pattern: final stage.

Figure 5.3: Applying CPU-intensive resource usages pattern on GALFA-HI workflow (Continued).

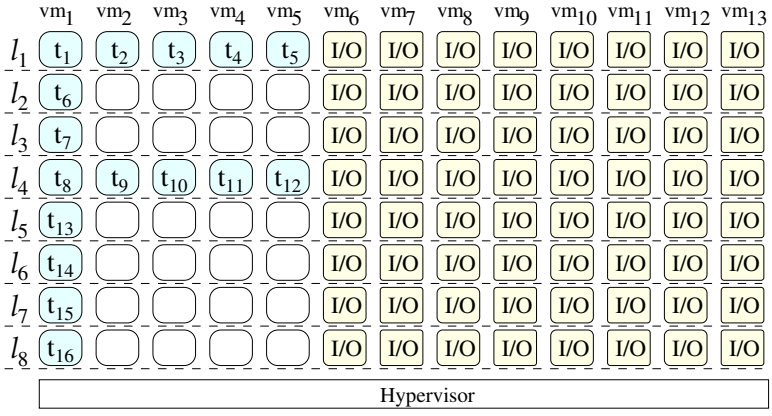
of a combination of resources is shown in Figure 5.4c, it is for CPU-Memory. Here, the process is repeated as described above. However, one CPU-intensive and one memory-intensive VM have been added at each stage, instead of two CPU-intensive ones. Other combinational resource contentions, like CPU-I/O and Memory-I/O, are created in the same process. Above discussion shows that experimental procedures like the ICBM require handling many task schedules. The exact sequence of task executions on VMs and their mutual performance inferences due to consolidation have to be investigated.

There are other tasks and resource scheduling software available. However, they are not designed to do experiments with task execution time variations on consolidated VMs. Other software uses high-level interfaces and hides almost all scheduling complexities from the user. That may be convenient for average Cloud users, who are interested only in the outcome and not the detail scheduling procedure. However, it is beneficial for researchers conducting experiments with resource contention and consolidation to control the scheduling process actively.

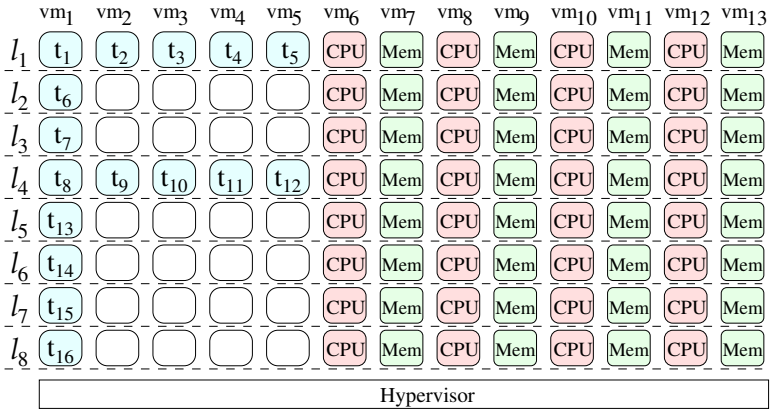
The primary objective of this work is to present a low-level, lightweight framework for experimenting with various workload patterns automatically. This framework should be able to act as both a scheduler and profiler of task execution times and connect to any Cloud. In this work, the design goals, implantation issues and experimental results of the framework are discussed in detail.



(a) Mem. resource usages pattern: final stage.



(b) I/O resource usages pattern: final stage.



(c) CPU-Mem. resource usages pattern: final stage.

Figure 5.4: Various resource usages pattern applied on GALFA-HI workflow.

5.4 Design goals of the framework

This section discusses both the primary goals and tradeoffs considered while designing and implementing the framework. The framework is modular in design, and different modules serve different purposes. The design goals of the framework are discussed first in this section, while the framework modules are discussed in the following section. The design goals of the framework are listed below.

1. Easy to perform experiments with workflow.
2. Execute resource usages patterns.
3. Easy to check the workload patterns.
4. Connect to any Cloud technology.
5. Easy to deploy.

Design goals are necessary for any project. Discussion about the goals makes it easier to understand what features are required for the proposed framework. Five design goals are further discussed below.

5.4.1 Easy to perform experiments with workflow

The first objective is to provide an easy interface to perform experiments with the workflows on virtualized servers. There are some Cloud management systems and programming paradigms. However, they are not designed for carrying out experiments with VM consolidation.

The new framework should be able to perform sophisticated experiments on the VMs independent of any management software. This work aims to provide an easy interface to design and carry out experiments with workflows on virtualized servers so that the performance variation of each task can be profiled independently. The primary application of the framework would be to discover the relationship between the execution time variations of consolidated VMs and resource utilization of the server.

5.4.2 Execute resource usages patterns

To study the effect of consolidation on the VM performance; it is necessary to create complex workload patterns and execute the tasks accordingly on the VMs. Therefore, the proposed framework should provide an easy way to run tasks according to predefined resource usages patterns as discussed above.

A human-readable file should contain all the workload patterns so that it would be easy to create and modify patterns. Researchers would be able to create those files to specify precisely the way they want the tasks to be executed on the system. Thus, the behavior of the system to resource contentions and consolidation performance can be scrutinized.

5.4.3 Easy to check the workload patterns

A workflow consists of a combination of several tasks; executing such one task may require running one or more sets of commands. Storing and managing too many commands in one workload pattern file often creates confusion. There should be an easy way to rectify any potential error in the workload pattern file. One way to achieve this is not to inscribe all the commands in the workload file rather store them in a separate file.

The workload file is created only with a short set of mnemonics. During runtime, the mnemonics are mapped to one or more actual set of commands. The process is described in more detail in the implementation section (Section 5.5).

5.4.4 Connect to any Cloud technology

Modern data centers have a countless number of servers, and various hypervisors are deployed on them. It is necessary for the framework to be able to connect to a large number of VMs running on multiple hypervisors. Therefore, the framework needs a method with a small connection overhead that can run any task on any Cloud system. The implementation section describes how this is achieved.

5.4.5 Easy to deploy

The framework should be easily deployable on a wide variety of systems. There are many operating systems today; the framework should be as universally deployable as possible. It should not be dependent on any Cloud management system or OS; thus, making it possible to initiate experiments from any machine regardless of the OS. Use of a common framework to perform experiments would give researchers the opportunity to collaborate and share experimental results more widely.

In this section, the motivations and design goals of the framework are described. The next section describes how those goals are achieved during implementation.

5.5 Implementation of the framework

This section describes the implementation process of the framework to achieve the design goals of the previous section. The framework is divided into seven modules, and each module accomplishes a specific task. The modules are listed below:

1. The command mapping module.
2. The workload loader module.
3. The hardware configuration loader module.
4. The scheduler module.
5. The connecting module.
6. The data formatting module.
7. The profile manager module

All modules are shown in Figure 5.5 and described below. Each module works independently of each other. The profile manager module coordinates among all the modules by sending signals. The modules also need to exchange data among themselves. Solid lines represent data transfer paths, while dashed lines represent command transfer paths.

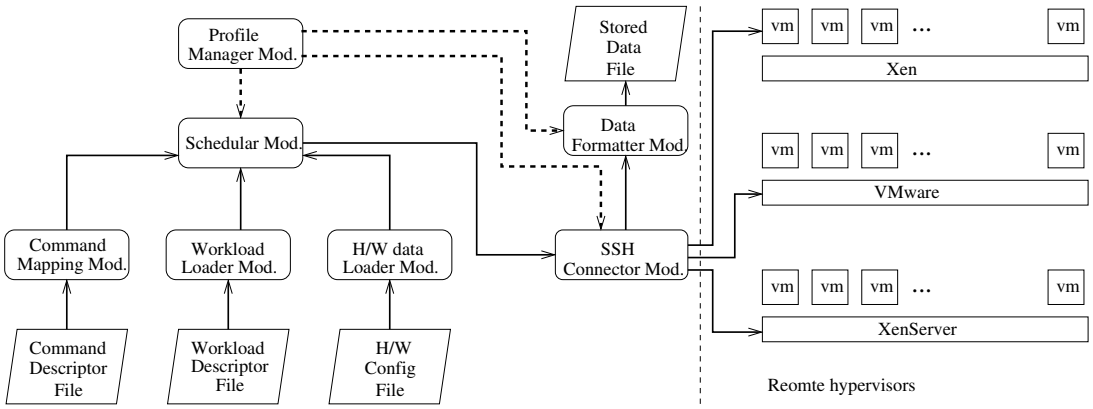


Figure 5.5: Modules of the framework.

5.5.1 The command mapping module

A workflow consists of many tasks, and each task requires one or more sets of commands to execute appropriately. Inscribing all commands to a workload file is counterproductive for several reasons. It makes the workload file long, and it becomes difficult to inspect the workload file for patterns. Furthermore, if an error is found in one of the commands, then all the occurrences have to be corrected in the workload file. This inconvenience can be avoided by storing all the actual commands in a separate file, called the *command descriptor* file.

The file stores one single mnemonic against a set of real commands required to run a task. The workload pattern files are created only with these mnemonics; they do not contain any real command. For the workload execution, first, the command mapping module loads all the actual commands into memory then all mnemonics are replaced with their corresponding actual command sets in the workload file. This design choice was made to make the workload file manageable in size and more straightforward to inspect by humans.

5.5.2 The workload loader module

All the experimental resource usage patterns are stored in a *workload descriptor* file, which is a human-readable file containing only mnemonics. This file describes line by line, the dependencies and exact execution sequence of the tasks. The tasks that are to be run simultaneously are stored in one line, while the tasks dependent on them are written in the line immediately below.

The workload loader module scans the tasks one by one so that they can be executed on the VMs precisely in the order intended on the workload file. The file is human-readable that makes it easier to identify how a virtualized system reacts to a particular pattern of resource usage.

5.5.3 The hardware configuration loader module

To execute the sequence of workload patterns some basic hardware information is required. The necessary hardware configuration of all the VMs and physical host are stored in the *hardware configuration* file.

The arrangements of VMs on physical hosts along with their MAC addresses are stored in this file. The *hardware configuration loader* module fetches this data from the file so that the framework can utilize it to connect and execute workloads on the VMs.

5.5.4 The scheduler module

The *scheduler module* collects information from the above three data loading modules, and allows the tasks to be executed on VMs. At first, memory mapped commands and hardware configuration file are used to check the consistency of the workload descriptor file. In the case of any inconsistency, the scheduling process has to be terminated.

After consistency checking the scheduler issues the necessary commands to VMs through the connecting module, which is described next. It is designed as a separate module so that it can be modified to implement any custom task scheduling algorithm for VMs in the future.

5.5.5 The connecting module

Another design goal is to make the framework universally connectable to as many systems as possible. The framework makes all connections through an SSH implementation in Java, called the JSch [317]. Thus, the entire framework is written in Java and can be run on any OS. It is completely portable and requires no installation.

The SSH is chosen over API to keep the framework lightweight. It allows the framework to connect to multiple hypervisors simultaneously without having to write

codes for multiple APIs. Furthermore, any new hypervisor can be easily supported without code modification.

5.5.6 The data formatting module

As the workflows are executed the raw data is sent back through the SSH channels to the remote machine; these data need to be formatted to use them with other applications. This module formats and stores the experimental results in the output files. The data is analyzed later to discover the relation between the resource usage patterns and task execution time variations.

5.5.7 The profile manager module

This module is responsible for coordination among all the modules so that they can work seamlessly. The profiler is modular in design so that a module can be easily customized if required. Also, adding new modules for future functionality is much more manageable in this way.

The next section describes the pseudocode for the framework. The pseudocode describes how the modules mentioned earlier work together.

5.6 Pseudocode for the framework

Figure 5.6 shows the pseudocode for the modules of the framework to run. First, all commands are loaded on the *COMM-LIST* from the command descriptor file. The command loader module does this by mapping all commands to their corresponding mnemonics in memory (lines 1-2). Then, the workload loader module parse the workload descriptor file and loads workload pattern on the *WL-LIST* (lines 3-4). The WL-LIST contains a detailed execution plan for both the parallel application and resource contention patterns.

Examples of such patterns are shown in Figures 5.3 and 5.4. Once the patterns are written in the workload file, then the framework executes all the patterns automatically. Afterward, the hardware configuration data is loaded from the file to the *VM-LIST* (lines 5-6). The VM-LIST contains all the data required for connecting to VMs during experiments.

1: Load all commands and mnemonics, from the Command Descriptor file to $COMM - LIST$.
2: Load workloads from the Workload Descriptor file to $WL - LIST$.
3: Load VMs configuration from file to the $VM - LIST$.
4: **for** Each line $\mathcal{L}_i \in WL - LIST$ **do**
5: **for** Each task, $t_j \in \mathcal{L}_i$ **do**
6: Let, $comm_j \in COMM - LIST$ be the command for t_j.
7: Let, $vm_j \in VM - LIST$ be the VM, where to run t_j .
8: Check the consistency of t_j against $comm_j$ on vm_j.
9: **if** t_j is consistent **then**
10: Put t_j, $comm_j$ and vm_j on $RUN - LIST$.
11: **else**
12: Exit.
13: **end if**
14: **end for**
15: Replace all mnemonics of $RUN - LIST$ with actual commands.
16: Simultaneously send commands to all vm_j of $RUN - LIST$.
17: Wait for their execution to finish and collect execution time data.
18: **end for**

Figure 5.6: Pseudocode for the framework.

Next, a *for* loop (lines 7-21) processes the WL-LIST, line by line. Recall that the tasks that are to be run simultaneously are written in a single line. The inner *for* loop (lines 8-17) removes one task at a time from the line and checks for consistency against hardware data and commands. The inner *for* loop processes all the tasks of a line, as those tasks are intended to be executed together. Therefore, the inner *for* loop do not execute the tasks inside the loop rather puts in a list so that all of them can be executed simultaneously.

After the consistency checking, if all the tasks are found to be consistent, then they are stored in a linked list, called the *RUN-LIST*. On the other hand, if a task is not compatible then the application exits immediately. Once all the tasks of a line are

processed the inner *for* loop exits.

Then, all the mnemonics of the RUN-LIST are replaced with the actual command set with the assistance of the COMM-LIST (line 15). Once this is done the commands are simultaneously sent to execute all tasks of the RUN-LIST (line 16). The framework then waits for the tasks to finish and collect the individual task execution time data (line 17). Afterward, the same process is repeated for the next line of WL-LIST on the next iteration of the outer *for* loop.

The outer *for* loop exit when all the lines of WL-LIST (entire pattern) have been processed. To experiment with another resource usage pattern the procedure repeated from the beginning.

5.7 Workloads used

Two categories of workload have been used in the experiments. The first category is the data-intensive scientific workflow, which is used to observe the execution time variations of tasks under consolidation. The second category is a set of benchmarks suites used to create resource contention patterns on servers.

The benchmark suites are used to build-up stage by stage resource contention in the server; to make the performance of the workflow tasks begin to vary on consolidated VMs. The framework runs the benchmark suites according to the pattern described in the workload file and retrieves the execution time of each task of the workflow at each stage.

5.7.1 Scientific workflow: GALFA-HI

The GALFA-HI survey continuously scans the sky for naturally occurring hydrogen atoms [314]. The survey divides the sky into several parts, and the parts are scanned at stages. After each stage, data cubes are released after a detailed scan of a part of the sky. Several data cubes have been released so far.

Five of those released data cubes have been processed with the Montage toolkit [315], to create a mosaic image of a part of the Milky Way galaxy. The workflow is shown in Figure 5.1 has 16 tasks and eight levels. It is a data-intensive workflow, which processes about 2 GB of raw data cubes. Experiments measure the execution time variation of tasks in this workflow due to consolidation.

5.7.2 Set of benchmark suites

Three sets of benchmark suites have been used to create resource contention patterns on the tasks of the above workflow. They are the sets of CPU, memory and I/O-intensive benchmark suites.

Each benchmark suite, in turn, consists of several similar types of tests. The benchmarks have been used in the experiments of previous chapters and already described in Section 3.4; therefore, a detail discussion is not given here. Next, each set is described in brief.

5.7.2.1 CPU-intensive benchmarks

Three CPU-intensive benchmarks have been used, they are the *Sysbench CPU* test, *Nbench* and *Unixbench*. The Sysbench CPU test has been widely used with multi-core server [318] and VM workload consolidation experiments [319].

The Nbench is a CPU-intensive benchmark suite, having ten different CPU-intensive tests [219]. The Unixbench is another CPU-intensive benchmark suite, which has been used for experiments on Amazon EC2 [221]. Thus, the benchmarks are have been used in previous works on CPU performance analysis.

5.7.2.2 Memory-intensive benchmarks

Three memory-intensive benchmarks have been used for creating resource contention patterns. The first is the *Cachebench*, which consists of eight different memory tests [224]. The second is the *Stream*, a syntactic benchmark program for measuring sustainable memory bandwidth [225]. The final one is the *Sysbench memory* test.

5.7.2.3 I/O-intensive benchmarks

Five I/O-intensive tests have been used to create resource contention patterns. The *Filebench* is an important I/O benchmark suite [229], which can be configured to perform various I/O-intensive tests. Five of them are used, they are the *file-server*, *web-server*, *web-proxy*, *video-server* and *online transaction processing* (OLTP) test.

5.7.3 Experimental setup

Three Dell XPS-8500 servers of identical hardware configuration had been set up for the experiments. Each server has one Intel i7-3770 processor and 32 GB memory. The i7-3770 has four cores and eight hardware threads, each is clocked at 3.4 GHz. Three different hypervisors are installed on three servers; they are, VMware ESXi 5.5, Citrix XenServer 6.5 and Xen 4.6 on *Centos* 7.

Each hypervisor has 14 VMs of identical configuration. Each VM has one processor, 2 GB of Ram and 50 GB virtual disk. During experiments, the framework connects to all 42 (14×3) VMs on three hypervisors and execute workload patterns simultaneously. The framework itself runs on a remote Dell OptiPlex 9010 machine and connects to hypervisors through the LAN. The results of the experiments are given next.

5.8 Experimental Results for GALFA-HI tasks

Recall that the GALFA-HI workflow (Figure 5.1) has 16 tasks comprised of seven functions. Average execution times of those seven functions without interferences are shown in Table 5.1. In this case, the tasks are scheduled precisely like that of Figure 5.3a. That is the tasks of the workflow are run without any other co-located VMs.

Table 5.1: Mean execution times of tasks of GALFA-HI workflow on VMs (in Figure 5.3a) without interferences.

Level	Task	Time (m)
1	mShrinkCube	3.878
2 & 5	mImgtbl	0.02
3	mMakeHdr	0.02
4	mProjectCube	39.774
6	mAddCube	12.32
7	mGetHdr	0.02
8	mViewer	0.04

Results are shown for two functions, the *mProjectCube* and *mShrinkCube*. The rest of the functions also show variations similar to that of these functions. The results are grouped according to the resources loads for the convenience of discussion. The three resource workload groups are the CPU, memory, and I/O. Each graph shows ETV of one task for one type of resource contention on three hypervisors.

5.8.1 Variations due to CPU-intensive workload

The three graphs in Figure 5.7 show execution time variations of the mProjectCube function for three CPU-intensive workloads. Again in each graph, the task ETV for three hypervisors are shown.

In each graph, the Y-axis represents the execution time variation. The X-axis represents how many CPU-intensive VMs were running on the server besides the workflow. The first point of the X-axis is zero meaning no other VMs were running when the execution time of the function was measured. That is only the tasks of the workflow are executed on the VM with zero co-located VMs. This execution schedule is shown in Figure 5.3a.

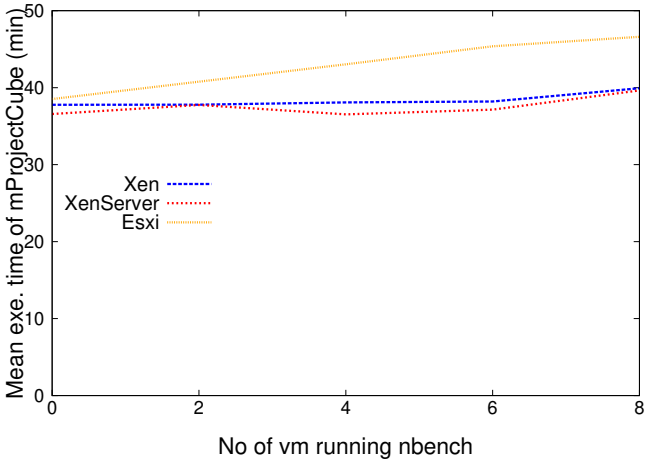
The next point on the X-axis is 2. That means two additional CPU-intensive VMs were running at every step of the workflow execution (schedule shown in Figure 5.3b).

In this way, the workflow is repeatedly executed with an increasing number of CPU-intensive VMs. The final point is 8, indicating eight additional CPU-intensive VMs were used, at each step of workflow execution as shown in Figure 5.3d.

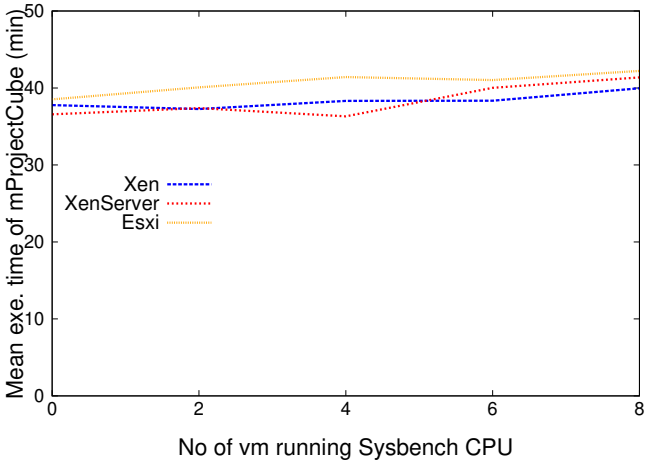
In each graph, from left to right on the X-axis the interference from the number of CPU-intensive VMs increases. The leftmost point is the execution time of a task without any interference from other VMs. The rightmost point is the execution time of the same task with maximum interference.

Figure 5.7 shows that the mProjectCube function shows relatively less execution time variation due to CPU-intensive VMs. It applies to all three hypervisors.

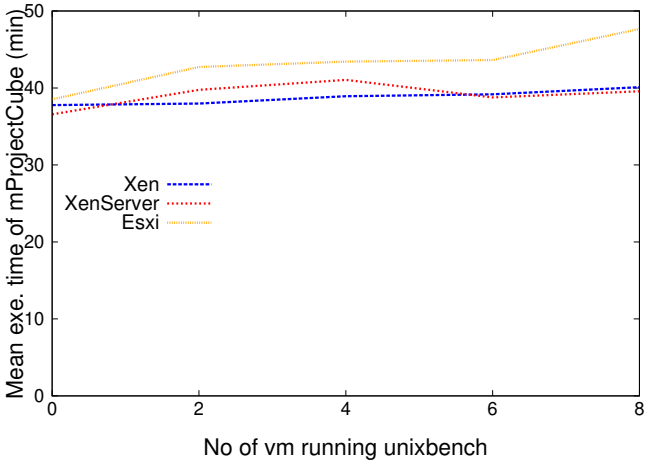
Figure 5.7c shows that on ESXi hypervisor the execution time of mProjectCube function without any co-located VM is 38.52 minutes. On the other hand, the same function takes 48.12 minutes to execute while eight other Unixbench suites are running on the co-located VMs. Thus, the execution time of mProjectCube function goes from 38.52 minute (the leftmost point on the graph) to 48.13 minute (rightmost point) due



(a) Task ETV of mProjectCube due to the Nbench.

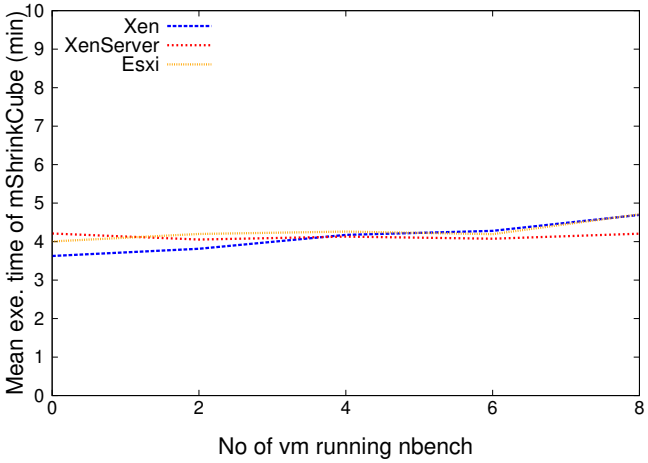


(b) Task ETV of mProjectCube due to the Sysbench CPU test.

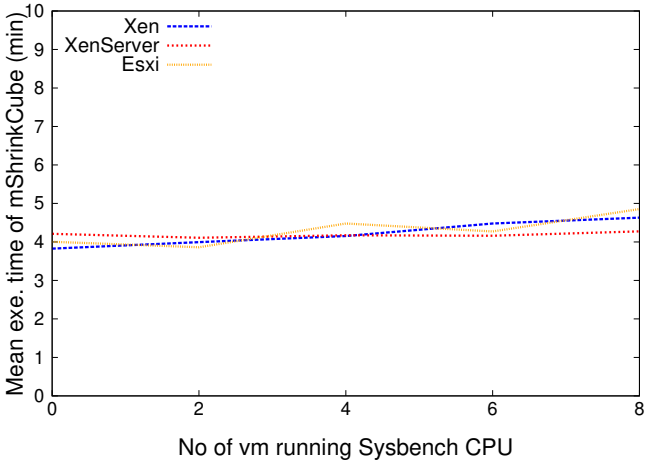


(c) Task ETV of mProjectCube due to the Unixbench.

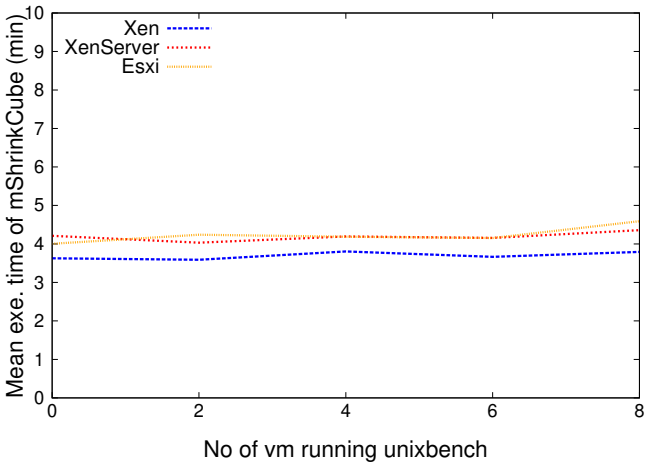
Figure 5.7: Task execution time variation (ETV) of the mProjectCube function due to the CPU-intensive workload patterns on VMs.



(a) Task ETV of mShrinkCube due to the Unixbench.



(b) Task ETV of mShrinkCube due to the Sysbench CPU test.



(c) Task ETV of mShrinkCube due to the Unixbench.

Figure 5.8: Task execution time variation (ETV) of the mShrinkCube functions due to the CPU-intensive workload patterns on VMs.

to the addition of 8 VMs. Here, each co-located VM is running a Unixbench benchmark suite. Therefore, consolidation with eight additional CPU-intensive VMs (in this case the Unixbench) causes 24.94% increase in execution time of the mProjectCube function. It is the highest of the three hypervisors.

For XenServer, the maximum execution time variation of the mProjectCube function happens due to Sysbench CPU tests (Figure 5.7b). In this case, the ETV is 13.49% and caused when the function is consolidated with eight co-located VMs running Sysbench CPU tests. For Xen, the mProjectCube function also shows the maximum variation due to the Unixbench (Figure 5.7c). In this case, ETV is 6.15% caused when eight VMs with Unixbench are consolidated with the function.

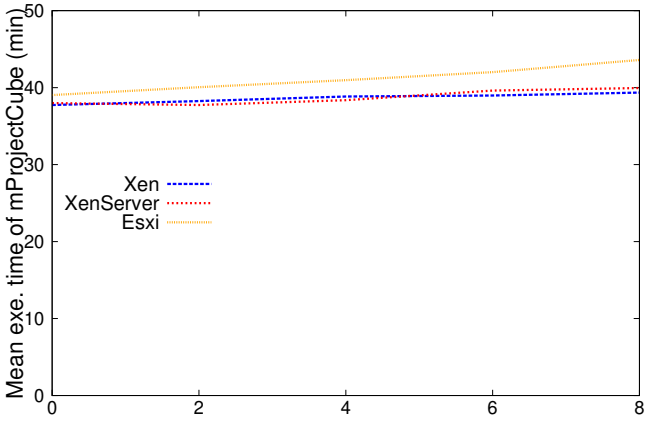
Figure 5.8 shows the execution time variation of the mShrinkCube function. The execution time variation data of three hypervisors show that the effect of CPU-intensive VM is also minimum on the mShrinkCube function.

5.8.2 Variations due to memory-intensive workload

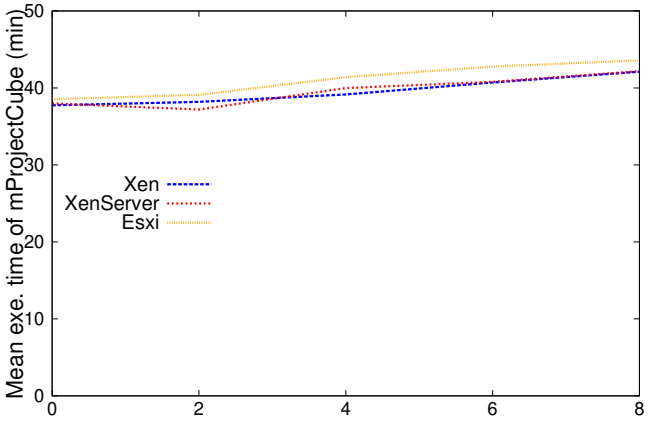
Figures 5.9, and 5.10 shows the execution time variations of the mProjectCube, and mShrinkCube functions due to the memory-intensive workloads, respectively. Figures 5.9a, 5.9b, and 5.9c show the execution time variation of the mProjectCube function due to consolidation with the Cachebench, Stream, and Sysbench memory test, respectively. Figures 5.10a, 5.10b, and 5.10c show the execution time variation of the mShrinkCube function for the same three benchmark suites, respectively.

In all cases, the mProjectCube shows more variation compared to the mShrinkCube function. For all three hypervisors, the mProjectCube function shows the maximum execution time variations when consolidated with the Stream benchmark. Figure 5.9b shows the maximum execution time variations of the mProjectCube function due to the consolidation with the Stream benchmark. The execution time increase of the mProjectCube function for ESXi, XenServer, and Xen hypervisors are 24.24%, 11.02%, and 11.56%, respectively.

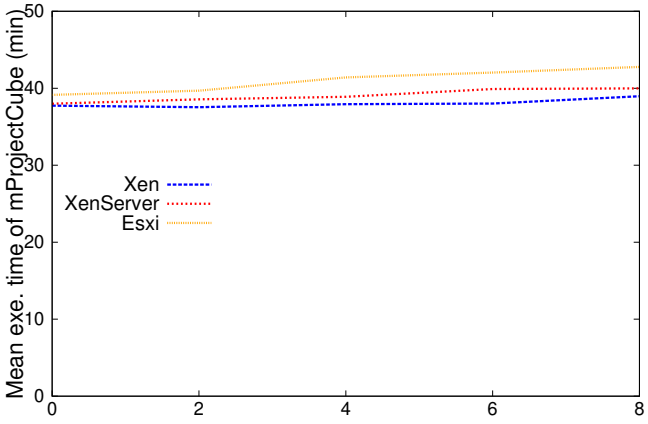
From the above graphs, it can be seen that the GALFA-HI workflow tasks do no show much execution time variation due to memory-intensive co-located VMs.



(a) Task ETV of mProjectCube due to the Cachebench.

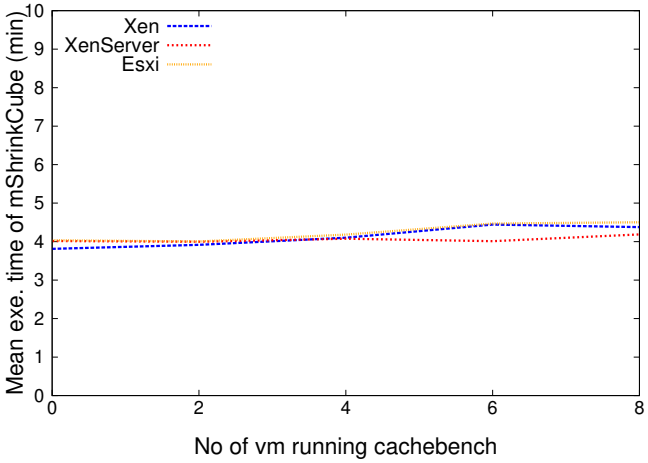


(b) Task ETV of mProjectCube due to the Stream.

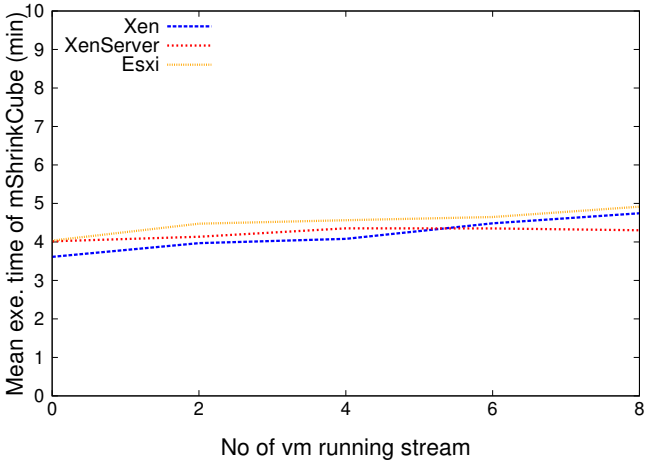


(c) Task ETV of mProjectCube due to the Sysbench Memory test.

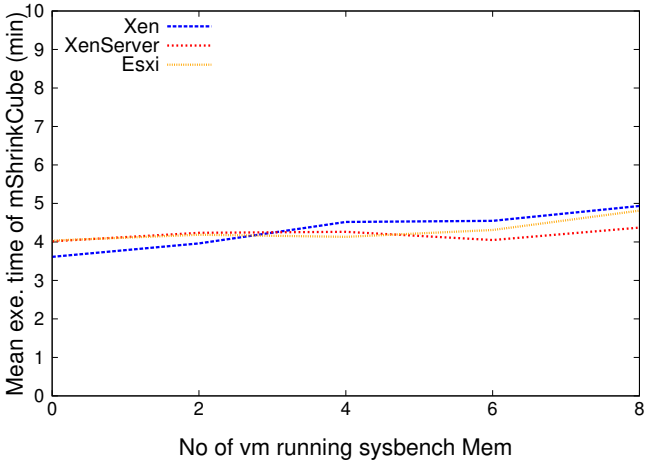
Figure 5.9: Task execution time variation (ETV) of the mProjectCube function due to the Memory-intensive workload patterns on VMs.



(a) Task ETV of mShrinkCube due to the Cachebench.



(b) Task ETV of mShrinkCube due to the Stream.



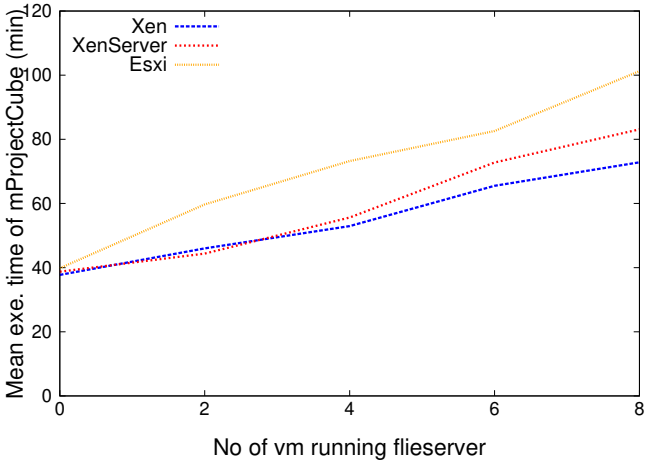
(c) Task ETV of mShrinkCube due to the Sysbench Mem test.

Figure 5.10: Task execution time variation (ETV) of the mShrinkCube function due to the Memory-intensive workload patterns on VMs.

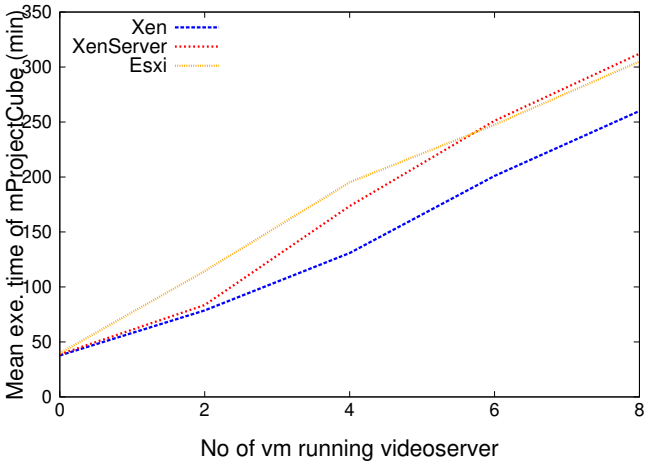
5.8.3 Variations due to I/O-intensive workload

During the experiments of previous chapters, it was observed that the I/O-intensive VMs tend to show a higher degree of execution time variation compared to CPU and memory-intensive VMs. Five I/O-intensive benchmarks have been used in the experiments. Figures 5.11, and 5.12 show the execution time variations of the mProjectCube, and mShrinkCube functions due to consolidation with five different I/O-intensive benchmarks, respectively.

Figure 5.11b shows the co-located VMs with video servers can cause significant execution time variation for the mProjectCube functions on all three hypervisors. When

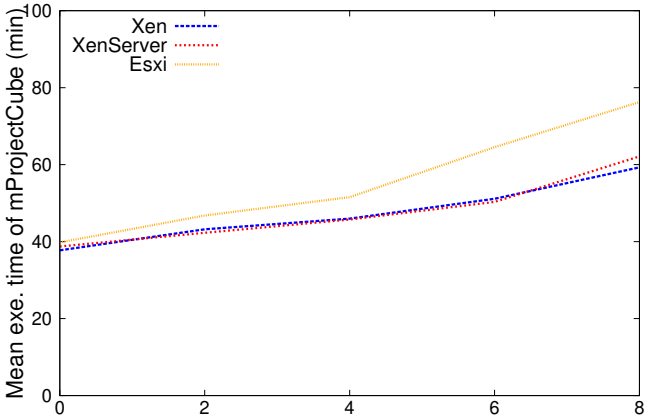


(a) Task ETV of mProjectCube due to File server.

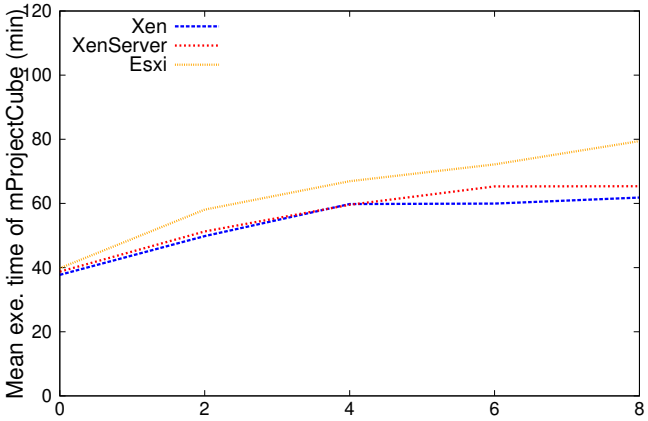


(b) Task ETV of mProjectCube due to Video server.

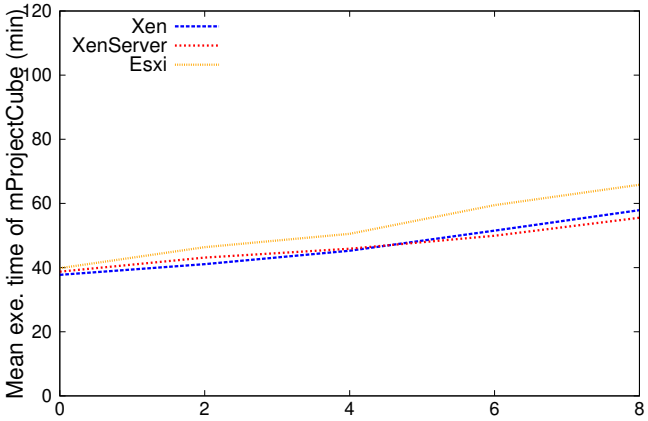
Figure 5.11: Task execution time variation (ETV) of the mProjectCube function due to the I/O-intensive workload patterns on VMs.



(c) Task ETV of mProjectCube due to Webproxy server.



(d) Task ETV of mProjectCube due to Web server.



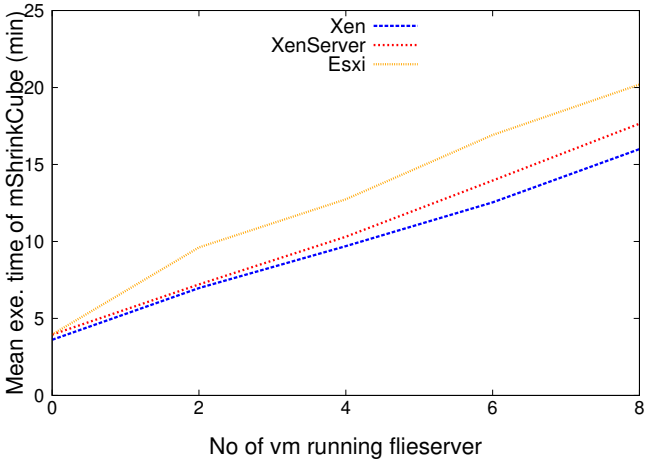
(e) Task ETV of mProjectCube due to OLTP.

Figure 5.11: Task execution time variation (TETV) of the mProjectCube function due to the I/O-intensive workload patterns on VMs (Continued).

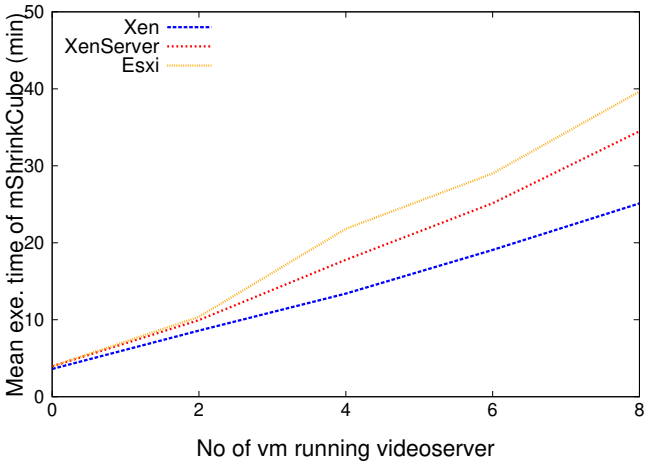
consolidation with eight VMs with video servers the execution times of the mProjectCube function increased by 683.30%, 705.83%, and 588.96% compared to the initial value on ESXi, XenServer, and Xen, respectively.

Figure 5.12b shows that the video servers also have a significant impact on the execution time variation of mShrinkCube function on all three hypervisors, too. The execution time increase of the mShrinkCube function due to consolidation on ESXi, XenServer, and Xen are 901.92%, 774.10%, and 595.34%, respectively.

For other I/O-intensive benchmarks, similar results can be obtained, too. For example, Figure 5.11a shows the execution time variation of the mProjectCube function due to the file-servers on all three hypervisors. Here, execution time increases for ESXi,

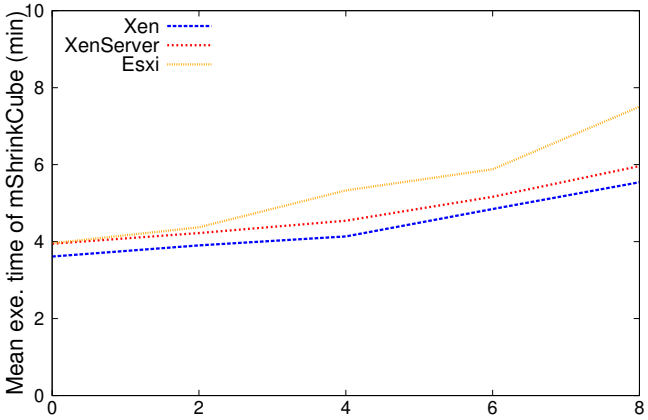


(a) Task ETV of mShrinkCube due to File server.

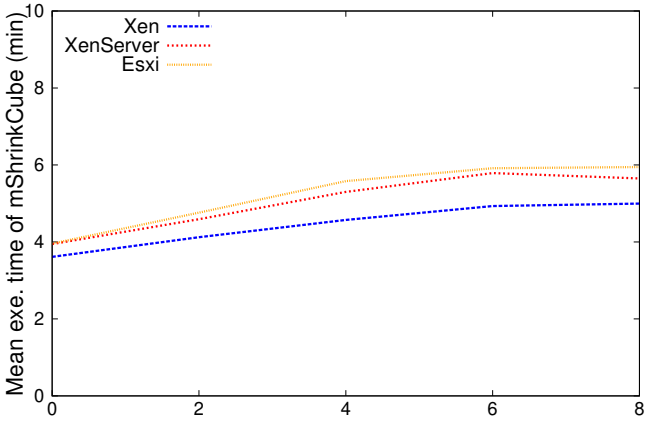


(b) Task ETV of mShrinkCube due to Video server.

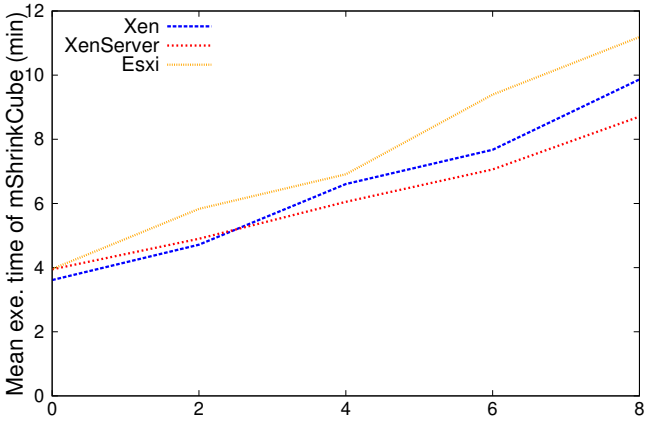
Figure 5.12: Task execution time variation (ETV) of the mShrinkCube function due to the I/O-intensive workload patterns on VMs.



(c) Task ETV of mShrinkCube due to Webproxy server.



(d) Task ETV of mShrinkCube due to Web server.



(e) Task ETV of mShrinkCube due to OLTP.

Figure 5.12: Task execution time variation (TETV) of the mShrinkCube function due to the I/O-intensive workload patterns on VMs (Continued).

XenServer, and Xen is 154.39%, 114.78%, and 92.95%, respectively.

The file-servers similarly cause execution time variation for the mShrinkCube function, too. Execution time increases for ESXi, XenServer, and Xen are 411.13%, 347.96%, and 343.15%, respectively.

From the presented execution time variation data it is clear that a combination of benchmarks can be used to create resource contention patterns for tasks on VMs. The significance of the above findings is discussed next.

5.8.4 Discussion

The experimental results show that resources like CPU, memory, and I/O, all have different types of effects on the task execution time. It is observed on all three hypervisors. From the above results, it is clear that the execution time variation directly depends on the cumulative resource requirement of the co-located VM of a server.

Previously it has been shown that by profiling the execution times of co-located VMs it is possible to predict the task execution time variations [43]. The resource requirement of the VMs plays a massive part in execution time variations. For example, both the mProjectCube and mShrinkCube functions are I/O-intensive tasks, and they have the maximum variation for I/O-intensive benchmarks. For both the CPU and memory intensive co-located VMs the functions shows much less execution time variation. Thus, resource intensities of co-located VMs play an essential part in consolidation performance.

The objective of the experiments is to show that the proposed framework can profile the tasks of a scientific workflow for any workload and hypervisor. It can help to design and carry out experiments with VM placement and consolidation for scientific workflows. Profiling execution time of VMs it is possible to determine which types of VMs show better performance when consolidated together.

5.9 Related work

Related works can be divided into two broad categories. The first category of works deals with application performance efficiency on the Cloud and VM consolidation [292–296]. However, the works do not provide any general framework to do experiments

with the tasks of parallel applications. In contrast, this work provides a simple and effective framework that can be used for experimental purposes on the Cloud.

The second category of works are the Cloud management, maintenance and scheduling software [39, 303–313]. They can provide many high-level functionalities for the Cloud, like running selected operations periodically. Many complex operations can be performed with just a few commands. However, they hide much of the operational complexity from the users and do not allow low-level control over the task execution process. On the other hand, this framework offers an easy interface for executing tasks according to the requirement of the experiment.

Although the works outlined above provide some high-level support for running tasks on the Cloud, none of them provides low-level functionality to carry out experiments with VM consolidation. Unlike, those previous works here a framework is presented to perform experiments with workloads on the Cloud.

5.10 Conclusion

There are many issues related to the Cloud that depend on consolidation, like application performance, energy efficiency, and resource utilization. Empirical data can help to choose a better VM combination for consolidation performance and efficiency. This work presents the design and implementation of a framework for performing experiments with execution time variation of scientific workflows on the Cloud. Profiling of task execution time is required for better understanding of VM consolidation.

The framework can apply any resource usage patterns to the tasks of a workflow. It does not compile the input files rather it behaves like an interpreter. There is no well-accepted theocratical model for task execution variation due to consolidation. Therefore, such a framework would help to set up large-scale experiments for achieving a practical solution. To show the capability of the framework experiments are performed with a real data-intensive workflow and three hypervisors. The resource contention patterns or workloads for VMs are created by combining various types of benchmarks.

The framework is lightweight and implemented in Java. It can be run on any OS and can connect to any hypervisor or the Cloud. Several sets of experiments have been conducted, and the results are retrieved successfully. The results demonstrate that the framework is capable of executing workflow schedules and resource usage pattern on

multiple hypervisors. This framework can be a powerful tool for experimenting with VM consolidation and task execution time variation of workflows. In the next chapters, the framework is used to set up large scale of experiments with various scientific workflows and diverse sets of resource usage patterns.

Chapter 6

Analysis and prediction of workflow and task execution time variations on multiple hypervisors

"Adversity is the first path to truth."

— George Gordon Noel Byron* (1788–1824)

6.1 Introduction**

This chapter is the continuum of the previous chapters. Chapter 4 introduces the *Incremental Consolidation Benchmarking Method* (ICBM) to predict the task *Execution Time variation* (ETV) of consolidated *Virtual Machines* (VMs). That chapter also presents task ETV prediction models for consolidated VM and experiments are done with Xen hypervisor. Then, Chapter 5 presents a framework to conduct experiments with parallel workflow and multiple hypervisors.

In this chapter, the framework is used to experiment with ICBM and workflows on multiple hypervisors. The ICBM is applied to collect data and build prediction models

*Image source: `http://www.facts-about.org.uk/famous-people-facts-starting-with-l/lord-byron.htm`

**This chapter contains materials that were published as [46]. Since the publication, the content has been extended with more experiments, results, and discussion. Author of the dissertation has designed and conducted all experiments. The author has also collected and analyzed the data and wrote the draft.

for multiple hypervisors. Ideas that are introduced in the previous chapters are further developed in this chapter. The scope of experiments has been expanded to multiple hypervisors. This chapter presents unified task ETV prediction models for multiple hypervisors.

A VM is a self-contained unit of execution created on top of a hypervisor in a physical *host*. Several VMs are *consolidated* on a single host to reduce the operational and maintenance cost. All the simultaneously running VMs of a physical host are collectively known as the *co-located VMs*. The *virtualization* has already taken a prominent place in modern data centers. Most common features of a data center, like the fault tolerance mechanism and pay-as-you-go model for the Cloud, are implemented through virtualization.

VMs are consolidated to increase the resource utilization of servers. However, it imposes a performance penalty, which is manifested through the ETV of co-located VMs [242–244]. It is an obstacle to efficiently scheduling parallel workflows on virtualized systems. The ETV depends on the server workload and resource contention among the VMs. As a result, the same task may take a different amount of time to be completed on different VMs depending on the resource contention level. To schedule a task of any parallel application the execution finish time of all the parent tasks must be known. Because of ETV, the task execution finish times of VMs may deviate, which makes scheduling parallel and high-performance applications on virtualized servers difficult [34, 103].

Using the ICBM as the basis; this chapter introduces unified models for predicting the ETV among multiple hypervisors. The contribution of this chapter is also summarized next:

1) This work introduces unified models for predicting the ETV on multiple hypervisors. The *Least Square Regression* (LSR) is used to build the unified models. These models can be used to predict the execution time variation of VMs for multiple resource combinations;

2) Experiments are conducted on three actual virtualized servers, and no simulation is used. All the prediction results presented here are real system data. This chapter proves the effectiveness of ICBM on multiple hypervisors;

3) In this chapter, experiments are conducted with parallel workflows on multiple

hypervisors, and prediction models are built with an *Artificial Neural Network* (ANN).

Rest of the chapter is structured as follows. The primary objectives of the experiments are discussed in Section 6.2, Next, Section 6.3 discusses the methodology used in the experiments. Section 6.3.2 discusses the benchmark suites used in the experiments. Sections 6.3.3 discusses the experiential setup. The experimental results for the execution time variation are shown in Sections 6.4, 6.5, and 6.6. The Section 6.8 discusses the prediction results for LSR models, while the Section 6.12 discusses the prediction results for ANN models. Finally, Section 6.13 concludes the chapter.

6.2 Resource contention and performance interference of consolidated VMs

This section presents the concepts behind the experimental setup of this chapter in a graphical form. Figures 6.1, and 6.2 in combination provides the conceptual overview of the objective of this chapter.

A consolidated server can have different numbers of VMs running at different times. Moreover, different VMs may be different types of resource intensive. Some of them may be CPU intensive, while others may be memory or I/O intensive. Thus, a combination of different types and number of resource intensive VMs may be present in the server at any given time. Each resource intensive VM has a particular kind of effect on the performance. For example, an increase in the number of the CPU intensive VMs will create more CPU resource contention on the system. The same goes for the memory or I/O intensive VMs.

The logical architectural construct of all modern computers are derived from the original *von Neumann model* [320, 321]. According to the model, all components of a computing system can be categorized into three parts; *Central processing unit* (CPU), *Memory Unit*, and *Input/Output Devices* [321, 322]. In modern computers, they also represent three types of resources required to run an application. This concept can be extended for VMs, too. The effects of these resources in the context of consolidation is discussed next.

Figure 6.1 depicts the effect of consolidation on a VM performance from the resource contention point of view. The figure has three axes, one for each primary resource type; CPU, memory, and I/O. Each axis indicates how many VMs of each resource type are running on the server. As mentioned above, three main resources can be identified for a server; CPU, Memory, and I/O.

In Cloud and data centers, it is not uncommon to run the same task or the same type of task repeatedly. However, on the repeated runs the server may have a different number of co-located VMs. In a data center, the numbers of co-located VMs on servers are continually changing. Thus, the performance of the same task may vary depending on the number of co-located VMs are running and their resource intensities.

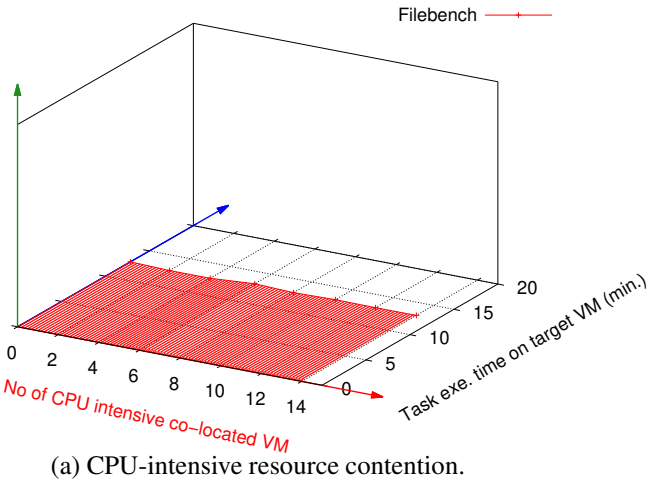
Previous two chapters (Chpaters 4 and 5) discuss a profiling method and tool for consolidated VM. Those chapters present a low-cost procedure for both training and testing of performance prediction models for consolidated VMs. The primary focus of this chapter is to apply those methods to a broader experimental setting with multiple hypervisors.

In a consolidated environment, the VMs compete for above mentioned three system resources with each other. One example of such a situation is given in Figure 6.1a, where the three axes represent the number of three resources intensive VMs on the system. The X-axis, which is shown in Red represents the number of CPU intensive VMs. Y-axis (shown in Blue) indicates the number of memory-intensive VMs. Lastly, the Z-axis (shown in Green) indicates the number of I/O intensive VMs.

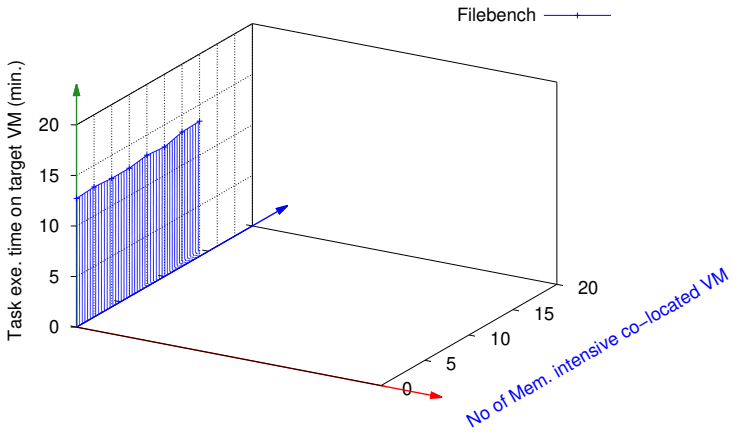
Initially, the execution times of the tasks are profiled for each resource types. Figure 6.1a shows the execution time variation of one of the tasks due to CPU-intensive co-located VMs. Here, the execution time variation of the Filebench is profiled for varying number of CPU-intensive VMs. Recall that the data profiling methodology is discussed in details in Section 4.4 of Chapter 4.

In the graph, the X-axis presents the number of CPU-intensive co-located VMs, while the Y-axis presents the execution time of the Filebench. This execution time variations are real system data, which is presented in Figure 6.13 and discussed in Section 6.6. These data are collected using the ICBM introduced in Chapter 4.

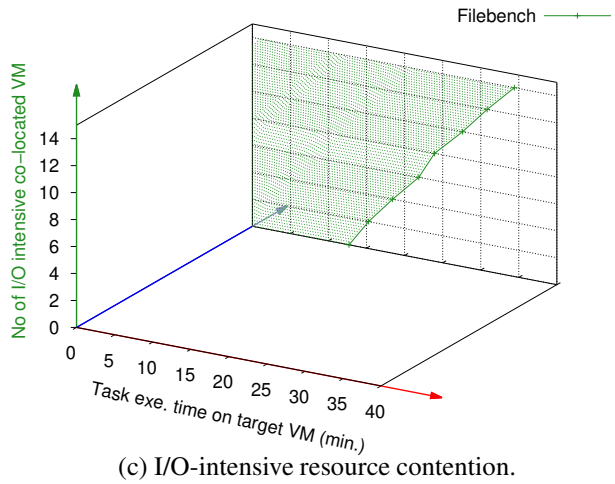
Filebench is an I/O intensive task; therefore, it does not show much variation due to CPU intensive co-located. The Filebench is already introduced in Section 3.4.3. Also, the effect of I/O intensive co-located VMs on the task execution time is already discussed in Section 3.7.3.



(a) CPU-intensive resource contention.



(b) Memory-intensive resource contention.



(c) I/O-intensive resource contention.

Figure 6.1: Resource contention due to single resources usage.

Next, Figure 6.1b shows the execution time variation of Filebench for memory intensive co-located VMs. In this case, the number of memory intensive co-located VMs presented in Y-axis, while the Z-axis represents the execution time variation of the Filebench. In this case, again, Filebench do not show much variation due to memory intensive co-located VMs as discussed before in Section 3.7.3.

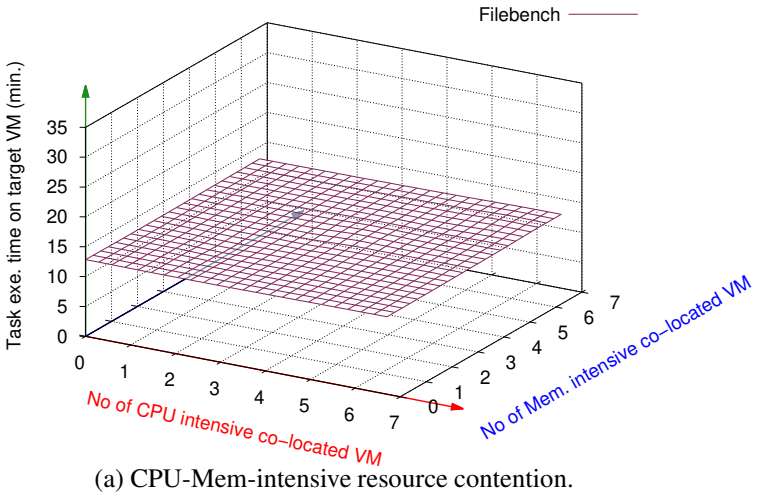
Lastly, Figure 6.1c shows the execution time variation of Filebench due to I/O intensive co-located VMs. In this case, the Filebench being an I/O-intensive task shows a much higher level of execution time variation compared to the previous two cases as explained in Section 3.7.3. Here, the number of I/O intensive co-located VMs is shown in the Z-axis and the execution time variation is shown on the X-axis.

In the three graphs of Figure 6.1, the number of three types of resource intensive co-located VMs are shown on three axes. On the other hand, the execution time variations are shown in cyclic order. That is, When the number of co-located VMs is shown in X-axis, then the execution time variation is shown in Y-axis. When the number of co-located VMs are shown in the Y-axis, the execution time variation is shown in Z-axis. Lastly, when the number of co-located VMs is in the Z-axis, the execution time variation is shown back in X-axis.

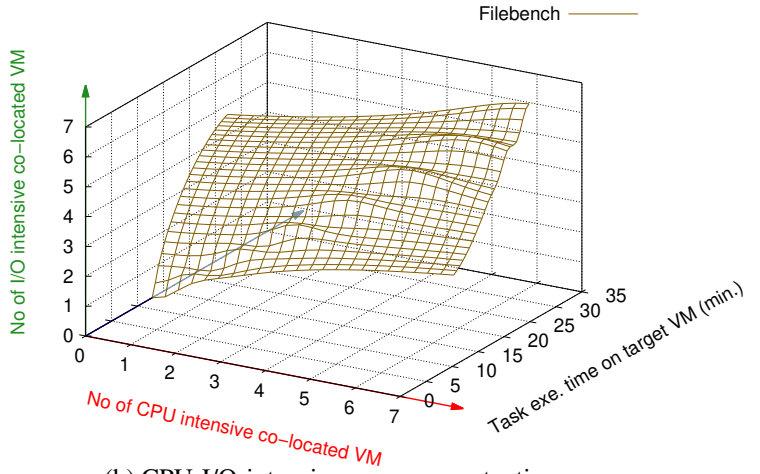
Even though the execution variation due to one type of resource intensive VMs can be placed into a two-dimensional space, the three-dimensional space is required to show their overall relation to each other. Next, Figure 6.2 shows how the execution time variation due to more than one type of resources.

Figure 6.2 shows the execution time variation of Filebench due to resource combinations on the co-located VMs. Figure 6.2a shows the execution time variation of Filebench due to CPU and memory intensive co-located VMs. In this case, the number of CPU-intensive co-located VMs are shown on the X-axis, while the memory-intensive VMs are shown on the Y-axis using the same convention as previously introduced in Figure 6.1. The execution time variation of Filebench is shown on the Z-axis. As it can be seen that the Filebench, an I/O intensive task do not show much execution time variation due to CPU and memory intensive co-located VMs.

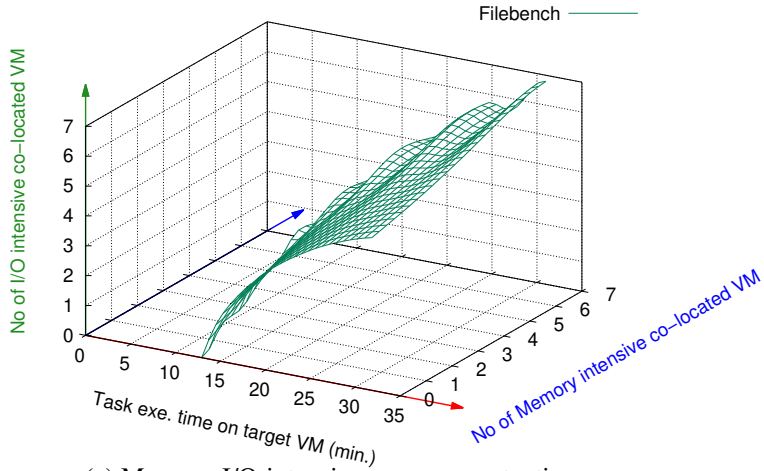
On the other hand, Figure 6.2b shows a more considerable amount of execution time variation for CPU and I/O intensive co-located VMs compared to the previous case. This is also expected as already explained in Section 3.7.3 of Chapter 3. In this case, the number of CPU and I/O intensive co-located VMs are shown in X and Z axes, respectively. The Z-axis shows the execution time variation of Filebench on the Y-axis.



(a) CPU-Mem-intensive resource contention.



(b) CPU-I/O-intensive resource contention.



(c) Memory-I/O-intensive resource contention.

Figure 6.2: Resource contention due to combination of multiple resources.

Lastly, Figure 6.2c shows the execution time variation of the Filebench for Memory and I/O intensive co-located VMs. In this case, the X-axis shows the execution time variation of Filebench while the number of memory and I/O intensive co-located VMs are shown in the Y and Z axis, respectively. As expected in this case the execution time shows much more variation compared to the previous two cases.

In Figure 6.2, all graphs are plotted in three-dimensional space. While plotting the number of co-located VMs are placed on the axis using the same convention as used in Figure 6.1 before. That is the number of CPU, memory, and I/O intensive co-located VMs are shown in X, Y, and Z-axes, respectively. In each graph of Figure 6.2, two of the axes are used to indicate the number of co-located VMs, while the third axis is used to represent the execution time variation.

Figures 6.1 and 6.2 are conceptually connected to each other; both figures have three axes representing the three types of resource contention; CPU, memory, and I/O. In Figure 6.1 execution time variation is shown due to single resource contention. On the other hand, in Figure 6.2 combination of resources are used to create a combined effect, which is represented in a three-dimensional space.

For example, in Figure 6.1a, the X-axis indicates the number of CPU intensive VMs are running on the system. In Figure 6.1b, the Y-axis represents the number of memory-intensive VMs are concurrently running on the same system. Then in Figure 6.2a, the X-Y plane located between those two axes represents the execution time variation due to CPU and memory intensive VMs combination. It is also true for other two resource combinations of the figure. Effect of CPU-I/O intensive resource combination in Figure 6.2b is presented with X-Z plane. Lastly, the Y-Z plane represents the memory-I/O intensive resource combination in Figure 6.2c.

Figures 6.1 and 6.2 show the objective of the experiments with the tasks of this chapter. Figure 6.1 shows that the execution time variation data is profiled for three resources separately. Prediction models are trained using this data. Then these models are used to predict the execution time variation of three resource combinations shown in Figure 6.2. Thus, the prediction models are trained with three sets of two dimensional execution time variation data and then used to predict the values in three-dimensional space.

6.2.1 Number of co-located VMs on multiple servers

As discussed in the previous section, a server can have different types of resource intensive VMs. Moreover, a data center houses many virtualized servers. Figure 6.3 depicts a typical situation in a data center. The data center may be operating many servers, and different kinds of hypervisors may be deployed on them. At different points in time, the servers may have different numbers of co-located VMs.

Figure 6.3 shows that at three different times the three servers have different numbers of simultaneously running VMs. For example, at time t_{i-1}, the Xen is running two VMs, and at t_i the number of VMs is increased to three. Similarly, the number of co-located VMs changes over time on the XenServer and ESXi hypervisors, too. Each of the co-located VMs has specific resource requirement and duly affect the performance of neighboring VMs. Depending on what type of resource intensive co-located VMs are running a task would take a different amount of time to be completed.

The CPU intensive co-located VMs would have one kind of effect on a CPU intensive task. Their effect would be different on a memory intensive task than an I/O intensive task. Different types of hypervisors and system configurations also have to be taken into account.

Two issues need to be addressed while analyzing the VM consolidation permanence. First one is to formulate a method to record the VM performance variations due to consolidation systematically. The second is to find a way to predict this execution time variation. Recall that, Chapter 4 introduces a methodology for profiling and predicting the ETV, called the ICBM [43]. In Chapter 4 the experimental results of ICBM

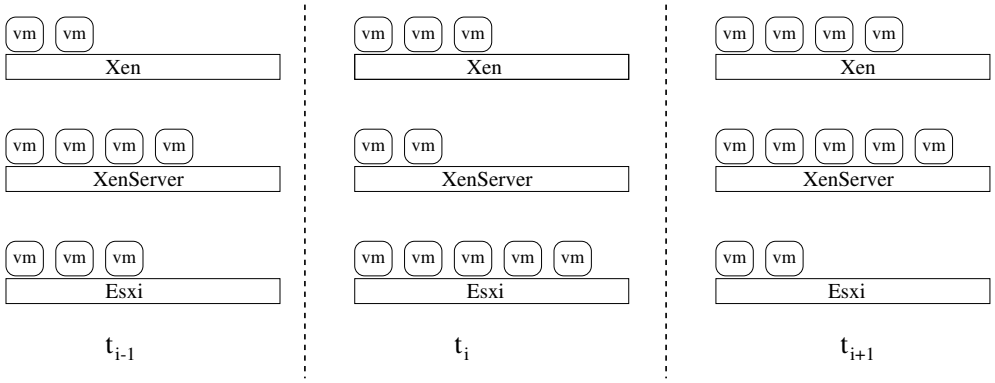


Figure 6.3: Multiple hypervisors with various numbers of simultaneously running VMs.

are shown for one hypervisor, Xen. In this chapter, the ICBM has been extended to multiple hypervisors.

This chapter shows that the ICBM can be applied not only to compare the performance of different stages of a consolidated hypervisor, also to make the comparison among multiple hypervisors. This work presents extended experimental results for three hypervisors. Thus showing that the ICBM can be used in a broader virtualization environment.

It is difficult to profile a virtualized system manually. A server may be running a different number of VMs with different amount of workloads at different points of time [323]. Recall that, Chapter 5 presents a framework for scheduling VMs on multiple hypervisors. In this chapter, the methodology from Chapter 4 (ICBM) and the framework from Chapter 5 are combined to profile and predict the ETV of VMs on multiple hypervisors.

6.3 Methodology

This section uses the methodology of ICBM, which is introduced previously in Chapter 4 and also used in Chapter 5. This chapter shows that the methodology can be extended to multiple hypervisors. A parallel application can have many different data flow paths, and an application can be decomposed into a set of tasks. The tasks need to be scheduled individually on the VMs. This work lays the ground for understanding the effect of consolidation at the VM level.

In the experimental setup, all VMs of a server has the same configuration. One of them is designated as the target VM (v_t), while rest are designated as (v_{co}). The v_t has been used to run different tasks and record their ETV. On the other hand, v_{co} have been used to create resource contention on the host collectively. As co-located VMs with different resource types are added to the server, the cumulative resources consumption create pressure on the server. The performance of VMs starts to degrade with the increase in the number of co-located VMs. The ETV data profiled from the above experiments demonstrate how the execution time of v_t varies on multiple hypervisors due to consolidation. In the experiments, each combination of v_{co} is considered as one workload. The co-located VM combinations can be changed to create new server workload.

6.3.1 Use of benchmark suites in the experiments

Various benchmarks suites are utilized in the ICBM. These benchmark suites are the result of detail analysis of commercially successful applications [222]. They are inspired by an interesting phenomenon that, the applications spend both 80% of their time and resources, executing just 20% of the code [234]. The syntactic benchmark suites are carefully crafted to mimic such essential parts, rather than running the entire application [222].

During consolidation, the VMs compete for shared server resources. As a result, the performance of each VM is affected. To get the real picture of consolidation; it is necessary to do the experiments with different resource consumption patterns. It is achieved by changing the number and type of benchmark on the co-located VMs. Each pattern of co-located VMs is a workload for the server.

The benchmark suites can be run in any order or quantity on the co-located VMs; thus, a wide variety of workloads can be created. The number of simultaneously running benchmarks of each type can be increased or decreased to create resource contention of a specific type. Thus, the overall server workload for a specific type of resource can be adjusted. The benchmarks suites are used here to get finer control over the server resource usage.

6.3.2 Benchmarks used

Three different categories of benchmarks have been used throughout the dissertation; they are CPU, memory and I/O intensive benchmarks.

Three CPU intensive benchmarks have been used in the experiments. They are the *Sysbench CPU test*, *Nbench* and *Unixbench*. Three memory intensive benchmarks have been used in the experiments, Those benchmarks are the *Cachebench*, *Stream* and *Sysbench memory* test. Three I/O intensive benchmarks have been used as well, they are the *Dbench*, *Filebench* and *Iozone*. Recall that the benchmarks are already described in Section 3.4 of Chapter 3.

6.3.3 Experimental setup

The experiments have been conducted with three Dell XPS-8500 systems. Each machine has one Intel i7-3770 processor and 32GB of RAM. The i7-3770 processor has

four cores and eight hardware threads, each clocked at 3.4 GHz.

Three different hypervisors have been used to compare the consolidation performance among them. They are the VMware ESXi 5.5, XenServer 6.5, and Xen 4.6. All the hypervisors have been deployed independently with identical VM settings.

Each hypervisor has 15 VMs setup, one to act as v_t, while the rest are v_{co}. Each VM centos 6 as guest OS, have one logical CPU, 2 GB of RAM and 50 GB of virtual disk space. The VMs is not pinned to any logical cores. A total of 30 GB RAM is allocated to the 15 VMs, and rest is available for the hypervisor. All the benchmarks are installed on VMs beforehand; a concurrent java application manages all benchmarks, VMs, and physical servers from a remote node.

Now, that the experimental setup and the benchmarks used are discussed in the next few sections, the experimental results are going to be discussed. Three sets of results are presented for three hypervisors; VMware ESXi, XenServer, and Xen. The prediction models derived from the ETV data is also presented in a later section. The discussion of experimental results starts with the ETV data of VMware ESXi hypervisor in the next section.

6.4 Execution time variation on Esxi

In this and following sections, the task execution time variations due to VM consolidation are discussed in details. The task execution time variation of tasks on VMs was the theme of the last few chapters. This section is just an extension of the works done previously in Chapters 3, 4, and 5.

The first set of experimental data presented here is collected from the VMware ESXi hypervisor. The ETV of v_t due to various types of workloads are shown in Figures 6.4, 6.5, and 6.6. ETVs of different resource-intensive tasks on ESXi is discussed in successive sections. ETVs of three CPU-intensive tasks are presented and discussed in the next section and Figure 6.4 shows the ETV data for those three CPU-intensive tasks.

Similarly, Sections 6.4.2, and 6.4.3 discuss the ETV of memory, and I/O intensive tasks on the ESXi hypervisor, respectively. The ETV of the three memory intensive tasks on VMs is shown in Figure 6.5. Finally, Figure 6.6 shows the ETV of the three I/O intensive tasks.

6.4.1 Execution time variation (ETV) of CPU intensive tasks

Figure 6.4 shows the ETV of CPU-intensive tasks on ESXi hypervisor for six different workloads.

In each graph of Figure 6.4, the X-axis shows the total number of VMs running on the system, while the Y-axis presents the task completion time of v_t. The execution times are profiled using ICBM as described in the Chapter 4.

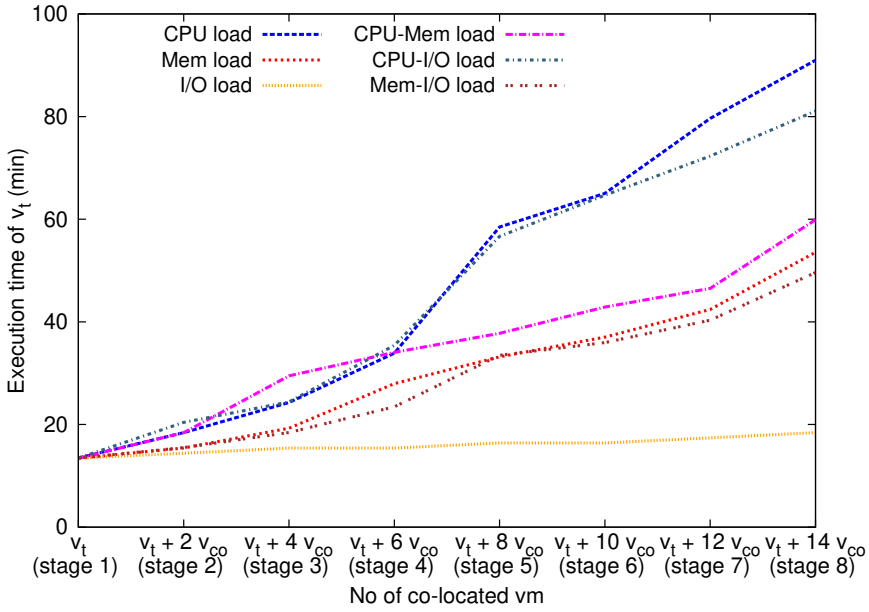
The first point of each graph is the execution time of v_t when it was run alone on the server (Stage 1). At stage 1, v_t is free from any interference from co-located VMs (v_{co}). As it is explained in Chapter 4, two v_{co} are added to the server at successive stages. Therefore, the successive points of X-axis represent the execution time of v_t with increasing amount of resource contention. In our experiential setup, the final stage has fourteen simultaneously running v_{co} besides the v_t.

In each graph, from left to right along the X-axis the interference from co-located VMs steadily increase. The first point of each graph gives the execution time of a task without any interference from co-located VMs. On the other hand, the last point gives the execution time with maximum interference. The results of this section demonstrate that different types and numbers of v_{co} can make the task execution time to vary at a different rate. Next, the ETV due to three categories of benchmarks on the ESXi is discussed. The discussion starts with TEV due to CPU-intensive co-located VMs.

Three graphs of Figure 6.4 show the ETV of three CPU intensive benchmarks for various types of resource intensive co-located VMs on ESXi hypervisor. Each graph shows results for six workload types; they are CPU, memory, I/O, CPU-memory, CPU-I/O and memory-I/O workload.

As the number of co-located VMs increase the execution time of target VM is increased, too. Thus, the execution time variations due to six different types of workload are shown in each graph. It shows that each workload on type v_{co} affects the task execution time uniquely.

Figure 6.4a shows the ETV of Nbench for the six types of workloads of v_{co}. Recall that Nbench is a CPU intensive benchmark. For this benchmark suffers the most performance degradation due to the CPU intensive workloads on v_{co}. Without any interference, the Nbench takes 13.48 minutes to execute. On the other hand, Nbench takes 90.99 minutes to execute with fourteen other co-located VMs running fourteen CPU intensive benchmarks. That, the execution times go from 13.48 to 90.99 minutes



(a) ETV of the Nbench, for six different workloads.

Figure 6.4: ETV of three CPU intensive tasks on the ESXi hypervisor.

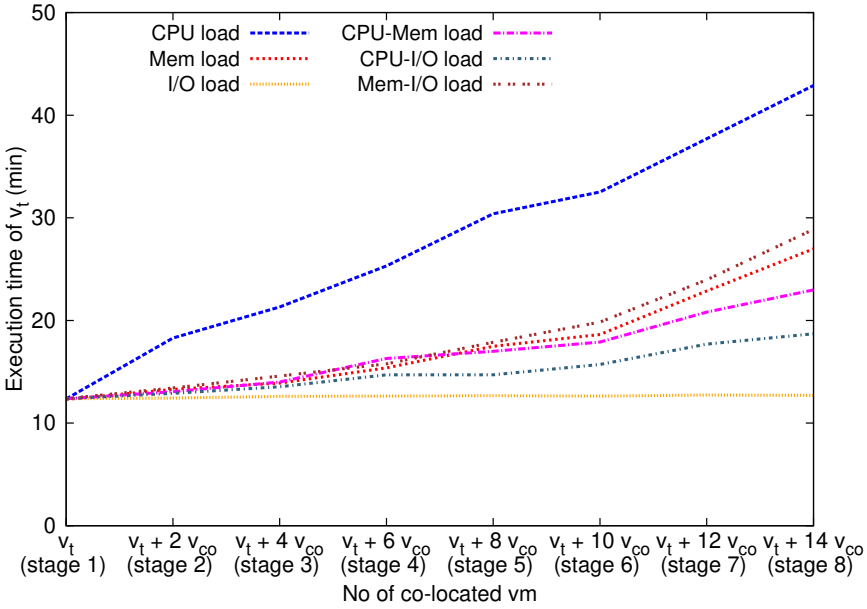
due to resource contention, this is a 575% increase in execution time.

Without any interference from any co-located VM, the Nbench takes 13.48 to execute on v_t. On the graph of Figure 6.4a, it is the leftmost point. As the co-located VMs are added to the server the execution time begins to extend. At the rightmost point, the v_t is simultaneously running with 14 more v_{co} and each co-located VM runs one CPU intensive benchmark. For this system configuration, the execution time due to the maximum interference is 90.99 minutes.

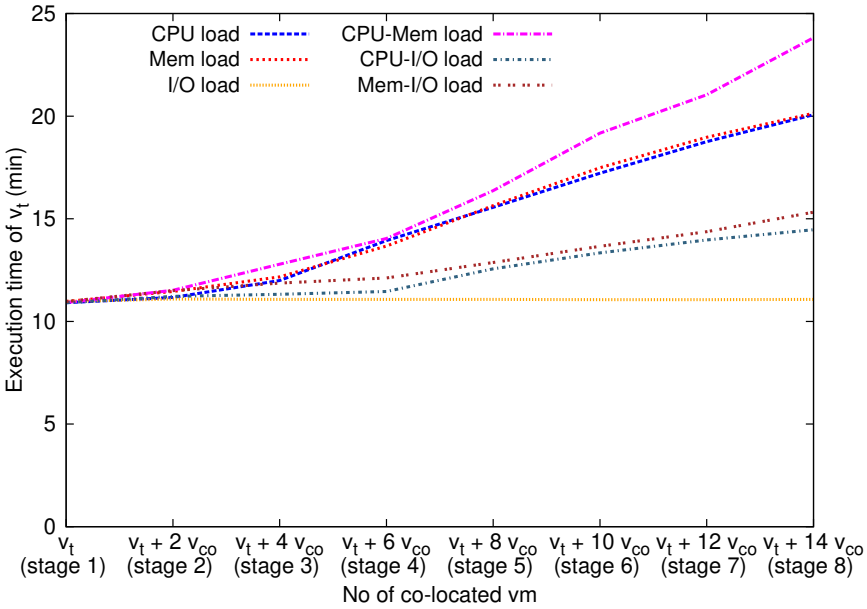
On the other hand, the Nbench shows much less variation for other types of workloads. For example, in the case of I/O workload, the variation of execution time is just 0.91%. Thus, CPU-intensive tasks have low variation due to I/O intensive workload.

Figure 6.4b shows ETV of the Unixbench, which is another CPU intensive benchmark; it also shows the maximum performance degradation for CPU intensive workload. On the other hand, it has low ETV for I/O intensive workload. For CPU intensive workload, the execution time variation is 247% (execution time goes from 12.36 min to 42.89 min). While for I/O intensive workload it is only 2.58%. The results show that the CPU intensive tasks have maximum ETV due to CPU intensive workload. On the other hand, ETV is the minimum due to I/O intensive workloads.

This work is a continuation of works of previous chapters. As results of Chapter 3



(b) ETV of the Unixbench, for six different workloads.



(c) ETV of the Sysbench CPU test, for six different workloads.

Figure 6.4: ETV of three CPU intensive tasks on the ESXi hypervisor (continued).

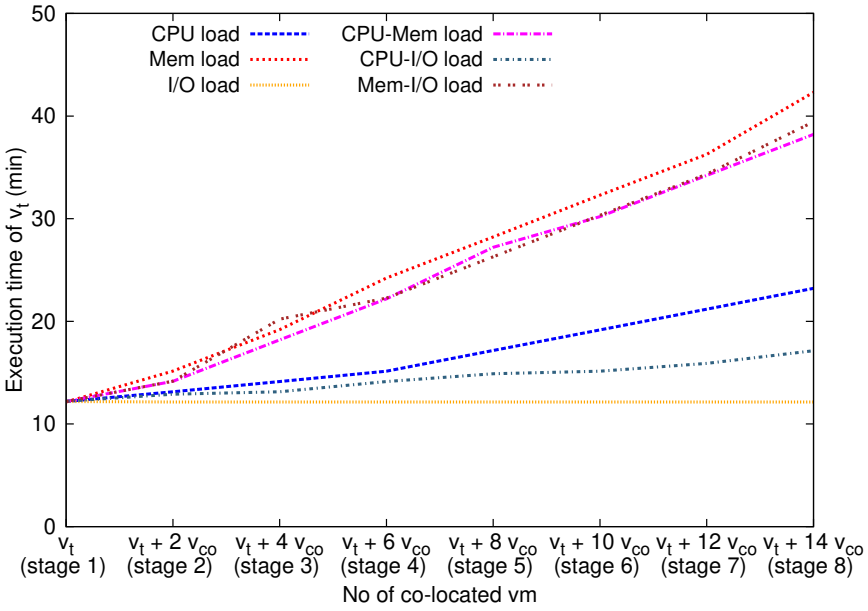
show different types of resource intensive workload affects the performance of consolidated VMs; as a result, the task execution times begin to vary. Furthermore, Chapter 4 discusses how different types of resource intensive co-located VMs have a different effect on the task execution time.

As shown in the Chapter 4, if the target (v_t) and co-located VMs (v_{co}) have the similar type of resource requirement then the performance degrades quickly. On the other hand, if they have dissimilar resource requirements, then the effect on performance is much less severe. That is why a CPU intensive task has much greater execution time degradation due to CPU intensive workload. The same task has much better performance when consolidated with I/O intensive workload. Thus, the results of this section are consistent with the results of the previous sections.

6.4.2 Execution time variation (ETV) of memory intensive tasks

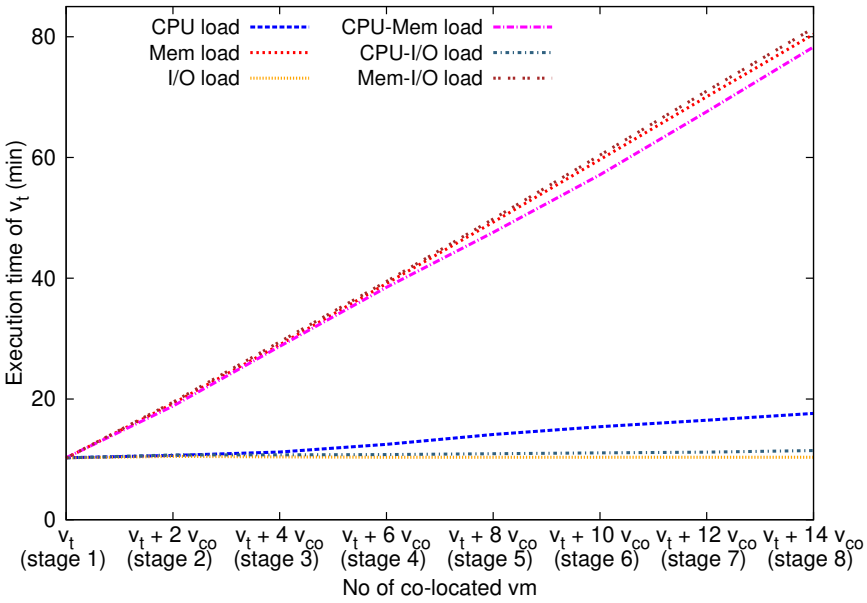
The ETV variation due to memory intensive tasks is shown in Figure 6.5. In this case, three memory tasks are the Cachebench, Stream and Sysbench memory test.

Figure 6.5a shows the ETV of Cachebench for six different workloads just as shown in the previous graphs. The Cachebench suffers the maximum change of execution

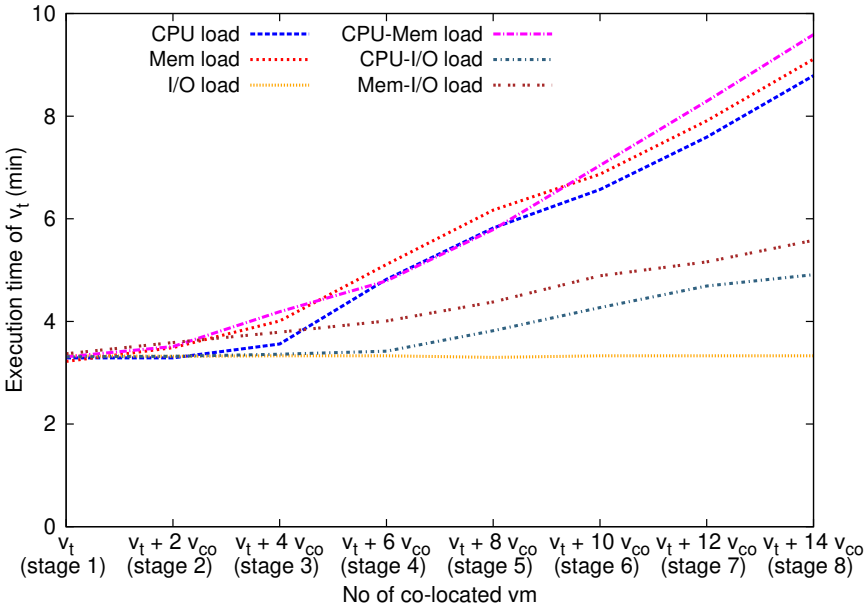


(a) ETV of the Cachebench, for six different workloads.

Figure 6.5: ETV of three memory intensive tasks on the ESXi hypervisor.



(b) ETV of the Stream CPU test, for six different workloads.



(c) ETV of the Sysbench memory test, for six different workloads.

Figure 6.5: ETV of three memory intensive tasks on the ESXi hypervisor (Continued).

time due to memory intensive workload of v_{co}. The execution time rises from 12.14 to 43.31 minutes, and it is a 248.51% increase in execution time.

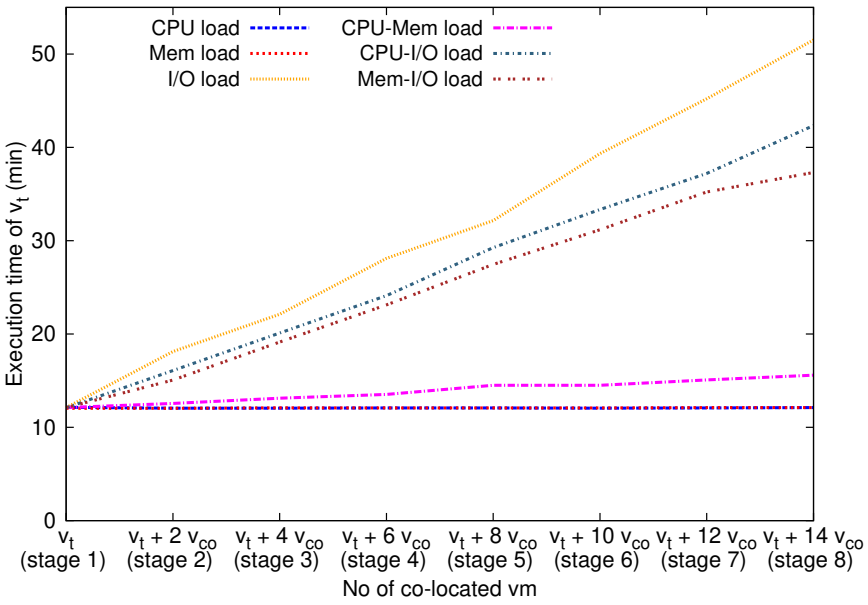
Afterward, Figure 6.5b shows the ETV of Stream for six different workloads. Stream also shows greatest ETV due to Memory-I/O intensive workload, and it is 698.33%.

Next, Figure 6.5c shows the task execution time variation for the Sysbench memory test. In this case, the highest variation happens due to CPU-Memory intensive workload, and it is 189.72%. In all three cases, least amount of variation is shown for I/O intensive workloads.

6.4.3 Execution time variation (ETV) of I/O intensive tasks

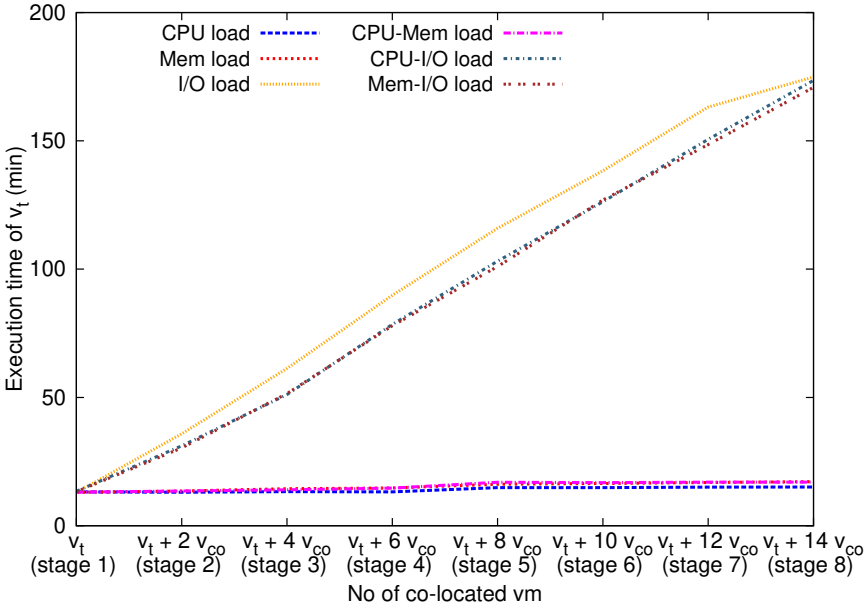
Finally, Figure 6.6 shows the ETV of three I/O intensive tasks on Exsi server; they are the Dbench, Iozone, and Filebench. In general, the benchmarks suffer the most performance degradation for I/O intensive v_{co}. On the other hand, they suffer the least for CPU and memory intensive v_{co}.

Firstly, Figure 6.6a shows the ETV of the Dbench for six different types of workload. Among the six, the performance degradation is worse for I/O intensive workload;

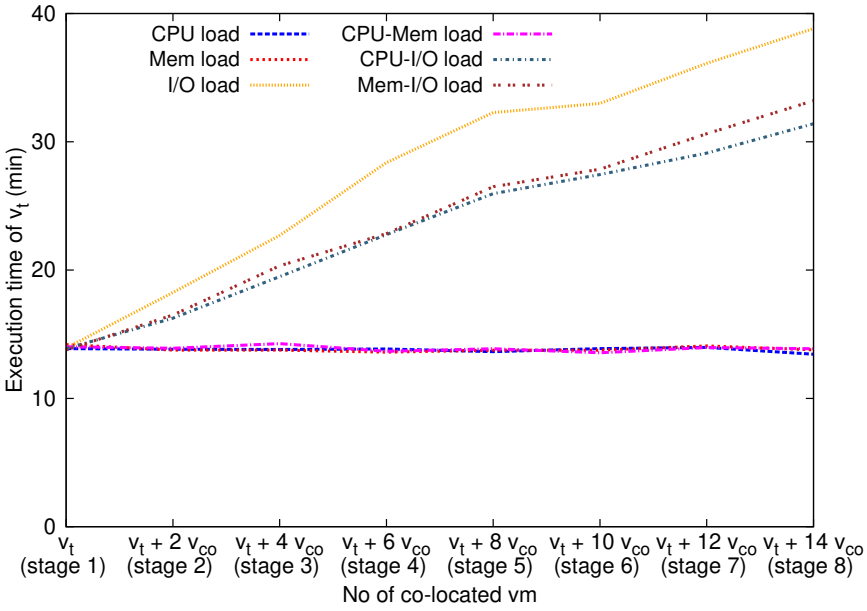


(a) ETV of the Dbench, for six different workloads.

Figure 6.6: ETV of three I/O intensive tasks on the ESXi hypervisor.



(b) ETV of the Iozone, for six different workloads.



(c) ETV of the Filebench, for six different workloads.

Figure 6.6: ETV of three I/O intensive tasks on the ESXi hypervisor (Continued).

it is 325%. Next, Figure 6.6b shows the ETV for the Iozone. In this case, also, I/O intensive workload shows the worse performance degradation, it is 1250.34%. Lastly, the Figure 6.6c shows the performance degradation of Filebench for six different workloads. It can be readily seen, that the performance is worse for I/O intensive workload and it is 178.67%.

This section discussed the ETV of tasks on ESXi hypervisor for six different types of workload. Next section, discusses the ETV of the same tasks and workloads on the XenServer hypervisor.

6.5 Task execution time variation on XenServer

This section presents the experimental results collected from the XenServer hypervisor. The ETV data of VMs for the XenServer is shown here in the same format as it is shown for ESXi hypervisor in the previous section. In this section, also, the experimental results show similar trends as discussed in the previous section; therefore, those discussions about similar patterns are not repeated.

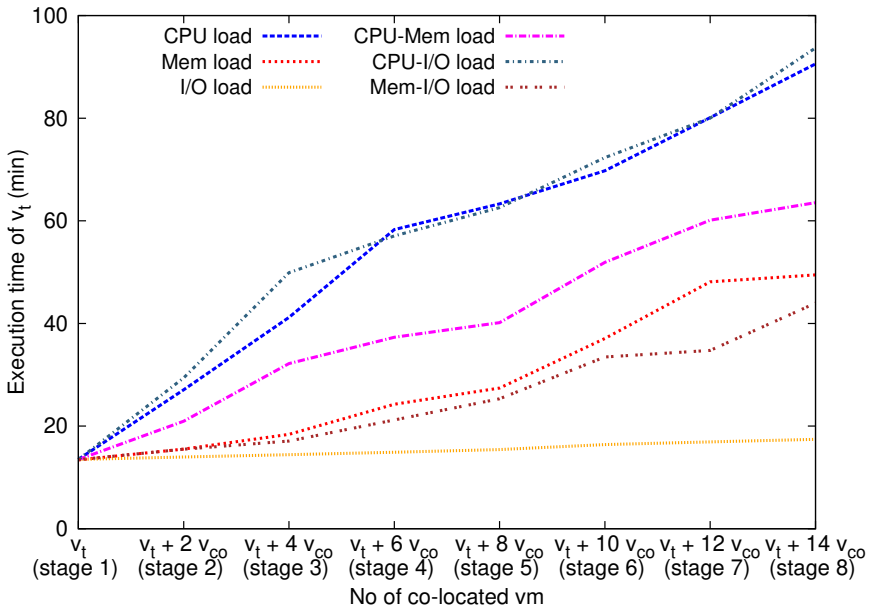
In this case, also, there are three sets of results for benchmarks of three primary resource groups. Three graphs of Figure 6.7, show how three CPU intensive tasks react to various resource workload types. Figures 6.8, and 6.10 show the execution time variations of memory, and I/O intensive tasks, respectively.

6.5.1 Execution time variation (ETV) of CPU intensive tasks

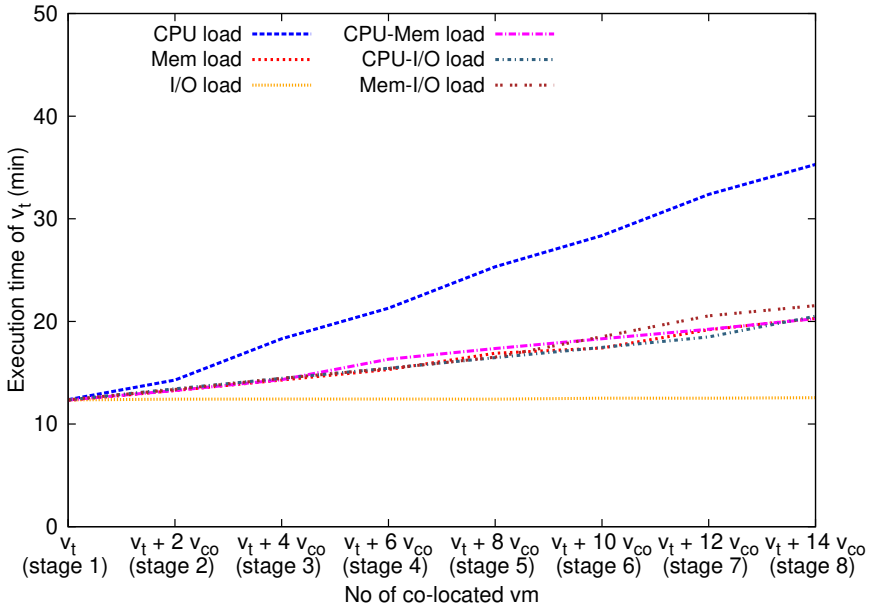
Figure 6.7 shows the ETV of three CPU intensive tasks for six different workloads each. These tasks are, the Nbench, Unixbench and Sysbench CPU test, and six workloads are CPU, memory, I/O, CPU-Memory, CPU-I/O and Memory-I/O.

The Y-axis shows the execution time variations, while the X-axis shows the number of simultaneously running VMs on the server. Figure 6.7a shows that the Nbench has most execution time variation for both CPU and CPU-I/O workloads.

When the Nbench is run alone on the server, it takes 13.54 minute to execute. On the other hand, when consolidated with 14 other CPU intensive VMs, it takes 90.57 minutes to execute. In other words, the Nbench shows 568.90% increase in execution time due to CPU intensive workload. Similarly, for CPU-I/O intensive workload, the

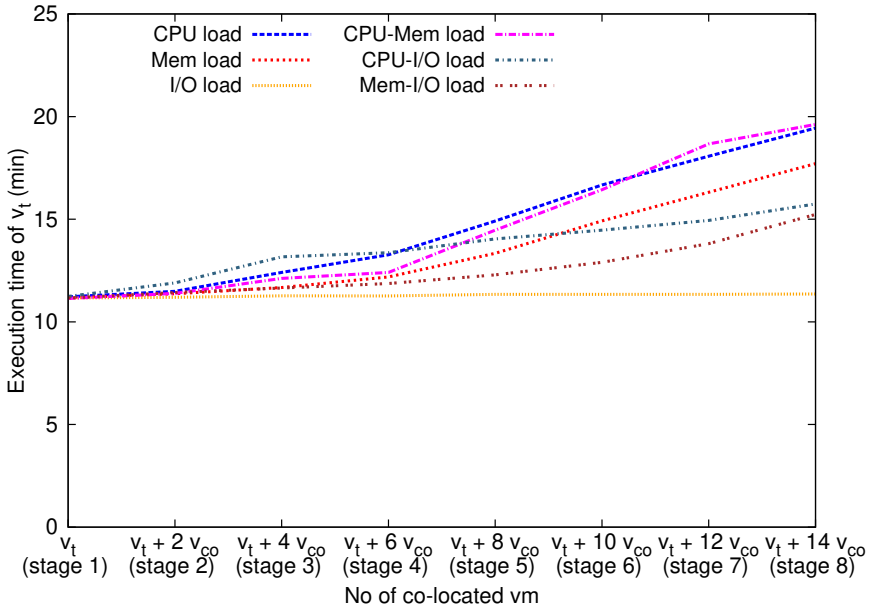


(a) ETV of the Nbench, for six different workloads.



(b) ETV of the Unixbench, for six different workloads.

Figure 6.7: ETV of three CPU intensive tasks on the XenServer hypervisor.



(c) ETV of the Sysbench CPU test, for six different workloads.

Figure 6.7: ETV of three CPU intensive tasks on the XenServer hypervisor (Continued).

Nbench has 594.51% increase in execution time. In contrast, the Nbench shows almost no execution time variation for I/O only workload.

Other two CPU intensive benchmarks also show similar variation patterns. That is, they have the highest degree of variation for CPU intensive workloads, while almost no variation due to I/O intensive workload.

Figure 6.7b shows that the Unixbench have 185.05% increase in execution time for the CPU workload; the execution time goes from 12.38 minute to 35.29 minutes. Figure 6.7c shows that the Sysbench CPU test has 73.26% increase in execution time for the similar case. However, both of the benchmarks show almost no variation for I/O intensive workload.

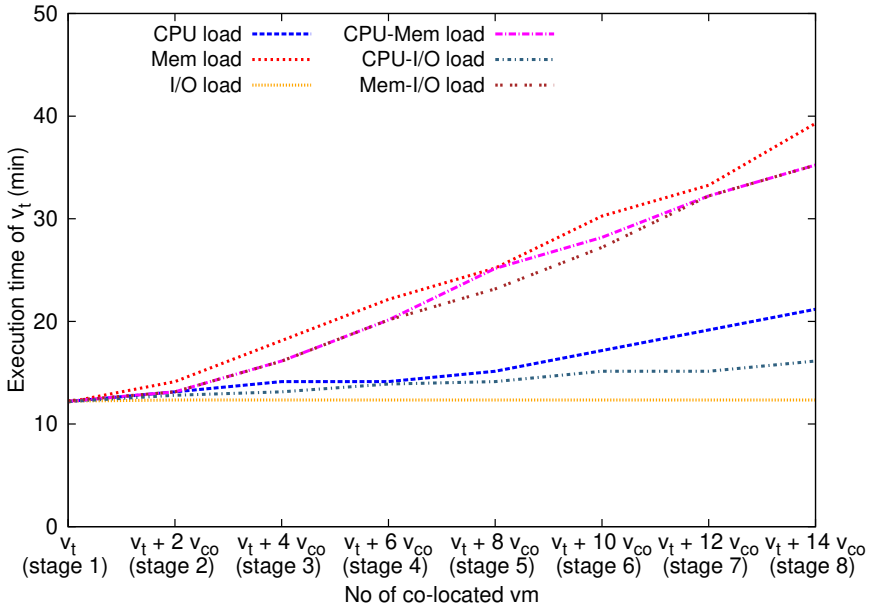
6.5.2 Execution time variation (ETV) of memory intensive tasks

Figure 6.8 shows the ETV of three memory intensive task for the same six workloads. The memory intensive benchmarks, are the Cachebench, Stream and Sysbench memory test. In this case, the three tasks show the highest amount of variation due to memory intensive workload and least for I/O intensive workload.

Figure 6.8a shows the ETV of Cachebench due to various workloads. For memory workload, the Cachebench has an execution time increase of 223.64%. The Cachebench takes 12.14 minutes to execute when it is run alone on the server; when consolidated with 14 other memory intensive VMs it takes 39.29 minutes to complete execution. On the other hand, the Cachebench shows almost no task execution time variation for I/O intensive workloads.

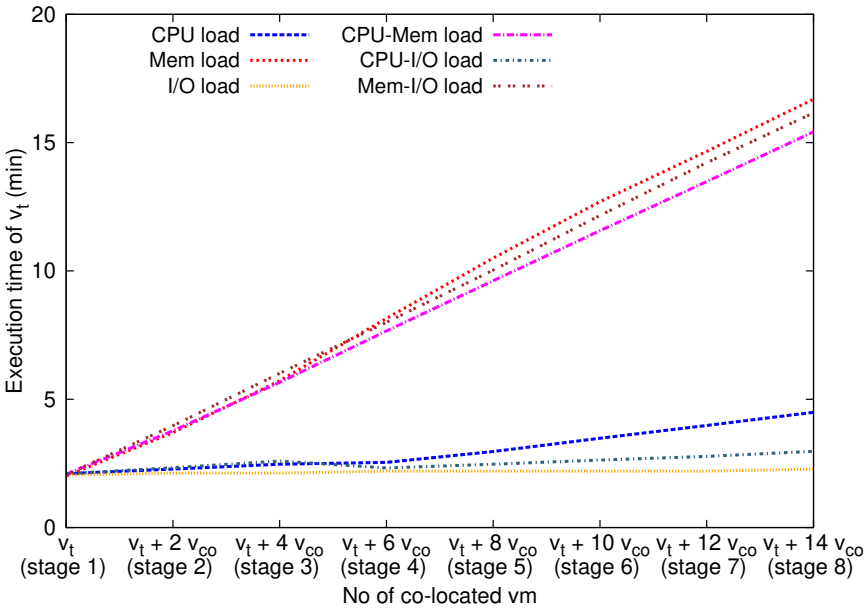
Next, the ETV of Stream benchmark is shown in Figure 6.8b. It shows that the Stream has 730.34% increase in execution time for memory intensive workload.

Next, Figure 6.8c shows the execution time variation of Sysbench memory test. In this case, both the CPU and CPU-Memory intensive workloads have the maximum amount of variation of task execution time. Here, the CPU intensive workload causes the execution time to increase by 140.30%, while the CPU-Memory intensive workloads cause 142.12%. For the above two benchmarks, the I/O intensive workloads cause the least amount of execution time variation.

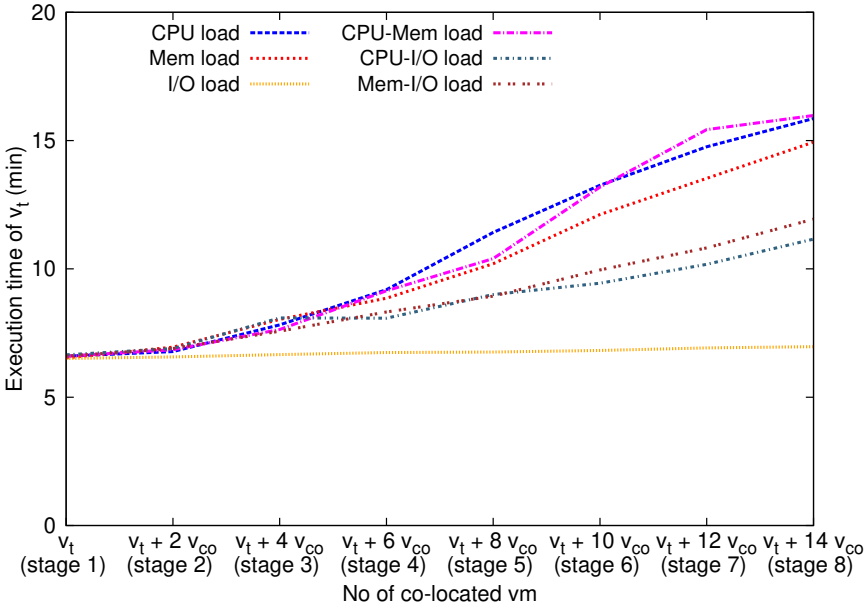


(a) ETV of the Cachebench, for six different workloads.

Figure 6.8: ETV of three memory intensive tasks on the XenServer hypervisor.



(b) ETV of the Stream, for six different workloads.



(c) ETV of the Sysbench memory test, for six different workloads.

Figure 6.8: ETV of three memory intensive tasks on the XenServer hypervisor (Continued).

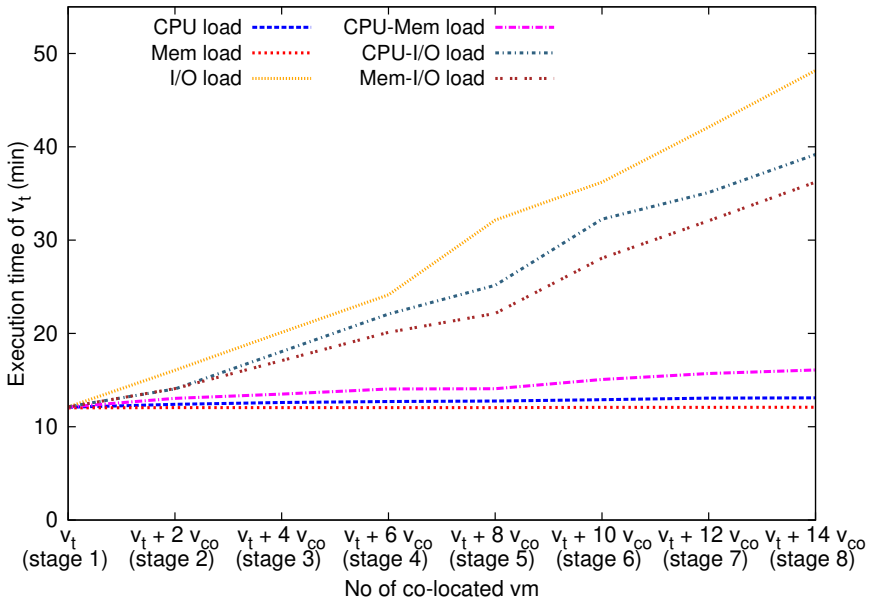
6.5.3 Execution time variation (ETV) of I/O intensive tasks

Next, Figure 6.10 shows the execution time variations of three I/O intensive tasks for six workload types. The three I/O intensive task are, the Dbench, Iozone, and Filebench. The experimental results show the similar patterns as discussed in the previous section and chapters. Therefore, those parts of the discussion are not repeated here.

First, the Figure 6.10a shows task execution time variation of the Dbench. It has the highest amount of variation for I/O intensive workload, and it is 297.77%. In this particular case execution time increases from 12.11 minutes to 48.17 minutes. Next, Figure 6.10b shows that the Iozone has 1455.57% increase in task execution time due to I/O intensive workload.

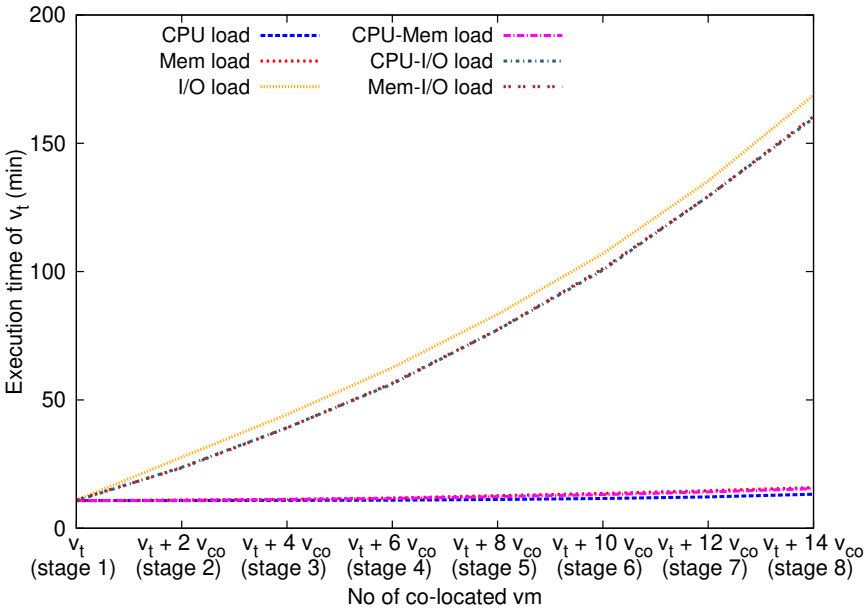
Similarly, the Filebench has 188.19% increase in execution time due to I/O workload. Thus, I/O workload causes the maximum amount of variation for all three benchmarks. On the other hand, CPU, and memory intensive workloads do not have much effect on these three tasks.

The next section discusses the ETV of the same tasks for another hypervisor, Xen.



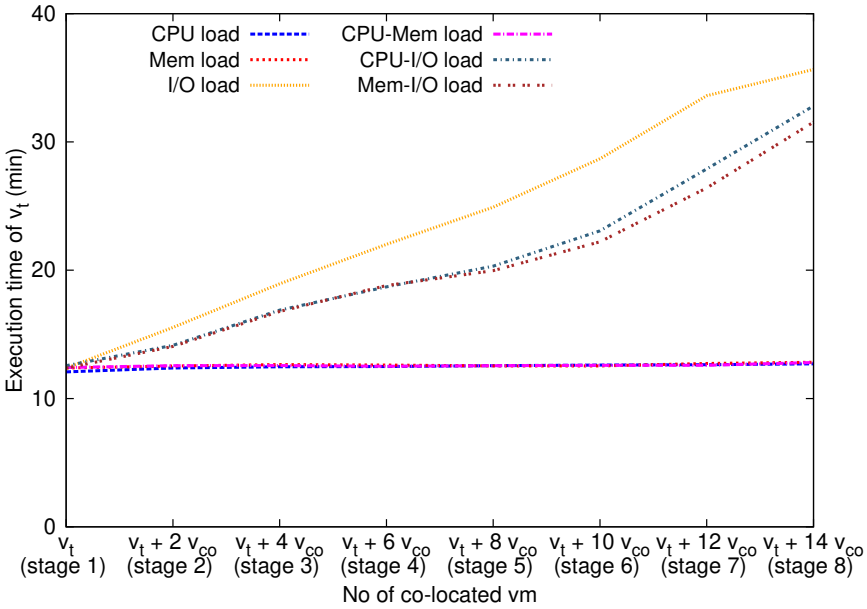
(a) ETV of the Dbench, for six different workloads.

Figure 6.9: ETV of three I/O intensive tasks on the XenServer hypervisor.



(b) ETV of the Iozone, for six different workloads.

Figure 6.9: ETV of three I/O intensive tasks on the XenServer hypervisor.



(c) ETV of the Filebench, for six different workloads.

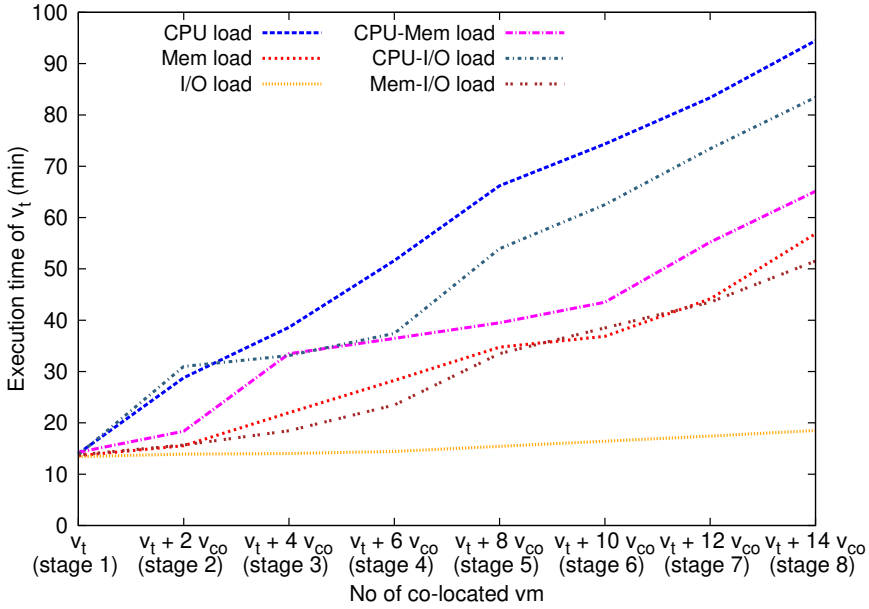
Figure 6.10: ETV of three I/O intensive tasks on the XenServer hypervisor (Continued).

6.6 Task execution time variation on Xen hypervisor

In this section, the experimental results for the Xen hypervisor are presented. Presented results for Xen hypervisor are in the same form as the results presented for the last two hypervisors in the previous sections. In this case, the same methodology and benchmark suites have been used as it is discussed in the last few sections and chapters. The results presented here also so similar patterns as discussed for the last two hypervisors (ESXi and XenServer). Therefore, those parts of the discussion are not repeated in this section.

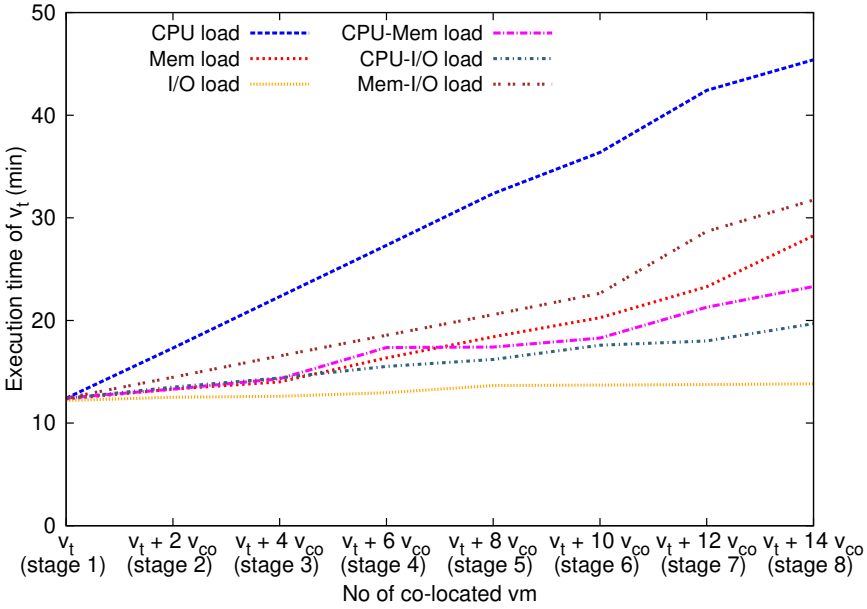
The ETV of the tasks on the Xen are shown in Figures 6.11, 6.12, and 6.13; the same tasks are used here. Figure 6.11 shows the ETV of three CPU intensive tasks on Xen; they are the Nbench, Unixbench, and Sysbench CPU test. Figure 6.12 shows the ETV of three memory intensive task for the same six workloads. They are the Cachebench, Stream, and Sysbench memory test. Finally, Figure 6.13 shows the ETV of three I/O intensive tasks; they are the Dbench, Iozone, and Filebench.

The tasks exhibit patterns similar to that observed in both ESXi and XenServer. Therefore, their ETVs are not discussed separately. In the next section, the maximum execution time variation percentage of each task is calculated and compared.

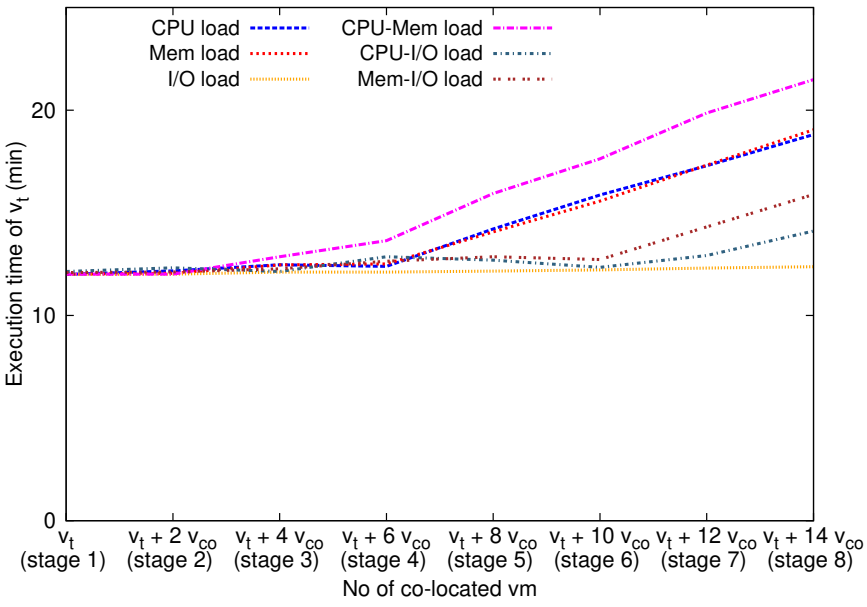


(a) ETV of the Nbench, for six different workloads.

Figure 6.11: ETV of three CPU intensive tasks on the Xen.

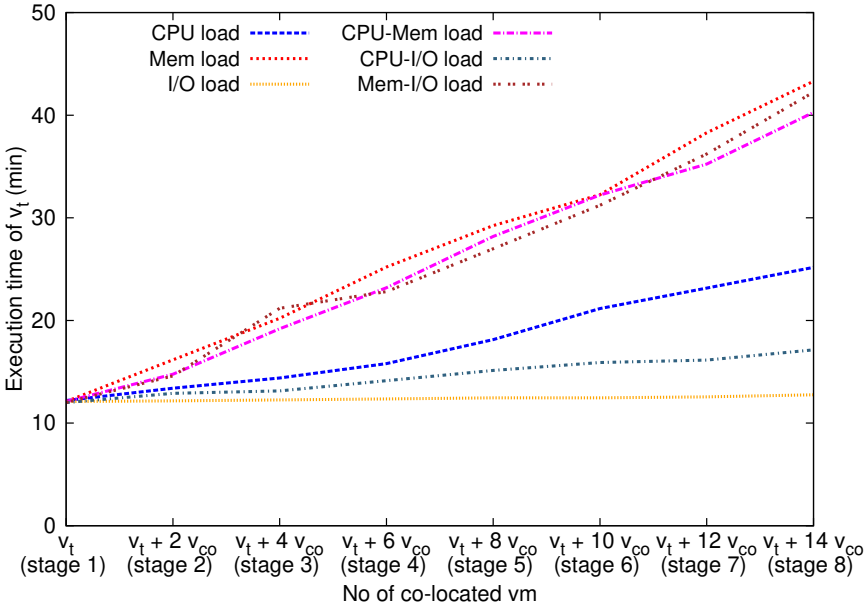


(b) ETV of the Unixbench, for six different workloads.

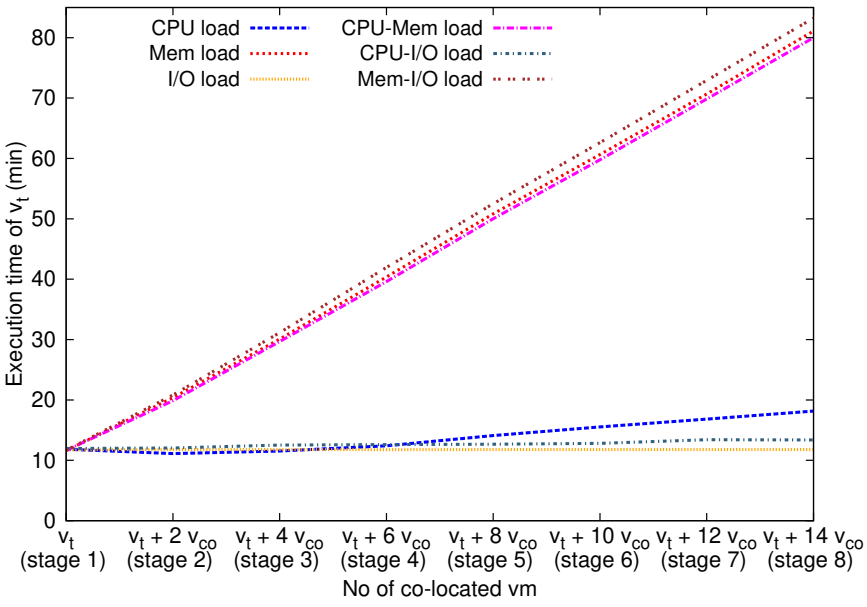


(c) ETV of the Sysbench CPU test, for six different workloads.

Figure 6.11: ETV of three CPU intensive tasks on the Xen (continued).

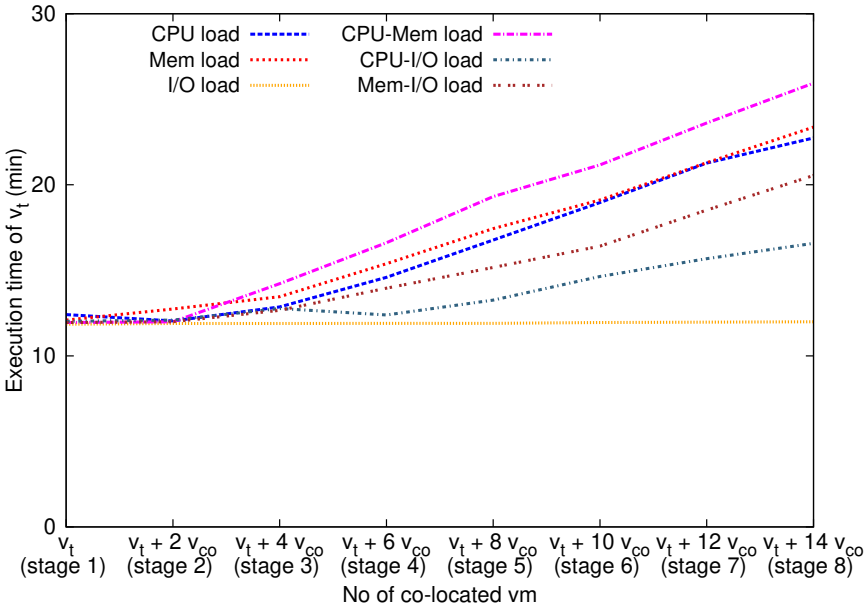


(a) ETV of the Cachebench, for six different workloads.



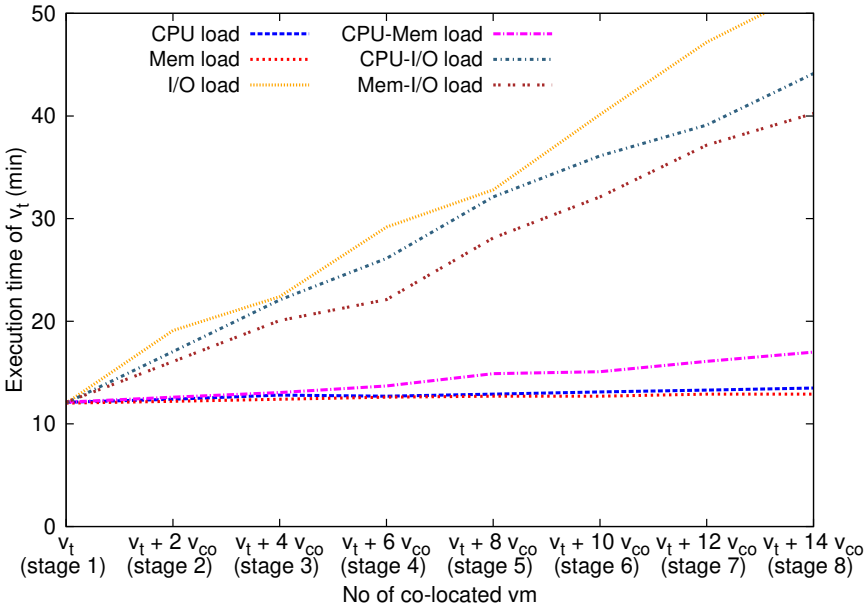
(b) ETV of the Stream CPU test, for six different workloads.

Figure 6.12: ETV of three memory intensive tasks on the Xen.



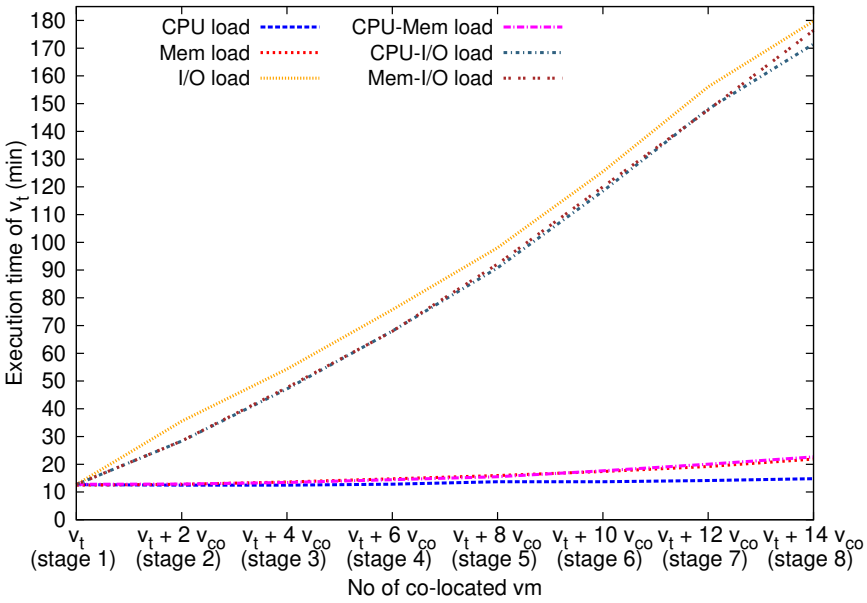
(c) ETV of the Sysbench memory test, for six different workloads.

Figure 6.12: ETV of three memory intensive tasks on the Xen (Continued).

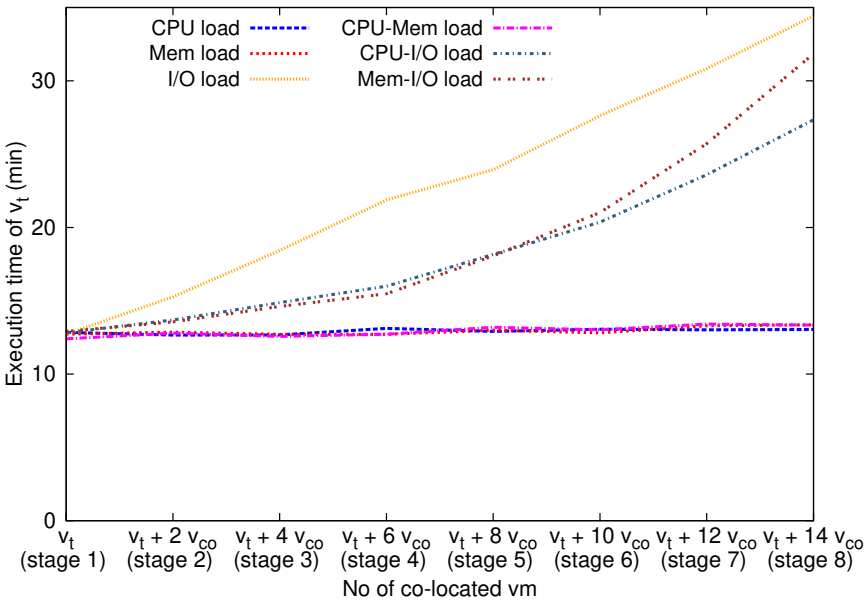


(a) ETV of the Dbench, for six different workloads.

Figure 6.13: ETV of three I/O intensive tasks on the Xen.



(b) ETV of the Iozone, for six different workloads.



(c) ETV of the Filebench, for six different workloads.

Figure 6.13: ETV of three I/O intensive tasks on the Xen (Continued).

6.7 The maximum execution time variation (ETV) percentage on three hypervisors

Tables 6.1, 6.2, and 6.3 show the maximum execution times variation the tasks on ESXi, XenServer, and Xen hypervisors, respectively. Those tables use the results presented in the three previous sections. The experimental data of Sections 6.4, 6.5, and 6.6 are used to calculate the values in Tables 6.1, 6.2, and 6.3. Thus, this section is a continuation of the discussion about the ETV results of the last three sections.

The maximum execution time variation is the difference between the initial execution time and the maximum execution time of a task. The task is run alone on the server and execution finish time is the initial execution time. Then the number of co-located VMs are increased on the system; as a result, the performance of the task begins to

Table 6.1: Maximum execution time variation percentage of tasks for six workload types on ESXi hypervisor.

Benchmark name	CPU intensive workload (%)	Mem. intensive workload (%)	I/O intensive workload (%)	CPU-Mem intensive workload (%)	CPU-I/O intensive workload (%)	Mem-I/O intensive workload (%)
SB CPU test	**83.95**	83.59	0.91	118.13	32.63	39.65
Nbench	**575.00**	299.32	36.77	345.75	502.37	268.57
Unixbench	**247.00**	119.17	2.58	85.62	51.05	133.01
Cachebench	91.81	**248.51**	0.41	213.53	40.37	223.54
Stream	71.56	688.33	1.07	663.15	11.80	**698.23**
SB Mem test	167.17	182.91	0.90	**189.72**	48.33	65.57
Dbench	0.91	0.58	**325.00**	28.63	250.20	207.83
Iozone	14.29	32.12	**1250.34**	29.79	1191.29	1190.48
Filebench	2.75	19.71	**200.23**	6.91	126.29	140.92

degrade. When the maximum number of co-located VMs are running on the server the execution finish time of a task is the maximum execution time. The difference between the final execution time and the initial time is the maximum execution time variation.

In the ETV graphs of above three sections (Sections 6.4, 6.5, and 6.6), the far left point on the X-axis represents the initial execution time, while the far right point represents the maximum execution time. Their difference is the maximum execution time variation. For example, in Figure 6.4a initial execution time of Nbench is 13.48 minutes and maximum execution time is 90.99 minutes. Therefore, the maximum execution time variation of Nbench due to CPU intensive workload is 77.51 minutes.

The maximum execution time variation percentage calculated from the ratio of the maximum execution time variation and initial execution time. Thus, the maximum execution time variation percentage of Nbench for CPU intensive workload is 575.00%,

Table 6.2: Maximum execution time variation percentage of tasks for six workload types on XenServer hypervisor.

Benchmark name	CPU intensive workload (%)	Mem. intensive workload (%)	I/O intensive workload (%)	CPU-Mem intensive workload (%)	CPU-I/O intensive workload (%)	Mem-I/O intensive workload (%)
SB CPU test	73.26	58.54	1.79	**76.12**	40.03	36.46
Nbench	568.90	268.90	28.94	470.37	**650.12**	226.35
Unixbench	**185.05**	65.09	1.69	64.26	65.93	74.13
Cachebench	73.83	**223.64**	0.81	188.37	32.40	187.04
Stream	113.80	**730.34**	11.21	641.34	41.42	684.46
SB Mem test	140.30	128.79	7.06	**142.12**	67.81	80.90
Dbench	8.27	0.58	**297.77**	32.86	222.73	198.76
Iozone	22.79	47.70	**1455.57**	43.26	1387.70	1356.08
Filebench	5.13	11.71	**188.19**	3.14	161.14	154.97

which is calculated as follows:

$$\begin{aligned} &((90.99 - 13.48)/13.48 \times 100) \\ &= (77.51/13.48 \times 100) \\ &= 575.00 \end{aligned}$$

In this way the maximum execution time variation percentage of each task is calculated for each workload and shown in the Tables 6.1, 6.2, and 6.3.

The maximum execution time variation percentage data allows comparing how the task execution time varies from the initial value due to consolidation. The three tables show a similar pattern; usually, a type of resource intensive task is most affected by

Table 6.3: Maximum execution time variation percentage of tasks for six workload types on Xen hypervisor.

Benchmark name	CPU intensive workload (%)	Mem. intensive workload (%)	I/O intensive workload (%)	CPU-Mem intensive workload (%)	CPU-I/O intensive workload (%)	Mem-I/O intensive workload (%)
SB CPU test	51.20	57.22	3.08	**78.04**	16.22	31.43
Nbench	**573.82**	317.63	38.20	356.48	505.95	274.41
Unixbench	**265.45**	129.01	13.36	87.37	58.66	155.14
Cachebench	106.31	**257.93**	5.45	232.80	42.83	251.70
Stream	53.29	598.62	0.51	587.19	14.55	**613.89**
SB Mem test	83.23	94.91	2.21	**115.09**	37.93	69.89
Dbench	11.48	7.32	**332.50**	40.18	266.91	232.28
Iozone	16.93	75.38	**1318.62**	76.06	1251.37	1289.99
Filebench	1.55	5.03	**170.81**	3.89	113.26	146.63

the same type of resource intensive workload. For example, the second column of Table 6.1 shows the maximum execution time variation percentage of nine tasks due to CPU-intensive workload. The three CPU intensive tasks, SB CPU test, Nbench, and Unixbench have the values 83.95%, 575.00%, and 247.00%, respectively. Those three values are highlighted; as they are the highest in the respective rows. In other words, CPU intensive tasks show maximum variation due to CPU-intensive workload.

The maximum execution time variation percentages of three memory intensive tasks are shown in fifth to seven rows. The three memory intensive tasks are Cachebench, Stream, SB Mem test. The highest percentage in each row is highlighted in bold. It can be seen that the maximum percentages are either for memory intensive workloads or a combination of memory-intensive workloads. For example, the highest percentage of the Cachebench is 248.51%, and it is for memory intensive workload. For the Stream, the highest value is 698.23%, and it is for memory and I/O workload combination.

Similarly, all three tables show that the CPU intensive tasks have the highest execution time variation usually for a CPU-intensive workload. On the other hand, memory-intensive workload shows the maximum amount of performance degradation for memory-intensive or combination of memory-intensive workloads and so on.

Following sections present the ETV prediction results and models; the models are derived from the ETV data discussed in the above sections.

6.8 Prediction with the Least Square Regression (LSR)

This section describes how the least square regression (LSR) method is utilized to build prediction models from the data obtained in the previous sections. Earlier, ETV data from three separate hypervisors are presented in Sections 6.4, 6.5, and 6.6. In this section, those data are used to build prediction models. Training and testing of the prediction models have been done with separate data sets, and there is no overlap between them.

In this section, it is explained how all those data are fitted to build the LSR prediction models. This section describes how the data set is arranged for training and testing and used in the subsequent sections for building and testing prediction models.

Recall that in Chapter 4 all the processes related to LSR has been thoroughly discussed. In Section 4.7, prediction process with LSR has been discussed in detail. In

Section 4.9, the process of obtaining parametric models from R statistical packages has been explained. Furthermore, Section 4.9.3 describes how to estimate model coefficients from the R statistical packages.

All data used during training and testing of LSR models are already presented in Sections 6.4, 6.5, and 6.6. In those sections, the ETV of nine benchmarks from three different resource intensive groups are discussed. For each set of experiments two separate hypervisors are used, they are the ESXi and XenServer. For example, Figures 6.4 and 6.7 show the ETV of three CPU intensive tasks on the Vmware ESX and XenServer, respectively. Similarly, Figures 6.5 and 6.8 show the ETV data of three memory intensive tasks, respectively. Figures 6.6 and 6.10 show the ETV of three I/O intensive tasks, respectively.

Each graph of the above figures shows data for six different workload types. For example, each graph of Figures 6.4 and 6.7 shows six different workload data. One single graph shows data for one task only; three of them are basic resource workloads, while the other three are combinational workloads. The basic workload types are CPU, memory and I/O. Three resource combinational workloads are, CPU-Memory, CPU-I/O, and Memory-I/O.

Data from above mentioned graphs (Figures 6.4, 6.7, 6.5, 6.8, 6.6, and 6.10) are used in the following sections for training and building prediction models. Data for each graph is divided into two categories; basic and combinational. The three resource combinations are used as the targets of three prediction models. On the other hand, the ETVs due to three basic resources are used as the inputs.

Following sections show how this data is used to build the models. The sections also discuss different prediction scenarios and accuracy. Finally, in Section 6.10 prediction models are built as the culmination of the works done in the following sections. Based on the works of following sections, three models built and they are shown in the Equations 6.1, 6.2 and 6.3. However, to understand the models the discussion of the following sections is essential.

The prediction models in Equations 6.1, 6.2 and 6.3 are created with the data from both the VMware ESX and XenServer hypervisors. Thus, the models presented in that section are unified models for both hypervisors. Later in the Section 6.10.1, it is also shown that the unified models can be used to predict the task execution time for random VM workloads. Thus, this chapter gives a section by section description of how the prediction models are built and tested for accuracy.

The ETV data of nine tasks are divided into two separate sets. One set contained six tasks data and used for the model training. Another set has three task data, which used for testing. The training set contains the Nbench, Unixbench, Cachebench, Sysbench memory test, Dbench, and Iozone. Three other tasks used for testing are the Sysbench CPU test, Stream, and Filebench. For each task, data is collected from all three hypervisors. Three sets of experiments have been done.

The ETVs data of six training tasks for three basic workloads are used as the model inputs. Three basic workloads are being CPU, memory, and I/O. Three resource combinations are used as the model outputs. The model is trained with multiple inputs and target data. Afterward, the models have been used to predict execution time variation for the combination of resource workloads.

For testing, three sperate tasks are used, they the Sysbench CPU test, Stream, and Filebench. Their ETV data for CPU-memory, CPU-I/O, and memory-I/O resource combinations have been predicted. Testing and training are done with entirely separate sets of data. Not only separate sets of tasks also separate hypervisor ETV data is used. Data from ESXi and Xen hypervisor have been used to train prediction models.

Once, the LSR models are built various model parameters are also obtained. The prediction results are discussed in the next section.

6.9 Prediction results for LSR

The ETV data for ESXi, XenServer, and Xen are discussed in the previous section. One example of input data from the ESXi hypervisor used for training can be seen in the Figure 6.4a. Figure 6.4a shows six different workloads ETV data of the Nbench on ESXi hypervisor; three are primary resources, and the other three are the combination of resources. To build a model for CPU-Memory intensive ETV prediction the ETV due to three primary resources (CPU, memory and I/O) are used as the inputs, while ETV due to CPU-Memory workload has been used as the target.

In the same way, to build the prediction model for CPU-I/O the same three primary resources ETV data are used as the inputs. However, in this case, the ETV due to CPU-I/O workload has been used as the target. A model for predicting the ETV due to memory-I/O workload is also created similarly.

The ETV data from XenServer is also fed to the model during the training phase.

One example of such input data from the XenServer can be seen in Figure 6.7a. Similarly, data from Xen is also used.

The accuracy of the built models is then tested against three separate test task data. An example of the test task for ESXi hypervisor is shown in Figure 6.4c. As usual, it has six different workloads related ETV data.

The predictions compared against the actual data. For example, to compare the prediction output for CPU-memory combination; three ETV data of the Sysbench CPU test (due to CPU, memory and I/O workload) are used as three inputs, respectively. On the other hand, the output from the model is compared with the actual CPU-memory ETV data of sysbench CPU test. Similarly, the accuracy of the other two models (CPU-I/O and memory-I/O) are also tested for accuracy.

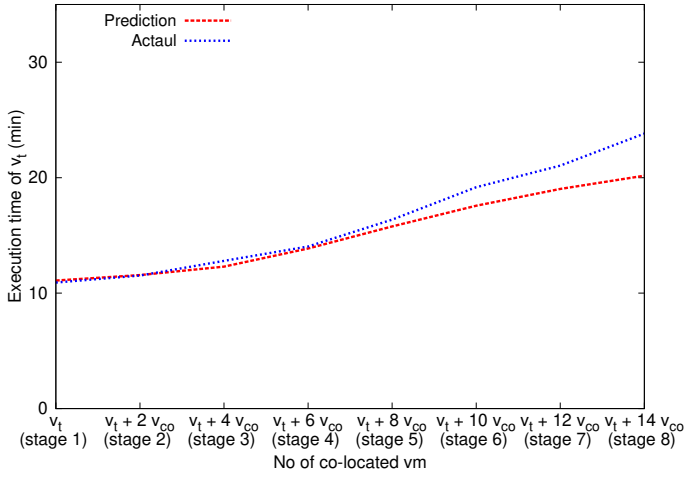
6.9.1 The prediction results for the ESXi

The prediction models are built with all three hypervisor data. That is the ETV data from ESXi, XenServer, and Xen is used to build a unified model. Then, the accuracy of the models is tested separately against two hypervisors, ESXi and XenServer. In this section prediction results for ESXi are presented.

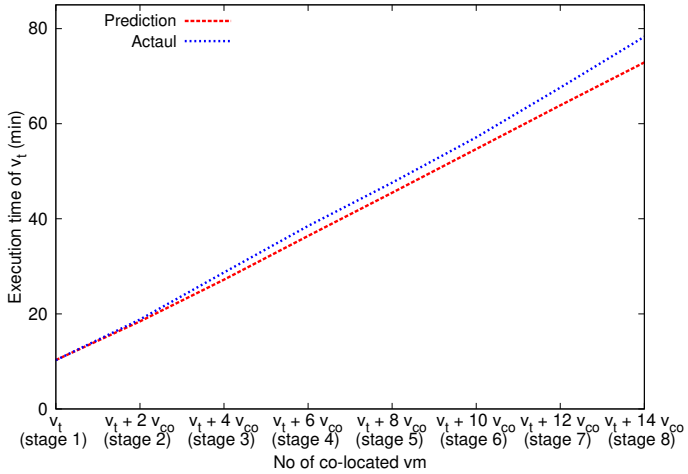
The prediction results for three resource combinations of ESXi hypervisor (CPU-memory, CPU-I/O and memory-I/O), for three test benchmarks (Sysbench CPU test, Stream, and Filebench) are shown separately in Figures 6.14, 6.15, and 6.16.

Figures 6.14 shows the three prediction results for CPU-memory resource combination on ESXi hypervisor. Figure 6.14a shows the prediction results for Sysbench CPU test for CPU and memory workload combination. The Sysbench is a CPU intensive task. The actual execution time variation is shown in blue, while the predicted execution time is shown in red. The prediction is made with the LSR models built from ETV data as described above.

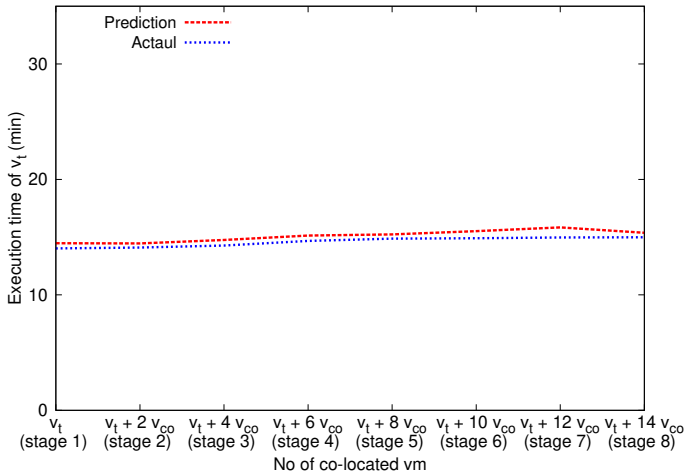
Next, Figure 6.14b shows the execution time variation prediction for the Stream. The Stream is a memory intensive task. Lastly, Figure 6.14c shows the execution time variation prediction of the Filebench for CPU and memory workload combination. The Filebench is an I/O intensive task. Each graph shows the ETV due to actual resource combination (shown in blue) and prediction from the respective models (shown in red). Thus, the three graphs of Figure 6.14 show the prediction results for one CPU, one memory, and one I/O intensive task, each.



(a) Prediction for the Sysbench CPU test.

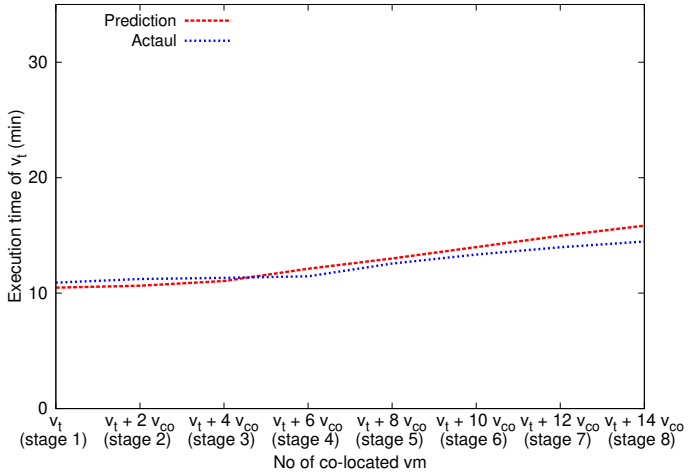


(b) Prediction for the Stream.

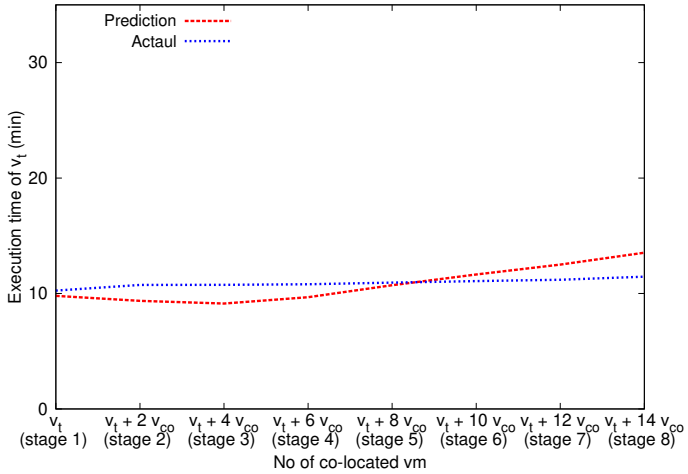


(c) Prediction for the Filebench.

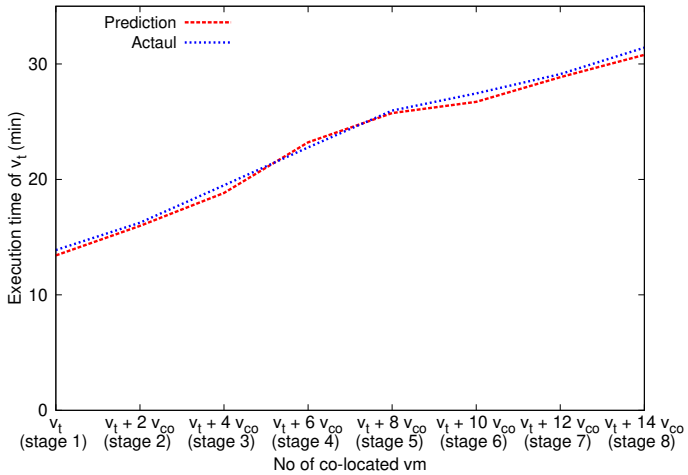
Figure 6.14: Prediction results for CPU-Memory workload combination on ESXi.



(a) Prediction for the Sysbench CPU test.

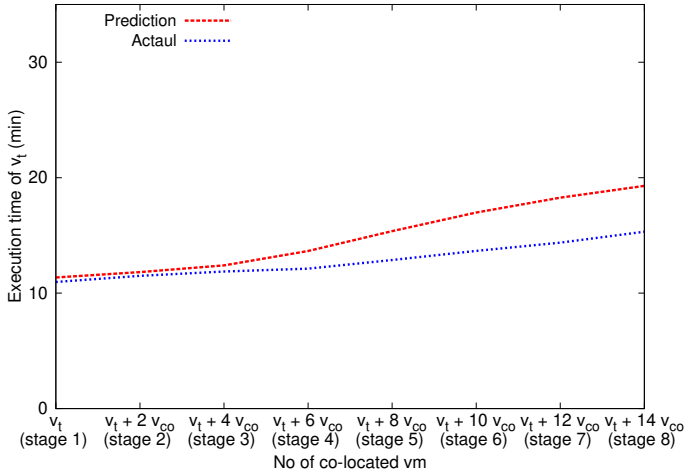


(b) Prediction for the Stream.

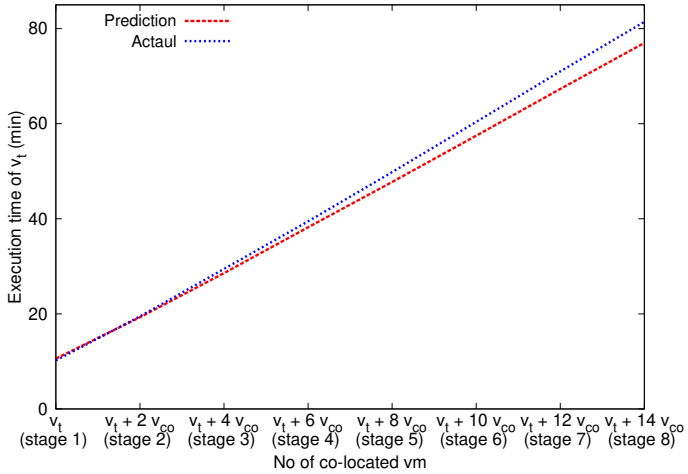


(c) Prediction for the Filebench.

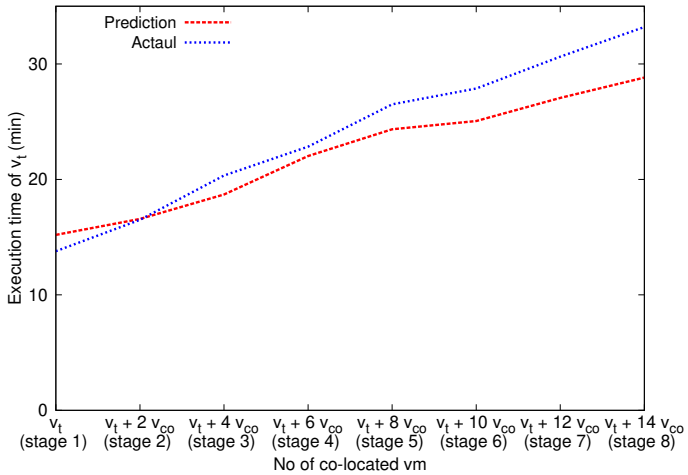
Figure 6.15: Prediction results for CPU-I/O workload combination on ESXi.



(a) Prediction for the Sysbench CPU test.



(b) Prediction for the Stream.



(c) Prediction for the Filebench.

Figure 6.16: Prediction results for memory-I/O workload combination on ESXi.

Similarly, the next two figures show the prediction for two other resource combinations, CPU-I/O and Memory-I/O. Figure 6.15 shows the task ETV prediction for CPU and I/O workload combination. Prediction for three task are shown, they are the Sysbench CPU test, Stream, and Filebench. Figure 6.15 shows the execution time variation prediction of the same three tasks for memory-I/O load combination.

6.9.2 The prediction results for the XenServer

In this section prediction results are presented for XenServer hypervisor. Figures 6.17, 6.18, and 6.19 show nine prediction results with the same tasks for the XenServer. The results are presented in the same format as above. That is, results from three resource combinations for three types of tasks (nine graphs in total) are shown separately.

Three resource combinations are CPU-memory, CPU-I/O, and memory-I/O, they are shown in Figures Figures 6.17, 6.18, and 6.19, respectively. Here also, each graph shows two data sets; one actual execution time variation due to resource workload combination and one predicted from LSR model. Again, the actual ETV is shown in blue, while the predicted ETV is shown in red.

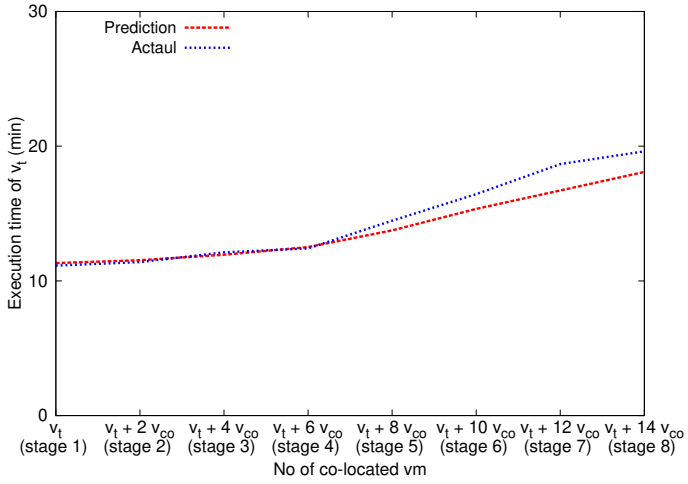
Previously, Section 6.8 described how the LSR prediction models built using the methodology introduced in Chapter 4. The LSR prediction models here are created by following the same methodology; therefore, it is not discussed here agagin.

Prediction results for the three sperate resource combinations are shown in the figures. Figure 6.17 shows the prediction results for the CPU-memory workload combination. Next, Figure 6.18 shows the predictions results for CPU-I/O workload combination. Lastly, Figure 6.19 shows the predictions results for memory-I/O workload combination.

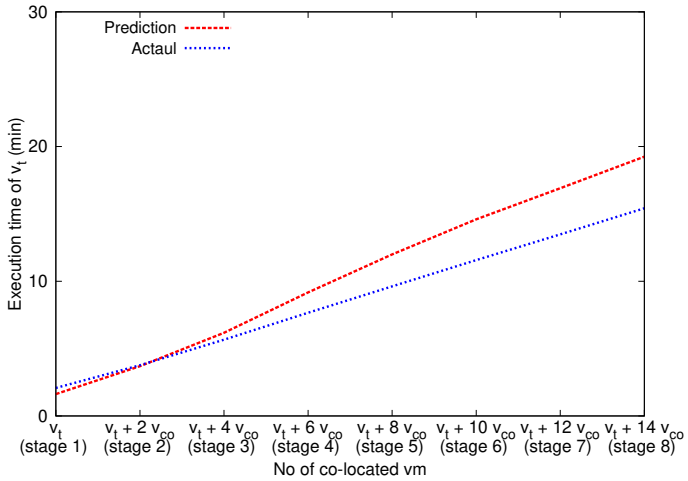
Each figure shows the prediction results for the same three tasks, which were used in the previous section for making predictions on ESXi. The prediction results are similar to that of the ESXi hypervisor discussed in the previous section; therefore, the discussion is not repeated here.

Similarly, predictions are also made on the Xen hypervisor using the same procedure and tasks. The LSR prediction results for the Xen are also similar to that of ESXi and XenServer. Therefore, those results are not presented here.

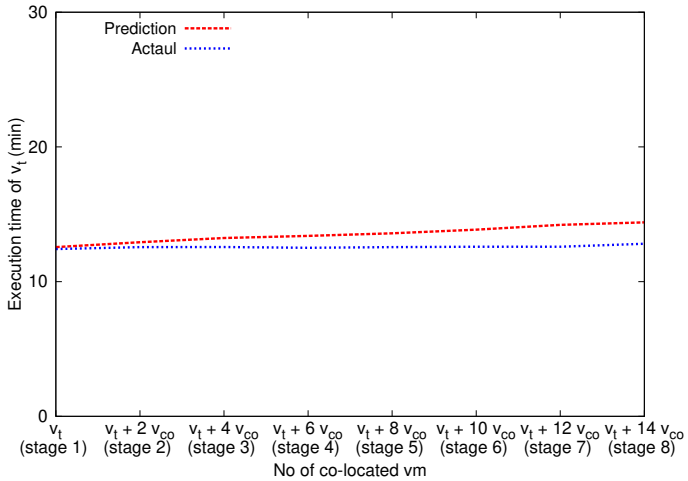
The accuracy of the prediction models in this and previous section are also calculated. The next section discusses the accuracy of the predictions by the models.



(a) Prediction for the Sysbench CPU test.

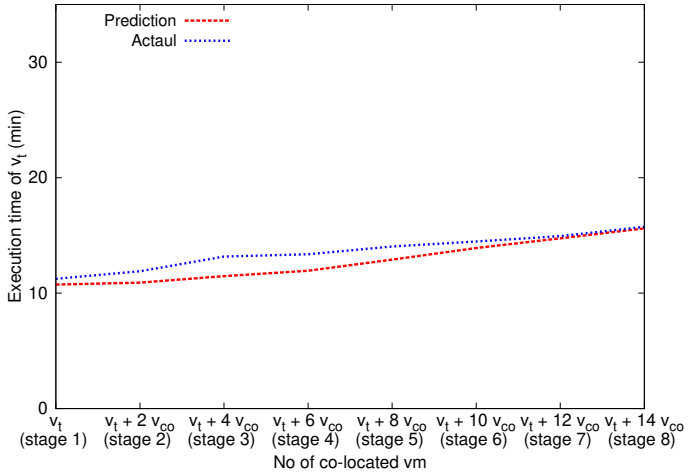


(b) Prediction for the Stream.

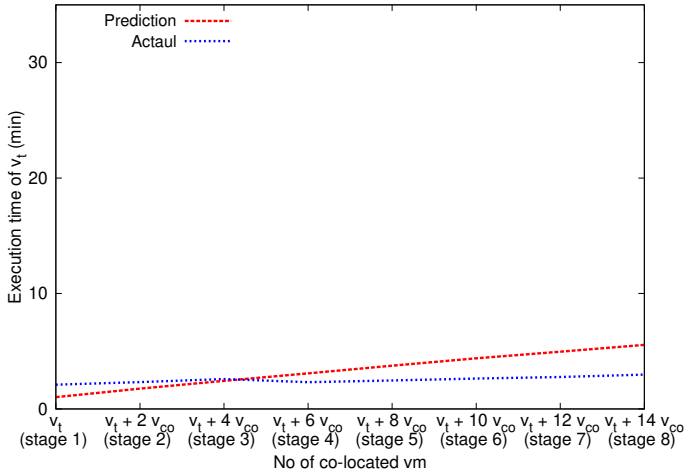


(c) Prediction for the Filebench.

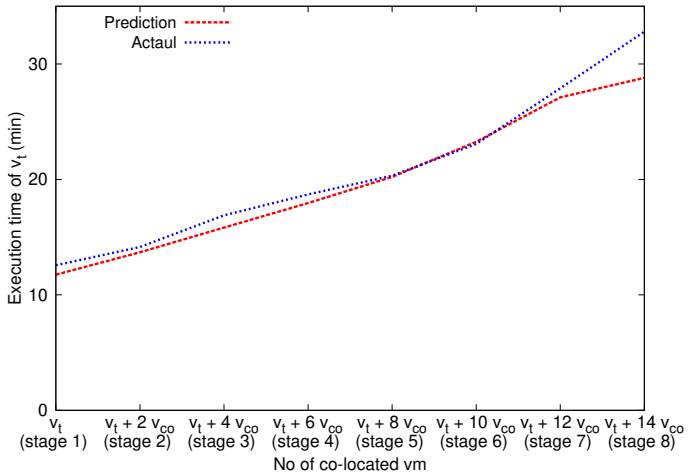
Figure 6.17: Prediction results for CPU-Memory workload combination on XenServer.



(a) Prediction for the Sysbench CPU test.

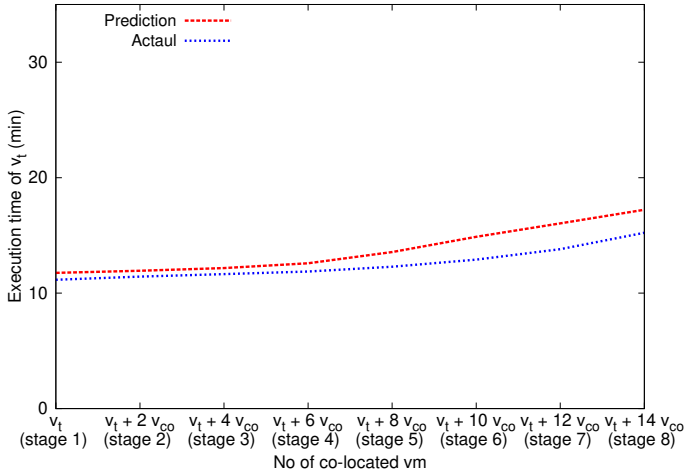


(b) Prediction for the Stream.

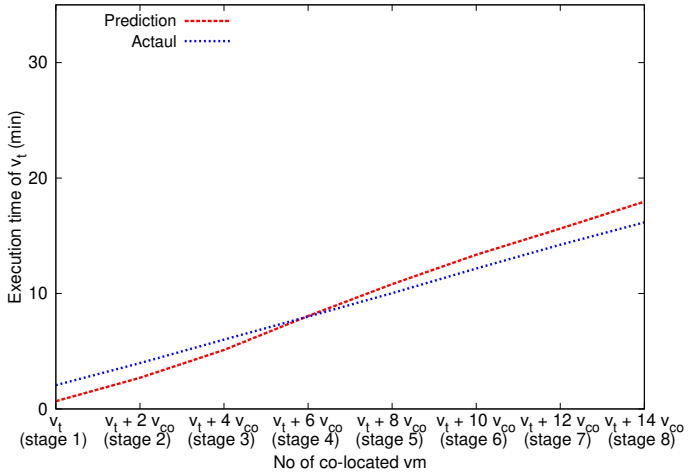


(c) Prediction for the Filebench.

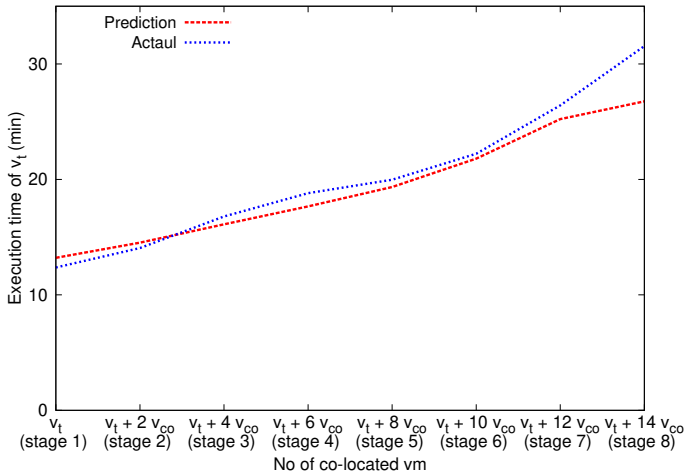
Figure 6.18: Prediction results for CPU-I/O workload combination on XenServer.



(a) Prediction for the Sysbench CPU test.



(b) Prediction for the Stream.



(c) Prediction for the Filebench.

Figure 6.19: Prediction results for Memory-I/O workload combination on XenServer.

6.9.3 Root mean square error (RMSE) for the prediction

The *Root Mean Square Error* (RMSE) of prediction for Exsi, XenServer, and Xen are shown in Tables 6.4, 6.4, and 6.4, respectively. The RMSE is an indicator of prediction accuracy; lower value indicates a better prediction accuracy. Recall that RMSE is already introduced and discussed in Section 4.8 of Chapter 4.

Previous two sections presented the prediction results for LSR prediction models. Section 6.9.1 shows the prediction results for the ESXi hypervisor, while Section 6.9.2 shows the prediction results for XenServer. The predictions are also made for Xen hypervisor. However, resulted graphs for Xen are not shown here as they are very similar to that of ESXi and XenServer. In this section, RMSE calculated from the prediction results for all three hypervisors are presented.

For each benchmark and workload types, the prediction errors are shown separately. The accuracy of the prediction for different resource combinations is different as shown in the tables. For each resource combination, the RMSE values of the three benchmarks are shown in separate rows.

Tables 6.4 shows the RMSE values for ESXi; values for the different combination of resources are shown in different columns. The third, fourth, and fifth columns of the table show the RMSE value for the CPU-memory, CPU-I/O, and memory-I/O workload combinations, respectively. For the CPU-memory load combination, the Stream has the highest RMSE value; it is 2.6298. For CPU-I/O load combination the Stream has the highest RMSE value again, it is 4.8102. For memory-I/O load

Table 6.4: RMSE of prediction for ESXi.

Task name	Resource intensity	CPU-Mem intensive workload combination	CPU-I/O intensive workload combination	Mem-I/O intensive workload combination
SB CPU test	CPU-intensive	2.2895	1.8205	3.0858
Stream	Mem.-intensive	**2.6298**	**4.8121**	3.8134
Filebench	I/O-intensive	1.0039	2.8102	**5.8760**

combination the Filebench has the highest value, it is 5.876. Highest values for each column is marked in bold.

Table 6.5 shows the RMSE of prediction for XenServer; the results are shown in the same format as above. Here, also the highest RMSE for each resource combination is shown in bold. Table 6.6 shows the RMSE values for the Xen hypervisor and highest values in each column are marked as well.

The second column of each table shows the resource intensity of the task. For example, the Sysbench (SB) CPU test is marked as CPU intensive. The Stream is memory intensive, and Filebench is I/O intensive. From the three tables, it can be seen that memory and I/O intensive tasks have higher RMSE values compared to CPU

Table 6.5: RMSE of prediction for XenServer.

Task name	Resource intensity	CPU-Mem intensive workload combination	CPU-I/O intensive workload combination	Mem-I/O intensive workload combination
SB CPU test	CPU-intensive	1.3541	1.3286	2.3954
Stream	Memory-intensive	**3.0901**	1.6091	2.6644
Filebench	I/O-intensive	0.3307	**2.5524**	**3.1346**

Table 6.6: RMSE of prediction for Xen.

Task name	Resource intensity	CPU-Mem intensive workload combination	CPU-I/O intensive workload combination	Mem-I/O intensive workload combination
SB CPU test	CPU-intensive	3.0849	1.2971	1.5593
Stream	Memory-intensive	**4.3019**	2.1410	4.3683
Filebench	I/O-intensive	2.3967	**3.5543**	**4.8173**

intensive tasks. As memory and I/O intensive tasks show more execution time variation due to the combination of resources, therefore, the prediction error is slightly higher.

he tables in this section present the accuracies of the prediction made for the three hypervisors. This accuracies are calculated from the prediction results presented in Sections 6.9.1, and 6.9.2. In the next section, it is shown how the parametric models can be built from the LSR prediction models trained in those sections.

6.10 Parametric models from LSR prediction results

Equations 6.1, 6.2, and 6.3 shows the model parameters and coefficients obtained from the LSR model trained above. Section 6.8 discusses the process of building LSR prediction models. Then, Section 6.9.1 presents the prediction results for ESXi hypervisor. Section 6.9.2 presents the prediction results for XenServer hypervisor. In this section, all the data from those sections is used to build parametric models.

Recall that, the process of obtaining parametric models from packages of the R programming language is already described in Section 4.9. As the same process from the Section 4.9 is used here to build parametric models the discussion about model building process is not repeated here.

Equations 6.1, 6.2, and 6.3 demonstrate the relationship between input and target data formats. The model terms are also described in Table 6.7 for clarification. Three prediction models are built for three resource combination targets, which are the CPU-Memory, CPU-I/O, and Memory-I/O.

The left side of equations denote the target terms that are to be predicted. For example, the term $ETV^{pred,t}_{cpu-mem,k}$ denotes the execution time variation prediction of a task, t on a VM for CPU and memory workload combinations. In this case, the VM is consolidated with the k number of co-located VMs, which run a combination of CPU and memory workloads. Recall that, the ICBM has several stages. At each stage, a different number of co-located VMs are simultaneously run with the task t. The $ETV^{pred,t}_{cpu-mem,k}$ is the predicted execution time variation of t when it is consolidated with total of k CPU and memory intensive VMs. Similarly, $ETV^{pred,t}_{cpu-io,k}$, and $ETV^{pred,t}_{mem-io,k}$ are the execution time variation predictions for k number of CPU-I/O, and memory-I/O VM combinations, respectively.

The right side of the equations presents the terms that are inputs to the models.

Table 6.7: Description of terms of Equations 6.1, 6.2, and 6.3

$ETV^{pred,t}_{cpu-mem,k}$	Predicted ETV of a task, t on a consolidated VM. Another k number of co-located VMs with CPU and memory intensive workload combination are consolidated with it.
$ETV^{pred,t}_{cpu-io,k}$	Predicted ETV of a task, t on a consolidated VM. Another k number of co-located VMs with CPU and I/O intensive workload combination are consolidated with it.
$ETV^{pred,t}_{mem-io,k}$	Predicted ETV of a task, t on a consolidated VM. Another k number of co-located VMs with CPU and memory intensive workload combination are consolidated with it.
H	A set of hypervisors.
T	A set of tasks.
$E^{h,t}_{cpu,k}$	Observed execution time of t on a consolidated VM on a hypervisor, h. It is consolidated with the k number of CPU-intensive co-located VMs.
$E^{h,t}_{mem,k}$	Observed execution time of t on a consolidated VM on a hypervisor, h. It is consolidated with the k number of memory-intensive co-located VMs.
$E^{h,t}_{io,k}$	Observed execution time of t on a consolidated VM on a hypervisor, h. It is consolidated with the k number of I/O-intensive co-located VMs.

All three equations (6.1, 6.2, and 6.3) use the same terms as inputs, they are the execution times due to the basic resources workload types. Each equation has six terms on the right side; they are quite self-explanatory. For example, $E^{h,t}_{cpu,k}$ represents the task execution time of a benchmark (t), when it is consolidated on a particular hypervisor (h) with k other CPU intensive co-located VMs. Where, $h \in H$ is one of the hypervisors and $t \in T$ is one of the tasks used during the training phase. The H and T are a set of hypervisors and tasks, respectively.

The model uses data from all training benchmarks from all hypervisors to create the models. In the same way, terms $E^{h,t}_{mem,k}$ and $E^{h,t}_{io,k}$ represent the task execution time, when t is consolidated with the k number of memory and I/O intensive co-located VMs, respectively. The α_i, β_i, and γ_i are the model coefficient, which is

being determined during the model training process.

$$
\begin{aligned}
ETV^{pred,t}_{cpu-mem,k} = \\
\alpha_1 + \sum_{h\in H}\sum_{t\in T}\Big(& \alpha_2 \times (E^{h,t}_{cpu,k}) + \alpha_3 \times (E^{h,t}_{mem,k}) + \alpha_4 \times (E^{h,t}_{io,k})^{0.9} \\
& + \alpha_5 \times (E^{h,t}_{cpu,k}) \times (E^{h,t}_{mem,k}) + \alpha_6 \times (E^{h,t}_{cpu,k}) \times (E^{h,t}_{io,k})^{0.9} \\
& + \alpha_7 \times (E^{h,t}_{mem,k}) \times (E^{h,t}_{io,k})^{0.9} \Big)
\end{aligned} \tag{6.1}
$$

$$
\begin{aligned}
ETV^{pred,t}_{cpu-io,k} = \\
\beta_1 + \sum_{h\in H}\sum_{t\in T}\Big(& \beta_2 \times (E^{h,t}_{cpu,k}) + \beta_3 \times (E^{h,t}_{mem,k})^{0.5} + \beta_4 \times (E^{h,t}_{io,k})^{1.5} \\
& + \beta_5 \times (E^{h,t}_{cpu,k}) \times (E^{h,t}_{mem,k})^{0.5} + \beta_6 \times (E^{h,t}_{cpu,k}) \times (E^{h,t}_{io,k})^{1.5} \\
& + \beta_7 \times (E^{h,t}_{mem,k})^{0.5} \times (E^{h,t}_{io,k})^{1.5} \Big)
\end{aligned} \tag{6.2}
$$

$$
\begin{aligned}
ETV^{pred,t}_{mem-io,k} = \\
\gamma_1 + \sum_{h\in H}\sum_{t\in T}\Big(& \gamma_2 \times (E^{h,t}_{cpu,k}) + \gamma_3 \times (E^{h,t}_{mem,k}) + \gamma_4 \times (E^{h,t}_{io,k})^{1.8} \\
& + \gamma_5 \times (E^{h,t}_{cpu,k}) \times (E^{h,t}_{mem,k}) + \gamma_6 \times (E^{h,t}_{cpu,k}) \times (E^{h,t}_{io,k})^{1.8} \\
& + \gamma_7 \times (E^{h,t}_{mem,k}) \times (E^{h,t}_{io,k})^{1.8} \Big)
\end{aligned} \tag{6.3}
$$

There is a total of six terms on the right side, of each equation. First three are being $E^{h,t}_{cpu,k}$, $E^{h,t}_{mem,k}$, and $E^{h,t}_{io,k}$. The next three are just combinations of the first three. Although, all the equations have the same six terms; however, their coefficients and exponents are different. Here, the presented models are unified models as the data from all three hypervisors are used to build the models.

6.10.1 LSR predictions with random workload

For further testing of the prediction accuracy of trained models, random workloads are used. Figures 6.20, 6.21, and 6.22 shows random workload are being run on ESXi, XenServer, and Xen. Figures show that the workloads are created using a random combination of tasks and VMs on the server.

First random locations for co-located VMs are chosen on the server then randomly selected tasks are executed on them. In each row of the figures, represents one stage of the experiment. At each stage, one target and several co-located VMs are running. The prediction models obtained in the previous section are used to predict the execution time of the target VM due to the presence of a random combination of co-located VMs.

The random workloads are run on the different hypervisors; which are VMware ESXi, XenServer, and Xen. Tables 6.8, 6.9, and 6.10 show the prediction results for the three hypervisors, respectively. In each table, the first column is the row number corresponding to the figures. The second column shows the names of the target VM tasks. The third column shows the actual execution time of the target VM task in each row. The fourth column shows the predicted execution time for the target VM task by the model. The fifth row contains the prediction error.

All training and testing are done with mutually exclusive hypervisor data. That is the hypervisor data used during training is not used for testing. Three hypervisors are used during the training process. Tables 6.8 shows the execution time prediction results for ESXi hypervisor. In this case, the models have trained with the execution time variation data from XenServer and Xen hypervisor. Then the trained models are used to predict the execution time of the tasks running on the ESXi hypervisor.

In the same way, the prediction models are trained with Xen and ESXi hypervisor data and then used to predict the execution time of tasks running on XenServer. The prediction results for the XenServer is shown in Table 6.9. Next, the models are again trained with ESXi and XenServer data, and the models are used to predict the execution time of tasks running on the Xen hypervisor. In this case, the prediction results are shown on Table 6.10. Thus, in all three Tables (6.8, 6.9, and 6.10) data from different hypervisors have been used to trained and test the prediction models.

Tables 6.8, 6.9, and 6.10 also show the *percentage prediction error* at the sixth column. The percentage prediction error is calculated using following formula [324]:

$$\text{percentage prediction error} = \left(\frac{\text{measured value} - \text{predicted value}}{\text{measured value}}\right) \times 100 \quad (6.4)$$

Above three tables (6.8, 6.9, and 6.10) also show the *Mean Absolute Percentage Error*(MAPE) in the very last row of each table. It is also known as the *Mean Absolute Percentage Deviation* (MAPD) is a measure of prediction accuracy of a forecasting models statistics. The MAPE is the arithmetic mean of the absolute percentage errors of prediction. The error is defined as measured value minus the predicted value. In this case, only the absolute value of the percentage error is summed up ignoring the sign of the error; this eliminates the possibility of positive and negative errors canceling each other during summation. As the error is presented in terms of percentage, this makes it easy to understand, and it is frequently used to test the accuracy of the prediction model. A smaller value of MAPE indicates a better prediction accuracy [325].

The MAPE is calculated using following formula [325, 326]:

Mean Absolute Percentage Error (MAPE)

$$= 100 \times \frac{1}{\text{N}} \sum_{t=1}^{N} \left| \frac{\text{measured value}_t - \text{predicted value}_t}{\text{measured value}_t} \right| \quad (6.5)$$

Each of the three figures (6.20, 6.21, and 6.22), shows nine randomly generated rows ($R1$-$R9$) of VMs. Among them, the first three rows ($R1$-$R3$) of the figures show that the Sysbench CPU test is consolidated with a various number of co-located VMs. In this case, the VMs are randomly placed on the server to generated workloads, which causes the execution time of the Sysbench CPU test to vary at a different amount. The LSR models built and presented previously in Section 6.10 are used to predict the execution time variation of the Sysbench CPU test. The objective here is to observe the prediction accuracy of the models for randomly generated workloads.

Similarly, the second three rows ($R4$-$R6$) of each figure (6.20, 6.21, and 6.22) show that the Stream is consolidated with various numbers of co-located VMs. The last three rows ($R7$-$R9$) show how a file server is consolidated with several combinations of co-located VMs. In each case, the co-located VMs are marked with their resource intensities.

Recall that the prediction models (Equations 6.1, 6.2, and 6.3) in Section 6.10 utilizes the number of co-located VMs and their resource intensities to predict the execution finish time of the target VM. In this section, each co-located VM in Figure 6.20, 6.21, and 6.22 is marked with resource intensity. A VM is either a CPU, memory or I/O intensive VM. They are some blank VMs shown in the figures they represent VMs with no task running on them.

Figure 6.20 shows that nine random workloads are created on the ESXi hypervisor. They are the combination of different types of resource intensive VMs. For example row 1 (top of Figure 6.20) has four CPU-intensive and nine memory-intensive co-located VMs. The execution finish time of the Sysbench CPU test due to consolidation with this workload on ESXi is 19.26 minute. The model trained in the previous section have been used to predict the execution time for this workload. Similarly, the models have been used to predict execution time variations of tasks for nine workloads on ESXi hypervisor. Their actual execution time and predicted execution times are listed in Table 6.8. For the workloads, the RMSE of prediction is 2.5867.

Table 6.8 shows measured and predicted execution time of each row. Each row is numbered along with the task name, whose execution time is to be predicted. The third column shows the measured actual execution time of the task due to the consolidated random workload on co-located VMs. The fourth column shows the execution time

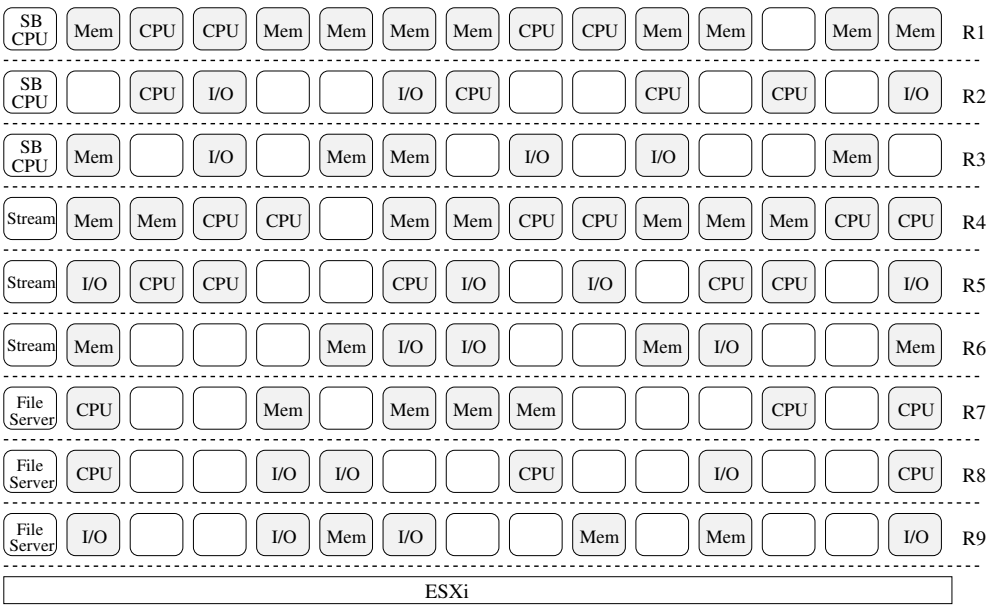


Figure 6.20: Random workload generated for ESXi.

Table 6.8: Execution time prediction for randomly generated workloads on ESXi.

VMware ESXi (Figure 6.20)					
Row no.	Task name	Measured execution time (minute)	Predicted execution time (minute)	Prediction Error (minute)	Percentage prediction error (%)
R1	SB CPU	19.26	15.7827	3.4773	**18.0545**
R2		78.30	75.0328	3.2672	4.1726
R3		13.71	14.5762	-0.8662	-6.3180
R4	Stream	11.47	11.0501	0.4199	3.6608
R5		11.10	10.3921	0.7079	6.3774
R6		23.64	20.7657	2.8743	12.1586
R7	File Server	11.87	13.2075	-1.3375	-11.2679
R8		39.56	36.1093	3.4507	8.7227
R9		25.01	21.2591	3.7509	14.9976
			RMSE	2.5867	
			MAE	2.2391	
			MAPE		9.5255

predicted by the LSR models. The fifth column shows the prediction error of each row. At the end of the column, the RMSE of the prediction errors is also given. The sixth column of the table shows the percentage prediction error. The column shows the largest magnitude of percentage error the Sysbench CPU test on the R1, and it is 18.0545%.

Similarly, execution time prediction for nine separate random workloads are made for XenServer, and Xen hypervisor; the results are shown in Figures 6.21, and 6.22, respectively. Their execution times are also predicted by models build with mutually exclusive hypervisor data. That is prediction model used for XenServer hypervisor is trained with data from ESXi and Xen hypervisor. Actual execution times and prediction data for XenServer are shown in Table 6.9.

Similarly, prediction models used for Xen is trained with data from ESXi and XenServer. Once the model is trained, they are used to predict the task execution time variation on Xen. Table 6.10 shows the measured and predicted task execution

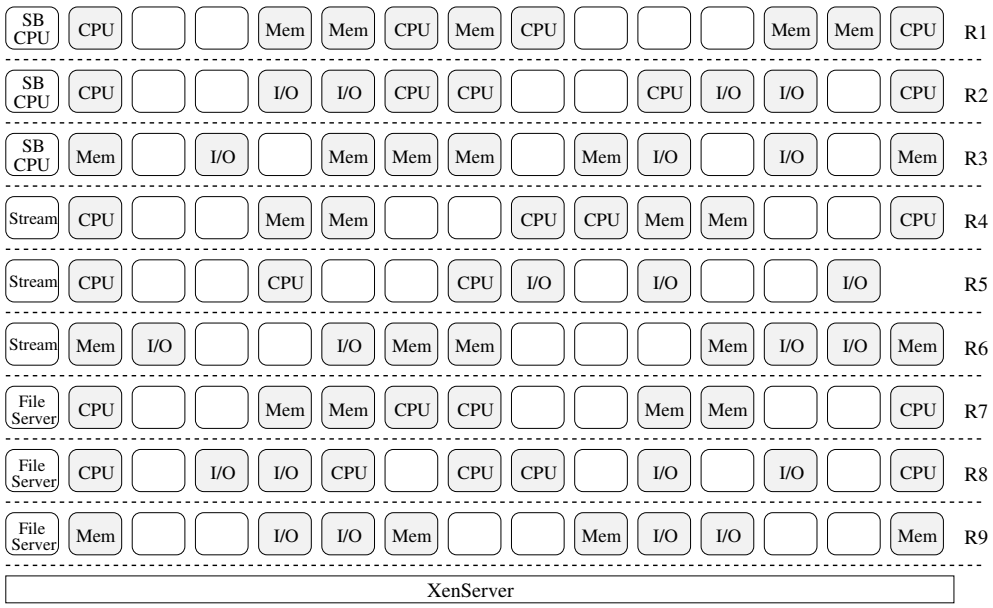


Figure 6.21: Random workload generated for XenServer.

Table 6.9: Execution time prediction for randomly generated workloads on XenServer.

XenServer (Figure 6.21)					
Row no.	Task name	Measured execution time (minute)	Predicted execution time (minute)	Prediction Error (minute)	Percentage prediction error (%)
R1	SB CPU	15.41	13.8446	1.5654	10.1583
R2		5.98	4.2006	1.7794	**29.7558**
R3		12.61	13.4495	-0.8395	-6.6574
R4	Stream	13.39	11.9921	1.3979	10.4398
R5		12.20	13.6815	-1.4815	-12.14344
R6		21.71	18.7382	2.9718	13.6886
R7	File Server	13.77	17.1161	3.3461	24.2999
R8		12.08	13.294	-1.2140	-10.0496
R9		21.12	23.1647	-2.0447	-9.6813
			RMSE	2.0042	
			MAE	1.8489	
			MAPE		14.0971

time for Xen.

The row by row prediction results for the VM settings of Figure 6.21 is shown in Table 6.9. In this case, all tasks are running on XenServer hypervisor. The sixth column of the Table 6.9 shows the percentage prediction error for the trained model. In can be seen that the percentage prediction error is significantly higher compared to that of ESXi (previously shown in Table 6.8). Particularly the third row of Table 6.9 show that for Sysbench CPU test the percentage prediction error is -29.7558%.

Recall that the hypervisors are trained with mutually exclusive hypervisor data. That is, the prediction models used for XenServer is trained with data from ESXi and Xen hypervisor. The prediction models for ESXi and Xen are trained in the same way using mutually exclusive training and testing data. However, prediction models for ESXi (Table 6.8) and Xen Table 6.10 show better percentage prediction error compared to that of XenServer (Table 6.9), this shows that the prediction accuracy of the unified prediction model is not the same for all hypervisors.

For each hypervisor, training situation can be unique. It is known that in different situation machine learning methods may require different amount of data to proper training [327–330]. Therefore, it is possible that the amount of data required for building a prediction model for different hypervisors can be different. It is possible that the accuracy of the prediction model for XenServer can be improved using a larger set of training data.

Lastly, Table 6.10 shows the accuracy of the prediction model for the Xen hypervisor. In this case, also nine random workloads are generated using various combination of resource-intensive co-located VMs. As the workloads are randomly generated the workloads are not the same on the three hypervisors.

The VM workloads used for this hypervisor is shown in Figure 6.22. The figure has nine rows showing nine randomly generated workload. The execution time of the task in target VM for the above nine workloads are shown from the second to the tenth column of Table 6.10.

In this case, the highest percentage prediction error is 22.5960%, and it is for the Stream benchmark. This value is shown in the fifth row of Table 6.10. The MAPE of all nine predictions is also shown in the thirteen row of the table, and it is 12.5249%. The RMSE of prediction for XenServer and Xen are 2.0042 and 2.6686, respectively.

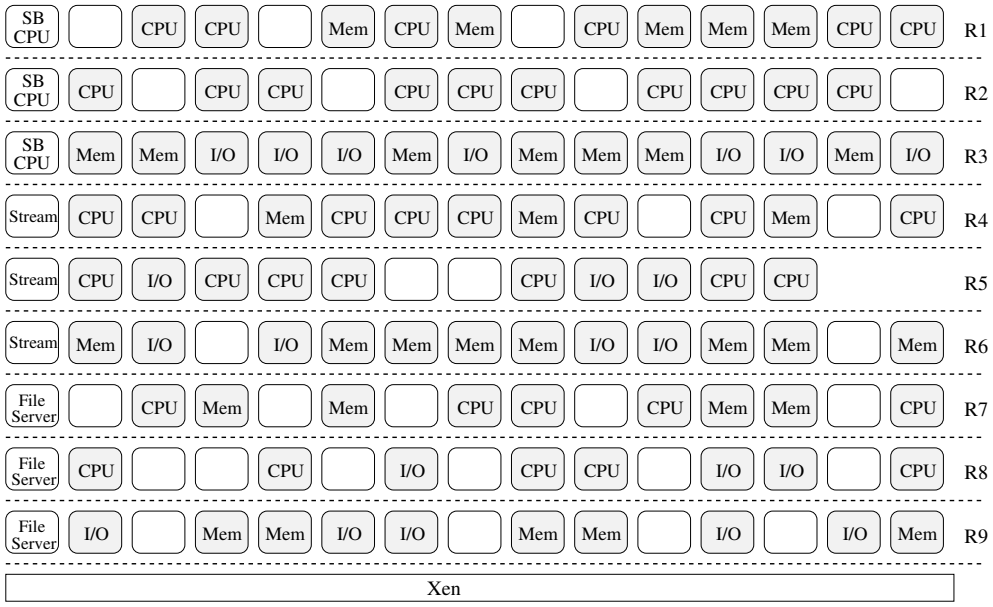


Figure 6.22: Random workload generated for Xen.

Table 6.10: Execution time prediction for randomly generated workloads on Xen.

Xen (Figure 6.22)					
Row no.	Task name	Measured execution time (minute)	Predicted execution time (minute)	Prediction Error (minute)	Percentage prediction error (%)
R1	SB CPU	18.79	16.3315	2.4585	13.0840
R2		40.33	37.4472	2.8828	7.1480
R3		13.11	15.1193	-2.0093	-15.3264
R4	Stream	16.91	13.0890	3.8210	**22.5960**
R5		13.55	11.6988	1.8512	13.6619
R6		17.44	14.5880	2.8520	16.3532
R7	File Server	15.86	14.4915	1.3685	8.6286
R8		93.74	96.9221	-3.1821	-3.3946
R9		21.91	19.1642	2.7458	12.5321
			RMSE	2.6686	
			MAE	2.5745	
			MAPE		12.5249

6.11 Artificial neural network (ANN) and Back propagation learning

Artificial neural network (ANN) is an widely used machine learning method for supervised learning [331–339, 339, 339–346]. ANNs are computer systems inspired initially by the biological neural networks.

In many cases the functional relationship between the input variables and the output variables are unknown. For example when modeling the relationship between complex diseases, and their potential risk factors that can be useful for disease prevention strategies. An ANN can be useful for building models for such complex scenarios. Unlike generalized linear models (GLM) [347], an ANN does not require to specify the relation between the input and output variables. Observed data can be used to train an ANN, which learns an approximation of the relationship by gradually adapting the parameters.

An ANN is a valuable statistical tool. They are an extension of GLMs and can be used similarly. An ANN consists of several layers of nodes and connects between them The connect usually exist between nodes on successive layers. A node is often called an artificial neuron, while a connection is also known as the synapsis. Figure 6.23 shows an example of an artificial neural network.

In the example of Figure 6.23, artificial neurons are shown as circles and synapses are shown as connections between the neurons. An artificial neural network can have one input layer, one output layer, and several hidden layers. Inputs are feed though input layers, data is transformed in the hidden layers, and finally, the output layer provides the output.

In Figure 6.23, the leftmost layer is the input layer, the middle layer is the hidden layer, and the rightmost layer is the output layer. In the figure, the input layer has four neurons, and the hidden layer has five neurons. Each node of the input layer is connected to all the nodes of the hidden layer. Data flow direction through the links are indicates with allows. It can be seen that each node of the hidden layer receives data from all the input nodes.

A weight is associated with each link. During the training phase, the learning algorithm updates the weights using various formulas. When the input data pass through a link, the associated weight is multiplied with the input and data is gathered in the

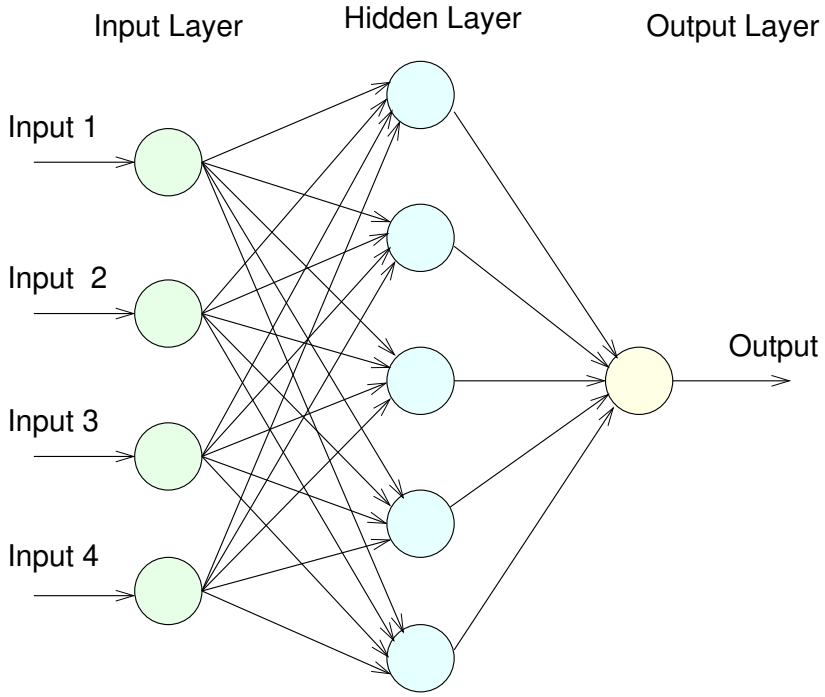


Figure 6.23: An Artificial neural network (ANN).

hidden layers. Data collected in the nodes of the hidden layer are then transformed using a predefined function. As the links have different weights, the data collected from different nodes of hidden layers are different. The transformed data is then propagated through another set of links either to another hidden layer or output layer.

ANN is used in a lot of real world situation like Handwriting Recognition [348–354], Speech Recognition [355–359], Remote Sensing and image classification [360–362], Image compression [363–367], Data mining [368–370], Robotics and Autonomous Systems [371, 372], Weather forecast [373, 374], Stock Exchange Prediction [375–377], etc. Thus, an ANN is an important method in the field of machine learning. In this chapter, An ANN is used here to predict the task ETV for various types of resource intensive co-located VMs. The backpropagation algorithm is used here for training the ANN. The algorithm is described in the next section.

6.11.1 Resilient Back propagation with weight backtracking

Backpropagation is one of the most widely used learning algorithms for multilayered forward feed networks [331–333, 378, 379]. The basic idea behind the backpropagation learning algorithm is to apply the chain rule repeatedly to compute the influence

1: **for all** $weight$ **do**
2: $weight \leftarrow weight - grad \times L.Rate$
3: **end for**

Figure 6.24: Traditional back propagation pseudocode.

of each weight with respect to an error function *E* [380–382]. *E* is an arbitrary error function. The relation can be mathematically expressed as:

$$\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial s_i} \frac{\partial s_i}{\partial net_i} \frac{\partial net_i}{\partial w_{i,j}} \tag{6.6}$$

Where, $w_{i,j}$ is the weight on the link, which connects neuron j to neuron i. s_i is the output from from neuron i. The net_i is the weighted sum of the inputs of neuron i. Once the partial derivatives for each weight of the network are known, the function tries to minimize the error by performing gradient descent:

$$w_{i,j}(t+1) = w_{i,j}(t) - \epsilon \frac{\partial E}{\partial w_{i,j}}(t) \tag{6.7}$$

Where ϵ is the learning rate. t is the index for iteration steps. It is essential to choose the value of ϵ carefully as it scales the derivative and effects the time needed for the solution to converge [332]. If the value of ϵ is set too small, then it will require too many steps to reach an acceptable solution. On the other hand, a large value of learning rate can lead to oscillation, which can prevent the error from reach an acceptable value.

All weights of the backpropagation is updated using the Equation 6.7, which is transformed into pseudocode in Figure 6.24. $weight$ is the weight of individual links that connect the neurons. All weight of the traditional backpropagation algorithm updated using the same formula. $grad$ is the gradient calculated according to Equation 6.6. $L.Rate$ is the learning rate for the neural network. For traditional backpropagation algorithm, the learning rate is a constant and applied to all weights equally. However, in resilient back propagation, each weight have separate learning rates and the rates may change during the learning process. The resilient backpropagation algorithm is discussed next in the section.

Resilient back propagation algorithm is based on the traditional back propagation [331–333]. In a traditional neural network, there is only one learning rate, and it is applied to all weights during the training process. On the other hand, resilient

backpropagation use a separate learning rate $\epsilon_{i,j}$ for each weight. The learning rates can also be changed during the training process. Equation 6.7 shows how the weights are adjusted during the training process of the traditional backpropagation algorithm. For the resilient backpropagation the equation is changed as follows:

$$w_{i,j}(t+1) = w_{i,j}(t) - \epsilon_{i,j} \; .sign\left(\frac{\partial E}{\partial w_{i,j}}(t)\right) \tag{6.8}$$

In Equation 6.8, only the sign of the partial derivatives have been used instead of the magnitude, this ensures an equal influence of the learning rate over the entire network [332].

Resilient back propagation allows a way to speed up the convergence in shallow areas of the search space. It is done by keeping an eye on the sign of the partial derivatives of the error function. If the partial derivative keeps the sign, the learning rate $\epsilon_{i,j}$ can be increased. On the other hand, if the sign changes that indicate that the minimum region is missed due to a large learning rate. In this case, the learning rate needs to be decreased. Also, the steps of the last iteration need to be undone. It is achieved by using the *weight backtracking* technique.

In the weight backtracking technique, the last step of the iteration is undone, and the learning rate is also decreased during the successive stages of iteration. Without the use of backtracking technique, the algorithm can oscillate over the minimum value several times. The pseudocode for resilient backpropagation algorithm with weight backtracking for training a feed-forward neural network is discussed below [332].

Figure 6.25 shows the basic algorithm for resilient backpropagation and weight backtracking. The **for** loop (from line 1 to 14) performs the same operations for all the weights of the network. Inside the **for** loop, one of three sets of actions are taken based on the sign of the multiplication of new and old gradient.

If condition in line 2 checks whether the old and new gradient have the same sign or not. The new gradient is calculated using Equation 6.6 and the old gradient is the stored gradient of the immediately previous step. Positive multiplication result indicates that both gradients have the same sign and the minimum region have not reached yet. In this case, the learning rate ($L.Rate$) is increased slightly for faster convergence. In line 3, present $L.Rate$ is multiplied with eta_{PLUS}, which is a predefined constant value. Then, the result is compared with another constant $L.Rate_{MAX}$ and the lesser value is chosen. $L.Rate_{MAX}$ is a predefined constant that represents the maximum

1: **for all** $weight$ **do**
2: **if** $(grad_{old} \times grad_{new} > 0)$ **then**
3: $L.Rate \leftarrow min\ (L.Rate \times eta_{PLUS},\ L.Rate_{MAX})$
4: $weight \leftarrow weight - sign(grad_{new}) \times L.Rate$
5: $grad_{old} \leftarrow grad_{new}$
6: **else if** $(grad_{old} \times grad_{new} < 0)$ **then**
7: $weight \leftarrow weight + sign(grad_{old}) \times L.Rate$
8: $L.Rate \leftarrow max\ (L.Rate \times eta_{MINUS},\ L.Rate_{MIN})$
9: $grad_{old} \leftarrow 0$
10: **else if** $(grad_{old} \times grad_{new} = 0)$ **then**
11: $weight \leftarrow weight - sign(grad_{new}) \times L.Rate$
12: $grad_{old} \leftarrow grad_{new}$
13: **end if**
14: **end for**

Figure 6.25: Resilient back propagation with weight backtracking.

learning rate. Thus, the learning rate is increased; however, making sure that it does not cross the maximum limit. In line 4, this updated value of $L.Rate$ is multiplied by the sign of the new gradient and subtracted from the present weight. Thus, depending on the sign of the new gradient $L.Rate$ amount is either removed from or added to the present weight. In line 5, the new gradient value is stored in the variable $grad_{old}$ to be used in the next step.

Next, a **else if** condition is used to check if the old and new gradients have different signs. In line 6, a negative multiplication result indicates that old and new gradients have different signs. That means the algorithm has crossed the minimum region and a backtracking step is required. First, the learning rate is multiplied by the sign of the old gradient and added to the weight. In this way, the weight of the last step is restored (line 7). Learning rate also needs to be decreased, and this is done in line 8. First the resent learning rate is multiplied with another predefined constant (eta_{MINUS}) and compared with the predefined minimum learning rate ($L.Rate_{MIN}$). Maximum of the values are chosen to make sure that learning rate does not fall below a certain limit.

At line 9, the old gradient is set to zero. Thus, the weight backtracking operation is performed.

Next, another **else if** condition checks if the multiplication of old and new gradient are zero or not. In this case, weights are updated (line 11); however, learning rate ($L.Rate$) is not changed. The gradient value is also stored in the variable $grad_{old}$. These same steps are performed in each stage of the iteration.

In this dissertation, the package `neuralnet` [383–388] from the R programming language [280–289, 389] has been used. The `neuralnet` package is described next.

6.11.2 An ANN package of R: `neuralnet`

The `neuralnet` is an R package to train a multi-layer neural network for regression analysis. This section provides an introduction to `neuralnet` function, which is used for training an ANN with a resilient backpropagation algorithm. The package gives options to select several different learning algorithms for the neural network, they are:

i) Traditional backpropagation algorithm.

ii) Resilient back propagation algorithm.

iii) Resilient back propagation with weight backtracking algorithm.

iv) Modified globally convergent resilient backpropagation algorithm.

In this dissertation, Resilient back propagation with weight backtracking algorithm is used for training an ANN. Recall that the resilient backpropagation with weight backtracking is described in Section 6.11.1 above.

The `neuralnet` package contains a flexible function to train neural networks. The package can handle an arbitrary number of input variables, output variables, and hidden layers. Functions are also provided to visualize the results and trained neural networks. The `neuralnet` function has following interface [383, 383]:

```
neuralnet(formula, data, hidden = 1, threshold = 0.01,
     stepmax = 1e+05, rep = 1, startweights = NULL,
     learningrate.limit = NULL,
     learningrate.factor = list(minus = 0.5, plus = 1.2),
```

```
    learningrate=NULL, lifesign = "none",
    lifesign.step = 1000, algorithm = "rprop+",
    err.fct = "sse", act.fct = "logistic",
    linear.output = TRUE, exclude = NULL,
    constant.weights = NULL, likelihood = FALSE)
```

Where parameter **formula** is a symbolic description of the model to be fitted. Parameter **data** is a data frame containing the variables specified in the formula. Parameter **hidden** is a vector of integer values that are used to specify the number of hidden neurons in each layer. Parameter **threshold** is a floating point value that specifies the threshold for the partial derivatives of the error function as the stopping criterion. Parameter **stepmax** defines the maximum number of training steps for the neural network. If the neural network does not converge to any particular value after the maximum number of steps, the training process is terminated. Parameter **rep** defines the number of times the whole training process should be repeated. Parameter **startweights** is a vector containing initial weights for the links of the neural network. If the weights are not specified then random values are assigned.

Parameter **learningrate.limit** is a list containing lowest and highest limit for the learning rate. Parameter **learningrate.factor** is a list containing multiplication factors for the upper and lower learning rate. Above two parameters are used with the resilient backpropagation algorithm. Parameter **learningrate** defines the learning rate for traditional back propagation algorithms. It is not used with the resilient backpropagation algorithm. Parameter **lifesign** is a string that indicates how much data will be printed during the iterations of the neural network.

Parameter **lifesign.step** is an integer that specifies the step size of the minimal threshold to print. Parameter **algorithm** defines which learning algorithm to use for training the neural network. Parameter **err.fct** is differentiable function, which is used for error calculation. Parameter **act.fct** is a differentiable function used for smoothing the results of the cross product of input and weights. Parameter **linear.output** is a logical parameter that determines whether the output should be linear or not. Parameter **exclude** is a matrix that specifies the weight, which should be excluded from the calculation. Parameter **constant.weights** is a list of weights that are treated as constant; thus, not altered during the training process. Parameter **likelihood** is a logical parameter that triggers further statistical calculations if error function shows specific

behavior.

In this section, the `neuralnet` function is introduced, and function parameters are discussed. This function of the R programming language is used for training an ANN prediction model. Next section describes the prediction data and results for the trained ANN.

6.12 Prediction with an Artificial neural network (ANN)

Previously in Section 6.9, the ETV data is used with LSR to build execution time prediction models. The ETV data is collected using ICBM from three hypervisors as described in the previous sections. This section shows that the ETV data can be used with other machine learning methods like the *Artificial Neural Network* (ANN). In this section, the prediction results from an ANN model are presented. The ANN is a widely used machine learning method [331–339, 339, 339–346]. An ANN model can learn from observed events. ANN is introduced in Section 6.11 and theory behind an ANN learning algorithm are discussed in Section 6.11.1. Also, the R language package used for training an ANN is discussed in Section 6.11.2. In this case, the ETV due to various workload is input to an ANN. Then, the trained model is used for prediction.

Figures 6.4, 6.5, 6.6, 6.7, 6.8, 6.10, 6.11, 6.12, and 6.13 show ETV results for two types of workloads. There are three basic types of workload; CPU, memory and I/O. There is also three combinational workload; CPU-memory, CPU-I/O, and memory-I/O. Prediction for two types of workloads is discussed in the next section.

6.12.1 Case 1: Execution time prediction for single resource contention

The task ETV results are discussed in Sections 6.4, 6.5, and 6.6. Figures 6.4, 6.5, 6.6, 6.7, 6.8, 6.10, 6.11, 6.12, and 6.13 show that some tasks can show huge level of variation during VM consolidation.

For example, Figure 6.6b shows that the execution time of the IOzone rapidly increases with the increase of I/O intensive workload in the co-located VMs. In the final stage, with 14 co-located I/O intensive VMs the IOzone takes 174.87 minutes to execute on ESXi. The summation of all execution times for eight combinations is 792.19 minutes or 13.20 hours.

Similarly, in the final stage of Figure 6.10b, it takes 168.78 minutes to execute on XenServer. The summation for all of the settings is 639.82 minutes or 10.66 hours. Thus, tasks can show a lot of ETV due to single resource contention and summation of all the execution times can be a large one. It would be a time-consuming process to profile all the tasks for all co-located VM combinations. A lot of profiling time can be saved if the ETV due to the higher number of co-located VMs can be predicted from the observed ETV due to the lower number of co-located VMs.

In this section, an ANN model is used for this purpose. The ANN model is trained with the task execution times due to the lower number of co-located VMs. Then, it is used to predict the execution time variation due to the higher number of co-located VMs. The prediction results for ESXi are given in Table 6.11.

The second row of the Table 6.11 shows the four actual execution time variation of the Nbench; they are taken from Figure 6.4a. Recall that, Figure 6.4a shows eight

Table 6.11: ANN prediction of execution time for higher number of VMs.

No of CPU-intensive co-located VMs	8	10	12	14	RMSE	MAE
Measured Nbench execution time (minute)(see Figure 6.4a)	58.48	65.03	79.64	90.99		
ANN prediction (minute)	63.08	70.48	84.75	97.91		
Prediction error (minute)	-4.60	-5.45	-5.11	-6.92	5.58	5.52
No of Memory-intensive co-located VMs	8	10	12	14	RMSE	MAE
Measured Stream execution time (minute)(see Figure 6.5b)	49.29	59.65	70.08	80.41		
ANN prediction (minute)	53.93	64.92	75.59	85.79		
Prediction error (minute)	-4.64	-5.27	-5.51	-5.38	5.21	5.20
No of I/O-intensive co-located VMs	8	10	12	14	RMSE	MAE
Measured IOzone execution time (minute)(see Figure 6.6b)	115.94	138.35	163.14	174.87		
ANN prediction (minute)	123.81	144.36	171.33	186.53		
Prediction error (minute)	-7.87	-6.01	-8.19	-11.66	8.67	8.43

execution time variations of Nbench due to CPU intensive co-located VMs. Data of four lower combinations of co-located VMs (0, 2, 4, 6) are used for training. Then, the ANN model is used for predicting the execution time variation of Nbench due to the higher number of co-located VMs (8, 10, 12, 14). Predictions are shown in the third row of Table 6.11, and the errors are shown on the fourth. The *Root Mean Squared Error* (RMSE) for this set of prediction is 5.58.

This process can be repeated for other resource-intensive co-located VMs, too. For example, the sixth row of the Table 6.11 shows the actual ETV of the Stream due to four higher configuration of memory intensive co-located VMs. This data is taken from Figure 6.5b. Here, also the lower four setting of co-located VMs have been used for training an ANN model. Then, the model is used to predict the execution times due to four higher combinations of co-located VMs. The RMSE for this set of prediction is 5.21.

Table 6.12: ANN prediction of execution time for lower number of VMs.

No of CPU-intensive co-located VMs	0	2	4	6	RMSE	MAE
Measured Nbench execution time (minute)(see Figure 6.4a)	13.48	18.41	24.29	33.89		
ANN prediction (minute)	16.7	22.18	28.38	39.56		
Prediction error (minute)	-3.22	-3.77	-4.09	-5.67	4.28	4.18
No of Memory-intensive co-located VMs	0	2	4	6	RMSE	MAE
Measured Stream execution time (minute)(see Figure 6.5b)	10.20	19.24	28.97	39.13		
ANN prediction (minute)	14.19	23.34	33.58	44.89		
Prediction error (minute)	-3.99	-4.10	-4.61	-5.76	4.66	4.61
No of I/O-intensive co-located VMs	0	2	4	6	RMSE	MAE
Measured IOzone execution time (minute)(see Figure 6.6b)	12.95	35.85	61.28	89.81		
ANN prediction (minute)	16.01	41.64	67.36	97.07		
Prediction error (minute)	-3.06	-5.69	-6.08	-7.26	5.73	5.54

Similarly, the last three rows of Table 6.11 show the prediction for the IOzone due to I/O intensive co-located VMs. ETV data of IOzone due to I/O intensive co-located VMs on the ESXi hypervisor is shown in Figure 6.6b.

In a data center, the number of co-located VMs may change in any order. Thus, a reverse scenario may also occur. That is, the number of co-located VMs may drop in a server after some time. In such case, the execution time variations due to the higher number of co-located VMs would be known and one needs to predict the variations due to the lower number of co-located VMs.

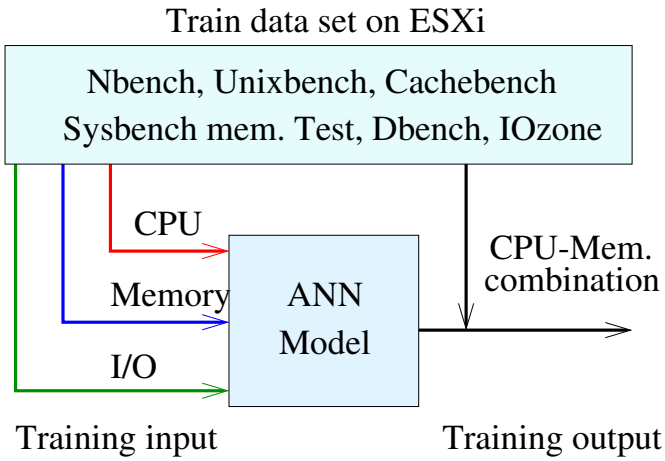
This reverse situation is considered in Table 6.12. In this case, the higher numbers of co-located VM data (8-14) is used to train the ANN model. Then, the execution time variations due to a lower number of co-located VMs is predicted. The RMSE of prediction in this case for the Nbench, Stream, and IOzone are 4.28, 4.66 and 5.73, respectively.

6.12.2 Case 2: Execution time prediction for multiple resources

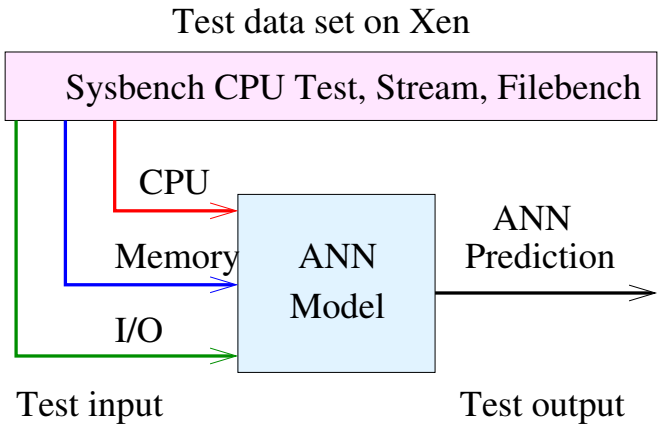
In the last section, execution time variations due to only single resource intensive workload have been considered. That is, at each stage only one type of resource intensive co-located VMs is increased. Either CPU, memory or I/O intensive workload is increased at a time. In this section, the contention due to the combination of workload is examined.

Three resource combinations are considered CPU-memory, memory-I/O, and CPU-I/O combinations. This section demonstrates the relationship between the execution time variation due to basic resource usages and that of the combination of resources. The objective is to train an ANN model with the performance variation data due to basic resource contention, then use it to predict the execution time variations due to the combination of resources.

For the ANN model, the training and testing are done with non-overlapping data sets. The training set contains six tasks; Nbench, Unixbench, Cachebench, Sysbench memory test, Dbench, and IOzone. The testing set contains three separate tasks; Sysbench CPU test, Stream, and Filebench. Furthermore, training and testing data are collected from different hypervisors. It is done to show the general applicability of the presented methodology. The prediction results hold across the hypervisors and VM configurations.



(a) ANN training with six tasks ETV data.



(b) ANN testing with separate three tasks ETV data.

Figure 6.26: ANN training and testing with separate data sets.

Figure 6.26a shows how input and target data are feed to an ANN model during the model training phase for predicting execution time variation due to CPU-memory workload combination. In this case, all training data is collected from the ESXi. Recall that ANN learning algorithms and R implementation packages are already discussed in Sections 6.11, 6.11.1, and 6.11.2. Figure 6.26a only demonstrates what tasks are used as input and target during the training process.

Execution time variation of the training set due to three basic resource intensive co-located VMs are used as the *training input*. Recall that, the ETV on ESXi due to the three basic and combinational workload are shown in Figures 6.4a, 6.5b, and Figure 6.6c. The execution time variation of the same tasks due to CPU-memory

workload combination is used as the *training output*. The training and testing data set are presented in the previous sections. The same data is used here, and instead of LSR, the ANN is used to build the models.

Next, Figure 6.26b shows the prediction process for CPU-memory resource combination. For testing data from three separate tasks on another hypervisor are used. Here, the execution time variation of the test set due to CPU, memory, and I/O intensive co-located VMs on Xen are used as the *test input*. Then, the ANN model prediction is compared with the actual execution time variation due to CPU-memory resource contention on Xen.

Table 6.13 shows the ETV prediction results of ANN for the Sysbench CPU test on Xen. As before, in one of the VMs, the Sysbench CPU test is run, and CPU and memory intensive workloads are run on co-located VMs. The first column of Table 6.13 shows the number of co-located VMs with CPU and memory intensive workload are used at each stage. The second column shows the execution times of the Sysbench

Table 6.13: ANN prediction of execution time for CPU & Mem workload combinations on Xen.

No of CPU & Mem. intensive co-located VMs	Sysbench CPU test execution time (minutes) (see Figure 6.11c)	ANN prediction (minutes)	Prediction error (minutes)	Percentage prediction error (%)
0	11.07	13.03	-1.96	-17.70
2	11.01	13.57	-2.56	-23.25
4	11.86	14.38	-2.52	-21.24
6	13.64	16.22	-2.58	-18.91
8	15.94	19.57	-3.63	-22.77
10	17.63	20.84	-3.21	-18.20
12	19.86	24.29	-4.43	-22.30
14	21.49	25.71	-4.22	-19.63
		RMSE	3.24	
		MAE	3.13	
		MAPE		20.50

CPU test at each stage. The ANN prediction, error and RMSE are shown in the following rows as well. The third column shows the execution time prediction of the ANN model at each stage. The fourth column lists the prediction error. At the end of the fourth column, the RMSE of prediction for this set of data is also given, it is 3.24.

It is also possible to train the ANN model for other workload combinations. ANN predictions for Memory-I/O, and CPU-I/O intensive combinational workload are shown in Tables 6.14, and 6.15, respectively. In each case, the prediction error and RMSE are also shown. The training and testing procedures followed here are similar to that are shown in Figs. 6.26a, and 6.26b, respectively. The total time taken by ANN training and prediction process is only 0.1330 seconds. Thus, the process is speedy and can be easily retrained in real time as new task execution variation data arrives.

Next, Tables 6.16, 6.17, and 6.18 show the execution time variation prediction for the same three resource combinations on ESXi, respectively. In this case, the same procedure described as above is followed. The difference only is that now the training

Table 6.14: ANN prediction of execution time for Mem. & I/O workload combinations on Xen.

No of Mem. & I/O intensive co-located VMs	Stream (minutes) execution time (see Figure 6.12b)	ANN prediction (minutes)	Prediction error (minutes)	Percentage prediction error (%)
0	11.66	9.21	2.45	21.01
2	20.83	17.47	3.36	16.13
4	31.16	27.46	3.70	11.87
6	41.95	37.46	4.49	10.70
8	52.49	47.33	5.16	9.83
10	62.63	57.37	5.26	8.39
12	72.94	67.01	5.93	8.12
14	83.24	76.42	6.82	8.19
		RMSE	4.83	
		MAE	4.64	
		MAPE		11.78

Table 6.15: ANN prediction of execution time for CPU & I/O workload combinations on Xen.

No of CPU & I/O intensive co-located VMs	Filebench execution time (minutes) (see Figure 6.13c)	ANN prediction (minutes)	Prediction error (minutes)	Percentage prediction error (%)
0	12.82	15.38	-2.56	-19.96
2	13.69	16.6	-2.91	-21.25
4	14.87	18.5	-3.63	-24.41
6	15.99	19.77	-3.78	-23.63
8	18.16	22.09	-3.93	-21.64
10	20.36	24.8	-4.44	-21.80
12	23.59	27.73	-4.14	-17.54
14	27.34	32.26	-4.92	-17.99
		RMSE	3.85	
		MAE	3.78	
		MAPE		21.02

Table 6.16: ANN prediction of execution time for CPU & Mem. workload combinations on ESXi.

No of CPU & Mem. intensive co-located VMs	Sysbench CPU test execution time (minutes) (see Figure 6.4c)	ANN prediction (minutes)	Prediction error (minutes)	Percentage prediction error (%)
0	10.92	13.44	-2.52	-23.07
2	11.52	14.09	-2.57	-22.30
4	12.79	15.41	-2.62	-20.48
6	14.04	17.31	-3.27	-23.29
8	16.37	20.28	-3.91	-23.88
10	19.17	23.18	-4.01	-20.91
12	21.04	25.91	-4.87	-23.14
14	23.82	28.66	-4.84	-20.31
		RMSE	3.69	
		MAE	3.57	
		MAPE		22.17

Table 6.17: ANN prediction of execution time for Mem. & I/O workload combinations on ESXi.

No of Mem. & I/O intensive co-located VMs	Stream execution time (minutes) (see Figure 6.5b)	ANN prediction (minutes)	Prediction error (minutes)	Percentage prediction error (%)
0	10.20	12.38	-2.18	-21.37
2	19.46	22.99	-3.53	-21.00
4	29.45	32.66	-3.21	-10.89
6	39.47	43.77	-4.30	-10.89
8	49.81	54.77	-4.96	-9.95
10	60.38	65.76	-5.38	-8.91
12	71.00	76.80	-5.80	-8.16
14	81.42	87.43	-6.01	-7.38
		RMSE	4.64	
		MAE	4.47	
		MAPE		12.31

Table 6.18: ANN prediction of execution time for CPU & I/O workload combinations on ESXi.

No of CPU & I/O intensive co-located VMs	Filebench execution time (minutes) (see Figure 6.6c)	ANN prediction (minutes)	Prediction error (minutes)	Percentage prediction error (%)
0	13.88	16.18	-2.30	-16.57
2	16.25	19.57	-3.32	-20.43
4	19.49	22.66	-3.17	-16.26
6	22.77	26.01	-3.24	-14.22
8	25.96	29.13	-3.17	-12.21
10	27.45	32.33	-4.88	-17.77
12	29.12	33.52	-4.40	-15.10
14	31.41	35.96	-4.55	-14.48
		RMSE	3.72	
		MAE	3.62	
		MAPE		15.88

tasks are run on the XenServer, and testing data is collected from the ESXi. These prediction results for XenServer hypervisor is similar to that of ESXi hypervisor; therefore, discussion about the results are not repeated. In this case, the total process of training an ANN model with the XenServer hypervisor data took only 0.1230 seconds.

6.12.3 Case 3: Execution time prediction for a parallel workflow

In previous sections, the task execution time variation due to different resource intensive co-located VMs are investigated. It is explained previously that a scientific workflow usually consists of a set of smaller tasks. Here, the methodology of the previous sections will be applied to the tasks of a scientific workflow.

In Chapter 5, experiments have been done with a scientific workflow, called the GALFA-HI. In this section, the same workload is used for testing the prediction ability of the ANN model. Recall, the GALFA-HI workflow in introduced in the Section 5.7.1 and shown in Figure 5.1.

The GALFA-HI is an ongoing survey and will eventually cover 13,000 square degrees of the sky on completion. That is a considerable portion of the sky where the sky is divided into smaller parts for detailed data collection. Thus, the GALFA-HI workflow is run again and again on the small data cubes to combine them all into a larger mosaic image. In a virtualized data center, the number of co-located VMs is going to be different from each run of the workflow. Reliable task execution time prediction for any combination of co-located VMs can improve the server resource utilization.

Performance prediction procedure used for the tasks of GALFA-HI workflow is shown in Algorithm 1. It is based on the experimental observation of the previous sections. Results from cases 1 and 2 show that task execution time depends on the total number of co-located VMs and their resource usages types.

Furthermore, recall that in Section 4.3.2 it was shown that the actual location of VMs on the server is not a major influencing factor. The cases also show that it is possible to predict the task execution time for any combination of resource-intensive co-located VMs using an ANN. It is low-cost and can be updated at run-time with new data. Those observations are put together to create the Algorithm 1.

The execution time variation of each task of GALFA-HI workflow is profiled individually using the ICBM, described previously in Section 4.4. Figure 5.1 shows that the GALFA-HI workflow has eight levels. Recall that, the execution time of

Algorithm 1 Procedure for task execution time prediction.

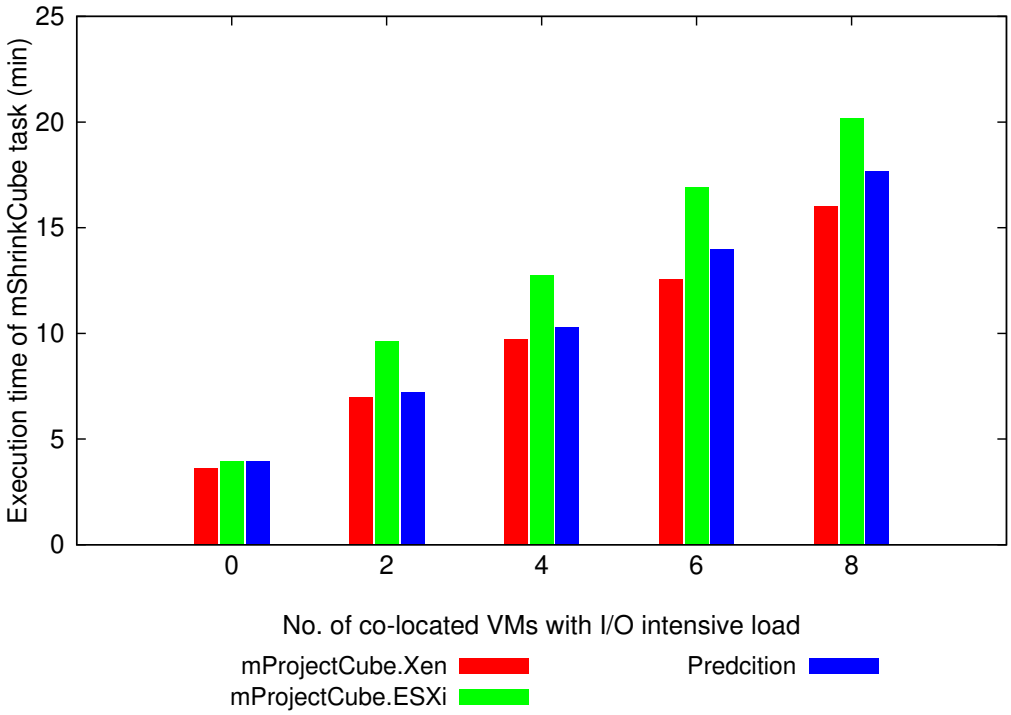
1: **Input**: A combination of VMs on the server and a task.
2: **Output**: Expected task execution time.
3: Categorize the co-located VMs according to their resource intensities
4: and count the total VM number for each resource type.
5: **if** Final count of VMs matches with a stored combination **then**
6: Output the stored expected task execution time.
7: **else**
8: Use the previously trained ANN to predict the task execution time
9: When task execution finishes, use the new data to update the ANN training.
10: **end if**

the tasks of each level are shown in Table 5.1. It shows the execution time of the tasks without any interference from co-located VMs. As shown in the Table 5.1 the tasks mShrinkCube, mImgtbl, mMakeHdr, mProjectCube, mAddCube, m-GetHdr, and mViewer take 3.878, 0.02, 0.02, 39.774, 12.32, 0.02, and 0.04 minutes to execute, respectively.

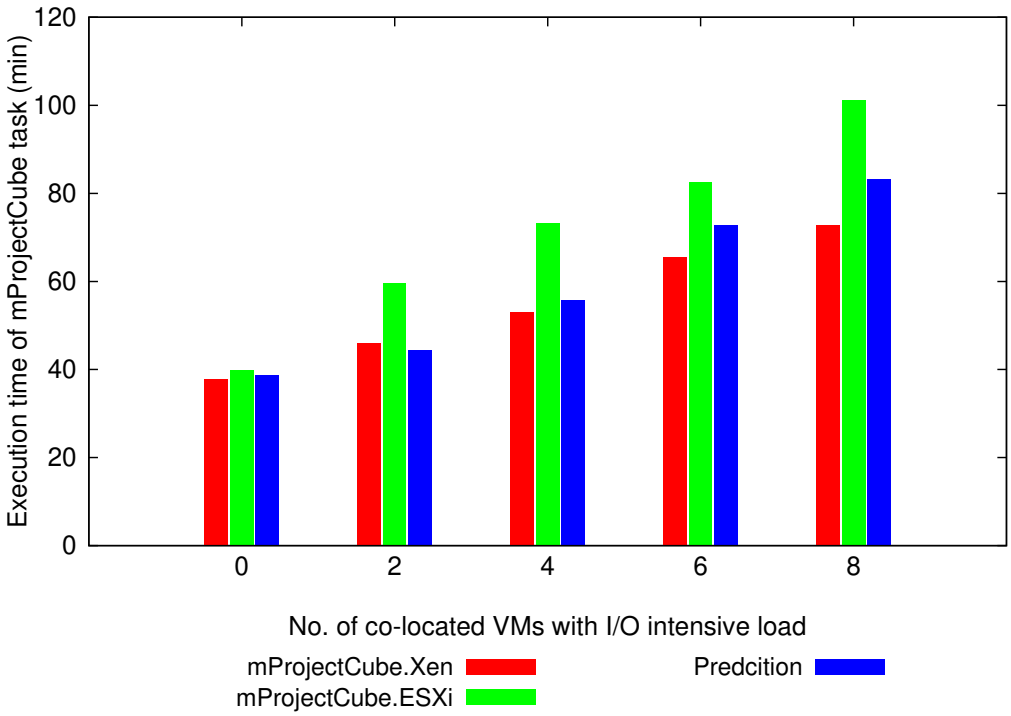
The execution time variation predictions for two of the tasks are shown in Figure 6.27. They are the *mShrinkCube* and *mProjectCube*. Figure 6.27a shows the performance variation of the mShrinkCube task for I/O-intensive resource contention in the server. Here, mShrinkCube is executed with a different combination of co-located VMs on the XenServer, just as described in cases 1 and 2. With those execution time variation data the ANN model is trained.

The prediction results are tested against data of two other hypervisors; Xen and ESXi. To test the prediction model, random numbers of I/O intensive VMs were run with the mShrinkCube task on both hypervisors. The collected execution time variations are then grouped and ordered according to the number of co-located VMs as shown in Figure 6.27a. The Xen hypervisor ETV data is shown in red, while ESXi data is shown in green. ANN execution time predictions are shown in blue. The RMSE of predictions for mShrinkCube is 2.32, and 1.02 on ESXi, and Xen, respectively.

Next, ANN prediction for the mProjectCube task is shown Figure 6.27b. In this



(a) mShrinkCube task.



(b) mProjectCube task.

Figure 6.27: Execution time prediction for two tasks of GALFA-HI workflow.

case, also the XenServer data is used to train the ANN model. The prediction ability of the model is tested against the data from Xen and ESXi, separately. The RMSE of prediction for the mProjectCube task (Figure 6.27b) on ESXi, and Xen are 13.93, and 5.82, respectively.

Algorithm 1 is straightforward and can be easily integrated with any existing system. Both the training and prediction overhead are low. Also, the ANN model can be easily retrained with new data during runtime to improve prediction accuracy.

6.13 Conclusion

This chapter addresses an important issue related to VM performance in data centers, the ETV of VMs due to consolidation on multiple hypervisors. Task performance variation due to consolidation is a major barrier to efficiently scheduling scientific workflow on virtual clusters. Accurate predict of task performance can be a crucial factor for increasing the server efficiency and resource utilization.

Recall that Chapter 4 presents a methodology (ICBM) for profiling and building performance model for consolidated VMs. Previously, in Chapter 5 a framework is implemented that allows applying ICBM to a large number of VMs and hypervisors. This chapter combines the works of the previous chapter to present several prediction models for multiple hypervisors based on profiling. The models can predict the performance variation of VMs due to three resource combinations; CPU-Memory, CPU-I/O, and Memory-I/O. Experimental results are shown from three well-known hypervisors; VMware ESXi, XenServer, and Xen. From the experimental data of the three hypervisors, unified execution time prediction models are built for consolidated hypervisors.

The experimental results presented here are real system data, and no simulation has been used in this chapter. First, the ETV due to basic workload types is profiled with the help of ICBM. The basic workload types are the CPU, memory and I/O. Then, this data is used to predict the ETV due to resource combinations, like CPU-Memory, CPU-I/O, and Memory-I/O.

The ICBM presented in Chapter 4 is a VM consolidation performance profiling technique that can be used with various machine learning and prediction algorithms. In Chapter 4, only LSR is used to build prediction models from the collected data. In this chapter, both LSR and ANN models are trained with the data collected to show

that the ICBM can work with both.

In this chapter, an R programming language package is used to build LSR prediction models. Detail of usages of the package is already given previously in Section 4.9. The unified LSR models are built using three hypervisors data; however, training and testing are done using mutually exclusive data set. Nine random workloads have been run on three hypervisors each to test the prediction accuracy of the models, and their results are shown in Tables 6.8, 6.9, and 6.10. The tables show that the mean absolute percentage error (MAPE) for ESXi, XenServer, and Xen hypervisors are 9.5255%, 29.7558%, and 12.5249%, respectively.

This chapter also presents prediction models built with an artificial neural network (ANN) with backpropagation. Section 6.11 discusses about ANN. Section 6.11.1 gives theoretical background about resilient Back propagation with weight backtracking. Section 6.11.2 describes the `neuralnet` package of R programming language that is used to build ANN models.

Tables 6.13, 6.14, and 6.15 shows the prediction results for Sysbench CPU test, Stream, and Filebench on the Xen hypervisor, respectively. Tables show that the prediction model MAPE for these three tasks on Xen are 20.50%, 11.78%, and 21.02%, respectively. On the other hand, Tables 6.16, 6.17, and 6.18 show the prediction results for the same three tasks on the ESXi hypervisor. The MAPEs of prediction for the three tasks are 22.17%, 12.31%, and 15.88%, respectively.

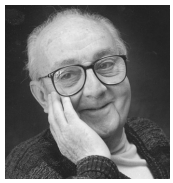
The MAPE results presented in this chapter show that in some cases the LSR models perform little better than ANN models. This is expected as it is known that the training data size can affect the accuracy of machine learning methods [327–330]. Neural networks are designed to work with a large set of data and usually produces a better result when hundreds and thousands of data samples are present. For a relatively smaller set of data, the accuracy of LSR prediction models can outperform that of an ANN.

Chapter 7

Analyzing the impact of memory allocation on the consolidated virtual machines performance using the ICBM

"Essentially, all models are wrong, but some are useful"

— George Edward Pelham Box* (1919–2013)



7.1 Introduction**

The *Incremental Consolidation Benchmarking Method* (ICBM) was introduced in Chapter 4. Then, Chapter 5 presents a framework to apply ICBM on multiple hypervisors effectively. Chapter 6 uses the data collected through the framework to train *Least Square Regression* (LSR) and *Artificial Neural Network* (ANN) models. This chapter continues experiments with the consolidation performance and resource allocation.

Virtualization is one of the most critical technologies behind the Cloud. It plays a significant role in providing the Cloud services. In this chapter, the principle of ICBM

*Image source: `http://bulletin.imstat.org/2013/07/obituary-george-e-p-box-1919-2013/`

**The contents of this chapter were published as [45]. The author of the dissertation has designed the study, conducted the experiments and collected the data. The author has also done all the data analysis and drafting.

is used to analyze the impact of resource allocation on consolidated *Virtual Machines* (VMs). A virtualized server may host many VMs, which are called the *co-located VMs*.

VMs are consolidated to increase the server resource utilization. However, it leads to VM performance degradation. In data centers, often the VM number per server is kept limited to maintain a certain performance level. As a result, the overall utilization of servers in data centers is low [390]. Performance prediction of consolidated VMs can help data center owners to take an informed decision about how many VMs to run without facing too much performance penalty.

The *Resilient Distributed Datasets* (RDDs) was introduced in 2012, it performs all operations in the main memory of a Cluster [391]. It claims to be twenty times (20x) faster than Mapreduce [231], Dryad [392] and other parallel data processing frameworks. However, it is not made clear whether such performance can be gained for consolidated VMs, too. This work looks into the relationship between VM resource allocation and consolidation performance. In particular, how the VM memory allocation and system events like interrupt and page-faults play a role in the consolidation.

System interrupt and page-fault handling are crucial issues for the hypervisor. VM scheduling is a hierarchical process [393–395]. The hypervisor is responsible for scheduling the vCPUs and other physical resources among the VMs. Virtualization process divides a physical CPU into several virtual CPUs or vCPUs, and this makes the interrupt and page fault handling process much more complicated for the hypervisor. Most Cloud workloads are either I/O or memory intensive [396, 397]. Both the I/O and memory intensive workload generates a lot of system-interrupts and page faults. Thus, handling too many interrupts acquires a lot of system resources leading to a reduced application performance [398].

Recently, the VM consolidation and application performance of virtualized servers have received a lot of attention from the researchers across-the-board [68, 292, 294, 295, 399–412]. Nonetheless, those works do not explore the connection between the system events and consolidation performance. Various host system event data can be logged and later analyzed [413]. However, getting an accurate picture of the system performance from them is much more challenging [414]. Previously, it is also shown that not all the host system event data is statistically significant [415].

A VM can not directly access the shared physical resources like the memory or I/O. The hypervisor is responsible for sharing such resources with the VMs. There are

three primary resources to be shared; CPU, memory and I/O. Among them, the CPU is the easiest to schedule. However, memory and I/O virtualization techniques are more complicated. In general, accessing any shared virtualized system bus is a costly and cumbersome process.

VMs cannot perform operations on the memory outside the region allocated by the hypervisor. For example, to handle a page fault first, the VM sends an interrupt signal to the hypervisor. The hypervisor initiates the appropriate course of actions. When the process is complete, the hypervisor sends back the new memory pages addresses to the VM. The hypervisor may receive interrupt requests from multiple VMs at once.

The sequences make the interrupt handling process for the hypervisor more costly compared to that of the physical machine. Similarly, when a VM needs to access the shared I/O bus, it needs to send an interrupt signal to the hypervisor. Since the memory and I/O intensive tasks can generate a lot of interrupts, their performance tends to suffer the most in consolidated servers. To improve the efficiency of virtualized servers their performance variation under consolidation is investigated here. The contributions of this work are briefly stated below:

i) This work investigates the effect of resource allocation and critical system events on the VM consolidation. Experimental results show that such system events have a significant influence on the consolidated VMs performance and by careful resource allocation it is possible to improve the performance;

ii) Results also indicate that the host hardware counters do not provide a reliable picture of the system events like the interrupts and page-faults. However, the co-located VMs system event data can be used better for performance prediction;

iii) We show that the execution time profiling can help to predict the consolidated VMs performance variation. VMs system counter data is used to train an Artificial Neural Network (ANN). Then, the ANN is used to predict the performance variations of the tasks of a parallel workflow.

7.2 Motivation: consolidated VMs performance and host event counters

A parallel workflow can be broken into smaller tasks. Figure 5.1 shows an example of parallel work flow [297] called the *GALFA-HI*. It is introduced and discussed in Section 5.2 of Chapter 5. The GALFA-HI data archive stores a large-scale survey data of the Galactic neutral hydrogen [416]. It is an ongoing project with the *Arecibo L-band Feed Array* (ALFA) radio telescope system [417].

The workflow comprises of sixteen tasks, which are discussed in detail in Section 5.7.1. The workflow is re-introduced here because it is required for clarification and discussion of this chapter. In the next section, the relevance of data and code reuse in the context of workflow scheduling is discussed.

7.2.1 Data and code reuse and Experimental setup

To schedule the tasks of a scientific workflow (like that of Figure 5.1 in Section 5.2) on VMs it is necessary to take into account the effect of consolidation on VMs. When a group of tasks is scheduled on a group of separate physical machines, the performance interference is not an issue. However, as shown in the last several chapters the consolidated VMs exert pressure on each other, and as a result, their performance degrades.

In the last few chapters, the effect of consolidation on task execution time has been analyzed. In this chapter, the relation between the task execution time variation and the system event and page faults are examined. Concurrently running VMs on a host server can generate a lot of event counter data. It is intuitive to collect the host data and analyze. However, results of the next section will demonstrate that it is not the best approach for the analyzing the VM performance interference.

Data and code reuse are standard practice in the Cloud [391]. Some examples include iterative data mining applications, like the *MapReduce* [418, 419] and *Dryad* [392]. Reuse of data is also common in iterative machine learning and graph algorithms, like the *PageRank*, *K-mean clustering*, and *Logistic regression*. In above cases, the same function is repeatedly applied to the subset of data. With each iteration the data volume reduces.

An experiment is set up to replicate the above scenario with a Dell XPS-8500

system. It has one Intel i7-3770 processor with four cores, eight threads, and 32 GB memory. Xen 4.6 is deployed on the system, and sixteen VMs are set up. Each VM has one virtual CPU (vCPU), 2GB RAM, and 50 GB hard disk space.

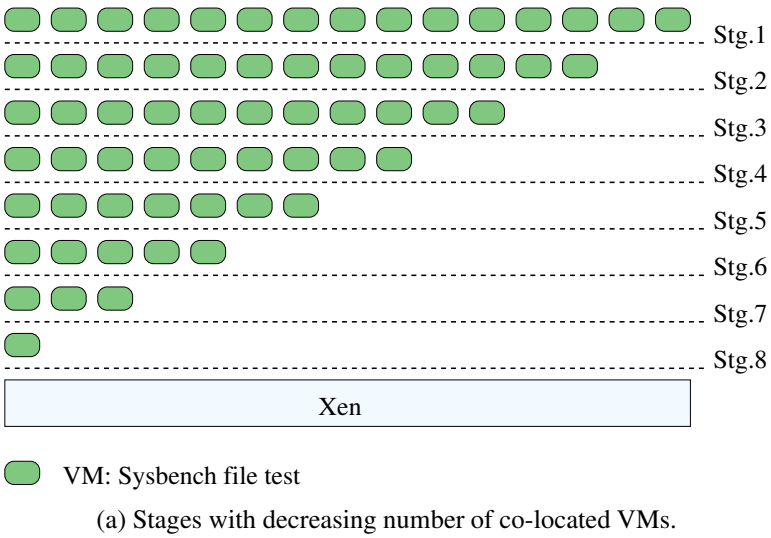
Now that the basic idea and set up of the experiments are described the next section explains how the experiment is performed.

7.2.2 Consolidation performance of an I/O intensive task

The Sysbench file I/O test is used in each co-located VM. The test performs random reads and writes operations on 1GB file data. Each task instance issues 100,000 random reads and writes requests. Experiments are done in stages as shown in Figure 7.1a. At stage 1, fifteen instances of the test are run on fifteen VMs and their execution finish times are recorded. In stage 2, the same tests are run on thirteen VMs.

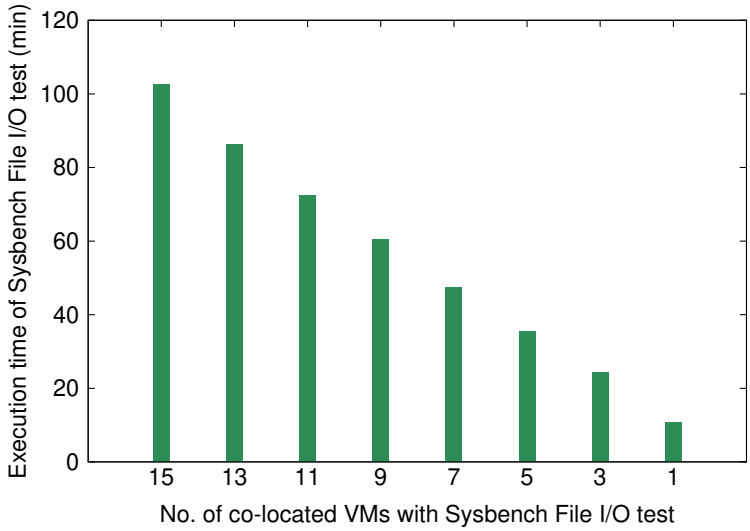
The iterative machine learning and graph algorithms perform repeated operations on the sub-datasets. At each iteration, the data is refined and reduced in volume. The process continues until the desired result set is obtained. In this experimental setup the I/O-intensive VMs are repeatedly run; at each stage, the number of co-located VMs is reduced. Thus, the total volume of data to be processed becomes smaller in successive stages.

Figure 7.1b shows the arithmetic mean of the execution times of the tasks at each stage. The experiment is run three times and the execution time of each task at each



(a) Stages with decreasing number of co-located VMs.

Figure 7.1: Experiment with the Sysbench File I/O test in Xen.



(b) The arithmetic mean of execution times of the stages.

Figure 7.1: Experiment with the Sysbench File I/O test in Xen (Continued).

stage is collected. For example, at stage 1, fifteen VMs simultaneously run the Sysbench file I/O test. The execution finish times of the fifteen tasks are 102.61, 102.39, 102.62, 102.32, 102.40, 102.05, 102.44, 102.02, 102.57, 102.47, 102.62, 102.19, 102.64, 102.54, and 102.45 minutes, respectively. The first bar of Figure 7.1b shows their arithmetic mean; 102.42 minute.

Similarly, the next bar shows the arithmetic mean of the tasks of the second stage where thirteen VMs are simultaneously running (second line of Figure 7.1a), and it is 86.30 minute.

It is clear that as the number of VMs are decreasing so is the arithmetic mean of the execution times. In other words, the co-located VMs performances are improving in the successive stages. Thus, from the data, it is easy to see how the task performances are affected by the co-located VMs.

Next, it is going tested how the consolidation performance correlates with the host system event counters.

7.2.3 Host system counter data for Sysbench File I/O test

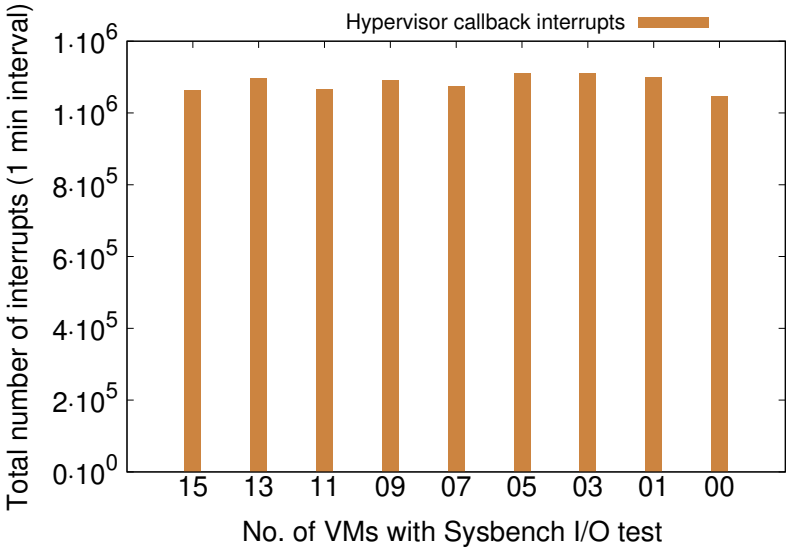
The I/O-intensive VMs generate a significant number of system interrupts and page faults. The objective of the experiment is to observe how the host system event counters are affected by the consolidated I/O workload. Figure 7.2 shows changes in the host

system interrupt, and page fault counter data during the experiments of Figure 7.1a. All data is collected from the *dom0* while the different stages of VMs were running. The *proc* filesystem keeps detail record about the system performance and processes [420]. Data is collected at a one-minute interval for the duration of the execution of each stage (each line in Figure 7.1). Figure 7.2 shows the arithmetic mean of the collected data.

Figure 7.2a shows the arithmetic mean of the *Hypervisor callback* (HYP) interrupts of the host. It is intuitive to think that more VMs with I/O activities will noticeably increase the host interrupt counters. However, it is shown in Figure 7.2a that the host system-interrupt values do not deviate much due to the change in the number of co-located VMs. For example, the HYP interrupts due to fifteen co-located VMs is 1061583 interrupts/minute (stage 1 of Figure 7.1a).

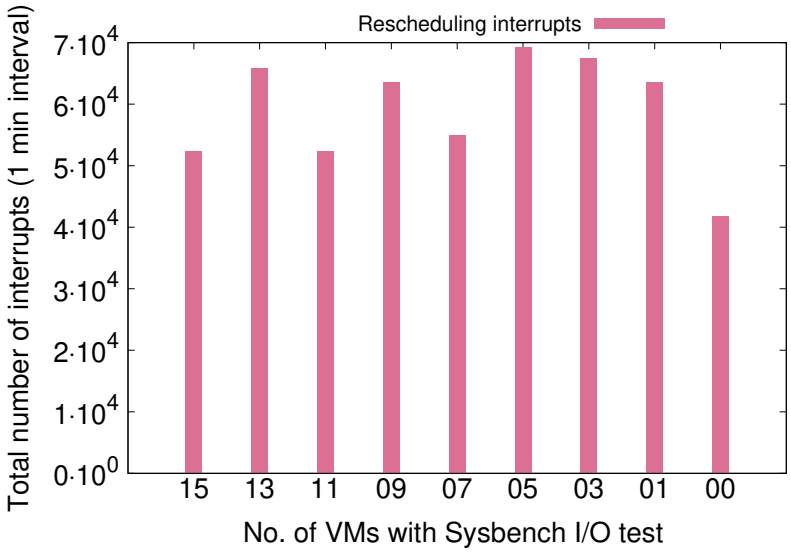
Next bar, shows the arithmetic mean of interrupts collected at the one-minute interval for thirteen co-located VMs, and it is 1096830. Similarly, following bars show the interrupt numbers for successive stages. There is no apprehensible pattern among the values.

The last bar on Figure 7.2a shows the HYP interrupts when no VM is running on the system; the host is still registering 1046581 HYP interrupts per minute. Even though no tasks are running the hypervisor has to perform necessary VM scheduling and maintenance services. The residual interrupts are the result of those services.

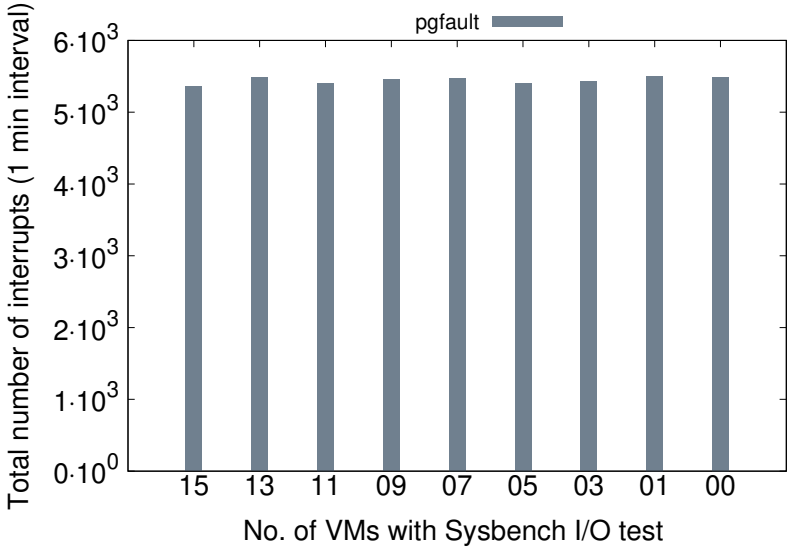


(a) Hypervisor callback (HYP) interrupts.

Figure 7.2: The host event counter data for various stages of the Sysbench File I/O consolidated tests in Xen.



(b) Rescheduling (RES) interrupts.



(c) Page fault.

Figure 7.2: The host event counter data for various stages of the Sysbench File I/O consolidated tests in Xen (Continued).

Next, Figure 7.2b shows the number of *Rescheduling* (RES) interrupts occurred in the host during each stage. The data is collected at a minute interval. Linux uses the RES interrupts to wake up an idle CPU-core to spread the load across the available CPU cores. It can be seen that RES interrupt of the host does not show a significant difference with the change in VM number.

Similarly, Figure 7.2c shows the number of page fault happening at each stage of the experiment. The page fault data are collected from the proc file system with a

one-minute interval. Again, there is no significant pattern appearing among the bars.

Above results show that the host interrupt and page fault data are not good indicators of consolidated VMs performance. It is also true for other types of resource intensive VMs, as shown next.

7.2.4 Consolidation performance of a memory intensive task

Figure 7.3 shows results for consolidated memory-intensive VMs. In this case, the experimental setup is similar to that of Figure 7.1. However, instead of I/O-intensive tasks now memory-intensive tasks are used, and stages number is reduced to five. The stages contain 15, 8, 4, 2, and 1 VMs, respectively. Figure 7.3 shows the arithmetic mean of execution times of the Sysbench memory test for five stages.

The Sysbench memory allows setting data volume for the test. Figure 7.3 shows the execution time variations for two different volumes, 512GB and 1024 GB. The first pair of bars shows the task execution times of both volumes for fifteen co-located VMs. For 512GB data the execution completion time is 17.47 minute.

The Sysbench is designed to test the overall memory throughput of the system, not only the main memory. Recall that each VM is allocated 2GB RAM only. Thus, not all test data is unique, and the same values may be used multiple times, this is a part of the test, too.

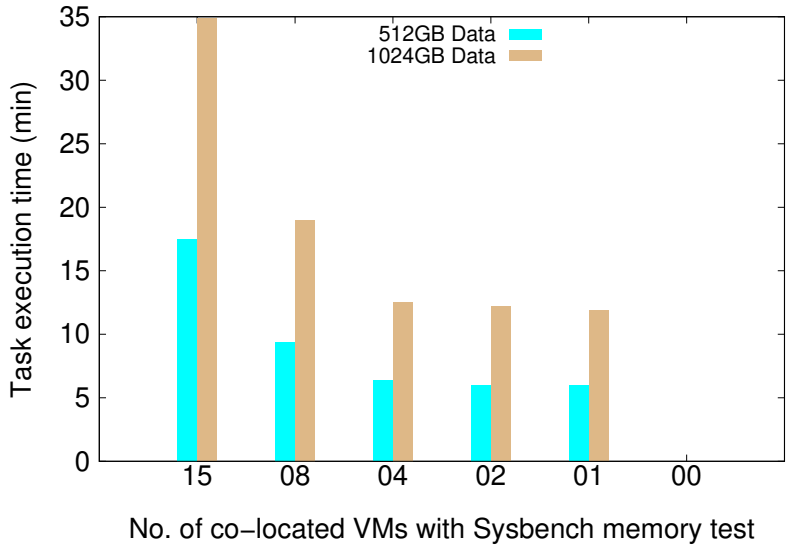


Figure 7.3: Execution times (min) data for various stages of the Sysbench memory consolidation tests in Xen.

The second bar of Figure 7.3 (shown in light brown) shows the arithmetic mean of execution times for 1024GB data; it is 34.84 minute. That is now on all fifteen VMs the Sysbench memory is set to process 1024GB data. Hence, the execution time increase is 99.42% compared to the previous volume of data.

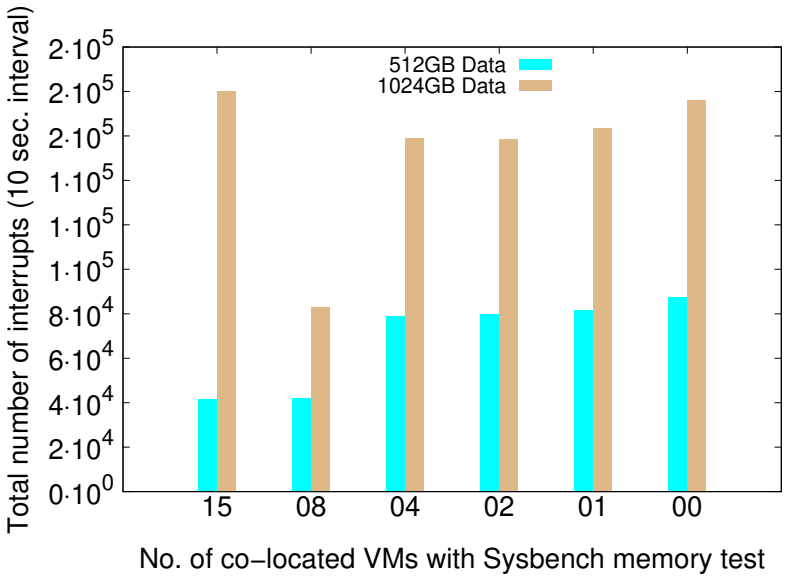
Similarly, next set of bars shows the execution times of other stages of VMs. For 512GB data, the arithmetic mean of execution times of 4, 2, and 1 co-located VMs are 6.34, 5.99, and 5.96 minutes, respectively.

Notice, for up to four, co-located VMs the overall consolidation performance is almost the same. However, the above four VMs the performance starts to degrade rapidly. The same thing happens to the 1024GB data, too. That is, up to four co-located VMs the arithmetic mean of execution times is about 12 minute. Beyond four VMs the execution times start to deteriorate rather quickly.

Next, we check the host system counter data for the same stages of experiment.

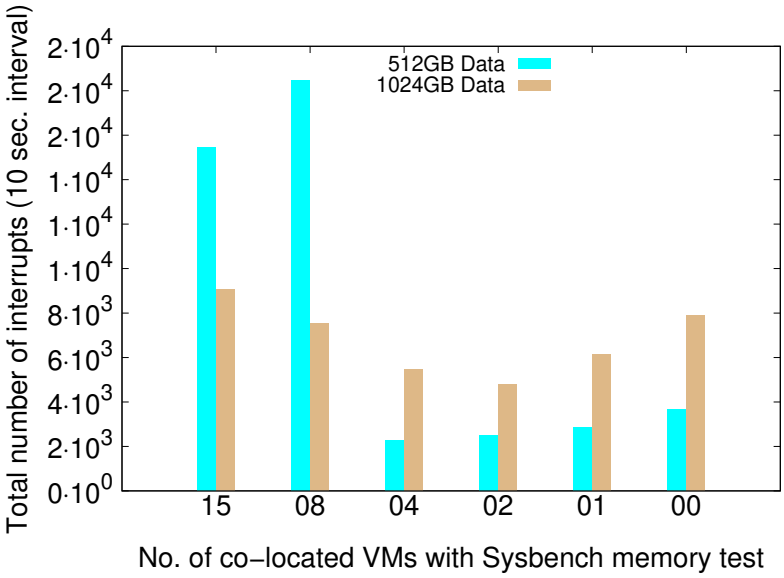
7.2.5 Host system counter data for Sysbench memory test

Figure 7.4a shows the host HYP interrupts number changes during the same stages of consolidation. Here, the interrupt data is pulled from the host proc filesystem with 10 seconds interval. The HYP interrupt data does not show any pattern of change due



(a) Hypervisor callback (HYP) interrupts.

Figure 7.4: The host event counter data for various stages of the Sysbench memory consolidation tests in Xen.



(b) Rescheduling (RES) interrupts.

Figure 7.4: The host event counter data for various stages of the Sysbench memory consolidation tests in Xen (Continued).

to consolidation stages. Next, Figure 7.4b shows the RES interrupt of the host during the same stages. The RES values do not show any statistically significant pattern, either. Similarly, experiments have been done with other host counter data and resource intensive VM, too. However, in all cases findings are quite similar; the host system counters do not show any pattern of change due to consolidation stages.

7.2.6 Discussion

In summary, the increase in consolidation decreases the overall VMs performances. It is known that, I/O and memory intensive tasks increases system interrupt and page faults [421]. However, the experimental results show that the host event counters do not show any significant pattern. The results are consistent with earlier findings that the host system data are not statistically significant for the consolidation performance monitoring [415].

It is not easy to extract performance information from the hypervisor cache, and page-fault event traces [414, 422]. The main reason behind is that the hypervisor uses many techniques to reduce the overall system interrupt and page-fault overhead. For example, the *Interrupt Coalescing* is used to reduce the interrupt handling cost [393, 396, 423, 424].

Similarly, other techniques are applied to merge the page fault events of co-located VMs. Thus, by just logging the host event counters, it is not possible to get the accurate picture of the performance. On the other hand, profiling task execution times can give a better idea of the performance in practice.

7.3 The consolidated task performance and VM memory allocation

The previous section demonstrates that profiling the system event counter is not a good way to monitor the consolidation performance. On the other hand, the task execution time variation shows a clear pattern with the consolidation performance changes.

This section examines the task execution time variation due to memory allocation of the co-located VMs. In the previous sections, the numbers of VMs are increased in stages. However, the task memory usage is kept fixed. Here, both the number of VMs and the task memory usages are raised to observe their effect on consolidation.

At first, the tasks with small memory requirement is used that can be fitted inside the RAM of the VM. In successive stages, the size of memory is increased so that the memory size becomes too big to fit in the RAM. The objective is to examine the performance variations of two types of consolidated tasks; tasks that can be entirely fit into the RAM and the tasks that can not.

Here, the Cachebench [425] is used as the memory-intensive benchmark. It offers the option to specify the memory size to be used. The Cachebench does not allocate all the RAM at once. First, it allocates a small block in the RAM. Then in successive stages, it allocates slightly larger memory blocks to perform tests. It continues to increase the memory allocation size until a predefined point is reached or runs out of the RAM to allocate.

The Cachebench suite contains two sets of experiments. The first test set includes the memory read, write, and update operations being performed on an array of elements in the RAM.

The second test set uses the *memset()* and *memcpy()* functions to set values and copy memory contents in the RAM. Above two functions primarily used by the C++ applications to acquire memory during the runtime. They are widely used by the memory-intensive application to manipulate the memory elements, and the efficiency

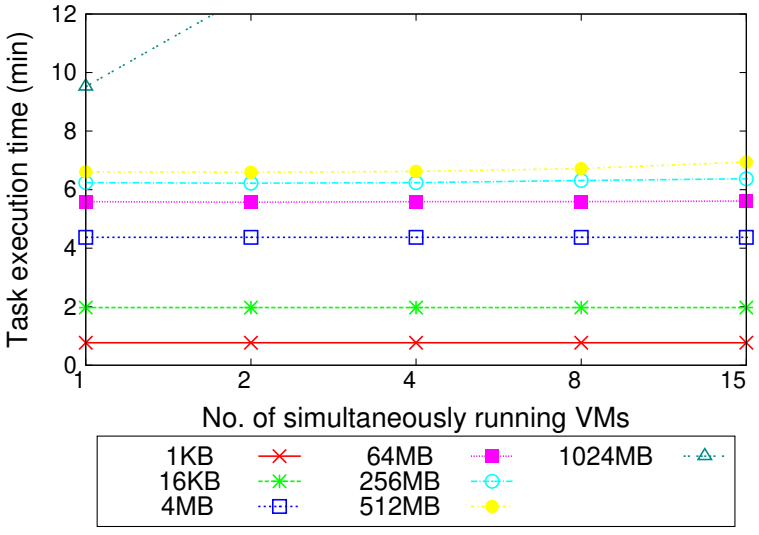
of those applications largely depends on the performance of the above two functions.

7.3.1 Experimental results for Xen

Figure 7.5 shows the execution time variations of two types of consolidated tasks in Xen. For each task, seven different memory setting are used ($1KB$, $16KB$, $4MB$, $64MB$, $256MB$, $512MB$ and $1024MB$).

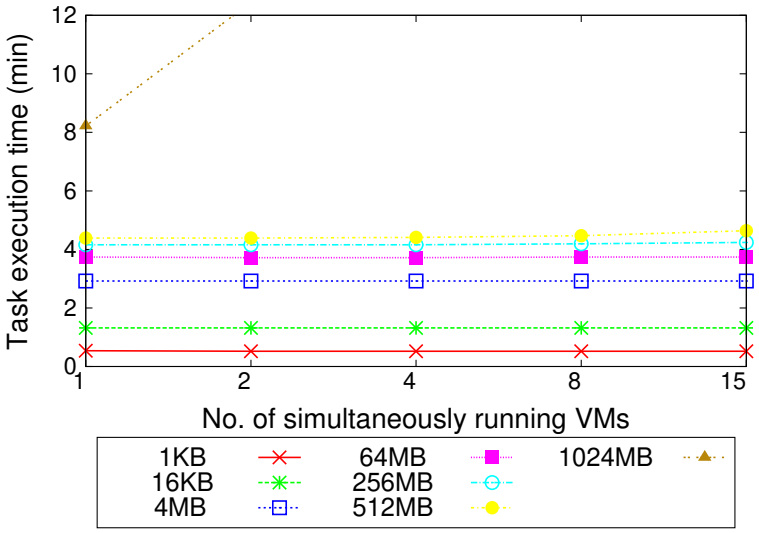
Figure 7.5a shows the execution time variation of the Read/Write/Update test during consolidation. The experimental stages are similar to that of Figure 7.1. First, one VM is run alone on the hypervisor, and in successive stages, the co-located VM number is increased. Each VM has 2GB RAM. The X-axis shows the number of co-located VMs running on the hypervisor.

Figure 7.5a shows that for first six task memory settings ($1KB$, $16KB$, $4MB$, $64MB$, $256MB$, and $512MB$) the execution time of the Read/Write/Update test does not increase much due to the consolidation. The reason is that the task occupies less space in memory than the available VM RAM. In this case, the results show that the consolidation has little effect on the task performance. However, when the task memory allocation is increased to 1024MB, it can not be entirely fit in the VM main memory.

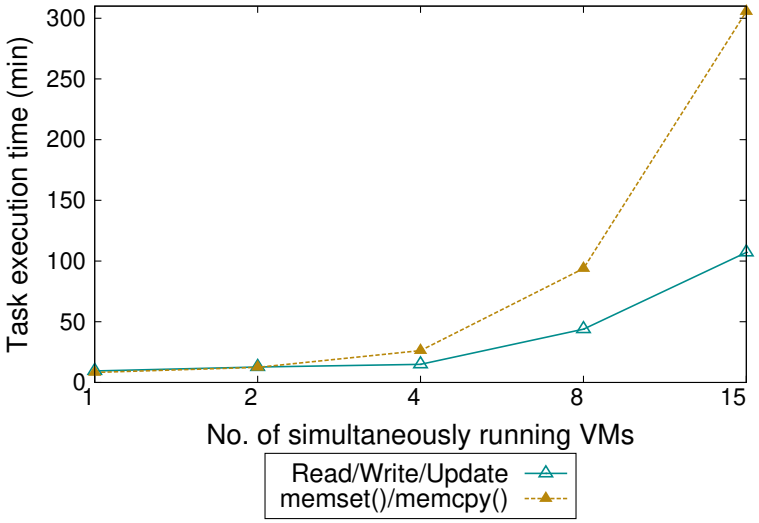


(a) Read/Write/Update test.

Figure 7.5: The Cachebench consolidation performance variation test on co-located VMs with 2GB RAM in Xen.



(b) *memset()/memcpy()* test.



(c) Results for 1024GB data.

Figure 7.5: The Cachebench consolidation performance variation test on co-located VMs with 2GB RAM in Xen (Continued).

Although the VM has 2GB RAM, the kernel processes occupy a significant portion of it. Practically it is not possible to allocate 1024MB memory to the Cachebench now, and that leads to an increased number of page faults and interrupts in the host. The hypervisor has to switch between kernel and user mode more frequently to handle the events. The effect of it is apparent in the consolidation performance.

Thus, for the tasks requiring 1024MB main memory, the consolidated performance degrades rapidly. As the task execution time increases rapidly compared to the previous

six cases, it is shown separately in Figure 7.5c.

The same phenomenon can be shown for other memory-intensive tasks, too. Figure 7.5b shows the execution time variations of the *memset()/memcpy()* tests. Here, the same stages are used as described above. The results show a similar pattern. That is, up to 512MB memory allocation the consolidated task performance degradation is minimum due to the increase of co-located VMs. However, once the task memory requirement becomes too big (like 1024MB) for the VM memory, the consolidated VMs performance starts to degrade rather quickly.

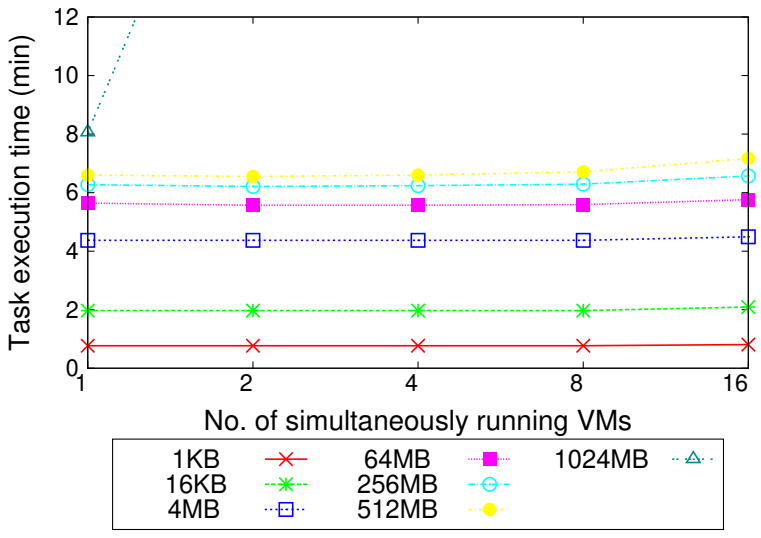
The execution time variation for 1024MB is shown in Figure 7.5c again as it does not completely fit in Figure 7.5b.

7.3.2 Experimental results for ESXi

Above findings are not dependent on the hypervisor; VMs in any hypervisor would react in the same way. Figure 7.6 shows the results of the same experimental stages on the ESXi. Here, sixteen VMs are set up on the ESXi with each VM having 2GB RAM.

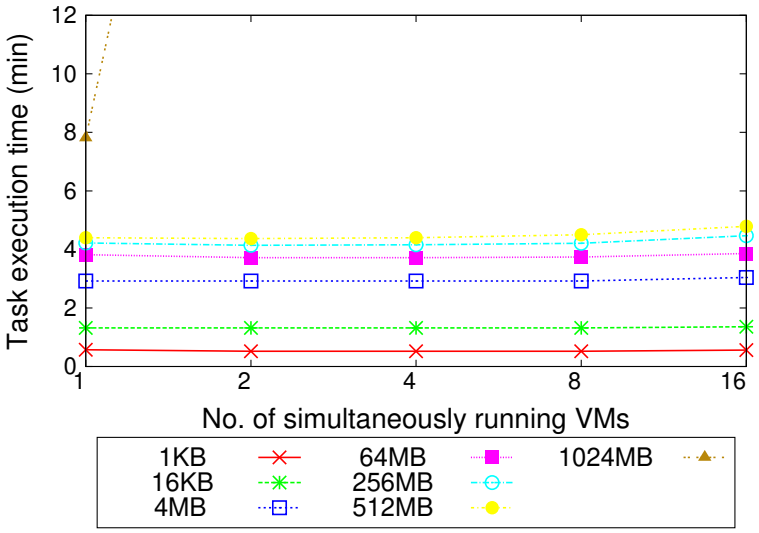
Figure 7.6a shows the results for the Read/Write/Update test. The execution finish time shows little variation as the number of co-located VMs is increased.

Figure 7.6b shows the performance of the *memset()/memcpy()* tests and the results

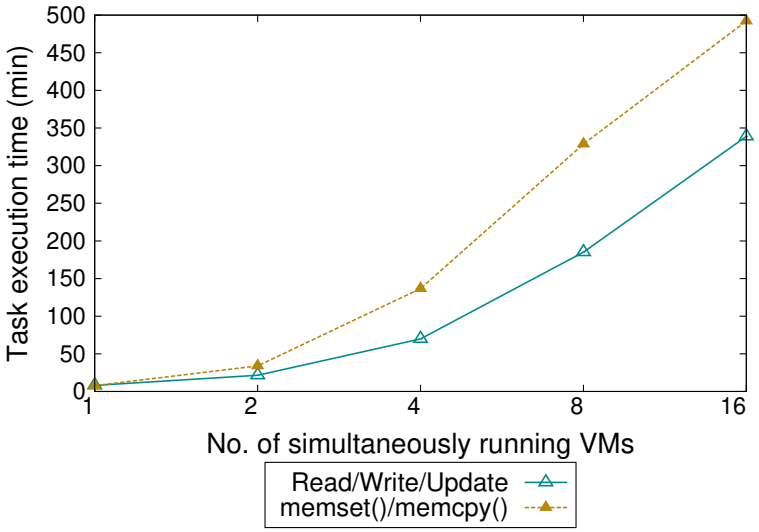


(a) Read/Write/Update test.

Figure 7.6: The Cachebench consolidation performance variation test on co-located VMs with 2GB RAM in ESXi.



(b) *memset()/memcpy()* tests.



(c) Results for 1024GB data.

Figure 7.6: The Cachebench consolidation performance variation test on co-located VMs with 2GB RAM in ESXi (continued).

are also similar. Figure 7.6c shows the consolidation performance variation of both tests for 1024MB memory allocation. The same tests are done on the XenServer, and the results are quite similar. However, experimental results for XenServer are not shown here.

7.3.3 Increased memory size of VMs

One more test is conducted to make sure that the memory size is the primary factor behind the observed results. The VM main memory size is raised to 4GB, and the consolidation experiment stages are repeated with increased VM RAM allocation. Due to the increased memory size now the 1024MB task can comfortably fit inside the VM RAM.

Figure 7.7 show the execution times of the tasks under consolidation on both the Xen and ESXi. First, the Cachebench is run on a VM with 4GB memory. In successive stages, the number of co-located VMs are increased. In contrast to Figs. 7.5c and 7.6c now the 1024MB memory size task show almost no variation with the increase of co-located VM number.

Results of this section prove that necessary memory allocation is critical for VM consolidation performance. Thus, accurate determining of the tasks memory requirements and VMs memory allocation can effectively improve the task performance in the VMs.

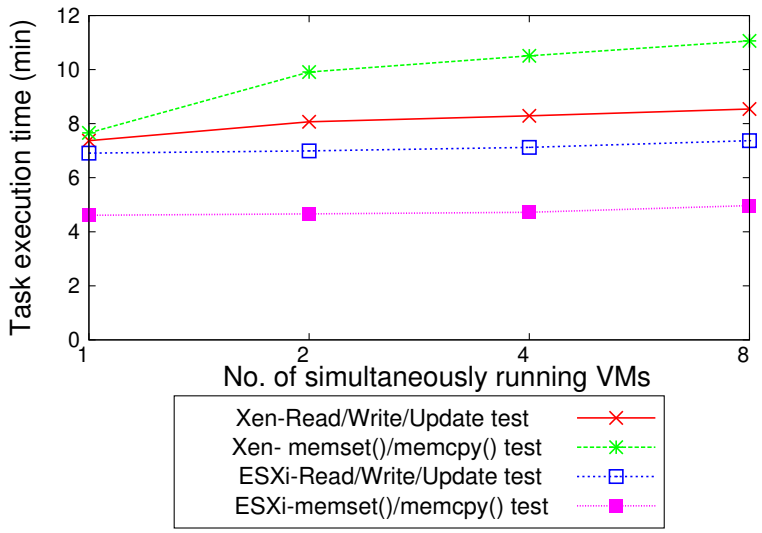


Figure 7.7: The Cachebench consolidation performance variation test on co-located VMs with 4GB RAM.

7.4 Non-memory resident tasks and $\times$-$factor$ graph

The results of the last section show that the task memory requirement has a real impact on the co-located VMs performance. If a task can be fit entirely in the VM main memory, then the task performs better under VM consolidation, this is true for all hypervisors.

In this section, we consider the performance variation of the tasks that can not be entirely fit inside the main memory. Here, the Sysbench memory test is used and it performs a different type of test compared to the Cachebench, which is used in the previous section. The test uses a fixed region of the main memory to perform all the operations, and that size is not altered during the test.

7.4.1 Sysbench memory read test

Figure 7.8a shows the execution time variations of the Sysbench memory read test during consolidation. The test continuously performs read operations on a fixed size VM memory block.

Figure 7.8a shows the time taken to complete the reading of four separate data volumes. The main memory allocation is not altered in this test, and the same values may be read multiple times. The objective is to see how the consolidated tasks performance varies with the increase of data volume. The four different data volumes are 256GB, 512GB, 1024GB, and 2048GB, respectively. Figure 7.8a shows the execution time variation of each volume.

Next, the ratio of execution time increase from the initial value is calculated and shown in Figure 7.8c. The calculation of the execution time increase ratio is straightforward as described below. From Figure 7.8a, one single VM without consolidation takes 2.96 minutes to complete reading of 256GB data.

With two simultaneously running co-located VMs the execution time increases to 3.09 minute. Thus, for two co-located VMs the execution time increase to initial value ratio is calculated as $(3.09 - 2.96)/2.96 = 0.043$.

For 4, 8, and 15 co-located VMs the execution times rises to 3.11, 4.77, and 8.99 minutes, respectively. Thus, their execution time increase ratios are 0.050, 0.611, and 2.037, respectively.

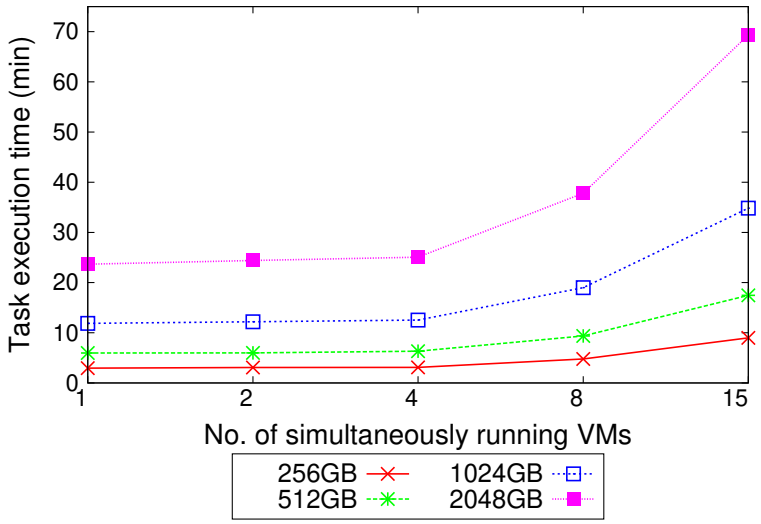
The execution time increase ratios for 512GB, 1024GB, and 2048GB data are calculated as shown in Figure 7.8c. For 512GB data the ratios for 2, 4, 8, and 15 co-located VMs are 0.005, 0.063, 0.572, and 1.931, respectively. Notice, even though Figure 7.8a has different ranges of execution time values; the execution time increase ratios of them in Figure 7.8c show a similar pattern.

7.4.2 Sysbench memory write test

Figure 7.8b shows the results for the Sysbench memory write test. Here, the Sysbench performs the write operations on a fixed region of the VM memory. The task execution time variations due to four different data volumes are shown and their execution time increase ratios are shown in Figure 7.8c.

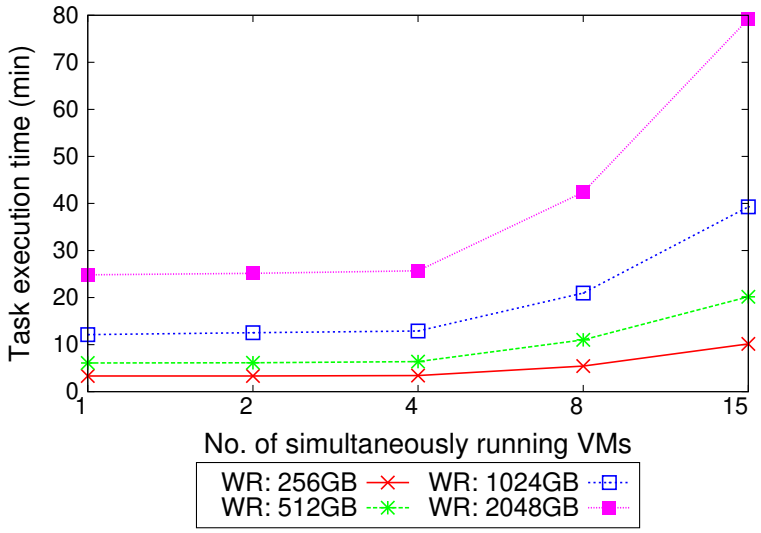
Although Figs. 7.8a and 7.8b have different execution time value ranges their execution time increase ratios are quite similar (Figure 7.8c). Besides the execution time increases ratios for eight data sets the Figure 7.8c has another curve called the $\times$-$factor$ (times factor). It is the arithmetic mean of all the eight execution time increase ratios in Figure 7.8c.

Notice, that the execution time increase ratios of a task remain the same even though data volumes are changed. Thus, a group of task execution time variations can be represented through one $\times$-$factor$ curve.

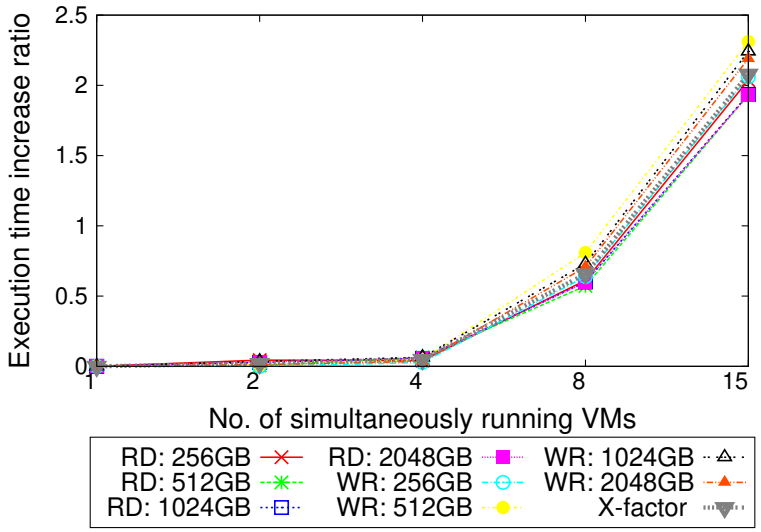


(a) Read from main memory.

Figure 7.8: The Sysbench memory test performance under consolidation in Xen.



(b) Write in the main memory.



(c) Exe. increase ratio.

Figure 7.8: The Sysbench memory test performance under consolidation in Xen (Continued).

This section explains how the phenomenon of $\times$-$factor$ is observed among the profiled execution times of the consolidated VMs. The benefit of using $\times$-$factor$ is that instead of using a wide range of execution time values (like shown in Figure 7.8a and 7.8b) only one set of ratios can be used (Figure 7.8c).

Later in Section 7.5.2, $\times$-$factor$ is further discussed. The section will show how the experimental data can be arranged to calculate $\times$-$factor$. However, to understand the process it is necessary to understand how the experiments are designed and

performed with the execution time variations. That is done in the next few sections.

Section 7.5.1 describes how the experiments are performed with system interrupt and task execution time variation. Then, Section 7.5.2 describes how the collected data is used to calculate the $\times$-$factor$. What is more, the subsequent sections also show how the $\times$-$factor$ data can be efficiently used with a machine-learning process to predict VM execution time variations from the system interrupt data.

7.5 Execution time variation due to system interrupt

In Section 7.3, it is shown that the task memory requirements affect the consolidated VMs performances. The memory and I/O-intensive tasks on co-located VMs increase the number of interrupts and page faults in the system. As the hypervisor interrupt handling process is more complicated compared to a physical machine, the task performance degrades quickly with the increase of co-located VMs.

However, Section 7.2 demonstrated that it is not possible to correlate the performance of co-located VMs with the host hardware counter data. That is because modern hypervisors use various techniques to coalesced VMs interrupts and page-faults. Thus, their performance impact on the consolidated VMs performances cannot be determined from the host event data alone.

Here, an indirect method is used to investigate the performance effect of the interrupts and page-faults on the consolidated VMs. A measured number of interrupts and page faults are injected into the consolidated host to observe the performance variation. First, the total number of interrupts and page faults caused by a selected set of tasks in an isolated system is measured. Then, the VMs are consolidated with the selected set of tasks so that the total amount of injected system interrupts and page faults are known. Next, the process is described in more detail in the next section.

7.5.1 Patterns for injecting system events

Figures 7.9 (on page 305), 7.10 (on page 306), and 7.11 (on page 307) show three different VM arrangements for experimenting with system interrupts and page faults. In this experimental setup, the Sysbench CPU test is used as the test task (shown in red), while the Stress context switching test is employed in the co-located VMs (shown in green). However, the same experiments can be repeated with any other test task or

co-located VMs.

Figures 7.9, 7.10, and 7.11 show how VMs are increased in the system in a step by step process. In all the Figures, first, the Sysbench CPU test is run on a single VM and the execution time is recorded. Then, in the successive stages, the number of co-located VMs are changed. That is how resource contention among the VMs are changed and as a result, the execution time of the co-located VMs starts to vary. Changing the number of co-located VMs is possible in several ways. Three possible patterns are shown in the Figures 7.9, 7.10, and 7.11. Those patterns are discussed next.

Figure 7.9 shows one possible pattern and it has eight stages. The observed execution time variations of the Sysbench CPU test task on all eight stages are shown in Table 7.1. The second column of Table 7.1 shows how many co-located VMs are running on the server at each stage, while the third column shows the execution time variation of the Sysbench CPU test task for the respective stages.

The fourth column of Table 7.1 shows the total system interrupt of all the co-located VMs at each stage. During each stage the system interrupts are collected from all the co-located VMs, and the fourth column is the summation of all those values. Similarly, the fifth column shows the summation of page-faults of all the co-located VMs.

Next, Figure 7.10 (on page 306) shows another pattern of increasing co-located VMs. Here, the positions of co-located VMs on the hypervisor are different from that of the Figure 7.9 (on page 305). The execution time variations, interrupt, page faults and other data are shown in from second to fifth columns of Table 7.2 (on page 306).

From Figures 7.9 and 7.10 it can be seen that VMs are arranged differently at each stage of those two figures. However, Tables 7.1 and 7.2 show that the stage-wise the execution time variations, interrupts, and page-faults are still similar. Thus, the results indicate that VM execution time, system interrupt, and page-fault values are dependent on the total number of VMs not on their location on the server.

Next, Figure 7.11 (on page 307) shows another pattern of co-located VMs. Here, the VMs are arranged in yet another way, differently to that of Figure 7.9 or 7.10. The stage-wise the execution time variations, interrupt, and page-faults of this pattern are shown in Table 7.3 (on page 307).

Even though, the three patterns of Figures 7.9, 7.10, and 7.11 are spatially different; however, Tables 7.1, 7.2, and 7.3 show there is a connection among them. If the data from all the three tables are sorted according to increasing order of co-located

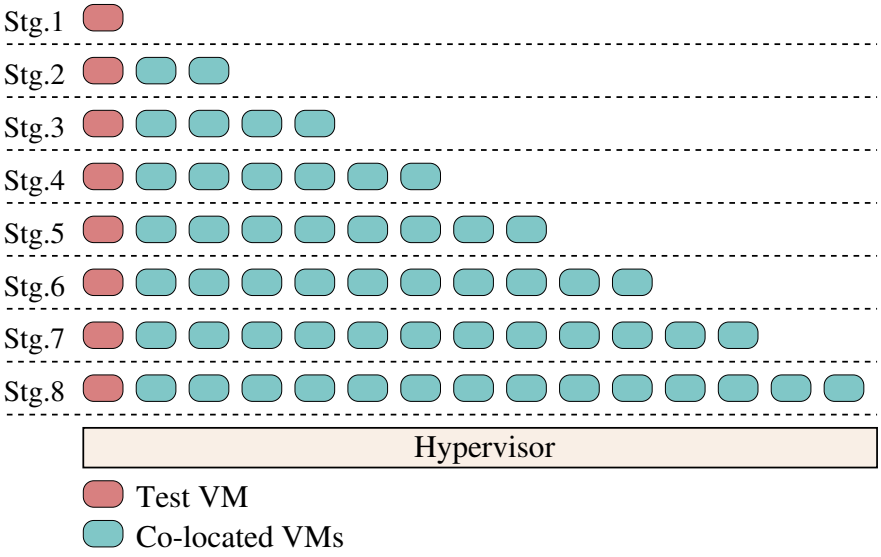


Figure 7.9: Execution time variation of the Sysbench CPU test for VM pattern 1.

Table 7.1: Observed VM execution time variation, interrupts and page-faults for the pattern of Figure 7.9 in ESXi.

Stage	Data for the Pattern 1 in Figure 7.9			
	Number of co-located VMs	Exe. time of Sysbench CPU test	System interrupts	page-faults
1	0	2.39	–	–
2	2	2.47	112802600	7439
3	4	2.57	217010717	14151
4	6	2.86	321218834	20448
5	8	3.39	434021434	28406
6	10	4.14	544675413	33838
7	12	4.82	638892442	41969
8	14	5.51	762760441	49478

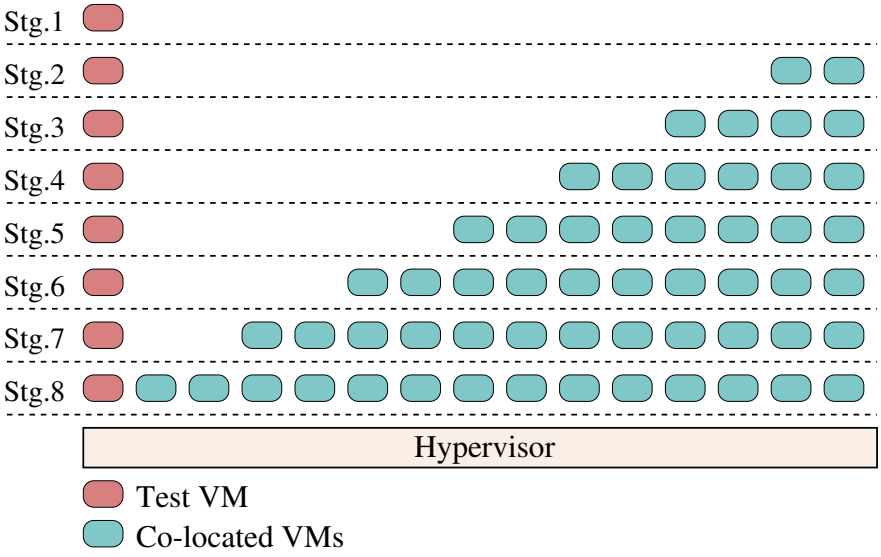


Figure 7.10: Execution time variation of the Sysbench CPU test for VM pattern 2.

Table 7.2: Observed VM execution time variation, interrupts and page-faults for the pattern of Figure 7.10 in ESXi.

Stage	Data for the Pattern 2 in Figure 7.10			
	Number of co-located VMs	Exe. time of Sysbench CPU test	System interrupts	page-faults
1	0	2.31	–	–
2	2	2.36	113534826	7369
3	4	2.39	231979265	12698
4	6	2.61	368834758	20656
5	8	3.16	466413338	26884
6	10	3.77	600200321	33700
7	12	4.42	730305094	40136
8	14	5.02	849363236	47955

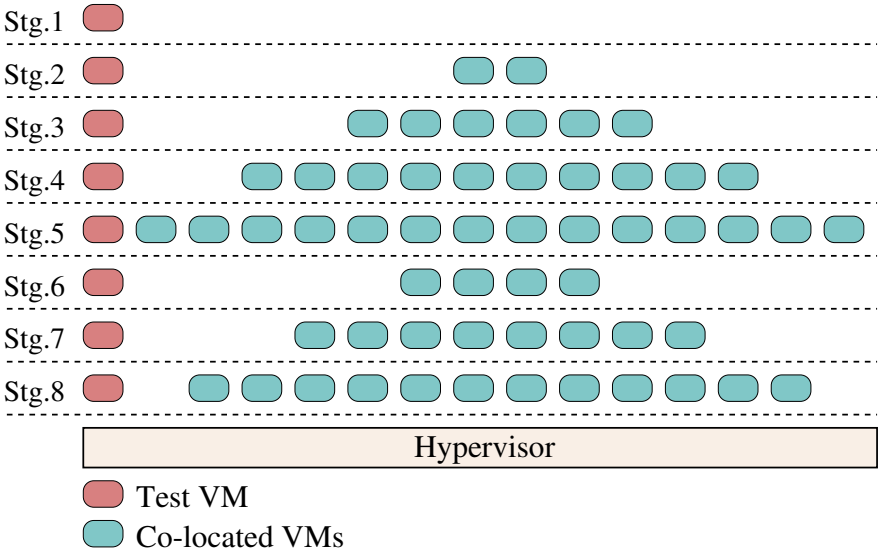


Figure 7.11: Execution time variation of the Sysbench CPU test for VM pattern 3.

Table 7.3: Observed VM execution time variation, interrupts and page-faults for three patterns of Figure 7.11 in ESXi.

Stage	Data for the Pattern 3 in Figure 7.11			
	Number of co-located VMs	Exe. time of Sysbench CPU test	System interrupts	page-faults
0	0	2.29	–	–
2	2	2.34	111113767	6772
4	6	2.64	218363680	20152
6	10	3.81	333507654	32559
8	14	5.02	437369925	45484
10	4	2.41	558735105	11918
12	8	3.19	661446011	26228
14	12	4.44	797065406	39142

VMs number, then it can be seen that their execution times, interrupts and page-faults values are almost the same.

Experiments have been done with other patterns, too. However, when the results are arranged according to the number of co-located VMs at each, they show a similar pattern again. It is observed that no matter in how many ways the co-located VMs are arranged, those values depend on the total number of co-located VMs, not on their locations.

Next section describes how the $\times$-$factor$ can be calculated from the experimental data presented here.

7.5.2 Calculating the $\times$-$factor$

This section describes how $\times$-$factor$ is calculated for the patterns shown in Figures 7.9, 7.10, and 7.11 above. The concept of $\times$-$factor$ is introduced in the Section 7.4. Then, in Section 7.5.1 experiments are conducted to collect execution time, system interrupt, and page-fault data from consolidated VMs. Consolidated VMs are run according to the three patterns of Figures 7.9, 7.10, and 7.11 as described in Section 7.5.1 above. Experimental data for those three patterns are are shown in Tables 7.1, 7.2, and 7.3, respectively.

To calculate the $\times$-$factor$, first the values of three patterns of Tables 7.1, 7.2, and 7.3 are sorted in ascending order according to the number of co-located VMs. After the sorting, the arithmetic mean of the three values of each stage is calculated and shown in Table 7.4.

The second column of Table 7.4 shows the arithmetic mean of the execution time variation of three patterns of Figures 7.9, 7.9, and 7.9. First the values of stages from Tables 7.1, 7.2, and 7.3 are sorted according to the ascending order of the number of co-located VMs. After sorting the stages in ascending order, then the arithmetic mean is calculated from three values at each stage. An example of the calculation process is discussed next.

Recall third column of that Table 7.1 shows all the execution time variation data for the pattern of Figure 7.9. The seventh row of Figure 7.9 shows the execution time variation of Sysbench CPU test for eight co-located VMs, and it is 3.39 minutes.

Similarly, Table 7.2 shows the execution time variation data for the pattern of Figure 7.10. In this particular table, the execution time of the Sysbench CPU test for eight

Table 7.4: $\times$-$factor$ calculated from the data of Tables 7.1, 7.2, and 7.3.

Stage no.	VM no.	Arithmetic mean of execution time $(Exe.time_i)$	$\times$-$factor$ (f_i)	Arithmetic mean of Interrupts	Arithmetic mean of page-faults
1	0	2.33	0.00	–	–
2	2	2.39	0.02	111113767	7193
3	4	2.45	0.05	218363680	12922
4	6	2.70	0.15	333507654	20418
5	8	3.24	0.39	437369925	27172
6	10	3.90	0.67	558735105	33365
7	12	4.56	0.95	661446011	40415
7	14	5.18	1.22	797065406	47639

co-located VMs is shown in the seventh row, and it is 3.16 minutes.

Next, Table 7.3 shows execution time variation data for the third pattern in Figure 7.11. In this table, the Sysbench CPU test execution time due to eight co-located VMs is shown in the ninth row, and it is 3.19 minutes. The arithmetic mean of those three execution time values is 3.24 minutes, and it is shown in the sixth row of Table 7.4. Thus, the sixth row shows the arithmetic mean of Sysbench CPU test execution times due to eight co-located VMs.

In the same way, the arithmetic mean of execution times is calculated for all co-located VM number and stored in the third row of Table 7.4. The rows of the table are sorted according to the ascending order of the number of co-located VMs.

Similarly, the system interrupts, and page-faults data is collected from three previous tables (Tables 7.1, 7.2, and 7.3) and their arithmetic means are calculated. The fourth, and fifth columns of Table 7.4 show the arithmetic means of the system interrupts, and page-faults, respectively. Next, it is explained how the $\times$-$factor$ is calculated these arithmetic mean values.

The fourth column of Table 7.4 is called the $\times$-$factor$, which is the ratio of the execution time increase from the initial value. For any stage, i the ratio is calculated

as follows:

$$\times - factor_i = f_i = \frac{(Exe.Time_i - Exe.Time_0)}{Exe.Time_0}$$

Where, $Exe.Time_i$ is the task execution time at stage i and $Exe.Time_0$ is the initial task execution time. The second column of Table 7.4, shows the $\times$-$factor$ for the Sysbench CPU test data. For example, $\times$-$factor$ for two co-located VMs is 0.02 (third row). It is calculated as follows $(2.39 - 2.33)/2.33 = 0.02$. Similarly, rest of the $\times$-$factors$ are calculated.

The $\times$-$factors$ essentially represents the ratio of how much the execution time varies at each stage compared to the initial value. Table 7.4 shows the $\times$-$factor$ of only one task, the Sysbench CPU test. However, it is possible to calculate the $\times$-$factor$ for other tasks also. Next section presents the calculated $\times$-$factors$ values for two hypervisors; VMware ESXi and Xen.

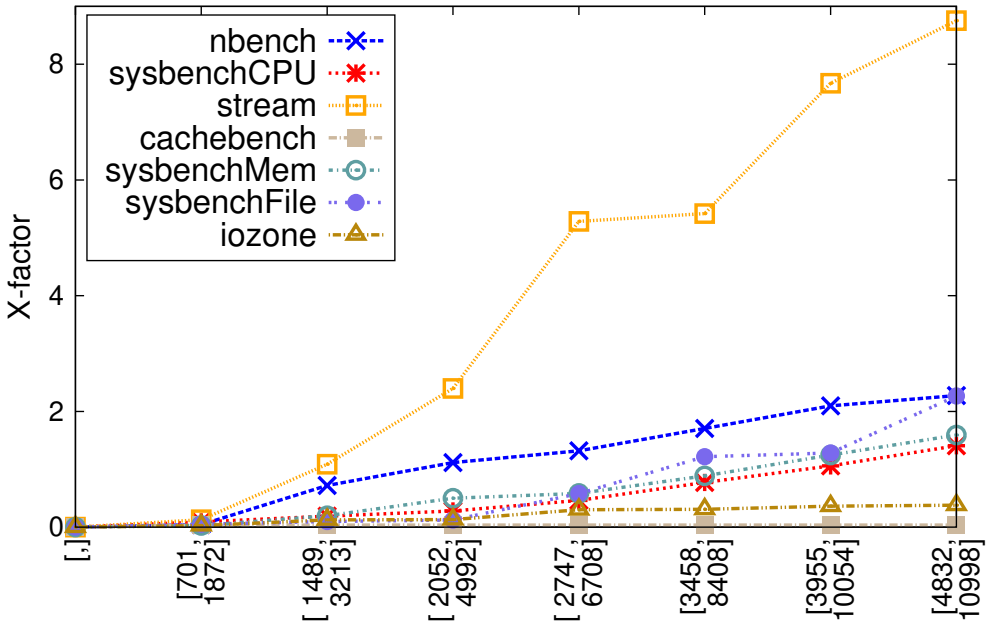
Figures 7.12, and 7.13 of the next section show $\times$-$factor$ of seven tasks on EXSi and Xen, respectively. The values for all the tasks are obtained using the same process as discussed in this section.

7.6 $\times$-$factor$ of tasks on ESXi and Xen

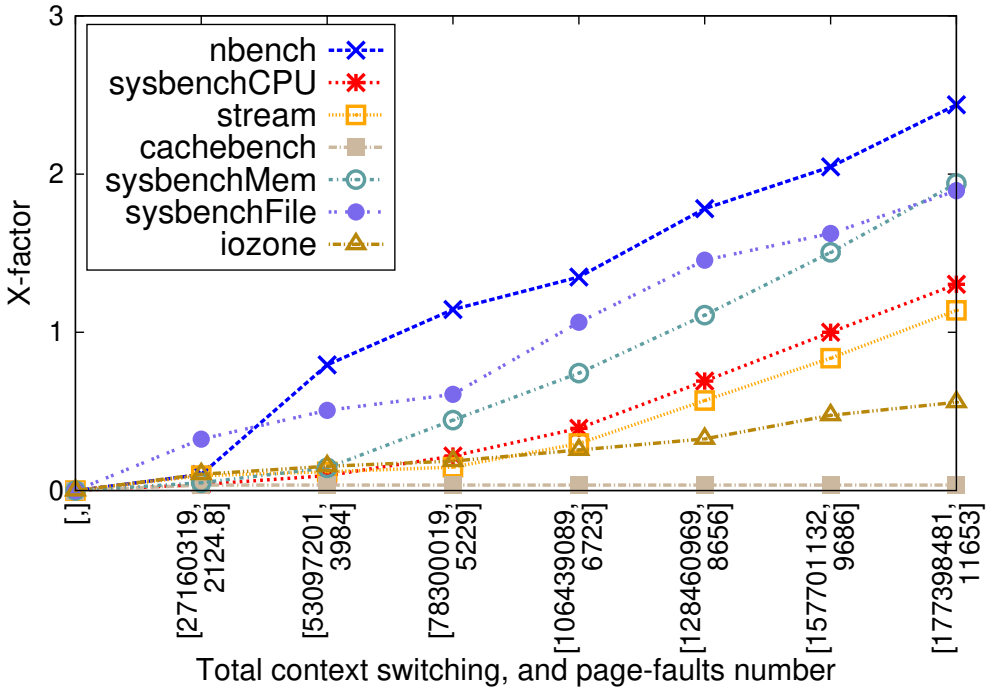
The seven tasks used in each graph of Figure 7.12 are the Nbench [219], Sysbench CPU test [233], Stream [225–227], Cachebench [224], Sysbench memory test, Sysbench file I/O test [233] and Iozone [230]. Recall that those benchmarks are introduced and discussed in the Section 3.4 of the Chapter 3. That is why those benchmarks are not discussed here again.

For those tasks, the execution complition times are collected for various patterns of co-located VMs as described above in Section 7.5.1. Then, the $\times$-$factor$ for each task is calculated according to the process described in Section 7.5.2.

The four graphs of Figure 7.12 are created using four types of tasks in the co-located VMs; the tasks are from the Sysbench [233] and Stress [426] benchmark suites. The four types of tasks are the Stress cache contention, Stress context switching, Sysbench mutex, and Sysbench threading tests; their results are shown in Figs. 7.12a, 7.12b, 7.12c, and 7.12d, respectively.

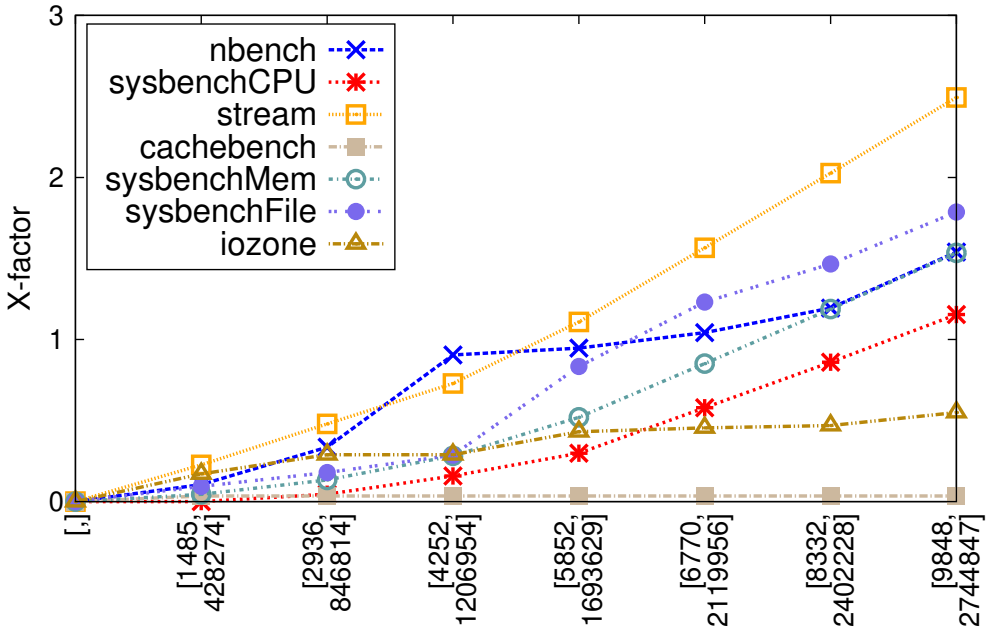


(a) $\times$-$factor$ due to the Cache contention test in co-located VMs.

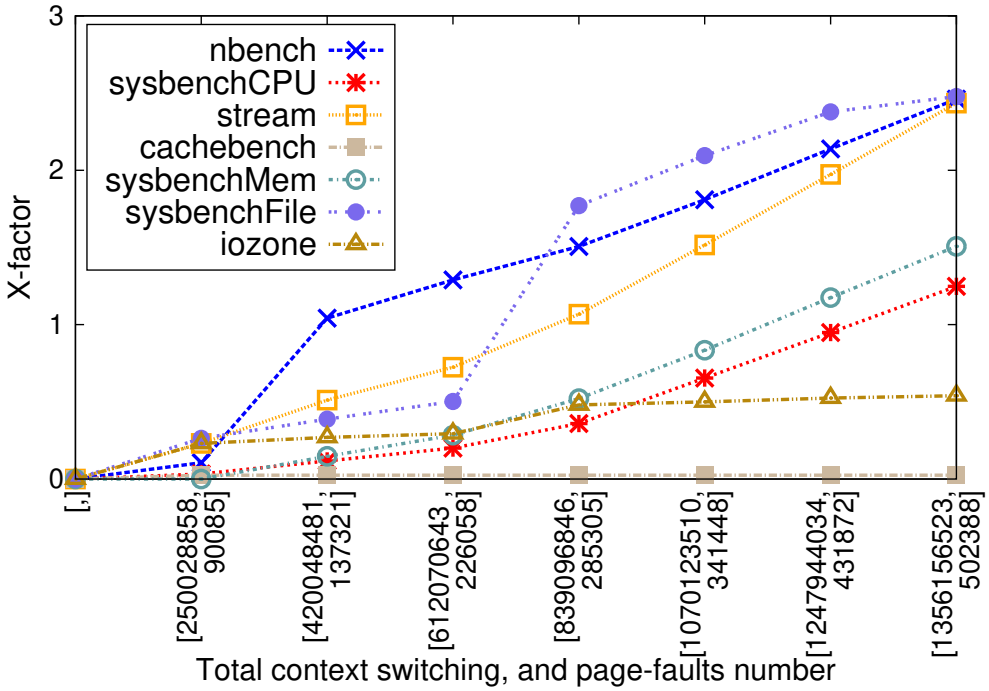


(b) $\times$-$factor$ due to the Stress Context switching test in co-located VMs.

Figure 7.12: Calculating $\times$-$factor$ in the ESXi.



(c) $\times$-$factor$ due to the Sysbench Mutex test in co-located VMs.



(d) $\times$-$factor$ due to the Sysbench thread test in co-located VMs.

Figure 7.12: Calculating $\times$-$factor$ in the ESXi (Continued).

Figure 7.12a shows the $\times$-$factor$ values of seven tasks for the Stress cache contention tests in the co-located VMs. As the number of co-located VMs are increased in the system the execution time of the tasks, start to vary, and this variation is the basis for calculating the $\times$-$factor$.

The system interrupts, and page-faults are collected from all the co-located VMs during the experimental stages. The x-axis of graphs shows the arithmetic mean of the interrupts and page-faults values. The arithmetic means are collected for three different test patterns (Figures 7.9, 7.10, and 7.11) as discussed in Section 7.5.1.

Next, Figure 7.12b shows the $\times$-$factor$ for context-switching tasks in the co-located VMs. The steps are similar to that described above only now the Stress context-switching benchmarks is used instead of the cache benchmark. The same goes for other two graphs (7.12c, and 7.12d).

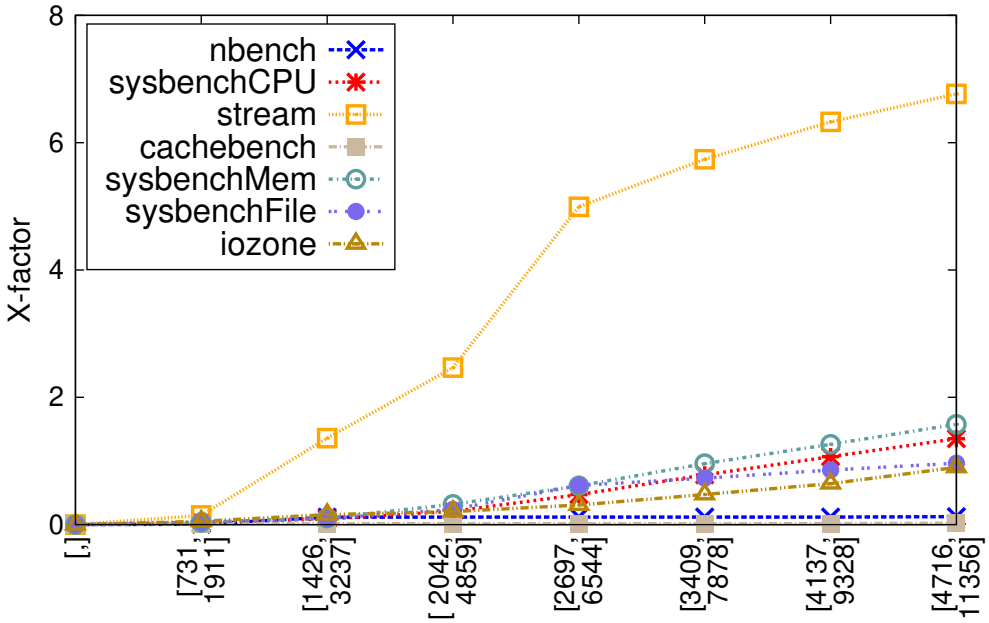
Four graphs of Figure 7.12 shows the relation between the execution time increases of tasks and the total number of system interrupts and page-faults in the co-located VMs. Similar experiments can be performed for other hypervisors, too.

Figure 7.13 shows the execution time variation of the same seven tasks on the Xen for four different types of co-located VMs. For example, Figure 7.13a shows the $\times$-$factor$ values of seven tasks for the Stress cache contention tests in the co-located VMs. In this case of Figure 7.13, the all the experiments are conducted on the Xen hypervisor.

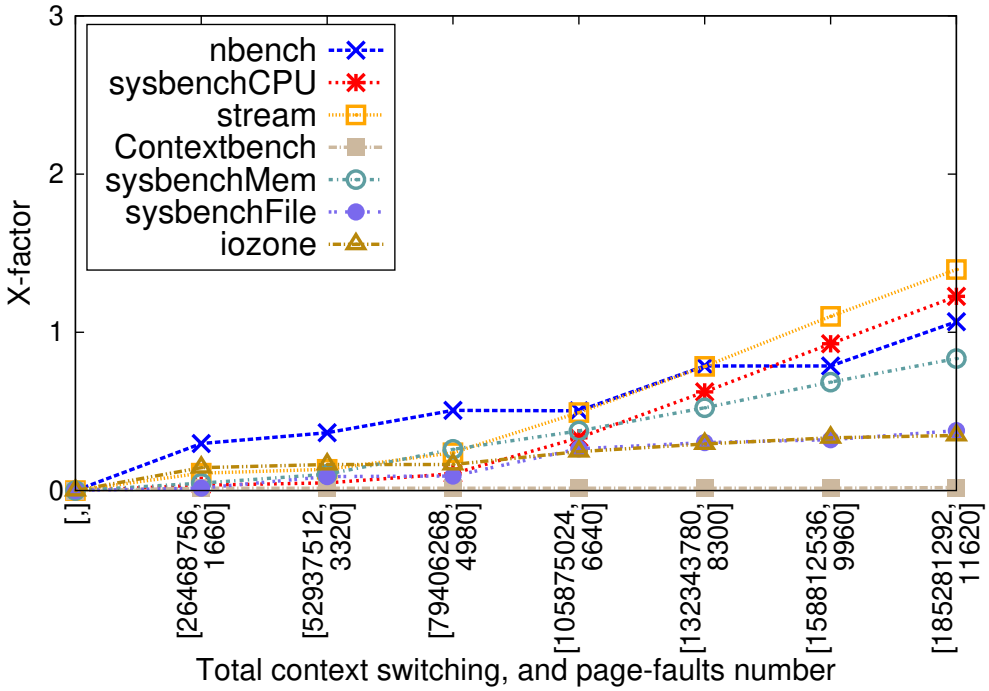
Similarly, Figure 7.13b Figure 7.12b shows the $\times$-$factor$ for context-switching tasks in the co-located VMs. In Figure 7.13c, $\times$-$factor$ due to the Sysbench Mutex test in co-located VMs is calculated and shown. Lastly, Figure 7.13d shows the $\times$-$factor$ due to the Sysbench thread test in co-located VMs. These results are similar to that of ESXi hypervisor shown in Figure 7.12 and discussed above in the section. That is why the discussion about the significance of the results are not repeated.

In this section, the $\times$-$factor$ of seven tasks are presented for two hypervisors, ESXi and Xen. For each hypervisor, seven tasks are run with four types of tests on the co-located VMs. Each of those four tests has different types of system resource requirement and effect the seven tasks differently. The $\times$-$factor$ for ESXi hypervisor is shown on the four graphs of Figure 7.12, while the $\times$-$factor$ for Xen hypervisor is shown on the four graphs of Figure 7.13.

The concept of the $\times$-$factor$ is vital for this chapter, and it is a reoccurring theme in the sections of this chapter. In this chapter, an ANN is used to predict the ETV of

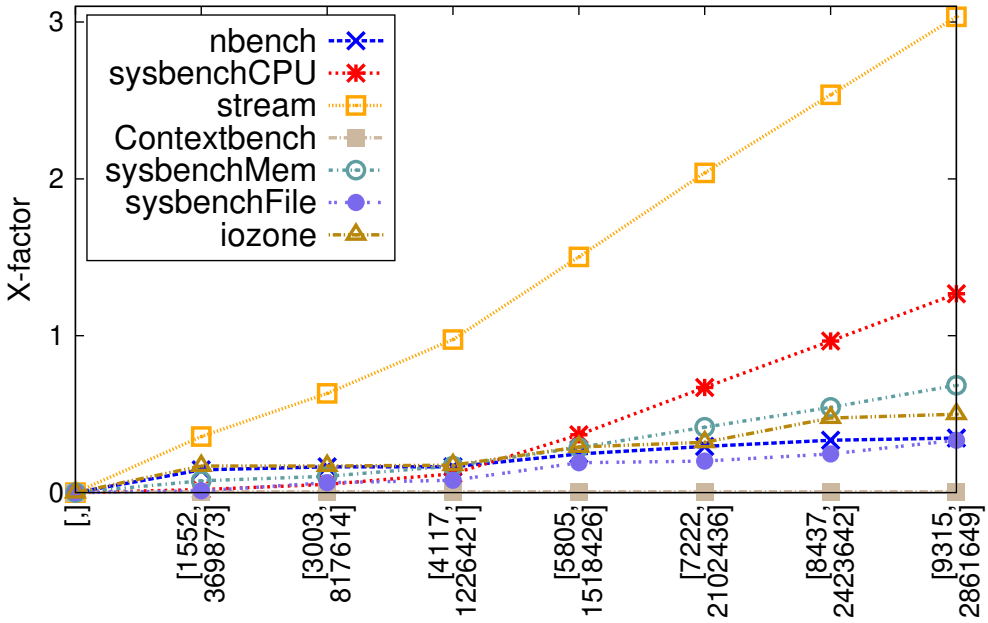


(a) $\times$-$factor$ due to the Cache contention test in co-located VMs.

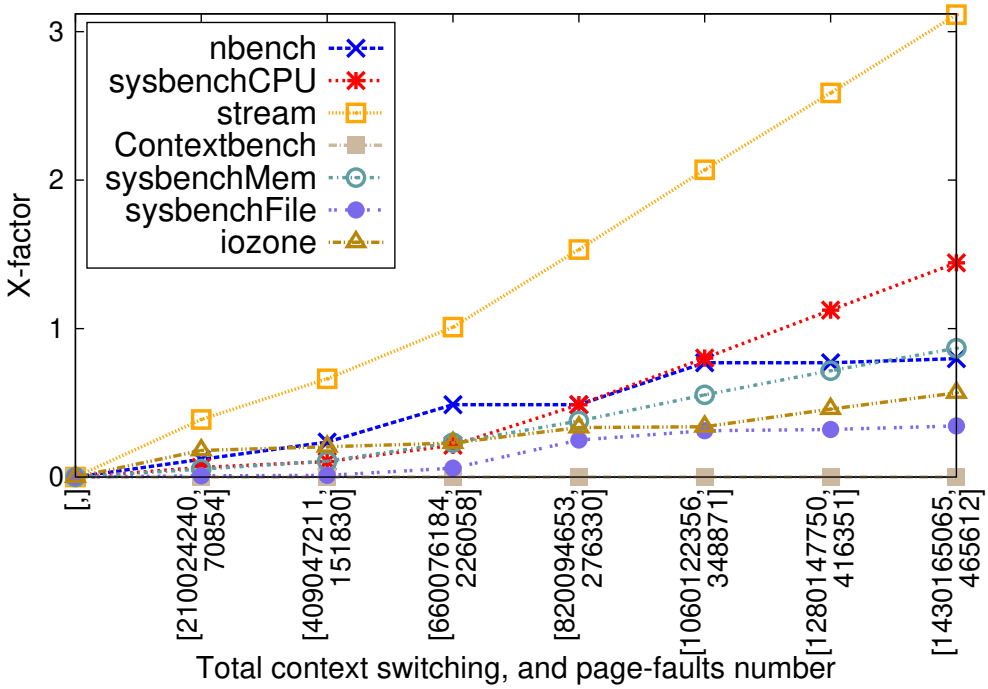


(b) $\times$-$factor$ due to the Stress Context switching test in co-located VMs.

Figure 7.13: Calculating $\times$-$factor$ in the Xen.



(c) $\times$-$factor$ due to the Sysbench Mutex test in co-located VMs.



(d) $\times$-$factor$ due to the Sysbench thread test in co-located VMs.

Figure 7.13: Calculating $\times$-$factor$ in the Xen (Continued).

tasks on consolidated VMs. The $\times$-$factor$ plays an integral part in the training of $\times$-$factor$ models. Next, section shows that these values can be used to train an ANN and predict the performance variation of tasks of a parallel workflow under consolidation.

7.7 Performance prediction for tasks of a parallel workflow

The previous section shows how the execution time variation of tasks can be linked to the interrupts and page-fault numbers of co-located VMs. In this section, it is shown that an ANN can be trained with those values to predict the performance of consolidated tasks.

A parallel workflow consists of many tasks. To schedule those tasks on the VMs the estimation of execution finish time is critical. However, in a consolidated server, the VMs performance may vary depending on the interference of co-located VMs. As discussed in the earlier chapters, this variation depends on the types of tasks running on the co-located VMS and amount of system events they are generating. In this section, the collected system event counter data is used to train an ANN to predict the performance of the tasks of a parallel workflow under consolidation.

ANN is a well-known machine learning technique [336, 339, 343–346, 427]. The neural network is consist of links and nodes. The nodes are arranged in layers. Recall that the construct of an ANN is introduced and discussed in Section 6.11 of Chapter 6. Training instances are fed to the network through input layers, then passes through hidden layers finally ranching to output layers. The layers are connected through links, and each link is associated with a particular weight. The training process involves adjusting the weights with the help of training instances. Recall that, theoretical background for training an ANN is also discussed in Section 6.11.1.

Recall that the GALFA-HI workflow introduced in the Section 5.2 of Chapter 5. The workload has seven different functions. Table 7.5 shows the execution time, interrupts, and page faults of those functions. The workflow diagram is shown again in Figure 7.14 for referential purposes only. It is done so that the tasks of the workflow can be easily related to the data of Table 7.5. That is also required for the clarification of the execution time prediction process for workflows that are discussed next.

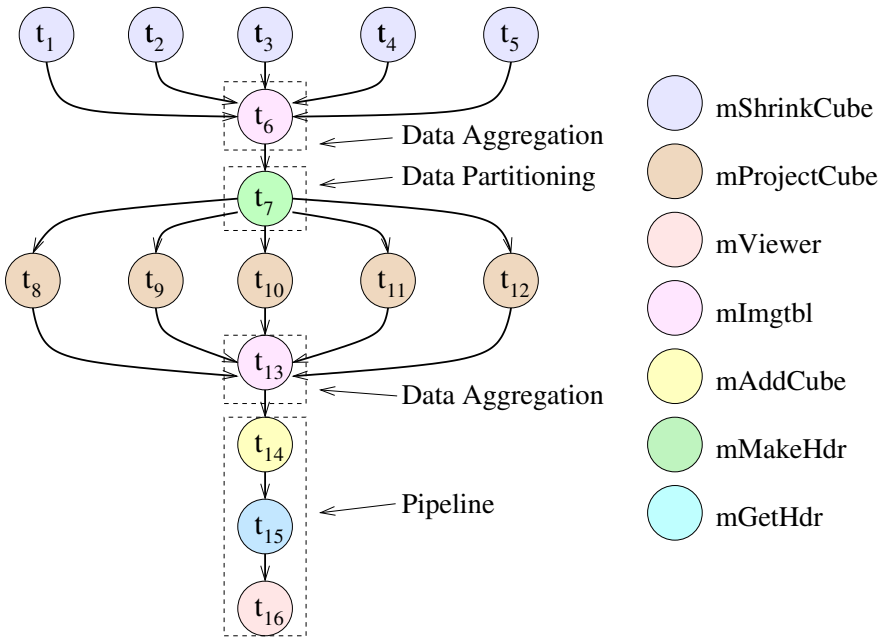


Figure 7.14: A parallel work flow example: the GALFA-HI.

An ANN model is trained with the $\times$-$factor$ data from the previous section (Figures 7.12 and 7.13). First, the total interrupts and page faults of each function are measured in an isolated system. Then those functions are run with a various combination of co-located VMs as described in the previous section. Then, an ANN model is trained

Table 7.5: Execution time and system event data of the functions of Figure 7.14 in Xen.

Level	GALFA-HI Task	Exe. time (min.)	Context-switches	Page-faults
1	mShrinkCube	3.74	5,486	362
2 &5	mImgtbl	0.09	29	410
3	mMakeHdr	0.02	8	1,063
4	mProjectCube	1.12	3,734	292,660
6	mAddCube	9.92	75,556	1,798
7	mGetHdr	0.07	12	271
8	mViewer	0.07	372	15,276

with all the $\times$-$factor$ and system event data.

7.7.1 Prediction result for mAddCube on Xen

In this section, the prediction results discussed here are collected by conducting experiments on the Xen hypervisor. Table 7.6 shows the prediction results for one the functions, the *mAddCube*. First, the mAddCube is run alone without any co-located VM and the execution time is collected (9.94 minutes). Then co-located VMs are added to the system and execution time of the mAddCube starts to vary. The total interrupts and page faults of the co-located VMS are profiled.

The first, and second columns of Table 7.6 shows the total interrupts, and page faults of the co-located VMs, respectively. Then an ANN model is used to predict the $\times$-$factor$ of mAddCube for a various number of co-located VMs. The third column shows the ANN prediction of $\times$-$factor$ for the mAddCube. The expected execution time is estimated from the $\times$-$factors$ using following formula:

Table 7.6: $\times$-$factor$ and execution time prediction for the mAddCube from the VMs system event data in the Xen.

Total context switching	Total page-fault	Predicted X-$factor$ (f_i^p)	Predicted execution time (minute)	Measured execution time (minute)	Prediction error (minute)	Percentage prediction error (%)
- -	- -	- -	- -	9.94	- -	
2317	5460	0.3259557	13.18	11.11	-2.07	-18.63
2979	7020	0.4768612	14.68	12.04	-2.64	-21.92
3641	8580	0.6026157	15.93	12.79	-3.14	-24.55
4303	10140	0.9688129	19.57	15.69	-3.88	-24.72
4965	11700	1.1941650	21.81	17.52	-4.29	-24.48
				RMSE	3.30	
				MAE	3.20	
				MAPE		22.86

$$Exe.Time_i = (1 + f_i^p) * Exe.Time_0$$

Where, the $Exe.Time_i$ is the predicted execution time for the task. f_i^p is the $\times$-$factor$ prediction from the ANN. $Exe.Time_0$ is the initial execution time of the task without any co-located VM. The fourth column of Table 7.6 shows the estimated execution times. The fifth column shows the actual execution time of the mAddCube, while the sixth column shows the prediction error. Lastly, the *Root Mean Squared Error* (RMSE), and *Mean Absolute Error* (MAE) of the prediction is shown in the last two rows, respectively.

7.7.2 Prediction result for mAddCube on ESXi

The previous section discusses the ANN prediction results for the mAddCube function on the ESXi hypervisor. In this section, the experimental procedure of the previous

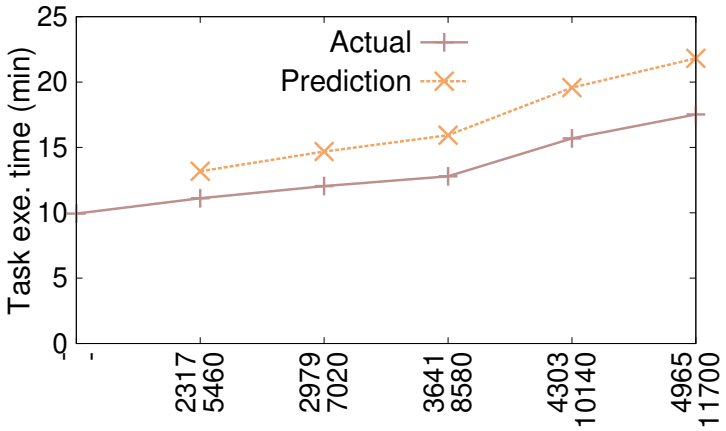
Table 7.7: $\times$-$factor$ and execution time prediction for the mAddCube from the VMs system event data in the ESXi.

Total context switching	Total page-fault	Predicted X-$factor$ (f_i^p)	Predicted execution time (minute)	Measured execution time (minute)	Prediction error (minute)	Percentage prediction error (%)
- -	- -	- -	- -	9.91	- -	
26470080	4780	0.2976791	12.86	10.72	-2.14	-19.96
26470742	6340	0.5459132	15.32	12.53	-2.79	-19.96
26471404	7900	0.8526741	18.36	16.25	-2.11	-12.98
26472066	9460	1.4823411	24.60	19.85	-4.75	-23.92
26472728	11020	1.6579213	26.34	22.23	-4.11	-18.48
				RMSE	3.35	
				MAE	3.18	
				MAPE		19.52

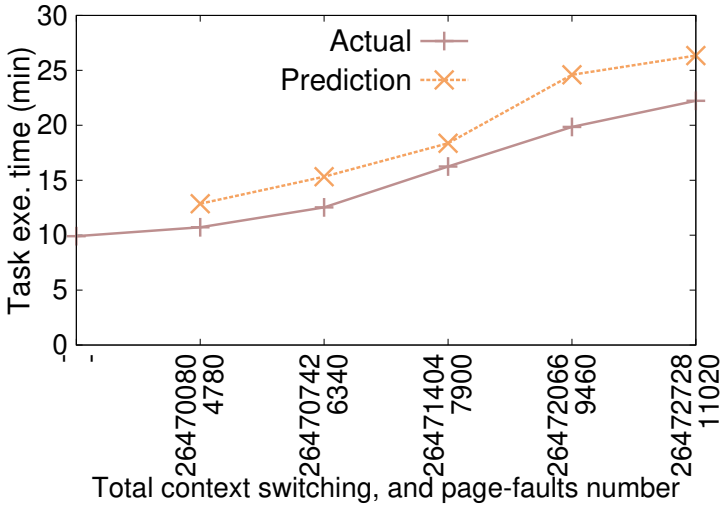
section is repeated for the Xen hypervisor.

Table 7.7 shows the execution time variation prediction of the mAddCube on ESXi. The previously trained ANN model is used to predict the $\times$-$factor$. As before, the mAddCube is first executed alone in the system. Then, it is repeatedly run with various co-located VMs. The total numbers of interrupt and page-fault of the VMs are collected and used for the $\times$-$factor$ predict. From the predicted $\times$-$factor$ values the expected execution times of the mAddCube is estimated. The RMSE and MAE of the prediction are 3.35, and 3.18, respectively.

Figure 7.15 shows actual and predicted execution times of the mAddCube for both



(a) Prediction for Xen.



(b) Prediction for ESXi.

Figure 7.15: The mAddCube execution time variation prediction.

Xen and ESXi hypervisors. The first points of the graphs show the actual mAddCube execution time without any interference from co-located VMs. Subsequent points show the execution time variation of the mAddCube due to the increased number of system interrupts and page-faults. Also, the execution time prediction for the total number of system interrupt and page-faults are shown.

The advantage of using $\times\text{-}factor$ for training an ANN model is the *feature scaling* effect. Actual execution time values have a wider range; it is known that the feature scaling of values can improve the prediction ability of an ANN model [428–430]. With the $\times\text{-}factor$ the training process converges more efficiently compared that of the actual execution time. The training with an ANN is also swift; it takes only 0.102 seconds to train the model. Therefore, any new data from co-located VMs can be easily used to retrain an ANN model.

7.8 Conclusion

This chapter examines the impact of VM resource allocation and system events on the consolidation performance. As data centers are getting bigger, the efficiency of the virtualized servers is becoming more of a pressing issue. Thus, it is essential to find ways to extract the maximum performance from the consolidated VMs.

The experimental results of this chapter show that the system interrupts and page-faults are one of the main causes of performance degradation in virtualized systems. Furthermore, it is not a straightforward process to predict the effect of the system interrupts and page-faults from the system event counters. Instead, this chapter proposes a method to profile the co-located VMs system event counters and use the data to predict the performance variation due to consolidation.

The presented methodology is straightforward; it uses various combination of workload on consolidated VMs to record the interrupts, page faults, and execution finish times. As the number of co-located VMs increase in the system, the task execution times begin to vary. Then, the $\times\text{-}factor$ is calculated, it is the ratio of execution time increase to the initial execution time value.

An ANN can be easily trained to predict this $\times\text{-}factor$ from the co-located VMs system interrupt and page fault numbers. Application of $\times\text{-}factor$ instead of execution time makes the training process more efficient. As it is a low-cost training process,

it can be easily retrained with new co-located VM data at any time.

Presented predicting methodology was applied to the tasks of scientific workflow running on both Xen and ESXi hypervisor. Tables 7.6, and 7.7 show the prediction results for mAddCube function running on Xen and ESXi hypervisors, respectively. From the prediction results, the mean absolute percentage error (MAPE) is also calculated for both hypervisors. The tables show that the MAPE of prediction for Xen and ESXi are 22.86% and 19.52%, respectively.

Chapter 8

Conclusions

"Do not judge me by my successes, judge me by how many times I fell down and got back up again."

— Nelson Rolihlahla Mandela* (1918–2013)

This dissertation explores the performance related issues with the consolidated *Virtual Machines* (VMs). Virtualization is an essential technology for the Cloud and data centers, many necessary features of the Cloud are dependent on it. The pay-as-you-go model for Cloud is implemented with the help of VMs. Furthermore, in data centers, features like the fault-tolerance and high-availability of services are achieved through the use of virtualization. Nowadays, Cloud and VMs are almost synonymous terms.

Chapter 2 of this dissertation, explains how the virtualization is essential for the Cloud. Virtualized allows physical servers to run multiple VM simultaneously; the process is known as the consolidation, the VMs are referred to as the co-located VMs. Consolidation allows to increase resource utilization and reduce operational cost. Running multiple VMs on a server also helps to reduce energy usage. On the other hand, the co-located VMs interfere with the performance of each other, that leads to performance degradation.

As data centers are getting bigger, the resource utilization and energy usage are becoming major issues. In this dissertation, experiments are done with the consolidated

*Image source: `https://www.20minutes.fr/monde/diaporama-3253-nelson-mandela-vie-lutte`

VMs on real systems. The effect of various types of resource contention on the consolidated VMs is examined. The experimental data is collected to train models, which can predict the performance variation of VMs.

All results presented here are real system data. Three hypervisors are used in the experiments; they are the *ESXi*, *Xen*, and *XenServer*. During the experiments, each hypervisor run up to sixteen VMs simultaneously. A benchmarking methodology for VMs and hypervisors called the *Incremental Consolidation Benchmarking Method* (ICBM) [43] is presented in Chapter 4. In subsequent chapters, the effectiveness of the ICBM is demonstrated with various experimental results. In the experiments, several resource-intensive benchmarks and a parallel workflow are used.

Setting up experimenting with a lot of VMs on multiple hypervisors can be time-consuming, that is why a new VM scheduling framework [44] is presented in Chapter 5. This framework can run workload on multiple hypervisors and collect execution time data. The collected data are used in Chapters 6 and 7 to build prediction models [45, 46]. The *Least Square Regression* (LSR) and *Artificial Neural Network* (ANN) models are trained with the profiled data. The results show that the models can predict the task execution time variation due to resource contention of co-located VMs. The next section summarizes the chapter-wise achievements.

8.1 Summary of the dissertation

In Chapter 3, initial experiments are conducted with the consolidated VMs. Various resource intensive benchmarks are consolidated to record their performance variation. Results show that different resource intensive tasks react differently to the resource contention. However, in all cases, the execution time of the tasks show some amount of variation due to consolidation. For a different number of consolidated VMs, the execution time varies differently; this indicates that the task execution time variation can be a good indicator of consolidation performance.

In Chapter 4, further experiments have been done with the task execution time variation. In this chapter, the ICBM is introduced [43]. The experimental results show that the task execution time variation depends on the number of co-located VMs and their resource usage. As the number of co-located VMs are increased in the system the execution time starts to vary. The variation is profiled and used for training prediction

models. The physical host has fourteen VMs, and a lot of different combinations of tasks can be run on them. A combination of tasks and co-located VM is considered as one workload.

The experimental results show that the task execution time variation does not depend on the location of the VMs, rather it depends on the total number of co-located VMs. That is if the order of the same number of VMs is shuffled in the host, their collective effect on the server resources remains the same. During the profiling stage, the execution time is collected for a set of the VM combinations. Then, the data is used to train LSR models. The experimental results show that the built LSR models are capable of predicting the execution time variation of tasks and co-located VM combinations.

The experimental results of applying the ICBM on consolidated VMs encourages one to do more experiments with the VM execution time variation in a wider-scale. In Chapter 5, three hypervisors are used in the experiments, each hypervisor run up to sixteen VMs at a time. Handling a large number of VMs during the tests is difficult; therefore, the necessity for a new VM scheduler is felt. Chapter 5 also introduces a new scheduling framework for VMs; it can run any number of VMs in any combination [44]. The scheduler can run VMs in a wide variety of patterns and collect the VM execution time data. The design goals, components and implementation detail of the framework are discussed in the chapter.

The framework can run different categories of resource-intensive benchmarks on VMs. The results show that different resource intensive VMs, like CPU, memory, and I/O-intensive VMs have a different effect on the consolidation performance. For example, CPU intensive co-located VMs show less performance degradation compared to I/O intensive co-located VMs. In addition to the benchmark suites, a parallel workflow was also used in the experiments. The results show that the framework can run and collect execution time variation data of each task of the parallel workflow.

Next, in chapter 6, large-scale experiments are carried out with the framework and ICBM [46]. This chapter demonstrates that ICBM is capable of handling a large number of VMs. Various experiments have been done with three hypervisors; ESXi, Xen, and XenServer. Three categories of tasks are used in the experiments; they are CPU, memory, and I/O-intensive tasks. The execution time variation data is collected from all three hypervisors. The collected data is then used to build a unified model for the VM consolidation execution time prediction. The results show that the unified

model can predict the execution time variation for both the ESXi and XenServer.

Recall that in Chapter 4, LSR models were introduced for predicting the execution time variation on one hypervisor. The models were built with the execution time variation data from the Xen hypervisor. In Chapter 6, a unified LSR model is created with the data from multiple hypervisors, and it is used to predict the VM execution time variation on multiple hypervisors. Thus, showing that the ICBM can be generalized among multiple virtualized systems.

Furthermore, Chapter 6 indicates that the ICBM data can be used for training an ANN model, too. In this case, the VM execution time variation data is collected from three hypervisors and used to train an ANN model. Then, the ANN model is used to predict the execution time variation of the tasks of a parallel workflow. The experimental results of Chapter 6 show that the ICBM is a versatile method that can be applied to a lot of experimental cases.

Chapter 7 shows that the ICBM can be used for conducting a further investigation with the individual resources. In this chapter, the effect of memory allocation and system events on the VM consolidation performance is examined [45]. The experimental results show that the memory usage of the tasks on VMs plays a significant role in the consolidation performance. By careful VM memory allocation, it is possible to improve the consolidated VMs performance.

Furthermore, experiments are conducted with the system events like interrupt and page faults. For VMs, the system events handling overhead is higher compared to that of physical servers. In this chapter, the effect of system events on the consolidated VMs is examined. Various system event data is collected from the VMs and used to train an ANN model. The trained ANN model is then used to predict the execution time variation of the tasks of a parallel workflow. Using the VM system event counter data the model can predict the performance of co-located VMs.

Throughout this project, the VM consolidation performance is investigated from various angles. The experiments are conducted to understand the effect of resource contention on the co-located VMs. The task execution time variation on consolidated VMs has been used as a metric in this project. Several VM combinations are run on the hypervisor to collect the VM execution time variation data. The data gathered from three hypervisors are analyzed and then used to build the execution time prediction models. The results encourage one to do further experiment with the task execution time variation and VM consolidation performance.

Bibliography

[1] James O'Neill, "Cloud-Based Financial Services: A Banker's Guide," November 2014, https://www.celent.com/insights/205230576 [Online: Accessed August-2018].

[2] Peter M. Mell and Timothy Grance, "The NIST Definition of Cloud Computing," Tech. Rep. 800-145, National Institute of Standards & Technology, Gaithersburg, MD, United States, 2011.

[3] Enterprise Architecture, "Deployment Model," 2016, https://eaexplorer.hana.ondemand.com/_item.html?id=12340#!/overview [Online: Accessed August-2018].

[4] Kent Langley, "Cloud Deployment Models," May 2009, http://productionscale.com/blog/2009/5/27/cloud-deployment-models.html [Online: Accessed August-2018].

[5] Michael Pearce, Sherali Zeadally, and Ray Hunt, "Virtualization: Issues, Security Threats, and Solutions," *ACM Computing Surveys*, vol. 45, no. 2, pp. 17:1–17:39, Mar. 2013.

[6] Yan Junhao and Lv Aili, "Research on the application of virtualization technology in high performance computing," in *2012 IEEE Symposium on Electrical Electronics Engineering (EEESYM)*, June 2012, pp. 386–388.

[7] Principled Technologies, "The benefits of server virtualization for the small and medium business," 2017, www.principledtechnologies.com/Dell/2-2-1_virtualization_benefits_1111.pdf [Online: Accessed 31-July-2017].

[8] Joshua E. Simons and Jeffrey Buell, "Virtualizing High Performance Computing," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 4, pp. 136–145, Dec. 2010.

[9] Suzanne Niles and Patrick Donovan, "Virtualization and Cloud Computing: Optimized Power, Cooling, and Management Maximizes Benefits," 2012, `https://www.anixter.com/content/dam/Suppliers/APC/WhitePaper/VirtualizationandCloudComputing.pdf` [Online: Accessed 31-July-2017].

[10] ISACA, "Virtualization: Benefits and Challenges ," 2010, `http://www.thedatachain.com/materials/virtulization_wp_27oct2010_research.pdf` [Online: Accessed 31-July-2017].

[11] Jim Metzler, "Virtualization: Benefits, Challenges, and Solutions," 2011, `http://www.stotthoare.com.au/sites/default/files/files/1_16100_WhitePaper_VirtualizationBenefits_by_Webtorials.pdf` [Online: Accessed 31-July-2017].

[12] Leja Christine, Richard C. Barnier, Charles L. Brown, Paul F. Dittmann, Paul Koziel, Mark Welle, and J.T. Westermeier, "Virtualization and Its Benefits," 2008, `http://www.stotthoare.com.au/sites/default/files/files/1_16100_WhitePaper_VirtualizationBenefits_by_Webtorials.pdf` [Online: Accessed 31-July-2017].

[13] G. A. Blaauw and F. P. Brooks, "The Structure of SYSTEM/360: Part I - Outline of the Logical Structure," *IBM System Journal*, vol. 3, no. 2, pp. 119–135, June 1964.

[14] W. Y. Stevens, "The structure of SYSTEM/360, Part II: System implementations," *IBM Systems Journal*, vol. 3, no. 2, pp. 136–143, 1964.

[15] G. M. Amdahl, "The Structure of SYSTEM/360: Part III - Processing Unit Design Considerations," *IBM Systems Journal*, vol. 3, no. 2, pp. 144–164, June 1964.

[16] G. A. Blaauw, "The Structure of SYSTEM/360: Part V - Multisystem Organization," *IBM Systems Journal*, vol. 3, no. 2, pp. 181–195, 1964.

[17] A. Padegs, "The structure of SYSTEM/360, Part IV: Channel design considerations," *IBM Systems Journal*, vol. 3, no. 2, pp. 165–179, 1964.

[18] S. G. Tucker, "Microprogram control for SYSTEM/360," *IBM Systems Journal*, vol. 6, no. 4, pp. 222–241, 1967.

[19] Richard P. Case and Andris Padegs, "Architecture of the IBM System/370," *Communications of the ACM - Special issue on computer architecture*, vol. 21, no. 1, pp. 73–96, Jan. 1978.

[20] Edward M. Davis Jr., William E. Harding, Robert S. Schwartz, and John J. Corning, "Solid Logic Technology: Versatile, High-Performance Microelectronics," *IBM Journal of Research and Development*, vol. 8, no. 2, pp. 102–114, 1964.

[21] A. D. Falkoff, K. E. Iverson, and E. H. Sussenguth, "A Formal Description of SYSTEM/360," *IBM Systems Journal*, vol. 3, no. 2, pp. 198–261, June 1964.

[22] P. Fagg, J. L. Brown, J. A. Hipp, D. T. Doody, J. W. Fairclough, and J. Greene, "IBM System/360 Engineering," in *Proceedings of the October 27-29, 1964, Fall Joint Computer Conference, Part I*, New York, NY, USA, 1964, AFIPS '64 (Fall, part I), pp. 205–231, ACM.

[23] Antonio Corradi, Mario Fanelli, and Luca Foschini, "VM Consolidation: A Real Case Based on OpenStack Cloud," *Future Generation Computer Systems*, vol. 32, pp. 118–127, Mar. 2014.

[24] Akshat Verma, Juhi Bagrodia, and Vimmi Jaiswal, "Virtual Machine Consolidation in the Wild," in *Proceedings of the 15th International Middleware Conference*, New York, NY, USA, 2014, Middleware '14, pp. 313–324, ACM.

[25] Xiong Fu and Chen Zhou, "Virtual machine selection and placement for dynamic consolidation in Cloud computing environment," *Frontiers of Computer Science*, vol. 9, no. 2, pp. 322–330, Apr 2015.

[26] Md Hasanul Ferdaus, Manzur Murshed, Rodrigo N. Calheiros, and Rajkumar Buyya, "Virtual Machine Consolidation in Cloud Data Centers Using ACO Metaheuristic," in *Euro-Par 2014 Parallel Processing: 20th International Conference, Porto, Portugal, August 25-29, 2014. Proceedings*, Fernando Silva, Inês

Dutra, and Vítor Santos Costa, Eds., pp. 306–317. Springer International Publishing, Cham, 2014.

[27] Hui Lv, Yaozu Dong, Jiangang Duan, and Kevin Tian, "Virtualization challenges: a view from server consolidation perspective," *SIGPLAN Not.*, vol. 47, no. 7, pp. 15–26, Mar. 2012.

[28] Kevin D. Bowers, Ari Juels, and Alina Oprea, "HAIL: A High-availability and Integrity Layer for Cloud Storage," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, New York, NY, USA, 2009, CCS '09, pp. 187–198, ACM.

[29] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia, "A View of Cloud Computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[30] Mina Nabi, Maria Toeroe, and Ferhat Khendek, "Availability in the Cloud," *J. Netw. Comput. Appl.*, vol. 60, no. C, pp. 54–67, Jan. 2016.

[31] Xiangping Bu, Jia Rao, and Cheng-zhong Xu, "Interference and Locality-aware Task Scheduling for MapReduce Applications in Virtual Clusters," in *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing*, New York, NY, USA, 2013, HPDC '13, pp. 227–238, ACM.

[32] Shin-gyu Kim, Hyeonsang Eom, and Heon Y. Yeom, "Virtual machine consolidation based on interference modeling," *The Journal of Supercomputing*, vol. 66, no. 3, pp. 1489–1506, Dec 2013.

[33] Ludmila Cherkasova and Rob Gardner, "Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor," in *ATEC '05*, Berkeley, CA, USA, 2005, pp. 24–24, USENIX Association.

[34] Diwakar Krishnamurthy, Mehrnoush Alemzadeh, and Mahmood Moussavi, "Towards automated HPC scheduler configuration tuning," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 15, pp. 1723–1748, 2011.

[35] Edward Haletky, *VMware ESX and ESXi in the Enterprise: Planning Deployment of Virtualization Servers*, Prentice Hall Press, Upper Saddle River, NJ, USA, 2nd Edition, 2011.

[36] Carl A. Waldspurger, "Memory Resource Management in VMware ESX Server," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 181–194, Dec. 2002.

[37] Al Muller and Seburn Wilson, *Virtualization with Vmware Esx Server*, Syngress Publishing, first Edition, 2005.

[38] Bryan Clark, Todd Deshane, Eli Dow, Stephen Evanchik, Matthew Finlayson, Jason Herne, and Jeanna Neefe Matthews, "Xen and the Art of Repeated Research," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, Berkeley, CA, USA, 2004, ATEC '04, pp. 47–47, USENIX Association.

[39] David E. Williams, *Virtualization with Xen(Tm): Including XenEnterprise, XenServer, and XenExpress: Including XenEnterprise, XenServer, and XenExpress*, Syngress Publishing, 2007.

[40] Ron C. Chiang and H. Howie Huang, "TRACON: Interference-aware Scheduling for Data-intensive Applications in Virtualized Environments," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2011, SC '11, pp. 47:1–47:12, ACM.

[41] Evangelos Angelou, Konstantinos Kaffes, Athanasia Asiki, Georgios I. Goumas, and Nectarios Koziris, "Improving virtual host efficiency through resource and interference aware scheduling," *CoRR*, vol. abs/1601.07400, 2016.

[42] Alan Roytman, Aman Kansal, Sriram Govindan, Jie Liu, and Suman Nath, "PACMan: Performance Aware Virtual Machine Consolidation," in *10th International Conference on Autonomic Computing, ICAC'13, San Jose, CA, USA, June 26-28, 2013*, 2013, pp. 83–94.

[43] Maruf Ahmed and Albert Y. Zomaya, "Profiling and Predicting Task Execution

Time Variation of Consolidated Virtual Machines," in *Proceeding of The Seventh International Conference on Cloud Computing, GRIDs, and Virtualization, CLOUD COMPUTING 2016*. 2016, pp. 103–112, IARIA.

[44] Maruf Ahmed and Albert Y. Zomaya, " An Automated Lightweight Framework for Scheduling and Profiling Parallel Workflows Simultaneously on Multiple Hypervisors," in *Proceeding of The Eighth International Conference on Cloud Computing, GRIDs, and Virtualization, CLOUD COMPUTING 2017*. 2017, pp. 16–25,, IARIA.

[45] M. Ahmed and A. Y. Zomaya, "The Effect of Resource Allocation and System Events on VM Consolidation," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, Sept. 2017, pp. 557–562.

[46] Maruf Ahmed and Albert Y. Zomaya, "A Framework for Scheduling and Profiling Task Execution Time Variations of Parallel Workflow on Virtual Machines," in *Poster session presented at Seventh ACM Symposium on Cloud Computing, SoCC 2016*, New York, NY, USA, 2016, ACM.

[47] John Venable, Jan Pries-Heje, and Richard Baskerville, " A Comprehensive Framework for Evaluation in Design Science Research ," in *Design Science Research in Information Systems. Advances in Theory and Practice*, Ken Peffers, Marcus Rothenberger, and Bill Kuechler, Eds., Berlin, Heidelberg, 2012, pp. 423–438, Springer Berlin Heidelberg.

[48] Roel Wieringa, "Design Science Methodology: Principles and Practice," in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 2*, New York, NY, USA, 2010, ICSE '10, pp. 493–494, ACM.

[49] Salvatore T. March and Veda C. Storey, "Design Science in the Information Systems Discipline: An Introduction to the Special Issue on Design Science Research," *MIS Q.*, vol. 32, no. 4, pp. 725–730, Dec. 2008.

[50] Richard Baskerville, Jan Pries-Heje, and John Venable, "Soft Design Science Methodology," in *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, New York, NY, USA, 2009, DESRIST '09, pp. 9:1–9:11, ACM.

[51] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram, "Design Science in Information Systems Research," *MIS Q.*, vol. 28, no. 1, pp. 75–105, Mar. 2004.

[52] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble, "Denali: Lightweight Virtual Machines for Distributed and Networked Applications," in *In Proceedings of the USENIX Annual Technical Conference*, 2002.

[53] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield, "Xen and the Art of Virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, Oct. 2003.

[54] VMware, "Virtualization Overview," Tech. Rep., VMware, Inc., 3145 Porter Drive, Palo Alto, CA 94304, USA, 2006, `https://www.vmware.com/pdf/virtualization.pdf` [Online: Accessed August-2018].

[55] The Greaves Group, "Virtualization in Education," Tech. Rep., IBM Systems and Technology Group, 3039 Cornwallis Road, Research Triangle Park, NC 27709, U.S.A, 2007, `http://www-07.ibm.com/solutions/in/education/download/Virtualization%20in%20Education.pdf` [Online: Accessed August-2018].

[56] David Berlind , "Etymology of 'hypervisor' surfaces," August 2005, `https://www.zdnet.com/article/etymology-of-hypervisor-surfaces/` [Online: Accessed August-2018].

[57] Stackexchange.com , "How did the term hypervisor come into use?," April 2013, `https://softwareengineering.stackexchange.com/questions/196405/how-did-the-term-hypervisor-come-into-use` [Online: Accessed August-2018].

[58] IBM, *IBM System/360 Principles of Operation*, IBM Press, 1964.

[59] Daniel Bovet and Marco Cesati, *Understanding The Linux Kernel* , Oreilly & Associates Inc, 2005.

[60] Robert Love, *Linux Kernel Development* , Addison-Wesley Professional, 3rd Edition, 2010.

[61] IBM Archives, "The IBM 700 Series - Computing Comes to Business," 2011, `http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/ibm700series/` [Online: Accessed August-2018].

[62] C. J. Bashe, "The IBM 7080," *IEEE Annals of the History of Computing* , vol. 5, no. 1, pp. 74–, Jan. 1983.

[63] William Aspray, "The Intel 4004 Microprocessor: What Constituted Invention?," *IEEE Ann. Hist. Comput.*, vol. 19, no. 3, pp. 4–15, July 1997.

[64] Barry B. Brey, *The Intel Microprocessors (5th Ed.): 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium II Processors: Architecture, Programming, and Interfacing*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2000.

[65] Ryan Riley, Xuxian Jiang, and Dongyan Xu, " Guest-Transparent Prevention of Kernel Rootkits with VMM-Based Memory Shadowing ," in *Recent Advances in Intrusion Detection*, Richard Lippmann, Engin Kirda, and Ari Trachtenberg, Eds., Berlin, Heidelberg, 2008, pp. 1–20, Springer Berlin Heidelberg.

[66] A. M. Nguyen, N. Schear, H. Jung, A. Godiyal, S. T. King, and H. D. Nguyen, " MAVMM: Lightweight and Purpose Built VMM for Malware Analysis ," in *2009 Annual Computer Security Applications Conference*, Dec 2009, pp. 441–450.

[67] Xuxian Jiang, Xinyuan Wang, and Dongyan Xu, " Stealthy Malware Detection Through Vmm-based "Out-of-the-box" Semantic View Reconstruction ," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, New York, NY, USA, 2007, CCS '07, pp. 128–138, ACM.

[68] Jiuxing Liu, Wei Huang, Bulent Abali, and Dhabaleswar K. Panda, "High Performance VMM-bypass I/O in Virtual Machines," in *ATEC '06*, Berkeley, CA, USA, 2006, pp. 3–3, USENIX Association.

[69] David Chisnall, *The Definitive Guide to the Xen Hypervisor*, Prentice Hall Press, Upper Saddle River, NJ, USA, first Edition, 2007.

[70] Irfan Habib, "Virtualization with KVM," *Linux Journal*, vol. 2008, no. 166, February 2008.

[71] Christoffer Dall and Jason Nieh, "KVM/ARM: The Design and Implementation of the Linux ARM Hypervisor," *SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 333–348, Feb. 2014.

[72] Yu-Ju Huang, Hsuan-Heng Wu, Yeh-Ching Chung, and Wei-Chung Hsu, "Building a KVM-based Hypervisor for a Heterogeneous System Architecture Compliant System," *SIGPLAN Not.*, vol. 51, no. 7, pp. 3–15, Mar. 2016.

[73] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori, "kvm: the Linux virtual machine monitor," in *Proceedings of the Ottawa Linux Symposium*, July 2007, pp. 225–230.

[74] Anthony Velte and Toby Velte, *Microsoft Virtualization with Hyper-V*, McGraw-Hill, Inc., New York, NY, USA, 1 Edition, 2010.

[75] Aidan Finn, Patrick Lownds, Michel Luescher, and Damian Flynn, *Windows Server 2012 Hyper-V Installation and Configuration Guide*, SYBEX Inc., Alameda, CA, USA, 1st Edition, 2013.

[76] Jon Watson, "VirtualBox: Bits and Bytes Masquerading As Machines," *Linux J.*, vol. 2008, no. 166, Feb. 2008.

[77] Sander van Vugt, *VMware Workstation - No Experience Necessary*, Packt Publishing, 2013.

[78] Jeremy Sugerman, Ganesh Venkitachalam, and Beng-Hong Lim, "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor," in *Proceedings of the General Track: 2001 USENIX Annual Technical Conference*, Berkeley, CA, USA, 2001, pp. 1–14, USENIX Association.

[79] Michael D. Schroeder and Jerome H. Saltzer, "A Hardware Architecture for Implementing Protection Rings," *Commun. ACM*, vol. 15, no. 3, pp. 157–170, Mar. 1972.

[80] Tom Shanley, *X86 Instruction Set Architecture*, Mindshare Press, 2010.

[81] Henry N. Palit, Xiaorong Li, Sifei Lu, Lars C. Larsen, and Joseph A. Setia, "Evaluating Hardware-assisted Virtualization for Deploying HPC-as-a-service," in *Proceedings of the 7th International Workshop on Virtualization Technologies*

in Distributed Computing, New York, NY, USA, 2013, VTDC '13, pp. 11–20, ACM.

[82] Rich Uhlig, Gil Neiger, Dion Rodgers, Amy L. Santoni, Fernando C. M. Martins, Andrew V. Anderson, Steven M. Bennett, Alain Kagi, Felix H. Leung, and Larry Smith, " Intel Virtualization Technology ," *Computer*, vol. 38, no. 5, pp. 48–56, May 2005.

[83] Michael Brengel, Michael Backes, and Christian Rossow, "Detecting Hardware-Assisted Virtualization," in *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721*, Berlin, Heidelberg, 2016, DIMVA 2016, pp. 207–227, Springer-Verlag.

[84] Hojoon Lee, Chihyun Song, and Brent ByungHoon Kang, "Lord of the x86 Rings: A Portable User Mode Privilege Separation Architecture on x86," *CoRR*, vol. abs/1805.11912, 2018.

[85] Gustavo Duarte, "CPU Rings, Privilege, and Protection," August 2008, `https://manybutfinite.com/post/cpu-rings-privilege-and-protection/` [Online: Accessed August-2018].

[86] Fabrice Bellard, "QEMU, a Fast and Portable Dynamic Translator," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, Berkeley, CA, USA, 2005, ATEC '05, pp. 41–41, USENIX Association.

[87] E. C. Hendricks and T. C. Hartmann, "Evolution of a Virtual Machine Subsystem," *IBM Syst. J.*, vol. 18, no. 1, pp. 111–142, Mar. 1979.

[88] R. J. Creasy, "The Origin of the VM/370 Time-sharing System," *IBM J. Res. Dev.*, vol. 25, no. 5, pp. 483–490, Sept. 1981.

[89] Richard P. Parmelee, Theodore I. Peterson, Coyt C. Tillman Jr., and Donald J. Hatfield, "Virtual Storage and Virtual Machine Concepts," *IBM Systems Journal*, vol. 11, no. 2, pp. 99–130, 1972.

[90] Michael M. Gold, "Time-sharing and Batch-processing: An Experimental Comparison of Their Values in a Problem-solving Situation," *Commun. ACM*, vol. 12, no. 5, pp. 249–259, May 1969.

[91] Fernando J. Corbató, Marjorie Merwin-Daggett, and Robert C. Daley, "An Experimental Time-sharing System," in *Proceedings of the May 1-3, 1962, Spring Joint Computer Conference*, New York, NY, USA, 1962, AIEE-IRE '62 (Spring), pp. 335–344, ACM.

[92] Norman R. Nielsen, "An Approach to the Simulation of a Time-sharing System," in *Proceedings of the November 14-16, 1967, Fall Joint Computer Conference*, New York, NY, USA, 1967, AFIPS '67 (Fall), pp. 419–428, ACM.

[93] H. A. Reich, "An Experimental System for Time-shared, On-line Data Acquisition," *IBM Journal of Research and Development*, vol. 13, no. 1, pp. 114–118, Jan 1969.

[94] G. W. R. Luderer, J. F. Maranzano, and B. A. Tague, "UNIX time-sharing system: The UNIX operating system as a base for applications," *The Bell System Technical Journal*, vol. 57, no. 6, pp. 2201–2207, July 1978.

[95] F. J. Corbató and V. A. Vyssotsky, "Introduction and Overview of the Multics System," in *Proceedings of the November 30–December 1, 1965, Fall Joint Computer Conference, Part I*, New York, NY, USA, 1965, AFIPS '65 (Fall, part I), pp. 185–196, ACM.

[96] Peter J. Denning, "Virtual Memory," *ACM Comput. Surv.*, vol. 2, no. 3, pp. 153–189, Sept. 1970.

[97] Jack B. Dennis, "Segmentation and the Design of Multiprogrammed Computer Systems," *J. ACM*, vol. 12, no. 4, pp. 589–602, Oct. 1965.

[98] J. L. Keedy, "On Structuring Operating Systems with Monitors," *SIGOPS Oper. Syst. Rev.*, vol. 13, no. 1, pp. 5–9, Jan. 1979.

[99] Martez Reed, *Mastering Citrix XenServer*, Packt Publ., Birmingham, 2014.

[100] Z. Shao, H. Jin, and D. Zhang, " A performance study of web server based on Hardware-assisted Virtual Machine ," in *2009 IEEE/ACS International Conference on Computer Systems and Applications*, May 2009, pp. 837–840.

[101] Keith Adams and Ole Agesen, "A Comparison of Software and Hardware Techniques for x86 Virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 5, pp. 2–13, Oct. 2006.

[102] Stephen Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson, "Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors," *SIGOPS Oper. Syst. Rev.* , vol. 41, no. 3, pp. 275–287, Mar. 2007.

[103] Cristian Ruiz, Emmanuel Jeanvoine, and Lucas Nussbaum, " Performance Evaluation of Containers for HPC ," in *Euro-Par 2015: Parallel Processing Workshops: Euro-Par 2015 International Workshops, Vienna, Austria, August 24-25, 2015, Revised Selected Papers*, Sascha Hunold, Alexandru Costan, Domingo Giménez, Alexandru Iosup, Laura Ricci, María Engracia Gómez Requena, Vittorio Scarano, Ana Lucia Varbanescu, Stephen L. Scott, Stefan Lankes, Josef Weidendorfer, and Michael Alexander, Eds., pp. 813–824. Springer International Publishing, Cham, 2015.

[104] Dirk Merkel, " Docker: Lightweight Linux Containers for Consistent Development and Deployment ," *Linux J.*, vol. 2014, no. 239, Mar. 2014.

[105] Tim Lindholm, Frank Yellin, Gilad Bracha, and Alex Buckley, *The Java Virtual Machine Specification, Java SE 8 Edition*, Addison-Wesley Professional, 1st Edition, 2014.

[106] Tim Lindholm and Frank Yellin, *Java Virtual Machine Specification*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd Edition, 1999.

[107] Mel Cordero, Lucio Correia, Hai Lin, Vamshikrishna Thatikonda, and Rodrigo Xavier, *IBM PowerVM Virtualization Introduction and Configuration 2013*, Vervante, 6th Edition, June 2013.

[108] Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, and John Wilkes, "CPI2: CPU Performance Isolation for Shared Compute Clusters," in *Proceedings of the 8th ACM European Conference on Computer Systems*, New York, NY, USA, 2013, EuroSys '13, pp. 379–391, ACM.

[109] Weiming Zhao and Zhenlin Wang, "Dynamic Memory Balancing for Virtual Machines," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, New York, NY, USA, 2009, VEE '09, pp. 21–30, ACM.

[110] Wenjin Hu, Andrew Hicks, Long Zhang, Eli M. Dow, Vinay Soni, Hao Jiang, Ronny Bull, and Jeanna N. Matthews, "A Quantitative Study of Virtual Machine Live Migration," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, New York, NY, USA, 2013, CAC '13, pp. 11:1–11:10, ACM.

[111] Mendel Rosenblum and Carl Waldspurger, "I/O Virtualization," *Queue*, vol. 9, no. 11, pp. 30:30–30:39, Nov. 2011.

[112] Ravi Khadka, Belfrit V. Batlajery, Amir M. Saeidi, Slinger Jansen, and Jurriaan Hage, " How Do Professionals Perceive Legacy Systems and Software Modernization? ," in *Proceedings of the 36th International Conference on Software Engineering*, New York, NY, USA, 2014, ICSE 2014, pp. 36–47, ACM.

[113] E. W. Lerew, "Migration of legacy test programs to a modern computer platform [for avionics testing]," in *1999 IEEE AUTOTESTCON Proceedings (Cat. No.99CH36323)*, 1999, pp. 293–298.

[114] P. Iyer, A. Nagargadde, S. Gopalan, and V. Sridhar, " Two phase approach for grid enablement of legacy applications ," in *2006 IEEE International Conference on Granular Computing*, May 2006, pp. 194–199.

[115] Tommaso Cucinotta, Fabio Checconi, Luca Abeni, and Luigi Palopoli, " Adaptive Real-time Scheduling for Legacy Multimedia Applications ," *ACM Trans. Embed. Comput. Syst.*, vol. 11, no. 4, pp. 86:1–86:23, Jan. 2013.

[116] Michael Feathers, *Working Effectively with Legacy Code* , Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[117] Changyeon Jo, Erik Gustafsson, Jeongseok Son, and Bernhard Egger, "Efficient Live Migration of Virtual Machines Using Shared Storage," *SIGPLAN Not.*, vol. 48, no. 7, pp. 41–50, Mar. 2013.

[118] Michael R. Hines, Umesh Deshpande, and Kartik Gopalan, "Post-copy Live Migration of Virtual Machines," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 3, pp. 14–26, July 2009.

[119] Gang Sun, Dan Liao, Vishal Anand, Dongcheng Zhao, and Hongfang Yu, "A New Technique for Efficient Live Migration of Multiple Virtual Machines," *Future Gener. Comput. Syst.*, vol. 55, no. C, pp. 74–86, Feb. 2016.

[120] Takahiro Hirofuchi, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi, "Reactive Consolidation of Virtual Machines Enabled by Postcopy Live Migration," in *Proceedings of the 5th International Workshop on Virtualization Technologies in Distributed Computing*, New York, NY, USA, 2011, VTDC '11, pp. 11–18, ACM.

[121] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield, "Live Migration of Virtual Machines," in *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, Berkeley, CA, USA, 2005, NSDI'05, pp. 273–286, USENIX Association.

[122] Haikun Liu, Cheng-Zhong Xu, Hai Jin, Jiayu Gong, and Xiaofei Liao, "Performance and Energy Modeling for Live Migration of Virtual Machines," in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, New York, NY, USA, 2011, HPDC '11, pp. 171–182, ACM.

[123] Q. Tang, S. K S Gupta, and G. Varsamopoulos, "Thermal-aware task scheduling for data centers through minimizing heat recirculation," in *Cluster Computing, 2007 IEEE International Conference on*, 2007, pp. 129–138.

[124] Lizhe Wang, G. von Laszewski, J. Dayal, Xi He, A.J. Younge, and T.R. Furlani, "Towards Thermal Aware Workload Scheduling in a Data Center," in *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on*, 2009, pp. 116–122.

[125] S. Akoush, R. Sohan, A. Rice, A.W. Moore, and A. Hopper, "Predicting the Performance of Virtual Machine Migration," in *MASCOTS*, 2010, pp. 37–46.

[126] Senthil Nathan, Purushottam Kulkarni, and Umesh Bellur, "Resource Availability Based Performance Benchmarking of Virtual Machine Migrations," in *ICPE*, 2013, pp. 387–398.

[127] Christopher Clark Keir, Christopher Clark, Keir Fraser, Steven H, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield, "Live Migration of Virtual Machines," in *In Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI*, 2005, pp. 273–286.

[128] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang, "CloudCmp: Comparing Public Cloud Providers," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, New York, NY, USA, 2010, IMC '10, pp. 1–14, ACM.

[129] Fernando Pires Barbosa and Andrea Schwertner Charão, "Impact of pay-as-you-go Cloud Platforms on Software Pricing and Development: A Review and Case Study," in *Computational Science and Its Applications – ICCSA 2012: 12th International Conference, Salvador de Bahia, Brazil, June 18-21, 2012, Proceedings, Part IV*, Beniamino Murgante, Osvaldo Gervasi, Sanjay Misra, Nadia Nedjah, Ana Maria A. C. Rocha, David Taniar, and Bernady O. Apduhan, Eds., pp. 404–417. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[130] Christof Weinhardt, Arun Anandasivam, Benjamin Blau, Nikolay Borissov, Thomas Meinl, Wibke Michalk, and Jochen Stößer, " Cloud Computing – A Classification, Business Models, and Research Directions ," *Business & Information Systems Engineering*, vol. 1, no. 5, pp. 391–399, Oct 2009.

[131] Ian Foster, "What is the Grid? - a three point checklist," *GRIDtoday*, vol. 1, no. 6, July 2002.

[132] VMware, "VMware vCloud Service Definition for a Public Cloud," Tech. Rep., VMware, Inc, 3401 Hillview Ave, Palo Alto, CA 94304, USA, 2011, `https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/whitepaper/cloud/vmware-servicedef-public-cloud-11q1-white-paper.pdf` [Online: Accessed August-2018].

[133] Intel Information Technology, "Intel Cloud Computing Taxonomy and Ecosystem Analysis," Tech. Rep., Intel Corporation, 2200 Mission College Blvd. Santa Clara, CA 95054-1537, USA, 2010, `https://www.intel.com/content/dam/doc/technology-brief/cloud-computing-intel-cloud-computing-taxonomy-and-ecosystem-analysis-brief.pdf` [Online: Accessed August-2018].

[134] Nelson M. Gonzalez, Charles Miers, Fernando F. Redígolo, Marcos A. Simplício Jr., Tereza Cristina M. B. Carvalho, Mats Näslund, and Makan Pourzandi, "A Taxonomy Model for Cloud Computing Services," in *CLOSER*. 2011, pp. 56–65, SciTePress.

[135] David B. Skillicorn, "The Case for Datacentric Grids," in *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, Washington, DC, USA, 2002, IPDPS '02, pp. 313–, IEEE Computer Society.

[136] George Lawton, "Moving the OS to the Web," *Computer*, vol. 41, no. 3, pp. 16–19, Mar. 2008.

[137] Christof Weinhardt, Jochen Ster, Arun Anandasivam, and Benjamin Blau, "Business Models in the Service World," *IT Professional*, vol. 11, pp. 28–33, 2009.

[138] Aaron Weiss, "Computing in the Clouds," *netWorker*, vol. 11, no. 4, pp. 16–25, Dec. 2007.

[139] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel, "The Cost of a Cloud: Research Problems in Data Center Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, Dec. 2008.

[140] Chunye Gong, Jie Liu, Qiang Zhang, Haitao Chen, and Zhenghu Gong, "The Characteristics of Cloud Computing," in *Proceedings of the 2010 39th International Conference on Parallel Processing Workshops*, Washington, DC, USA, 2010, ICPPW '10, pp. 275–279, IEEE Computer Society.

[141] Neil Caithness, Michel Drescher, and David Wallom, "Can functional characteristics usefully define the cloud computing landscape and is the current reference model correct?," *Journal of Cloud Computing*, vol. 6, no. 1, pp. 10, Jun 2017.

[142] David Hilley, "Cloud Computing: A Taxonomy of Platform and Infrastructure-level Offerings," Tech. Rep. GIT-CERCS-09-13, Georgia Institute of Technology, Georgia Institute of Technology. Center for Experimental Research in Computer Systems, 4 2009.

[143] C. N. Hoefer and G. Karagiannis, "Taxonomy of cloud computing services," in *2010 IEEE Globecom Workshops*, Dec 2010, pp. 1345–1350.

[144] R. Teckelmann, C. Reich, and A. Sulistio, "Mapping of Cloud Standards to the Taxonomy of Interoperability in IaaS," in *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, Nov 2011, pp. 522–526.

[145] C. N. Höfer and G. Karagiannis, " Cloud computing services: taxonomy and comparison ," *Journal of Internet Services and Applications*, vol. 2, no. 2, pp. 81–94, Sep 2011.

[146] Adrian Roberts, "Public Cloud Service Definition," Tech. Rep., VMware, Inc., 3401 Hillview Ave, Palo Alto, CA 94304, USA, January 2018, `https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/vcat/vmware-public-cloud-service-definition.pdf` [Online: Accessed August-2018].

[147] Henry Li, *Introducing Windows Azure*, Apress, Berkely, CA, USA, 2009.

[148] L. Pilosu, P. Ruiu, K. Goga, and M. A. Budroni, "Automated Cloud Computing Approach for the Simulation of Chemo-Hydrodynamic Problems," in *2016 10th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*, July 2016, pp. 438–443.

[149] S. Hwangbo and K. Lee, "Cloud Services for Modeling and Simulation: A Simulation of a Chemical GasDiffusion in the Cloud," in *2016 IEEE/ACM 20th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, Sept 2016, pp. 187–188.

[150] M. J. Harvey and G. De Fabritiis, "AceCloud: Molecular Dynamics Simulations in the Cloud," *Journal of Chemical Information and Modeling*, vol. 55, no. 5, pp. 909–914, 2015, PMID: 25849093.

[151] Tim Mather, Subra Kumaraswamy, and Shahed Latif, *Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance*, O'Reilly Media, Inc., 2009.

[152] Hassan Takabi, James B. D. Joshi, and Gail-Joon Ahn, "Security and Privacy Challenges in Cloud Computing Environments," *IEEE Security and Privacy*, vol. 8, no. 6, pp. 24–31, Nov. 2010.

[153] Z. Tari, "Security and Privacy in Cloud Computing," *IEEE Cloud Computing*, vol. 1, no. 1, pp. 54–57, May 2014.

[154] Wayne Jansen and Timothy Grance, "SP 800-144. Guidelines on Security and Privacy in Public Cloud Computing," Tech. Rep., National Institute of Standards & Technology, Gaithersburg, MD, United States, 2011.

[155] Joel Scheuner, Jürgen Cito, Philipp Leitner, and Harald Gall, "Cloud WorkBench: Benchmarking IaaS Providers Based on Infrastructure-as-Code," in *Proceedings of the 24th International Conference on World Wide Web*, New York, NY, USA, 2015, WWW '15 Companion, pp. 239–242, ACM.

[156] Hamoun Ghanbari, Bradley Simmons, Marin Litoiu, Cornel Barna, and Gabriel Iszlai, " Optimal Autoscaling in a IaaS Cloud ," in *Proceedings of the 9th International Conference on Autonomic Computing*, New York, NY, USA, 2012, ICAC '12, pp. 173–178, ACM.

[157] Luis M. Vaquero, Luis Rodero-Merino, and Daniel Morn, " Locking the sky: a survey on IaaS cloud security. ," *Computing*, vol. 91, no. 1, pp. 93–118, 2011.

[158] Alexandru Iosup, "IaaS Cloud Benchmarking: Approaches, Challenges, and Experience," in *HotTopiCS*, 2013, pp. 1–2.

[159] Amazon Inc, *Amazon Elastic Compute Cloud (Amazon EC2)*, Amazon Inc., http://aws.amazon.com/ec2/#pricing, 2008.

[160] Rostyslav Zabolotnyi, Philipp Leitner, Waldemar Hummer, and Schahram Dustdar, "JCloudScale: Closing the Gap Between IaaS and PaaS," *ACM Transactions on Internet Technology (TOIT)*, 2015.

[161] Rajdeep Dua, A Reddy Raja, and Dharmesh Kakadia, "Virtualization vs Containerization to Support PaaS," in *Proceedings of the 2014 IEEE International Conference on Cloud Engineering*, Washington, DC, USA, 2014, IC2E '14, pp. 610–614, IEEE Computer Society.

[162] Cláudio Teixeira, Joaquim Sousa Pinto, Ricardo Azevedo, Tiago Batista, and André Monteiro, "The Building Blocks of a PaaS," *J. Netw. Syst. Manage.*, vol. 22, no. 1, pp. 75–99, Jan. 2014.

[163] C. Krainer and C.M. Kirsch, "Cyber-Physical Cloud Computing Implemented as PaaS," in *Proc. Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems (CyPhy)*. 2014, ACM.

[164] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart, "Cross-Tenant Side-Channel Attacks in PaaS Clouds," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, 2014, CCS '14, pp. 990–1003, ACM.

[165] Luis R. Merino, Luis M. Vaquero, Eddy Caron, Adrian Muresan, and Frédéric Desprez, "Building Safe PaaS Clouds: A Survey on Security in Multitenant Software Platforms," *Comput. Secur.*, vol. 31, no. 1, pp. 96–108, Feb. 2012.

[166] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch, "Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis," in *SoCC '12*, New York, NY, USA, 2012, pp. 7:1–7:13, ACM.

[167] D. Jacobs, "Implementing Software as a Service," in *12th International IEEE Enterprise Distributed Object Computing Conference*, Sept 2008.

[168] H. Zhu, I. Bayley, M. Younas, D. Lightfoot, B. Yousef, and D. Liu, "Big SaaS: The Next Step beyond Big Data," in *2015 IEEE 8th International Conference on Cloud Computing*, June 2015, pp. 1131–1140.

[169] Bryce Allen, John Bresnahan, Lisa Childers, Ian Foster, Gopi Kandaswamy, Raj Kettimuthu, Jack Kordas, Mike Link, Stuart Martin, Karl Pickett, and Steven Tuecke, "Software As a Service for Data Scientists," *Commun. ACM*, vol. 55, no. 2, pp. 81–88, Feb. 2012.

[170] Cor-Paul Bezemer and Andy Zaidman, "Multi-tenant SaaS Applications: Maintenance Dream or Nightmare?," in *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, New York, NY, USA, 2010, IWPSE-EVOL '10, pp. 88–92, ACM.

[171] Melvin B. Greer, *Software As a Service Inflection Point: Using Cloud Computing to Achieve Business Agility*, iUniverse, Inc., 2009.

[172] Bret Waters, "Software as a service: A look at the customer benefits," *Journal of Digital Asset Management*, vol. 1, no. 1, pp. 32–39, Jan 2005.

[173] Mingdi Xin and Natalia Levina, "Software-as-a Service Model: Elaborating Client-Side Adoption Factors.," in *ICIS*. 2008, p. 86, Association for Information Systems.

[174] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[175] Kien Le, Ricardo Bianchini, Jingru Zhang, Yogesh Jaluria, Jiandong Meng, and Thu D. Nguyen, "Reducing Electricity Cost Through Virtual Machine Placement in High Performance Computing Clouds," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2011, SC '11, pp. 22:1–22:12, ACM.

[176] Qun Huang and Patrick P. C. Lee, "An experimental study of cascading performance interference in a virtualized environment," *SIGMETRICS Performance Evaluation Review*, vol. 40, no. 4, pp. 43–52, 2013.

[177] Yiduo Mei, Ling Liu, Xing Pu, Sankaran Sivathanu, and Xiaoshe Dong, "Performance Analysis of Network I/O Workloads in Virtualized Data Centers," *IEEE Trans. Services Computing*, vol. 6, no. 1, pp. 48–63, 2013.

[178] Xing Pu, Ling Liu, Yiduo Mei, Sankaran Sivathanu, Younggyun Koh, and Calton Pu, "Understanding Performance Interference of I/O Workload in Virtualized Cloud Environments," in *IEEE International Conference on Cloud Computing, CLOUD 2010, Miami, FL, USA, 5-10 July, 2010*, 2010, pp. 51–58.

[179] Xi Chen, Lukas Rupprecht, Rasha Osman, Peter Pietzuch, Felipe Franciosi, and William Knottenbelt, "CloudScope: Diagnosing and Managing Performance Interference in Multi-tenant Clouds," *2015 IEEE 23rd International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, vol. 00, pp. 164–173, 2015.

[180] Warren Smith, Ian T. Foster, and Valerie E. Taylor, "Predicting Application Run Times Using Historical Information," in *IPPS/SPDP '98*, London, UK, UK, 1998, pp. 122–142, Springer-Verlag.

[181] Dongsung Kim, Hwanju Kim, Myeongjae Jeon, Euiseong Seo, and Joonwon Lee, "Guest-Aware Priority-Based Virtual Machine Scheduling for Highly Consolidated Server," in *Proceedings of the 14th international Euro-Par conference on Parallel Processing*, Berlin, Heidelberg, 2008, Euro-Par '08, pp. 285–294, Springer-Verlag.

[182] Ron Chi-Lung Chiang, Jinho Hwang, H. Howie Huang, and Timothy Wood, "Matrix: Achieving Predictable Virtual Machine Performance in the Clouds," in *11th International Conference on Autonomic Computing, ICAC '14, Philadelphia, PA, USA, June 18-20, 2014.*, 2014, pp. 45–56.

[183] Mohiuddin Solaimani, Latifur Khan, and Bhavani M. Thuraisingham, "Real-time anomaly detection over VMware performance data using storm," in *Proceedings of the 15th IEEE International Conference on Information Reuse and Integration, IRI 2014, Redwood City, CA, USA, August 13-15, 2014*, 2014, pp. 458–465.

[184] Christina Delimitrou and Christos Kozyrakis, "iBench: Quantifying interference for datacenter applications," in *Proceedings of the IEEE International Symposium on Workload Characterization, IISWC 2013, Portland, OR, USA, September 22-24, 2013*, 2013, pp. 23–33.

[185] Yiduo Mei, Ling Liu, Xing Pu, and Sankaran Sivathanu, "Performance Measurements and Analysis of Network I/O Applications in Virtualized Cloud," in *IEEE International Conference on Cloud Computing, CLOUD 2010, Miami, FL, USA, 5-10 July, 2010*, 2010, pp. 59–66.

[186] Girish Keshav Palshikar, Amrit Lal Ahuja, and Harrick M. Vin, "Utilization Analysis of Servers in a Data Centre," in *Data Engineering and Management: Second International Conference, ICDEM 2010, Tiruchirappalli, India, July 29-31, 2010. Revised Selected Papers*, Rajkumar Kannan and Frederic Andres, Eds., pp. 173–180. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[187] Stephan Kraft, Giuliano Casale, Diwakar Krishnamurthy, Des Greer, and Peter Kilpatrick, "Performance models of storage contention in cloud environments," *Software and System Modeling*, vol. 12, no. 4, pp. 681–704, 2013.

[188] Stephan Kraft, Giuliano Casale, Diwakar Krishnamurthy, Des Greer, and Peter Kilpatrick, "IO performance prediction in consolidated virtualized environments (abstracts only)," *SIGMETRICS Performance Evaluation Review*, vol. 39, no. 3, pp. 17–18, 2011.

[189] Xing Pu, Ling Liu, Yiduo Mei, Sankaran Sivathanu, Younggyun Koh, Calton Pu, and Yuanda Cao, "Who Is Your Neighbor: Net I/O Performance Interference in Virtualized Clouds," *IEEE Trans. Services Computing*, vol. 6, no. 3, pp. 314–329, 2013.

[190] Wei Kuang, Laura E. Brown, and Zhenlin Wang, "Modeling Cross-Architecture Co-Tenancy Performance Interference," in *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2015, Shenzhen, China, May 4-7, 2015*, 2015, pp. 231–240.

[191] Indrani Paul, Sudhakar Yalamanchili, and Lizy K. John, "Performance impact of virtual machine placement in a datacenter," in *31st IEEE International Performance Computing and Communications Conference, IPCCC 2012, Austin, TX, USA, December 1-3, 2012*, 2012, pp. 424–431.

[192] Qais Noorshams, Kiana Rostami, Samuel Kounev, and Ralf H. Reussner, "Modeling of I/O Performance Interference in Virtualized Environments with Queueing Petri Nets," in *IEEE 22nd International Symposium on Modelling, Analysis*

& Simulation of Computer and Telecommunication Systems, MASCOTS 2014, Paris, France, September 9-11, 2014, 2014, pp. 331–336.

[193] Younggyun Koh, Rob C. Knauerhase, Paul Brett, Mic Bowman, Zhihua Wen, and Calton Pu, "An Analysis of Performance Interference Effects in Virtual Environments," in *2007 IEEE International Symposium on Performance Analysis of Systems and Software, April 25-27, 2007, San Jose, California, USA, Proceedings*, 2007, pp. 200–209.

[194] Sankaran Sivathanu, Ling Liu, and Cristian Ungureanu, "Modeling the performance and energy of storage arrays," in *International Green Computing Conference 2010, Chicago, IL, USA, 15-18 August 2010*, 2010, pp. 229–242.

[195] Jóakim v. Kistowski, Hansfried Block, John Beckett, Klaus-Dieter Lange, Jeremy A. Arnold, and Samuel Kounev, "Analysis of the Influences on Server Power Consumption and Energy Efficiency for CPU-Intensive Workloads," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, New York, NY, USA, 2015, ICPE '15, pp. 223–234, ACM.

[196] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Toward energy-efficient cloud computing: Prediction, consolidation, and overcommitment," *Network, IEEE*, vol. 29, no. 2, pp. 56–61, March 2015.

[197] Jae-Wan Jang, Myeongjae Jeon, Hyo-Sil Kim, Heeseung Jo, Jin-Soo Kim, and Seungryoul Maeng, "Energy Reduction in Consolidated Servers through Memory-Aware Virtual Machine Scheduling," *Computers, IEEE Transactions on*, vol. 60, no. 4, pp. 552–564, 2011.

[198] Etienne Le Sueur and Gernot Heiser, "Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns," in *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, Berkeley, CA, USA, 2010, HotPower'10, pp. 1–8, USENIX Association.

[199] Faruk Caglar, Shashank Shekhar, and Aniruddha S. Gokhale, "Towards a performance interference-aware virtual machine placement strategy for supporting soft real-time applications in the cloud," in *REACTION 2014, 3rd IEEE International Workshop on Real-time and distributed computing in emerging applications, Proceedings, Rome, Italy. December 2nd, 2014*, 2014.

[200] Brian J. Watson, Manish Marwah, Daniel Gmach, Yuan Chen, Martin Arlitt, and Zhikui Wang, "Probabilistic performance modeling of virtualized resource allocation," in *Proceedings of the 7th international conference on Autonomic computing*, New York, NY, USA, 2010, ICAC '10, pp. 99–108, ACM.

[201] Sebastien Godard, "Sysstat : utilities," 2017, `http://sebastien.godard.pagesperso-orange.fr/` [Online: Accessed 17-September-2017].

[202] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mallor, Ken Shwaber, and Jeff Sutherland, "The Agile Manifesto," Tech. Rep., The Agile Alliance, 2001.

[203] Marshall Kirk McKusick, Keith Bostic, Michael J. Karels, and John S. Quarterman, *The Design and Implementation of the 4.4BSD Operating System*, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1996.

[204] Keigo Matsubara, Edison Kwok, Inge Rodriguez, and Murali Paramasivam, *Developing and Porting C and C++ Applications on Aix (Ibm Redbooks.)*, IBM Press, 2003.

[205] Lizhe Wang, G. von Laszewski, J. Dayal, and T.R. Furlani, "Thermal aware workload scheduling with backfilling for green data centers," in *Performance Computing and Communications Conference (IPCCC), 2009 IEEE 28th International*, 2009, pp. 289–296.

[206] Francois Baccelli, Alain Jean-marie, Zhen Liu, and Sophia antipolis Cedex, "A Survey on Solution Methods for Task Graph Models," 1993.

[207] Yu-Kwong Kwok and Ishfaq Ahmad, "Benchmarking and Comparison of the Task Graph Scheduling Algorithms," *J. Parallel Distrib. Comput.*, vol. 59, no. 3, pp. 381–422, 1999.

[208] Amol Bakshi, Viktor K. Prasanna, Jim Reich, and Daniel Larner, "The Abstract Task Graph: A Methodology for Architecture-independent Programming

of Networked Sensor Systems," in *Proceedings of the 2005 Workshop on End-to-end, Sense-and-respond Systems, Applications and Services*, Berkeley, CA, USA, 2005, EESR '05, pp. 19–24, USENIX Association.

[209] Rajiv Gupta and Madalene Spezialetti, "A Compact Task Graph Representation for Real-Time Scheduling," *Real-Time Systems*, vol. 11, no. 1, pp. 71–102, 1996.

[210] Salah E. Elmaghraby, *Activity Networks: Project Planning and Control by Network Models*, Wiley, New York, 1977.

[211] Yu-Kwong Kwok and Ishfaq Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," *ACM Comput. Surv.*, vol. 31, no. 4, pp. 406–471, Dec. 1999.

[212] Pierre-Francois Dutot, Tchimou N'Takpe, Frederic Suter, and Henri Casanova, "Scheduling Parallel Task Graphs on (Almost) Homogeneous Multicluster Platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 7, pp. 940–952, 2009.

[213] Robert P. Dick, David L. Rhodes, and Wayne Wolf, "TGFF: Task Graphs for Free," 1998.

[214] Kunal Agrawal, Charles E. Leiserson, and Jim Sukha, "Executing task graphs using work-stealing," in *IPDPS*, 2010, pp. 1–12.

[215] Jeffrey P. Casazza, Michael Greenfield, and Kan Shi, "Redefining Server Performance Characterization for Virtualization Benchmarking," *Intel Technology Journal*, vol. 10, no. 03, pp. 243–252, 2006.

[216] Vikram Makhija, Bruce Herndon, Paula Smith, Eric Zamost, and Jennifer Anderson, "VMmark: A Scalable Benchmark for Virtualized Systems," Tech. Rep. TR-2006-002, VMware, 2006.

[217] Hewlett-Packard Enterpris, "HPE ProLiant DL380 Gen10 - VMmark V2.5.2 Results," 2017, `https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/vmmark/2017-09-26-HPE-ProLiant-DL380Gen10-6143.pdf` [Online: Accessed August-2018].

[218] VMware, "VMmark 2.x Results," 2018, `https://www.vmware.com/products/vmmark/results2x.html` [Online: Accessed August-2018].

[219] Cornelia Cecilia Eglantine, *NBench*, TypPRESS, 2012, ISBN: 9786136257211.

[220] Ben Smith, Rick Grehan, and Tom Yager, "Unixbench: BYTE UNIX benchmark suite," 2017, `http://github.com/kdlucas/byte-unixbench` [Online: Accessed 17-September-2017].

[221] Zhonghong Ou, Hao Zhuang, Jukka K. Nurminen, Antti Ylä-Jääski, and Pan Hui, "Exploiting Hardware Heterogeneity Within the Same Instance Type of Amazon EC2," in *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*, Berkeley, CA, USA, 2012, HotCloud'12, pp. 4–4, USENIX Association.

[222] Reinhold P. Weicker, "Dhrystone: A Synthetic Systems Programming Benchmark," *Commun. ACM*, vol. 27, no. 10, pp. 1013–1030, Oct. 1984.

[223] H. J. Curnow and Brian A. Wichmann, "A Synthetic Benchmark," *Comput. J.*, vol. 19, no. 1, pp. 43–49, 1976.

[224] Phillip J. Mucci, Kevin London, and Philip J. Mucci, "The CacheBench Report," Tech. Rep., The Department of Electrical Engineering and Computer Science, The University of Tennessee, Knoxville, November 1998.

[225] John D. McCalpin, "Memory Bandwidth and Machine Balance in Current High Performance Computers," *IEEE CS TCCA Newsletter*, pp. 19–25, Dec. 1995.

[226] John D. McCalpin, "STREAM: Sustainable Memory Bandwidth in High Performance Computers," Tech. Rep., University of Virginia, Charlottesville, Virginia, 2007.

[227] Lars Bergstrom, "Measuring NUMA effects with the STREAM benchmark," *CoRR*, vol. abs/1103.3225, 2011.

[228] Andrew Tridgell, "The dbench benchmark," `https://www.samba.org/ftp/tridge/dbench/README`, 2017, [Online: Accessed 17-September-2017].

[229] OpenSolaris Project, "Filebench," `https://github.com/filebench/filebench/wiki`, 2017, [Online: Accessed 17-September-2017].

[230] W. Norcott and D. Capps, "IOzone Filesystem Benchmark," `www.iozone.org`, 2017, [Online: Accessed 17-September-2017].

[231] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[232] Carl Staelin and Hewlett packard Laboratories, "LMbench: Portable Tools for Performance Analysis," in *In USENIX Annual Technical Conference*, 1996, pp. 279–294.

[233] Alexey Kopytov, "SysBench manual," 2004, `http://imysql.com/wp-content/uploads/2014/10/sysbench-manual.pdf` [Online: Accessed August-2018].

[234] G. Gordon Schulmeyer and James I. McManus, Eds., *Handbook of Software Quality Assurance (3rd Ed.)*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

[235] Thomas Williams, Colin Kelley, and many others, " Gnuplot 4.6: an interactive plotting program ," April 2013, `http://gnuplot.sourceforge.net/` [Online: Accessed August-2018].

[236] Philipp K. Janert, *Gnuplot in Action*, Manning Publications Co., Greenwich, CT, USA, 2nd Edition, 2015.

[237] David W. Juedes, "UNIX Writing Tools Primer for CS Graduate Students: Ispell, LaTeX, BibTex, Xfig, GNUPLOT, XV, etc.," January 2001, `https://pdfs.semanticscholar.org/7d8e/fa1afe063807b097cfb82d06ec736923b9c9.pdf` [Online: Accessed August-2018].

[238] Joydeep Mukherjee, Diwakar Krishnamurthy, and Jerry Rolia, "Resource Contention Detection in Virtualized Environments," *IEEE Trans. Network and Service Management*, vol. 12, no. 2, pp. 217–231, 2015.

[239] Yasaman Amannejad, Diwakar Krishnamurthy, and Behrouz Homayoun Far, "Managing Performance Interference in Cloud-Based Web Services," *IEEE Trans. Network and Service Management*, vol. 12, no. 3, pp. 320–333, 2015.

[240] Joydeep Mukherjee, Mea Wang, and Diwakar Krishnamurthy, "Performance Testing Web Applications on the Cloud," in *Seventh IEEE International Conference on Software Testing, Verification and Validation, ICST 2014 Workshops Proceedings, March 31 - April 4, 2014, Cleveland, Ohio, USA*, 2014, pp. 363–369.

[241] Joydeep Mukherjee, Diwakar Krishnamurthy, Jerry Rolia, and Chris Hyser, "Resource contention detection and management for consolidated workloads," in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), Ghent, Belgium, May 27-31, 2013*, 2013, pp. 294–302.

[242] T. Janpan, V. Visoottiviseth, and R. Takano, "A virtual machine consolidation framework for CloudStack platforms," in *ICOIN*, 2014, pp. 28–33.

[243] F. Farahnakian, T. Pahikkala, P. Liljeberg, J. Plosila, and H. Tenhunen, "Utilization Prediction Aware VM Consolidation Approach for Green Cloud Computing," in *CLOUD*, 2015, pp. 381–388.

[244] Alain Tchana, Noel De Palma, Ibrahim Safieddine, Daniel Hagimont, Bruno Diot, and Nicolas Vuillerme, "Software Consolidation as an Efficient Energy and Cost Saving Solution for a SaaS/PaaS Cloud Model," in *Euro-Par*, 2015, pp. 305–316.

[245] Menno Dobber, Robert D. van der Mei, and Ger Koole, "Effective Prediction of Job Processing Times in a Large-Scale Grid Environment," in *HPDC*, 2006, pp. 359–360.

[246] Valerie E. Taylor, Xingfu Wu, Jonathan Geisler, and Rick L. Stevens, "Using Kernel Couplings to Predict Parallel Application Performance.," in *HPDC*, 2002, pp. 125–134.

[247] Wei Gao, Yong Li, Haoyang Lu, Ting Wang, and Cong Liu, "On Exploiting Dynamic Execution Patterns for Workload Offloading in Mobile Cloud Applications," in *ICNP*, 2014, pp. 1–12.

[248] Abhishek Gupta, Laxmikant V. Kalé, Filippo Gioachin, Verdi March, Chun Hui Suen, Bu-Sung Lee, Paolo Faraboschi, Richard Kaufmann, and Dejan S. Milojicic, "The Who, What, Why, and How of High Performance Computing in the Cloud," in *CloudCom, Volume 1*, 2013, pp. 306–314.

[249] José Simão and Luís Veiga, "Flexible SLAs in the Cloud with a Partial Utility-Driven Scheduling Architecture," in *CloudCom, Volume 1*, 2013, pp. 274–281.

[250] Sisu Xi, Meng Xu, Chenyang Lu, Linh T. X. Phan, Christopher D. Gill, Oleg Sokolsky, and Insup Lee, "Real-time multi-core virtual machine scheduling in Xen," in *EMSOFT*, 2014, pp. 27:1–27:10.

[251] Jiaqi Zhao, Jie Tao, Lizhe Wang, and Andreas Wirooks, "A Toolchain For Profiling Virtual Machines," in *ECMS*, 2013, pp. 497–503.

[252] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears, "Benchmarking Cloud Serving Systems with YCSB," in *SoCC '10*, New York, NY, USA, 2010, pp. 143–154, ACM.

[253] Anh Vu Do, Junliang Chen, Chen Wang, Young Choon Lee, Albert Y. Zomaya, and Bing Bing Zhou, "Profiling Applications for Virtual Machine Placement in Clouds," in *CLOUD*, 2011, pp. 660–667.

[254] Zoltán Ádám Mann, "Allocation of Virtual Machines in Cloud Data Centers&Mdash;A Survey of Problem Models and Optimization Algorithms," *ACM Comput. Surv.*, vol. 48, no. 1, pp. 11:1–11:34, Aug. 2015.

[255] Q. Noorshams, K. Rostami, S. Kounev, P. Tuma, and R. Reussner, "I/O Performance Modeling of Virtualized Storage Systems," in *MASCOTS*, 2013, pp. 121–130.

[256] Yasaman Amannejad, Diwakar Krishnamurthy, and Behrouz Homayoun Far, "Detecting performance interference in cloud-based web services," in *IFIP/IEEE International Symposium on Integrated Network Management, IM 2015, Ottawa, ON, Canada, 11-15 May, 2015*, 2015, pp. 423–431.

[257] Richard McDougall and Jennifer Anderson, "Virtualization Performance: Perspectives and Challenges Ahead," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 4, pp. 40–56, Dec. 2010.

[258] T. Deshane, Z. Shepherd, J. Matthews, M. Ben-Yehuda, A. Shah, and B. Rao, "Quantitative comparison of Xen and KVM," in *Xen summit*, Berkeley, CA, USA, June 2008, USENIX association.

[259] Padma Apparao, Ravi Iyer, and Don Newell, "Towards Modeling & Analysis of Consolidated CMP Servers," *SIGARCH Comput. Archit. News*, vol. 36, no. 2, pp. 38–45, May 2008.

[260] Omesh Tickoo, Ravi Iyer, Ramesh Illikkal, and Don Newell, "Modeling Virtual Machine Performance: Challenges and Approaches," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 3, pp. 55–60, Jan. 2010.

[261] Hai Jin, Wenzhi Cao, Pingpeng Yuan, and Xia Xie, "VSCBenchmark: Benchmark for Dynamic Server Performance of Virtualization Technology," in *IFMT '08*, 2008, pp. 5:1–5:8.

[262] Padma Apparao, Ravi Iyer, Xiaomin Zhang, Don Newell, and Tom Adelmeyer, "Characterization & Analysis of a Server Consolidation Benchmark," in *VEE '08*, New York, NY, USA, 2008, pp. 21–30, ACM.

[263] Sajib Kundu, Raju Rangaswami, Ajay Gulati, Ming Zhao, and Kaushik Dutta, "Modeling Virtualized Applications Using Machine Learning Techniques," *SIGPLAN Not.*, vol. 47, no. 7, pp. 3–14, Mar. 2012.

[264] Diwakar Krishnamurthy, Jerry Rolia, and Min Xu, "WAM - The Weighted Average Method for Predicting the Performance of Systems with Bursts of Customer Sessions," *IEEE Trans. Software Eng.*, vol. 37, no. 5, pp. 718–735, 2011.

[265] Giuliano Casale, Stephan Kraft, and Diwakar Krishnamurthy, "A Model of Storage I/O Performance Interference in Virtualized Systems," in *31st IEEE International Conference on Distributed Computing Systems Workshops (ICDCS 2011 Workshops), 20-24 June 2011, Minneapolis, Minnesota, USA*, 2011, pp. 34–39.

[266] Andy Georges and Lieven Eeckhout, "Performance metrics for consolidated servers," in *System-level Virtualization for High Performance Computing, 4th Workshop, Proceedings*. 2010, p. 8, Association for Computing Machinery (ACM).

[267] Xiang Sun, Nirwan Ansari, and Ruopeng Wang, "Optimizing Resource Utilization of a Data Center," *Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2822–2846, Oct. 2016.

[268] Jonathan G Koomey, "Worldwide electricity used in data centers," *Environmental Research Letters*, vol. 3, no. 3, pp. 034008, 2008.

[269] Mohammed Rashid Chowdhury, Mohammad Raihan Mahmud, and Rashedur M. Rahman, " Implementation and Performance Analysis of Various VM Placement Strategies in CloudSim ," *J. Cloud Comput.*, vol. 4, no. 1, pp. 45:1–45:21, Dec. 2015.

[270] Steve G. Langer and Todd French, "Virtual Machine Performance Benchmarking," *Journal of Digital Imaging*, vol. 24, no. 5, pp. 883–889, Oct 2011.

[271] Timothy Wood, Ludmila Cherkasova, Kivanc Ozonat, and Prashant Shenoy, "Profiling and modeling resource usage of virtualized applications," in *Middleware '08*, New York, NY, USA, 2008, pp. 366–387, Springer-Verlag New York, Inc.

[272] Citrix Systems, *XenServer 7.5 Configuration Limits*, Citrix Systems, Inc., 851 West Cypress Creek Road, Fort Lauderdale, FL 33309, May 2018, `https://docs.citrix.com/content/dam/docs/en-us/xenserver/current-release/downloads/xenserver-config-limits.pdf` [Online: Accessed August-2018].

[273] Martin J. Bland and Douglas G. Altman, "Statistics Notes: Measurement error," *BMJ*, vol. 313, no. 7059, pp. 744+, Sept. 1996.

[274] Tianfeng Chai and R R. Draxler, "Root mean square error (RMSE) or mean absolute error (MAE)? Arguments against avoiding RMSE in the literature," *Geoscientific Model Development Discussions*, vol. 7, pp. 1247–1250, June 2014.

[275] R.E.Deakin and D.G.Kildea, "A Note on Standard Deviation and RMS," *THE AUSTRALIAN SURVEYOR* , vol. 44, pp. 74–79, June 1999.

[276] C. J. Willmott and K. Matsuura, "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance," *Climate Research*, vol. 30, pp. 79–82, 2005.

[277] Maxim Shcherbakov, Adriaan Brebels, N.L. Shcherbakova, A.P. Tyukov, T.A. Janovsky, and V.A. Kamaev, "A survey of forecast error measures," *World Applied Sciences Journal*, vol. 24, pp. 171–176, 01 2013.

[278] Carlos Ruberto Fragoso Jnior, *Model evaluation methods*, Federal University of Alagoas (UFAL), Macei, Alagoas, Brazil, 8 2010, `http://www.ctec.ufal.br/professor/crfj/Graduacao/MSH/Model%20evaluation%20methods.doc` [Online: Accessed August-2018].

[279] Anthony G. Barnston, "Correspondence among the Correlation, RMSE, and Heidke Forecast Verification Measures; Refinement of the Heidke Score," *Weather and Forecasting*, vol. 7, no. 4, pp. 699–709, 12 1992.

[280] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2013, ISBN 3-900051-07-0.

[281] Michael J. Crawley, *The R Book*, Wiley Publishing, 2nd Edition, 2012.

[282] Frank E Harrell Jr, "Regression Modeling Strategies (rms) v5.1-2 - rms Version of glm," `https://www.rdocumentation.org/packages/rms/versions/5.1-2/topics/Glm`, 2018, [Online: Accessed August-2018].

[283] R core R-core@R project.org, "The R Stats Package v3.5.0 - Fitting Linear Models," `https://www.rdocumentation.org/packages/stats/versions/3.5.0/topics/lm`, 2018, [Online: Accessed August-2018].

[284] R Core Team and contributors worldwide, "The R Stats Package version 3.6.0 - Fitting Generalized Linear Models," `https://stat.ethz.ch/R-manual/R-devel/library/stats/html/glm.html`, 2018, [Online: Accessed August-2018].

[285] Achim Zeileis and Yves Croissant, *Extended Model Formula*, 5 2018, `https://cran.r-project.org/web/packages/Formula/Formula.pdf` [Online: Accessed August-2018].

[286] P. Richard Hahn, *Statistical Formula Notation in R*, University of Chicago, Booth School of Business, Chicago, IL 60637, USA,

2012, http://faculty.chicagobooth.edu/richard.hahn/teaching/formulanotation.pdf [Online: Accessed August-2018].

[287] Felipe Rego, "Quick Guide: Interpreting Simple Linear Model Output in R," https://feliperego.github.io/blog/2015/10/23/Interpreting-Model-Output-In-R, October 2005, [Online: Accessed August-2018].

[288] NaLette Brodnax, Simo Goshev, Steven Worthington, and Ista Zahn, "Linear regression - R Tutorials," https://tutorials.iq.harvard.edu/R/Rstatistics/Rstatistics.html, September 2017, [Online: Accessed August-2018].

[289] Will Johnson, "Explaining the lm() Summary in R," http://www.learnbymarketing.com/tutorials/explaining-the-lm-summary-in-r/, 2015, [Online: Accessed August-2018].

[290] Marcio Silva, Michael R. Hines, Diego Gallo, Qi Liu, Kyung Dong Ryu, and Dilma da Silva, "CloudBench: Experiment Automation for Cloud Environments," in *IC2E '13*, Washington, DC, USA, 2013, pp. 302–311, IEEE CS.

[291] Christina Delimitrou, Daniel Sanchez, and Christos Kozyrakis, "Tarcil: Reconciling Scheduling Speed and Quality in Large Shared Clusters," in *SoCC '15*, New York, NY, USA, 2015, pp. 97–110, ACM.

[292] Timothy Zhu, Daniel S. Berger, and Mor Harchol-Balter, "SNC-Meister: Admitting More Tenants with Tail Latency SLOs," in *SoCC '16*, New York, NY, USA, 2016, pp. 374–387, ACM.

[293] Rebecca Taft, Willis Lang, Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, and David DeWitt, "STeP: Scalable Tenant Placement for Managing Database-as-a-Service Deployments," in *SoCC '16*, New York, NY, USA, 2016, pp. 388–400, ACM.

[294] Raja R. Sambasivan, Ilari Shafer, Jonathan Mace, Benjamin H. Sigelman, Rodrigo Fonseca, and Gregory R. Ganger, "Principled Workflow-centric Tracing of Distributed Systems," in *SoCC '16*, New York, NY, USA, 2016, pp. 401–414, ACM.

[295] Kaushik Rajan, Dharmesh Kakadia, Carlo Curino, and Subru Krishnan, "PerfOrator: Eloquent Performance Models for Resource Optimization," in *SoCC '16*, New York, NY, USA, 2016, pp. 415–427, ACM.

[296] Chien-Chun Hung, Leana Golubchik, and Minlan Yu, "Scheduling Jobs Across Geo-distributed Datacenters," in *SoCC '15*, New York, NY, USA, 2015, pp. 111–124, ACM.

[297] J. E. G. Peek, Kevin A. Douglas, Min-Young Lee, Jana Grcevich, Carl Heiles, Eric Korpela, and Jacob Cychosz, "GALFA HI Data Archive at Berkeley," 2018, `https://purcell.ssl.berkeley.edu/` [Online: Accessed August-2018].

[298] Joseph C. Jacob, Daniel S. Katz, G. Bruce Berriman, John C. Good, Anastasia C. Laity, Ewa Deelman, Carl Kesselman, Gurmeet Singh, Mei-Hui Su, Thomas A. Prince, and Roy Williams, "Montage: a Grid Portal and Software Toolkit for Science-Grade Astronomical Image Mosaicking," *Int. J. Comput. Sci. Eng.*, vol. 4, no. 2, pp. 73–87, July 2009.

[299] Ilia Pietri, Gideon Juve, Ewa Deelman, and Rizos Sakellariou, "A Performance Model to Estimate Execution Time of Scientific Workflows on the Cloud," in *WORKS '14*, Piscataway, NJ, USA, 2014, pp. 11–19, IEEE Press.

[300] Rafael Ferreira da Silva, Gideon Juve, Ewa Deelman, Tristan Glatard, Frédéric Desprez, Douglas Thain, Benjamin Tovar, and Miron Livny, "Toward Fine-grained Online Task Characteristics Estimation in Scientific Workflows," in *WORKS '13*, New York, NY, USA, 2013, pp. 58–67, ACM.

[301] Gurmeet Singh, Mei-Hui Su, Karan Vahi, Ewa Deelman, Bruce Berriman, John Good, Daniel S. Katz, and Gaurang Mehta, "Workflow Task Clustering for Best Effort Systems with Pegasus," in *MG 08*, New York, NY, USA, 2008, pp. 9:1–9:8, ACM.

[302] Vitor Viana, Daniel de Oliveira, and Marta Mattoso, "Towards a Cost Model for Scheduling Scientific Workflows Activities in Cloud Environments.," in *SERVICES*. 2011, pp. 216–219, IEEE Computer Society.

[303] Forbes Guthrie, Scott Lowe, and Kendrick Coleman, *VMware vSphere Design*, SYBEX Inc., Alameda, CA, USA, 2nd Edition, 2013.

[304] Matt Liebowitz, Christopher Kusek, and Rynardt Spies, *VMware vSphere Performance: Designing CPU, Memory, Storage, and Networking for Performance-Intensive Workloads*, SYBEX Inc., Alameda, CA, USA, 1st Edition, 2014.

[305] Gohar Ahmed, *Implementing Citrix XenServer Quickstarter*, Packt Publishing, Birmingham, UK, June 2013.

[306] Navin Sabharwal and Ravi Shankar, *Apache CloudStack cloud computing: leverage the power of CloudStack and learn to extend the CloudStack environment*, Community experience distilled. Packt Publishing, Birmingham, UK, 2013.

[307] Kevin Jackson, *OpenStack Cloud Computing Cookbook*, Packt Publishing, Birmingham, UK, 2012.

[308] Aaron Paradowski, Lu Liu, and Bo Yuan, "Benchmarking the Performance of OpenStack and CloudStack," in *ISORC '14*, Washington, DC, USA, 2014, pp. 405–412, IEEE CS.

[309] Salman A. Baset, "Open Source Cloud Technologies," in *SoCC '12*, New York, NY, USA, 2012, pp. 28:1–28:2, ACM.

[310] Peter Sempolinski and Douglas Thain, "A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus," in *CLOUDCOM '10*, Washington, DC, USA, 2010, pp. 417–426, IEEE Computer Society.

[311] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov, "The Eucalyptus Open-Source Cloud-Computing System," in *CCGRID '09*, Washington, DC, USA, 2009, pp. 124–131, IEEE Computer Society.

[312] Steve Pousty and Katie Miller, *Getting Started with OpenShift*, O'Reilly Media, Inc., 1st Edition, 2014.

[313] David Bernstein, "Cloud Foundry Aims to Become the OpenStack of PaaS," *IEEE Cloud Computing*, vol. 1, no. 2, pp. 57–60, 2014.

[314] J. E. G. Peek, Carl Heiles, Kevin A. Douglas, Min-Young Lee, Jana Grcevich, Sneana Stanimirovi, M. E. Putman, Eric J. Korpela, Steven J. Gibson, Ayesha Begum, Destry Saul, Timothy Robishaw, and Marko Kro, "The GALFA-HI Survey: Data Release 1," *The Astrophysical J. Supplement Series*, vol. 194, no. 2, pp. 20, 2011.

[315] G. B. Berriman, J. Good, B. Rusholme, and T. Robitaille, "The Next Generation of the Montage Image Mopsaic Engine," in *American Astronomical Society Meeting Abstracts*, Jan. 2016, vol. 227 of *American Astronomical Society Meeting Abstracts*, p. 348.13.

[316] Michael Sindelar, Ramesh Sitaraman, and Prashant Shenoy, "Sharing-aware algorithms for virtual machine colocation," in *Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures*, New York, NY, USA, 2011, pp. 367–378.

[317] JCraft, Inc., "JSch - Java Secure Channel," `http://www.jcraft.com/jsch/`, 2017, [Online: Accessed 17-September-2017].

[318] Heekwon Park, Seungjae Baek, Jongmoo Choi, Donghee Lee, and Sam H. Noh, "Regularities Considered Harmful: Forcing Randomness to Memory Accesses to Reduce Row Buffer Conflicts for Multi-core, Multi-bank Systems," *SIGPLAN Not.*, vol. 48, no. 4, pp. 181–192, Mar. 2013.

[319] Jiannan Ouyang, John R. Lange, and Haoqiang Zheng, "Shoot4U: Using VMM Assists to Optimize TLB Operations on Preempted vCPUs," in *VEE '16*, New York, NY, USA, 2016, pp. 17–23, ACM.

[320] B. Jack Copeland, "The Modern History of Computing," in *The Stanford Encyclopedia of Philosophy*, Edward N. Zalta, Ed. Metaphysics Research Lab, Stanford University, winter 2017 Edition, 2017.

[321] William Aspray, *John Von Neumann and the Origins of Modern Computing*, MIT Press, Cambridge, MA, USA, 1990.

[322] John Von Neumann, "First draft of a report on the EDVAC," Tech. Rep., Los Alamos National Laboratory, New Mexico, USA, 1945.

[323] Raoufehsadat Hashemian, Diwakar Krishnamurthy, and Martin F. Arlitt, "Web workload generation challenges - an empirical investigation," *Softw., Pract. Exper.*, vol. 42, no. 5, pp. 629–647, 2012.

[324] Wu Guang, Massimo Baraldo, and Mario Furlanut, " Calculating percentage prediction error: A user's note ," *Pharmacological Research*, vol. 32, no. 4, pp. 241 – 248, 1995.

[325] P. M. Swamidass, Ed., *MAPE (mean absolute percentage error)MEAN ABSOLUTE PERCENTAGE ERROR (MAPE)* , pp. 462–462, Springer US, Boston, MA, 2000.

[326] Sungil Kim and Heeyoung Kim, " A new metric of absolute percentage error for intermittent demand forecasts ," *International Journal of Forecasting*, vol. 32, no. 3, pp. 669 – 679, 2016.

[327] Rosa L. Figueroa, Qing Zeng-Treitler, Sasikiran Kandula, and Long H. Ngo, " Predicting sample size required for classification performance ," *BMC Medical Informatics and Decision Making*, vol. 12, no. 1, pp. 8, Feb 2012.

[328] Torgyn Shaikhina and Natalia A. Khovanova, "Handling limited datasets with neural networks in medical applications: A small-data approach," *Artificial Intelligence in Medicine*, vol. 75, pp. 51 – 63, 2017.

[329] Junghwan Cho, Kyewook Lee, Ellie Shin, Garry Choy, and Synho Do, "Medical Image Deep Learning with Hospital PACS Dataset," *CoRR*, vol. abs/1511.06348, 2015.

[330] Mark Johnson and Dat Quoc Nguyen, "How much data is enough? Predicting how accuracy varies with training data size," September 2017, `http://web.science.mq.edu.au/~mjohnson/papers/Johnson17Power-talk.pdf` [Online: Accessed August-2018].

[331] Martin Riedmiller, "Advanced supervised learning in multi-layer perceptrons From backpropagation to adaptive learning algorithms," *Computer Standards and Interfaces*, vol. 16, no. 3, pp. 265 – 278, 1994.

[332] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *IEEE International Conference on Neural Networks*, March 1993, pp. 586–591.

[333] Aristoklis D. Anastasiadis, George D. Magoulas, and Michael N. Vrahatis, "New globally convergent training scheme based on the resilient propagation algorithm," *Neurocomputing*, vol. 64, pp. 253 – 270, 2005, Trends in Neurocomputing: 12th European Symposium on Artificial Neural Networks 2004.

[334] Marvin Minsky and Seymour Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, MA, USA, 1969.

[335] K. Hornik, M. Stinchcombe, and H. White, "Multilayer Feedforward Networks Are Universal Approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, July 1989.

[336] Raúl Rojas, *Neural Networks: A Systematic Introduction*, Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[337] Christopher M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, Inc., New York, NY, USA, 1995.

[338] Kevin L. Priddy and Paul E. Keller, *Artificial Neural Networks: An Introduction (SPIE Tutorial Texts in Optical Engineering, Vol. TT68)*, SPIE- International Society for Optical Engineering, 2005.

[339] Simon Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd Edition, 1998.

[340] Mohamad H. Hassoun, *Fundamentals of Artificial Neural Networks*, MIT Press, Cambridge, MA, USA, 1st Edition, 1995.

[341] Howard B. Demuth, Mark H. Beale, Orlando De Jess, and Martin T. Hagan, *Neural Network Design*, Martin Hagan, USA, 2nd Edition, 2014.

[342] Tariq Rashid, *Make Your Own Neural Network*, CreateSpace Independent Publishing Platform, USA, 1st Edition, 2016.

[343] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., pp. 1097–1105. Curran Associates, Inc., 2012.

[344] Christian Szegedy, Alexander Toshev, and Dumitru Erhan, "Deep Neural Networks for Object Detection," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., pp. 2553–2561. Curran Associates, Inc., 2013.

[345] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[346] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury, "Deep Neural Networks for Acoustic Modeling in Speech Recognition," *Signal Processing Magazine*, 2012.

[347] P. McCullagh and J.A. Nelder, *Generalized Linear Models, Second Edition* , Chapman and Hall/CRC Monographs on Statistics and Applied Probability Series. Chapman & Hall, 1989.

[348] W. L. Goh, D. P. Mital, and H. A. Babri, "An artificial neural network approach to handwriting recognition," in *Knowledge-Based Intelligent Electronic Systems, 1997. KES '97. Proceedings., 1997 First International Conference on*, May 1997, vol. 1, pp. 132–136 vol.1.

[349] Huang Hanmin, Huang Xiyue, Zhang Ping, Chai yi, and Shi Weiren, "ANN-based handwritten character recognition," in *SICE Annual, 1999. 38th Annual Conference Proceedings of the*, Aug 1999, pp. 1177–1180.

[350] I. T. R, V. V, and B. V. K, "Multilingual Online Handwriting Recognition System: An Android App," in *2015 Fifth International Conference on Advances in Computing and Communications (ICACC)*, Sept 2015, pp. 33–36.

[351] F. Zamora-Martínez, V. Frinken, S. España Boquera, M. J. Castro-Bleda, A. Fischer, and H. Bunke, "Neural Network Language Models for Off-line Handwriting Recognition," *Pattern Recogn.*, vol. 47, no. 4, pp. 1642–1652, Apr. 2014.

[352] Alex Graves and Jürgen Schmidhuber, "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks," in *Proceedings of the 21st International Conference on Neural Information Processing Systems*, USA, 2008, NIPS'08, pp. 545–552, Curran Associates Inc.

[353] A. C. R. Hogervorst, M. K. van Dijk, P. C. M. Verbakel, and C. Krijgsman, " Handwritten character recognition using neural networks ," in *Neural Networks: Artificial Intelligence and Industrial Applications*, Bert Kappen and Stan Gielen, Eds., London, 1995, pp. 337–343, Springer London.

[354] Stephane Kouamo and Claude Tangha, " Handwritten Character Recognition with Artificial Neural Networks ," in *Distributed Computing and Artificial Intelligence*, Sigeru Omatu, Juan F. De Paz Santana, Sara Rodríguez González, Jose M. Molina, Ana M. Bernardos, and Juan M. Corchado Rodríguez, Eds., Berlin, Heidelberg, 2012, pp. 535–543, Springer Berlin Heidelberg.

[355] S. Kurogi, " Speech recognition by an artificial neural network using findings on the afferent auditory system ," *Biological Cybernetics*, vol. 64, no. 3, pp. 243–249, Jan 1991.

[356] Glin Dede and Murat Hsn Sazli, " Speech recognition with artificial neural networks ," *Digital Signal Processing*, vol. 20, no. 3, pp. 763 – 768, 2010.

[357] N. Morgan and H. Bourlard, " Continuous speech recognition ," *IEEE Signal Processing Magazine*, vol. 12, no. 3, pp. 24–42, May 1995.

[358] Joseph Michael Tebelskis, *Speech Recognition Using Neural Networks* , Ph.D. thesis, School of Computer Science, Pittsburgh, PA, USA, May 1995, UMI Order No. GAX96-22445.

[359] Hervé Bourlard and Nelson Morgan, " Hybrid HMM/ANN Systems for Speech Recognition: Overview and New Research Directions ," in *Adaptive Processing of Sequences and Data Structures, International Summer School on Neural*

Networks, "E.R. Caianiello"-Tutorial Lectures, London, UK, UK, 1998, pp. 389–417, Springer-Verlag.

[360] Yu guo Wang and Hua peng Li, " Remote sensing image classification based on artificial neural network: A case study of Honghe Wetlands National Nature Reserve ," in *2010 International Conference on Computer, Mechatronics, Control and Electronic Engineering*, Aug 2010, vol. 5, pp. 17–20.

[361] M. Iqbal Quraishi, J. Pal Choudhury, and M. De, " Image recognition and processing using Artificial Neural Network ," in *2012 1st International Conference on Recent Advances in Information Technology (RAIT)*, March 2012, pp. 95–100.

[362] Paul Dan Cristea, " Application of Neural Networks In Image Processing and Visualization ," in *GeoSpatial Visual Analytics*, Raffaele De Amicis, Radovan Stojanovic, and Giuseppe Conti, Eds., Dordrecht, 2009, pp. 59–71, Springer Netherlands.

[363] Stephane Kouamo and Claude Tangha, " Image Compression with Artificial Neural Networks ," in *International Joint Conference CISIS'12-ICEUTE´12-SOCO´12 Special Sessions*, Álvaro Herrero, Václav Snášel, Ajith Abraham, Ivan Zelinka, Bruno Baruque, Héctor Quintián, José Luis Calvo, Javier Sedano, and Emilio Corchado, Eds., Berlin, Heidelberg, 2013, pp. 515–524, Springer Berlin Heidelberg.

[364] P. V. Rao, S. Madhusudana, N. S. S., and K. Keerthi, " Image Compression using Artificial Neural Networks ," in *2010 Second International Conference on Machine Learning and Computing*, Feb 2010, pp. 121–124.

[365] Yijing Z. Watkins and Mohammad R. Sayeh, " Image Data Compression and Noisy Channel Error Correction Using Deep Neural Network ," *Procedia Computer Science*, vol. 95, pp. 145 – 152, 2016, Complex Adaptive Systems Los Angeles, CA November 2-4, 2016.

[366] Udo Seiffert, " ANNIE-Artificial Neural Network-based Image Encoder ," *Neurocomput.*, vol. 125, pp. 229–235, Feb. 2014.

[367] Hamdy S. Soliman and Mohammed Omari, " A Neural Networks Approach to Image Data Compression ," *Appl. Soft Comput.*, vol. 6, no. 3, pp. 258–271, Mar. 2006.

[368] S. Nirkhi, " Potential use of Artificial Neural Network in Data Mining ," in *2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE)*, Feb 2010, vol. 2, pp. 339–343.

[369] G. Peter Zhang, *Neural Networks For Data Mining* , pp. 419–444, Springer US, Boston, MA, 2010.

[370] Joseph P. Bigus, *Data Mining with Neural Networks: Solving Business Problems from Application Development to Decision Support* , McGraw-Hill, Inc., New York, NY, USA, 1996.

[371] Janusz Racz and Artur Dubrawski, " Artificial neural network for mobile robot topological localization ," *Robotics and Autonomous Systems*, vol. 16, no. 1, pp. 73 – 80, 1995, Intelligent Robotics Systems SIRS '94.

[372] Sameer M. Prabhu and Devendra P. Garg, " Artificial neural network based robot control: An overview ," *Journal of Intelligent and Robotic Systems*, vol. 15, no. 4, pp. 333–365, Apr 1996.

[373] Yuan Quan and Lu Yuchang, " Research on weather forecast based on neural networks ," in *Proceedings of the 3rd World Congress on Intelligent Control and Automation (Cat. No.00EX393)*, 2000, vol. 2, pp. 1069–1072 vol.2.

[374] Kumar Abhishek, M.P. Singh, Saswata Ghosh, and Abhishek Anand, " Weather Forecasting Model using Artificial Neural Network ," *Procedia Technology*, vol. 4, pp. 311 – 318, 2012, 2nd International Conference on Computer, Communication, Control and Information Technology(C3IT-2012) on February 25 - 26, 2012.

[375] Y. Yetis, H. Kaplan, and M. Jamshidi, " Stock market prediction by using artificial neural network ," in *2014 World Automation Congress (WAC)*, Aug 2014, pp. 718–722.

[376] Amin Hedayati Moghaddam, Moein Hedayati Moghaddam, and Morteza Esfandyari, "Stock market index prediction using artificial neural network," *Journal of Economics, Finance and Administrative Science* , vol. 21, no. 41, pp. 89 – 93, 2016.

[377] Amin Hedayati, Moein Hedayati, and Morteza Esfandyari, "Stock market index prediction using artificial neural network," *Journal of Economics, Finance and Administrative Science*, vol. 21, no. 41, pp. 89–93, 2016.

[378] Jing Li, Ji-hang Cheng, Jing-yuan Shi, and Fei Huang, " Brief Introduction of Back Propagation (BP) Neural Network Algorithm and Its Improvement ," in *Advances in Computer Science and Information Engineering*, David Jin and Sally Lin, Eds., Berlin, Heidelberg, 2012, pp. 553–558, Springer Berlin Heidelberg.

[379] T. Nitta, " A back-propagation algorithm for complex numbered neural networks ," in *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*, Oct 1993, vol. 2, pp. 1649–1652 vol.2.

[380] David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group, Eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, MIT Press, Cambridge, MA, USA, 1986.

[381] David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group, Eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 2: Psychological and Biological Models*, MIT Press, Cambridge, MA, USA, 1986.

[382] James L. McClelland and David E. Rumelhart, " A simulation-based tutorial system for exploring parallel distributed processing ," *Behavior Research Methods, Instruments, & Computers*, vol. 20, no. 2, pp. 263–275, Mar 1988.

[383] Stefan Fritsch, Frauke Guenther, Marc Suling, and Sebastian M. Mueller, *Training of Neural Networks - Package* `'neuralnet'` , The R Foundation, 1.33 Edition, 08 2016, `https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf` [Online: Accessed August-2018].

[384] cran.r project.org, " neuralnet: Training of Neural Networks ," 08 2016, https://CRAN.R-project.org/package=neuralnet [Online: Accessed August-2018].

[385] Frauke Guenther, "R Documentation - neuralnet," 07 2001, https://www.rdocumentation.org/packages/neuralnet/versions/1.33/topics/neuralnet [Online: Accessed August-2018].

[386] Michy Alice, "Fitting a Neural Network in R; neuralnet package," 02 2018, https://datascienceplus.com/fitting-neural-network-in-r/ [Online: Accessed August-2018].

[387] Michael Grogan, "neuralnet: Train and Test Neural Networks Using R," June 2018, http://www.michaeljgrogan.com/neural-network-modelling-neuralnet-r/ [Online: Accessed August-2018].

[388] Frauke Gnther and Stefan Fritsch, "neuralnet: Training of Neural Networks," *The R Journal*, vol. 2, no. 1, pp. 30–38, 2010.

[389] R Core Team and contributors worldwide, "The R Base Package version 3.6.0 - Inhibit Interpretation/Conversion of Objects," https://stat.ethz.ch/R-manual/R-devel/library/base/html/AsIs.html, 2018, [Online: Accessed August-2018].

[390] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks, "Profiling a Warehouse-scale Computer," *SIGARCH Comput. Archit. News*, vol. 43, no. 3, pp. 158–169, June 2015.

[391] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica, "Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, Berkeley, CA, USA, 2012, NSDI'12, pp. 2–2, USENIX Association.

[392] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly, " Dryad: Distributed Data-parallel Programs from Sequential Building Blocks

," in *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, New York, NY, USA, 2007, EuroSys '07, pp. 59–72, ACM.

[393] Hyoseung Kim, Shige Wang, and Ragunathan Rajkumar, "Responsive and Enforced Interrupt Handling for Real-Time System Virtualization," in *21st IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2015, Hong Kong, China, August 19-21, 2015*, 2015, pp. 90–99.

[394] Diego Ongaro, Alan L. Cox, and Scott Rixner, "Scheduling I/O in Virtual Machine Monitors," in *Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, New York, NY, USA, 2008, VEE '08, pp. 1–10, ACM.

[395] Ludmila Cherkasova, Diwaker Gupta, and Amin Vahdat, "Comparison of the Three CPU Schedulers in Xen," *SIGMETRICS Perform. Eval. Rev.*, vol. 35, no. 2, pp. 42–51, Sept. 2007.

[396] Irfan Ahmad, Ajay Gulati, and Ali Mashtizadeh, "vIC: Interrupt Coalescing for Virtual Machine Storage Device IO," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, Berkeley, CA, USA, 2011, USENIXATC'11, pp. 4–4, USENIX Association.

[397] Athanasios Antoniou, "Performance Evaluation of Cloud Infrastructure using Complex Workloads," M.S. thesis, Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, 2012.

[398] Xiaolin Chang, Jogesh K. Muppala, Zhen Han, and Jiqiang Liu, "Analysis of Interrupt Coalescing Schemes for Receive-Livelock Problem in Gigabit Ethernet Network Hosts," in *Proceedings of IEEE International Conference on Communications, ICC 2008, Beijing, China, 19-23 May 2008*, 2008, pp. 1835–1839.

[399] Takahiro Hirofuchi and Ryousei Takano, "RAMinate: Hypervisor-based Virtualization for Hybrid Main Memory Systems," in *SoCC*. 2016, pp. 112–125, ACM.

[400] Ning Li, Hong Jiang, Dan Feng, and Zhan Shi, "PSLO: Enforcing the Xth Percentile Latency and Throughput SLOs for Consolidated VM Storage," in *EuroSys*. 2016, pp. 28:1–28:14, ACM.

[401] Peng Wang, Hong Xu, Zhixiong Niu, Dongsu Han, and Yongqiang Xiong, "Expeditus: Congestion-aware Load Balancing in Clos Data Center Networks," in *SoCC*. 2016, pp. 442–455, ACM.

[402] Junji Zhi, Nilton Bila, and Eyal de Lara, "Oasis: Energy Proportionality with Hybrid Server Consolidation," in *EuroSys*. 2016, pp. 10:1–10:13, ACM.

[403] Jennie Duggan, Yun Chi, Hakan Hacigümüs, Shenghuo Zhu, and Ugur Çetintemel, "Packing light: Portable workload performance prediction for the cloud," in *Workshops Proceedings of the 29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, 2013, pp. 258–265.

[404] Jennie Duggan, Ugur Cetintemel, Olga Papaemmanouil, and Eli Upfal, "Performance Prediction for Concurrent Database Workloads," in *SIGMOD*. 2011, pp. 337–348, ACM.

[405] Hakan Hacigumus, Yun Chi, Wentao Wu, Shenghuo Zhu, Junichi Tatemura, and Jeffrey F. Naughton, "Predicting Query Execution Time: Are Optimizer Cost Models Really Unusable?," in *ICDE*. 2013, pp. 1081–1092, IEEE CS.

[406] Jeanna Neefe Matthews, Wenjin Hu, Madhujith Hapuarachchi, Todd Deshane, Demetrios Dimatos, Gary Hamilton, Michael McCabe, and James Owens, "Quantifying the Performance Isolation Properties of Virtualization Systems," in *ExpCS '07*, New York, NY, USA, 2007, ACM.

[407] Aravind Menon, Jose Renato Santos, Yoshio Turner, G. (John) Janakiraman, and Willy Zwaenepoel, "Diagnosing Performance Overheads in the Xen Virtual Machine Environment," in *VEE '05*, New York, NY, USA, 2005, pp. 13–23, ACM.

[408] Ming Tao, Shoubin Dong, and Liping Zhang, "A multi-strategy collaborative prediction model for the runtime of online tasks in computing cluster/grid.," *Cluster Computing*, vol. 14, no. 2, pp. 199–210, 2011.

[409] Eduardo Javier Huerta Yero and Marco Aurlio Amaral Henriques, "Contention-sensitive static performance prediction for parallel distributed applications.," *Perform. Eval.*, vol. 63, no. 4-5, pp. 265–277, 2006.

[410] Qingjia Huang, Kai Shuang, Peng Xu, Jian Li, Xu Liu, and Sen Su, "Prediction-based Dynamic Resource Scheduling for Virtualized Cloud Systems," *Journal of Networks*, vol. 9, no. 2, 2014.

[411] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica, "Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics," in *NSDI*, Santa Clara, CA, Mar. 2016, pp. 363–378, USENIX.

[412] Alexandru Agache, Razvan Deaconescu, and Costin Raiciu, "Increasing Datacenter Network Utilisation with GRIN," in *NSDI*, Oakland, CA, May 2015, pp. 29–42, USENIX.

[413] George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza A. Basrai, and Peter M. Chen, "ReVirt: Enabling Intrusion Analysis Through Virtual-machine Logging and Replay," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 211–224, Dec. 2002.

[414] Stephen T. Jones, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, " Geiger: Monitoring the Buffer Cache in a Virtual Machine Environment ," *SIGARCH Comput. Archit. News*, vol. 34, no. 5, pp. 14–24, Oct. 2006.

[415] Anna Melekhova and Larisa Markeeva, " Estimating Working Set Size by Guest OS Performance Counters Means ," in *The Sixth International Conference on Cloud Computing, GRIDs, and Virtualization, CLOUD COMPUTING 2015*. 2015, pp. 33–38, IARIA.

[416] J. Eli Goldston Peek, B. L. Babler, K. A. Douglas, Y. Zheng, S. Clark, M. E. Putman, S. Stanimirovic, C. E. Heiles, S. J. Gibson, and E. J. Korpela, " The Galactic Arecibo L-Band Feed Array Survey Data Release 2 ," in *American Astronomical Society Meeting Abstracts*, Jan. 2016, vol. 227 of *American Astronomical Society Meeting Abstracts*, p. 347.08.

[417] P. A. Henning, C. M. Springob, R. F. Minchin, E. Momjian, B. Catinella, T. McIntyre, F. Day, E. Muller, B. Koribalski, J. L. Rosenberg, S. Schneider, L. Staveley-Smith, and W. van Driel, " The Arecibo L-band Feed Array Zone of Avoidance Survey. I. Precursor Observations Through the Inner and Outer Galaxy ," *The Astronomical Journal*, vol. 139, no. 6, pp. 2130, 2010.

[418] Jeffrey Dean and Sanjay Ghemawat, " MapReduce: Simplified Data Processing on Large Clusters ," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, Berkeley, CA, USA, 2004, OSDI'04, pp. 10–10, USENIX Association.

[419] Wei Zhang, Sundaresan Rajasekaran, Shaohua Duan, Timothy Wood, and Mingfa Zhu, "Minimizing Interference and Maximizing Progress for Hadoop Virtual Machines," *SIGMETRICS Performance Evaluation Review*, vol. 42, no. 4, pp. 62–71, 2015.

[420] Roger Faulkner and Ron Gomes, " The Process File System and Process Model in UNIX System V ," in *Proceedings of the Usenix Winter 1991 Conference, Dallas, TX, USA, January 1991*, 1991, pp. 243–252.

[421] Luwei Cheng, Jia Rao, and Francis C. M. Lau, "vScale: Automatic and Efficient Processor Scaling for SMP Virtual Machines," in *Proceedings of the Eleventh European Conference on Computer Systems*, New York, NY, USA, 2016, EuroSys '16, pp. 2:1–2:14, ACM.

[422] Pin Lu and Kai Shen, " Virtual Machine Memory Access Tracing with Hypervisor Exclusive Cache ," in *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, Berkeley, CA, USA, 2007, ATC'07, pp. 3:1–3:15, USENIX Association.

[423] L. Cheng and F. C. M. Lau, "Offloading Interrupt Load Balancing from SMP Virtual Machines to the Hypervisor," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 11, pp. 3298–3310, Nov 2016.

[424] Luwei Cheng and Cho-Li Wang, "vBalance: Using Interrupt Load Balance to Improve I/O Performance for SMP Virtual Machines," in *Proceedings of the*

Third ACM Symposium on Cloud Computing, New York, NY, USA, 2012, SoCC '12, pp. 2:1–2:14, ACM.

[425] Qunying Huang, Jizhe Xia, Min Sun, Kai Liu, Jing Li, Zhipeng Gui, Chen Xu, Qunying Yang, Chaowei, Jizhe Xia, Min Sun, Kai Liu, Jing Li, Zhipeng Gui, Chen Xu, and Chaowei Yang, " How to test the readiness of open-source cloud computing solutions ," in *Spatial Cloud Computing*, Chaowei Yang and Qunying Huang, Eds., Chapter 14, pp. 241–260. CRC Press, Oxford, 2013.

[426] Colin Ian King, "Stress-ng benchmark," 2017, `http://kernel.ubuntu.com/~cking/stress-ng/` [Online: Accessed August-2017].

[427] Josep Ll. Berral, Ricard Gavalda, and Jordi Torres, "Adaptive Scheduling on Power-Aware Managed Data-Centers Using Machine Learning," in *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*, Washington, DC, USA, 2011, GRID '11, pp. 66–73, IEEE Computer Society.

[428] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su, " Scaling Distributed Machine Learning with the Parameter Server ," in *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, Berkeley, CA, USA, 2014, OSDI'14, pp. 583–598, USENIX Association.

[429] Paul S. Bradley and O. L. Mangasarian, " Feature Selection via Concave Minimization and Support Vector Machines ," in *Proceedings of the Fifteenth International Conference on Machine Learning*, San Francisco, CA, USA, 1998, ICML '98, pp. 82–90, Morgan Kaufmann Publishers Inc.

[430] Verónica Bolón-Canedo, Diego Peteiro-Barral, Amparo Alonso-Betanzos, Bertha Guijarro-Berdiñas, and Noelia Sánchez-Maroño, " Scalability Analysis of ANN Training Algorithms with Feature Selection ," in *Advances in Artificial Intelligence: 14th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2011, La Laguna, Spain, November 7-11, 2011. Proceedings*, Jose A. Lozano, José A. Gámez, and José A. Moreno, Eds., pp. 84–93. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

“Indeed we belong to Allah, and indeed to Him we will return.”

Al-Baqarah [20:156]