Aalto University
School of Science
Degree Programme of Computer Science and Engineering

Hao Zhuang

Performance Evaluation of Virtualization in Cloud Data Center

Master's Thesis
Espoo, June 18, 2012

Supervisors: Professor Antti Ylä-Jääski, Aalto University
Associate Professor Markus Hidell, Royal Institute of Technology, KTH
Instructor: Zhonghong Ou, PhD, Aalto University

Aalto University
School of Science
Degree Programme of Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Hao Zhuang

Title:
Performance Evaluation of Virtualization in Cloud Data Center

Date: June 18, 2012 **Pages:** 67

Professorship: Data Communication Software **Code:** T-110

Supervisors: Professor Antti Ylä-Jääski
Associate Professor Markus Hidell, Royal Institute of Technology, KTH

Instructor: Zhonghong Ou, PhD, Aalto University

Amazon Elastic Compute Cloud (EC2) has been adopted by a large number of small and medium enterprises (SMEs), e.g. foursquare, Monster World, and Netflix, to provide various kinds of services. There has been some existing work in the current literature investigating the variation and unpredictability of cloud services. These work demonstrated interesting observations regarding cloud offerings . However, they failed to reveal the underlying essence of the various appearances for the cloud services.

In this thesis, we looked into the underlying scheduling mechanisms, and hardware configurations, of Amazon EC2, and investigated their impact on the performance of virtual machine instances running atop. Specifically, several instances with the standard and high-CPU instance families are covered to shed light on the hardware upgrade and replacement of Amazon EC2. Then large instance from the standard family is selected to conduct focus analysis. To better understand the various behaviors of the instances, a local cluster environment is set up, which consists of two Intel Xeon servers, using different scheduling algorithms.

Through a series of benchmark measurements, we observed the following findings: (1) Amazon utilizes highly diversified hardware to provision different instances. It results in significant performance variation, which can reach up to 30%. (2) Two different scheduling mechanisms were observed, one is similar to Simple Earliest Deadline Fist (SEDF) scheduler, whilst the other one analogies Credit scheduler in Xen hypervisor. These two scheduling mechanisms also arouse variations in performance. (3) By applying a simple "trial-and-failure" instance selection strategy, the cost saving is surprisingly significant. Given certain distribution of fast-instances and slow-instances, the achievable cost saving can reach 30%, which is attractive to SMEs which use Amazon EC2 platform.

Keywords: Performance Evaluation, Xen Virtualization, Cloud Data Center, Energy Efficiency

Language: English

Acknowledgments

At this moment, a myriad of thoughts and thankfulness are filled my mind. I would like to give my sincere thanks for those who offer the generous help during my thesis project.

First and foremost, I would like to thank my supervisors, Prof. Antti Ylä-Jääski from Aalto University and Associate Prof. Markus Hidell from Royal Institute of Technology (KTH), for their high-level guidances and valuable and helpful comments for my thesis work. Also, I would like to express my sincere gratitude to my instructor, Dr. Ou Zhonghong from Aalto University, for his valuable guidance and mentorship. He not only gave me a lot of help on the experiments and thesis writing, but, more importantly, showed me how to conduct the scientific research with a positive attitude. These valuable experiences will bring many benefits to my future work and study.

Secondly, I would like express my thanks to my classmates Xin Gu, Antony Meyn and Vu Ba Tien Dung. They offers me a lot of useful suggestions and ideas. Our discussions have refreshed my minds, which gave a great help for my thesis work. Also, I would like to thank Daoyuan Li, Guohua Liu, and Xiang Gan. We always share the ideas as well as stories during lunch time, which kept me high motivated and enthusiasms.

Last but not least, I would like to gratitude my parents and girl friend Huizhen Li, who always support me in all aspects of my life. The thesis cannot finished without your supports.

Espoo, June 18, 2012

Hao Zhuang

Contents

Table of Contents iv

List of Tables vii

List of Figures viii

Abbreviations and Acronyms ix

1 Introduction 1
- 1.1 Overview . . . 1
- 1.2 Problem Statement . . . 2
- 1.3 Contributions . . . 3
- 1.4 Thesis Organization . . . 3

2 Background 5
- 2.1 Xen Virtualization . . . 5
  - 2.1.1 Introduction to Virtualization . . . 5
  - 2.1.2 Xen Hypervisor . . . 6
  - 2.1.3 CPU Schedulers in Xen . . . 7
- 2.2 Amazon EC2 . . . 8
  - 2.2.1 Instance Type . . . 8
  - 2.2.2 Multiple Locations . . . 9
- 2.3 Literature Review . . . 9
  - 2.3.1 Xen Virtualization . . . 9
  - 2.3.2 Performance Evaluation in Amazon EC2 . . . 10
  - 2.3.3 Mobile Clients for Cloud . . . 11
- 2.4 Benchmark Survey . . . 13
  - 2.4.1 Benchmark Classification . . . 13
  - 2.4.2 Benchmark Overview . . . 14

3 Methodology **18**
3.1 Environment Setup . . . 18
3.1.1 Amazon EC2 . . . 18
3.1.2 Local Cloud . . . 18
3.2 Measurement Metrics and Tools . . . 19
3.2.1 CPU Performance . . . 19
3.2.2 Memory Performance . . . 20
3.2.3 Disk Performance . . . 20
3.2.4 Network Throughput . . . 20
3.2.5 Network Latency . . . 21
3.2.6 Web Server Performance . . . 21
3.2.7 Energy Efficiency of Mobile Clients . . . 22

4 Evaluation **23**
4.1 Amazon EC2 API Tools . . . 23
4.2 CPU Performance . . . 24
4.2.1 Detect CPU Type . . . 24
4.2.2 CPU Performance . . . 26
4.2.3 CPU Process Sharing . . . 28
4.2.4 Amazon Scheduling Mechanisms . . . 29
4.2.4.1 CPU Utilization . . . 29
4.2.4.2 Scheduling Mechanisms . . . 31
4.2.4.3 Local Emulation . . . 33
4.3 Memory Performance . . . 34
4.4 Disk Performance . . . 35
4.5 Network Throughput . . . 37
4.6 Network Latency . . . 39
4.7 Web Server Performance . . . 40
4.7.1 Static HTTP Request . . . 40
4.7.2 Dynamic HTTP Request . . . 41
4.8 Energy Efficiency of Mobile Clients . . . 42
4.8.1 Traffic Characteristic . . . 42
4.8.2 Energy Efficiency Discussion . . . 43

5 Discussion **45**
5.1 Performance Review . . . 45
5.2 Cost Analysis . . . 46
5.3 Implications . . . 49
5.3.1 High Performance Computing . . . 49
5.3.2 Latency Sensitive Application . . . 49

6 Conclusions and Future Work **51**
6.1 Summary 51
6.2 Future Work 52

A Implementation Issues **60**
A.1 CPUBen 60
A.1.1 Single Process 60
A.1.2 Multiple Processes 61
A.2 TCP Benchmark 63
A.2.1 TCPBenServer.cpp 63
A.2.2 TCPBenClient.cpp 66

List of Tables

2.1 Instances in Amazon EC2 8

4.1 CPU type in different instances 25
4.2 CPU utilization rate 30
4.3 CPU utilization of emulated instances 33
4.4 Handling Latency of GET Request 35

5.1 Performance variations 45
5.2 Notations 46

List of Figures

4.1 CPU performance scores for the *m1.large* instance with different cpu type 27
4.2 CPU process sharing 28
4.3 Running time, waiting time, and period of m1.small instances (single-process) 31
4.4 Running time, waiting time, and period of m1.large instances (single-process) 32
4.5 Redis GET operation 34
4.6 Dbench workload throughput 36
4.7 TCP throughput of m1.large instances 37
4.8 UDP throughput of m1.large instances 38
4.9 Network latency of m1.large instances 39
4.10 Httperf performance of static HTTP requests 40
4.11 Httperf performance of dynamic HTTP requests 41
4.12 Traffic Shape 42
4.13 Energy Efficiency 43

5.1 Cost saving analysis 48

Abbreviations and Acronyms

Amazon EC2	Amazon Elastic Cloud Computing
AWS	Amazon Web Services
VM	Virtual Machine
OS	Operating System
IT	Information Technology
VMM	Virtual Machine Monitor
Intel VT	Intel Virtualization Technology
AMD-V	AMD Virtualization
Dom0	Domain 0
DomU	Guest Domain
BVT	Borrowed Virtual Time
SEDF	Simple Earliest Deadline First
WC	Work-Conserving
NWC	Non Work-Conserving
ECU	EC2 Compute Unit
HPC	High Performance Computing
S3	Simple Storage Service
SQS	Simple Queue Service
CDN	Content Delivery Network
REST	REpresentational State Transfer
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
AMI	Amazon EC2 Machine Image
FPS	First Person Shooter
API	Application Programming Interface
SSH	Secure SHell
RTT	Round Trip Time
HTTP	Hypertext Transmission Protocol
WNI	Wireless Network Interface
PSM	Power Saving Mode

Chapter 1

Introduction

1.1 Overview

Many organizations and companies process operational and business data in their in-house data centers, which can maximize IT efficiency and, in turn, improve overall organization efficiency. However, with increasing complexity and growing amount of data, organizations need a more cost and energy effective utilization of IT infrastructure and manage a variety of resources in a more flexible, efficient and safe manner. To meet these new demands, many leading cloud providers, such as Amazon, Microsoft and Google, have built their cloud data centers, which open new windows of provisioning services and applications in an on-demand and pay-as-you-go manner. Cloud data centers give users an appearance of having infinite computing resources. Users can rent virtual machine instances with different capabilities as needed and adjust resources according to various demands dynamically.

Meanwhile a wide range of related services, emerging during the development of cloud data centers, have simplified the operations for end-users, thereby significantly reducing maintenance costs. For example, Amazon EC2 provides an array of value-added services such as Management Console, CloudFront and Elastic MapReduce for end-users to manage compute, storage, security and other cloud resources conveniently. At present, hundreds of thousands of companies, especially the small and medium business(SME), have joined the Amazon Web Services (AWS) community and used AWS solutions to build their businesses, which is ranging among application hosting, content delivery, storage and high performance computing.

Along with ever-increasing cloud services used by various end users, it is es-

sential to conduct a comprehensive performance evaluation in the cloud data center. There have already been some studies focusing on different aspects of clouds services and performance variations. In this paper, we take a step further analyze the underlying scheduling mechanisms and hardware configurations for the VM instances running atop, to shed light on the essential causes of the various performance behaviors.

1.2 Problem Statement

Most of cloud data centers utilize machine virtualization technologies to provide agile and efficient IT infrastructure for the users. For example, Amazon EC2 applies Xen virtualization to provide multiple virtual machines on a single physical server while Microsoft Azure uses Azure VM and Google AppEngine has its own proprietary sandbox. However, virtualization of the underlying resources also yields a certain number of tradeoffs. For instance, CPU cores and network interfaces in a physical machine are shared among a bunch of instances, which usually do not belong to the same user. This sharing of resources brings processing latency, incurs unnecessary packet loss, and lowers the network throughput, which are not desirable for cloud applications.

On the other hand, it has been the sixth year since Amazon announced the first beta version of EC2 service in 2006. One could naturally speculate that due to hardware upgrade and replacement, Amazon EC2 will most likely utilize diversified hardware to provision various VM instances. Therefore, it is interesting to exploit the impact of hardware heterogeneity on performance variation of the same instance type. Moreover, we could make an cost analysis for cloud users, which helps them make large savings.

In this thesis, we focus on the Amazon EC2 cloud data center and its Xen virtualization technology. We carry out active measurements on Amazon EC2 and try to find out the impact of virtualization on performance in Amazon EC2. Specifically, we try to answer the following questions:

1. What are the CPU scheduling mechanisms adopted by Xen-based Amazon EC2. How do the scheduling mechanisms impact on the instance's performance such as CPU capacity, memory speed, disk I/O and network I/O?

2. Does Amazon EC2 utilize homogeneous or heterogeneous hardware configuration? If heterogeneous hardware are used, what is the re-

sulting performance variation?

3. With more mobile applications hosting in Amazon EC2, is there any specific mechanisms adopted by Amazon for mobile clients connecting to EC2? Does the virtualization impact the energy efficiency of mobile clients?

1.3 Contributions

The goal of this thesis is to study the impact of virtualization in cloud data center. Specifically, we perform the experiments on the standard large instance in Amazon EC2 data center. To further explore the profoundness, we also carry out the experiment in the local data center in Aalto University. Our contributions can be summarized as follows.

1. Amazon EC2 uses widely diverse hardware configurations to build up its platform; the diversity not only lies in different instance types, but also occurs within the same instance type;

2. To decrease or eliminate the diversity, distinct scheduling mechanisms have been employed to assure the fair-share of CPU processing capability of the same instance type;

3. The distinct scheduling mechanisms incur unpredictable performance on the other aspects of the VMs, e.g. network throughput, network latency, and Web server performance;

4. The variation of the same sub-type of instances, i.e. hosted by identical hardware, is relatively small, whilst the variation among different sub-types of instances , i.e. hosted by heterogeneous hardware, can reach up to 60%.

5. By selecting better performing instances rather than taking the random instances assigned by Amazon EC2 platform to complete the same task, EC2 users can acquire up to 30% of cost saving.

1.4 Thesis Organization

Chapter 2 introduces the background of the thesis. Firstly, Xen virtualization and its two CPU schedulers are discussed. Secondly, We introduce the Ama-

zon EC2 data center including different instance types, available regions and zones. Thirdly, we demonstrate our related work in three aspects, namely Xen virtualization technology, performance evaluation in Amazon EC2 and mobile clients. Finally, a comprehensive benchmark survey will be presented.

In Chapter 3, we describe the methodology of performance evaluation including the performance metrics, measurement environment setup and selected benchmarks.

Chapter 4 describe the implementation of performance evaluation. Furthermore, we give an in-depth analysis on the results of every test cases.

Chapter 5 discusses the implications based on the performance evaluations and also we make a cost analysis based on the results. From the analysis, we provide an mathematical model for cloud users to achieve better cost efficiency.

Finally, conclusions and future work are given in chapter 6. We summarize what we have evaluated as well as our findings. The thesis concludes with a discussion about what future work is needed.

Chapter 2

Background

2.1 Xen Virtualization

2.1.1 Introduction to Virtualization

Virtualization provides a software layer to host multiple guest operating systems(OS), which enable them to share a single underlying piece of hardware. Compare to guest OS, the OS running virtualization software is called host OS. The software that creates a guest on the host OS is called a *hypervisor* or *Virtual Machine Monitor*. For the guest OS, it is completely unaware that they are running on the top of a hypervisor rather than directly on the physical hardware. There are three types of vitualization techniques that can be used to virtualize a guest OS.

Full virtualization allows a guest OS running without any modifications. It is a complete simulation of the actual hardware and a technique called *binary translation* is employed to automatically modify x86 software on-the-fly to replace instructions with a different, virtual machine safe sequence of instructions.

Paravirtualization allows the guest OS to communicate with hypervisor. Thus, the guest OS needs to be specifically modified to translate non-virtualizable instruction with *hypercalls* which are instructions that make it possible for communication.

Hardware-assisted virtualization is to implement virtualization with support of hardware. CPUs with virtualization technology have some new instructions to control virtualization, which make it simpler for VMM, thereby improving performance compared to software-only solutions. For example,

Intel Virtualization Technology (Intel VT) and AMD Virtualization (AMD-V) have an extra instruction set called Virtual Machine Extensions or VMX to support hardware-assisted virtualization.

2.1.2 Xen Hypervisor

Xen [3], firstly developed by University of Cambridge Computer Laboratory, is an open source virtual machine monitor for multiple CPU architectures. It supports a wide range of guest operating systems including Windows, Linux, Solaris and various versions of the BSD operating system.

Xen, slightly different from their original form, employs para virtualization to expose the underlying hardware resources to the upper layer guest Operating System (OS). Specifically, Xen utilizes a synchronous *hypercall* mechanism to enable guest domains (in Xen terminology, DomUs) to conduct software trap into the hypervisor to perform privileged operations, e.g. page table updates. It uses an asynchronous event mechanism to deliver virtual interrupts and other notifications to guest domains. The latest Xen architecture introduces a dedicated privileged domain, i.e. driver domain (referred to as domain0, or Dom0) to deal with all I/O related operations. Dom0 implements unmodified device drivers, has direct access to the corresponding hardware devices, e.g. disk, and network interface card (NIC), and performs I/O operations on behalf of DomUs. DomUs run ported device drivers and communicate via the I/O channel with Dom0 to access the real hardware devices. Hardware interrupts are firstly captured and handled by the Xen VMM, and then notifications are sent as virtual interrupts via the event channel to Dom0. After receiving the virtual interrupts, when Dom0 is scheduled to run again, it initiates virtual interrupts to the corresponding destination DomUs.

With regard to the Xen network architecture, in the front-end, DomUs create virtual interfaces and forward the network requests of virtual interfaces to the Dom0. While in the back-end, Dom0 has created the virtual interfaces corresponding to each virtual interface in the DomUs and forward the network requests from the domUs. Furthermore, all the backend interfaces in the Dom0 are connected to the physical NIC and to each other through a virtual network bridge. Hypervisor is responsible for establishing event channels and I/O channels which enable the communication between Dom0 and DomUs. Based on network virtualization architecture, Dom0 acts as the proxy between DomUs and physical NIC. All the packets sent from DomUs will be first forwarded to Dom0 and then sent into the Internet. It immediately follows that all the packets sending to the DomUs transferred

by Dom0.

2.1.3 CPU Schedulers in Xen

Xen allows users to choose the proper schedulers for their applications. Three CPU schedulers are introduced in Xen, namely Borrowed Virtual Time (BVT) [] Simple Earliest Deadline First(SEDF) and Credit [7]. Currently, the default scheduler in Xen 4.1.2 is Credit1. In this paper, we will focus on Credit and SEDF scheduler since BVT is depreciated.

Credit is the Xen latest Proportional Share based schedulers which allocates CPU in proportion to the number of shares (weights) that VMs have been assigned. It assigns two parameters, namely *weight* and *cap*, to each guest domain. Each domain gets CPU time relative to other domains according to its weight. The cap parameter sets the upper bound of the CPU time a domain is eligible to require even if there is extra CPU capacity available. Two priority states exist in credit scheduler, i.e. under and over, whilst the former means the domain owns positive amount of credits, the latter implies that the domain has run out of more than its fair share of CPU time. Domains with the under priority is always scheduled to run ahead of domains with over priority. Within the same priority list, domains are scheduled to run in a round-robin manner. It has a good support of global load balancing on multiprocessors.

SEDF utilizes real-time algorithms to deliver time guarantees and provides weighted CPU sharing. Every domain expresses its CPU requirements through four parameters.

period - time interval during which a domain is guaranteed to receive its allocation of CPU time;

slice - time per period that a domain is guaranteed to run for;

extra - flag (0/1), which controls whether the domain can run in extra-time; put it in another way, extra=1 means Work-Conserving mode (WC-mode), whilst extra=0 means Non Work-Conserving mode (NWC-mode);

weight - mutually exclusive with period/slice and specifies another way of setting a domains CPU slice.

2.2 Amazon EC2

2.2.1 Instance Type

Table 2.1: Instances in Amazon EC2

Instance Family	Instance Type	cpu (ECU)	memory (GB)	disk (GB)	I/O
Standard	m1.small	1	1.7	160	moderate
	m1.medium	2	3.75	410	moderate
	m1.large	4	7.5	850	high
	m1.xlarge	8	15	690	high
Micro	t1.micro	up to 2	631MB	EBS only	low
High-Memory	m2.xlarge	6.5	17.1	420	Moderate
	m2.2xlarge	13	34.2	850	High
	m2.4xlarge	26	68.4	1690	High
High-CPU	c1.medium	5	1.7	350	moderate
	c1.xlarge	20	7	1690	high
Cluster	cc1.4xlarge	33.5	23	1690	very high
	cc1.8xlarge	88	60.5	3370	very high
GPU	cg1.4xlarge	33.5	22	1690	very high

In our thesis, we focus on Amazon EC2 data center, which provides a wide range of instances for different needs. Table 2.1 shows the instances in the Amazon data center. Amazon offer six different families of instances, namely Standard, Micro, High-Memory, High-CPU, Cluster and Cluster GPU, which meet the different types of applications (e.g., compute- and memory-intensive, graphics processing units, etc.). For each family, the instances are further divided into different types, such as micro, small, medium, large, extra large, double extra large, quadruple and eight extra large, with different configurations in CPU, memory, disk and I/O performance. The detailed configuration information are listed in the table 2.1. Among which, CPU performance is assessed by EC2 Compute Unit (ECU) which, according to Amazon, is equivalent to the CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. Amazon EC2 uses Xen virtualization, an open-source virtual machine monitor (VMM), to manage physical servers. Therefore each instance is actually a Xen virtual machine.

2.2.2 Multiple Locations

The instances of Amazon EC2 are placed in multiple locations which are composed of *regions* and *availability zones*. There are seven regions all over the world [1], namely US East (Virginia), US West(Oregon), US West (California), EU West(Ireland), Asia Pacific (Singapore), Asia Pacific (Tokyo) and South America(SaoPaulo). Each region has different available zones. For example, in US East region, there are five available zones: us-east-1a, us-east-1b,us-east-1c, us-east-1d and us-east-1e. These availability zones are distinct locations which provides inexpensive, low latency network connectivity to other Availability Zones in the same Region. The companies could deploy their applications in different zones and regions as replications to improve reliability, fault-tolerance, or accessibility.

2.3 Literature Review

Aimed at our goals of performance evaluation, most of the literature studies focus on three areas: Xen virtualization, performance evaluation in Amazon EC2 and mobile clients for cloud.

2.3.1 Xen Virtualization

For Xen virtualization, most of the studies are carried out in the local data centers to explain the tradeoffs brought by Xen virtualization. [3] presented the design of Xen virtualization and a thorough performance evaluation of Xen. They compared Xen with a number of alternative virtualization techniques. Moreover, they evaluated the Xen performance of running concurrent applications as well as the performance isolation of Xen. In the basement of [3] study, [7],[13],[6] mainly focused on the Xen CPU schedulers and its overhead while [28],[31] further investigated the network I/O of Xen. Different approaches are proposed to improve the fair-share CPU scheduling and network throughput, e.g., tuning the parameters of CPU scheduler and changing I/O related parameters.As for performance isolation, [15] presented XenMon to accurately measure per VM resource consumption,including work done on behalf of a particular VM in Xen's driver domains. [29], [41] and [12] conducted their studies on performance profiling in Xen virtual machine environment.Xenoprof [29] was implemented to provide system-wide profiling

[1] `http://aws.amazon.com/ec2/`

toolkit for Xen virtualization.

2.3.2 Performance Evaluation in Amazon EC2

With the widespread use of Amazon EC2 data center, an increasing number of studies have conducted to evaluate the impact of virtualization on network performance and cloud services. The existing research efforts can be divided into four categories: High Performance Computing(HPC) oriented, database oriented, systematic performance comparator and others.

1) HPC-oriented: [44] compared an Amazon EC2-based platform built upon its newly available HPC-oriented virtual machines with typical local cluster and supercomputer options, using benchmarks and applications with scale and problem size unprecedented in previous cloud HPC studies. Walker [37] studied the performance of Amazon EC2 high-performance cluster compute instances against a locally configured equivalent processors cluster, and showed that there existed a performance gap between the EC2 provisioned cluster and local traditional scientific cluster.[17] represented the most comprehensive evaluation to date comparing conventional HPC platforms to Amazon EC2, using real applications representative of the workload at a typical supercomputing center.

2) Database-oriented: Cooper et al. [10] developed Yahoo Cloud Serving Benchmark (YCSB) framework to facilitate performance comparison. They also defined several core workloads and introduced a workload generator to facilitate the comparison. Garfinkel [14] conducted a measurement study of various Amazon Web Services (AWS), including EC2, Simple Storage Service (S3) and Simple Queue Service (SQS) etc, to evaluate the feasibility and cost of moving a large-scale research application from localized server to Amazon offering. Later on, Palankar et al. [32] performed measurements focusing on Amazon S3 to testify its ability to provide stable storage support for large-scale scientific computation application. Their results showed that Amazon S3 service needs to be improved to support scientific computing community.

3) Systematic performance comparator: Li et al.[24] developed a performance and cost comparator, i.e. CloudCmp, to measure various cloud services from different cloud providers, including Amazon AWS, Microsoft Azure, Google AppEngine, and Rackspace ClousServers. Their study demonstrated that there was no single winner who outperformed the other counterparts in all aspects of its cloud service offerings. In order to provide apples-to-apples comparison of different cloud serving systems, e.g. HBase, BigTable, [23] designed a new performance measuring method for Infrastructure-as-a-Service

offerings which take into account the type of services running in cloud. [34] carried out a study of the performance variance of the most widely used Cloud infrastructure (Amazon EC2) from different perspectives(CPU,I/O and network). Their results showed that EC2 performance varies a lot and often falls into two bands,time and locations, having a large performance gap, e.g., the choice of availability zone also influences the performance variability.[25] presented CloudProphet, a low cost tool to accurately predict the end-to-end response time of an on-premise web application if migrated to a cloud.

4) Others: Wang et al. [38] presented a measurement study on the impact of virtualization on Amazon EC2 platform. Their findings indicated that virtualization causes instability and variation to network throughput and packet delay. [4] used a combination of micro-benchmarks and two real-world latency sensitive applications-the Doom 3 game server and Apple Darwin Streaming Server- for their experimental evaluation. Their results revealed that the jitter and the throughput seen by a latency-sensitive application can indeed degrade due to background load from other virtual machines. [36] found out an optimal file size for text analysis application on Amazon EC2 and then reshaped the input data by merging files in order to match as closely as possible the desired file size. [43] presented mathematical models to demonstrate that to achieve optimal performance in a heterogeneous cloud infrastructure, the response time of the slowest node should be no more than three times that of the fastest node.

2.3.3 Mobile Clients for Cloud

For mobile clients for cloud data center, the energy consumption and mobile offloading are hot topics of discussion. [30] provided an analysis of the critical factors affecting the energy consumption of mobile clients in cloud computing and also carried out measurements about the central characteristics of mobile handheld devices that define the basic balance between local and remote computing.

[8] presented the design and implementation of an Android/Amazon EC2-based mobile application outsourcing platform, leveraging the cloud for scalability, elasticity, and multi-user code/data sharing. They empirically demonstrated that the cloud is not only feasible but desirable as an offloading platform for latency-tolerant applications despite wide-area latencies.[39] surveyed and discussed various remarkable techniques toward green mobile networks, to mainly targeting mobile cellular networks.[21] suggested that not all applications are energy efficient when migrated to the cloud. Mobile cloud

computing services would be significantly different from cloud services for desktops because they must offer energy savings.In [22], authors had analyzed the trade-off between local and remote processing from the viewpoint of energy use, taking into account the energy used for transferring the task from the local system to a remote service. [9] demonstrated that mobile applications can be enhanced with REST based cloud computing technologies to create applications that exceed the capabilities of traditional mobile devices. Recent studies in [20] and [19] proposed ThinkAir, a framework that makes it simple for developers to migrate their smartphone applications to the cloud. They evaluated the execution time and energy consumption using a N-queens puzzle application, a face detection and a virus scan application. They also showed that a parallelizable application can invoke multiple VMs to execute in the cloud in a seamless and on-demand manner such as to achieve greater reduction on execution time and energy consumption. [2] presented a measurement study of the energy consumption of three mobile networking technologies: 3G, GSM and WiFi. They developed a protocol TailEnder which reduces energy consumption through scheduling transfers with user-specified deadlines. [33] demonstrated the network characteristics of the two most popular video streaming strategies Youtube and Netflix varying with the type of application and the containers. [18] found out the the impact of burst length and peak transmission rate on packet loss and delay as well as potential energy savings.

Meantime, many systems are developed to minimize the energy consumptions. [11] proposed a system named Catnap which reduces energy consumption of mobile devices by allowing them to sleep during data transfers. It combined small gaps between packets into meaningful sleep intervals, thereby saving energy consumption on transmission. [27] focused network contention and WiFi density among different APs. It designed a system Sleepwell that achieves energy efficiency by evading network contention. Xiao et al. [42] proposed CasCap, a novel cloud-assisted context-aware power management for mobile devices. It achieved energy savings by offloading to the cloud. Stratus in [1] leveraged cloud resources to make data communication on smart phones more efficient. It employed three mechanisms, namely aggregation, compression and opportunistic scheduling to reduce energy consumptions. [35] designed a power monitor, BattOr, to conduct power measurements on both mobile and stationary status. [16] presented a proxy-based solution that shapes an audio stream into bursts before relaying the traffic to the mobile device. [26] proposed a traffic scheduler that shapes the packets into consistent bursts based on per-packet performance constraints in order to reduce the overall transmission cost.

With respect to CDN, [40] proposed a Mobile Streaming Media Content Delivery Network (MSM-CDN), which is a network overlay consisting of overlay servers on top of the existing network. They presented an overview of the MSM-CDN system architecture, and described the testbed prototype that they built based on these architectural principles. In [5],they studied the problem of minimizing the retrieval latency considering both caching and retrieval capacity of the edge nodes and server simultaneously. They derived analytical models to evaluate the content retrieval latency under two source selection strategies, i.e., Random and Shortest-Queue, and three caching policies: selfish, collective, and adaptive caching policy.

2.4 Benchmark Survey

It is essential to understand the features and properties of various benchmarks during the process of performance evaluation. We give a comprehensive survey on the current benchmarks. However, we only focus on the open-source benchmarks for Linux operating system.

2.4.1 Benchmark Classification

Benchmarks can be classified by two different ways. The first classification is the most popular way to categorize benchmarks into three groups: Quick-hit, Synthetic and Application benchmark. Quick-hit are simple tests to measure a certain aspect of performance, but usually do not give a larger perceptive of system performance. It is useful in the cases where only one aspect needs to be assessed. For example, it is useful to identify the network problems or damaged hardware by using Quick-hit benchmarks. Synthetic benchmarks are usually more extensive tests that impose the workload on the component, which is useful for measuring the maximum capacity or throughput for a given component (e.g, CPU, network, disk, etc.). However, they do not give an evaluation on the system-level. Application benchmarks represent a real-world workload which tests systems in a simulated environment. Thus application benchmarks usually give a much better measure of real-world performance on a given system. Compared to application benchmarks, quick-hit and synthetic benchmarks could be regarded as Microbenchmarks.

The other classification is divided by the computer components including both hardware and software. Hardware-specific benchmarks include CPU, memory, disk and network I/O while software-specific contain web applica-

tion, in-memory database, blog, database and Parallel, to name but a few. In the following section, we will introduce the useful benchmarks classified by this way.

2.4.2 Benchmark Overview

Benchmarks we collected during the literature study are categorized in 7 classes.

1) CPU

UnixBench [34] provides a basic indicator of the performance of a Unix-like system. It consists of multiple tests that are targeted to various aspects of the system performance. If the system has multiple CPUs, the default behaviour is to run the selected tests twice – once with one copy of each test program running at a time, and once with N copies, where N is the number of CPUs. The selected tests include Dhrystone, Whetstone, Execl throughput, file copy, pipe throughput, shell scripts and system call overhead.

Crafty is a popular open-source chess engine that can be used to benchmark CPU speed. It analyzes pre-determined chess games positions and calculates the number of "nodes" (moves) per second (NPS) till certain "depth" is reached and displays the total NPS as well as the average NPS.

Openssl measures the CPU performance with RSA 4096-bit encryption algorithm of OpenSSL.

C-Ray is a simple raytracer designed to test the floating-point CPU performance.

Cpuid is a processor information retrieving tool which is written in the x86 assembly language completely. The processor information (i.e.,vendor, family, model and stepping data) is reported by *cpuid* instruction.

2) Memory

CacheBench is designed to test the memory and cache bandwidth performance.

Stream tests the system memory (RAM) performance including add, copy, scale and triad operation.

RAMspeed is a cache and memory benchmark which has different version for both uniprocessor and multiprocessor machines. It evaluates four aspects: Integer, Float, MMX and SSE. Each of them includes four subtests called copy, scale, add and triad.

Lmbench [3] is a suite of simple, portable benchmarks for bandwidth and latency. The bandwidth results include cached file read, memory copy/read-/write, pipe and TCP while latency results include context switching, network connections (e.g, TCP, UDP, RPC) establishment, process creation, signal handling, system call overhead and memory read latency.

3) Disk

Bonnie++ [34] is based on the Bonnie hard drive benchmark by Tim Bray. It is designed for testing hard disks and file system performances in sequential read/write, random seeks and CPU usage.

Dbench is a tool to generate I/O workloads to either a file system or to a networked CIFS or NFS server. It helps users to find out how many concurrent clients/applications performing disk I/O workload the server can handle.

IOzone is a filesystem benchmark tool. The benchmark generates and measures a variety of file operations, which provides a broad filesystem analysis of a computer.

dd is a Linux utility which can be also used to benchmark the disk throughput in sequential read/write and random read/write.

4) Network

Iperf [34] is designed for measuring maximum TCP and UDP bandwidth performance. Iperf allows the tuning of various parameters and UDP characteristics. Iperf reports bandwidth, delay jitter, datagram loss.

Netperf is a network performance benchmark that can be used to measure the performance of many different types of networking including TCP/UDP stream. It provides tests for both unidirectional throughput, and end-to-end latency.

Badabing[39] is the state-of-the-art loss rate estimation tool. Packet loss estimation is considered challenging because packet loss typically occurs rarely and lasts for very short time. Badabing use active probes and statistical estimations to measure the packet loss properties.

Ping [39] is to measure the round-trip time for messages sent from the originating host to a destination computer.

5) Application level

Sysbench is a modular, cross-platform and multi-threaded benchmark tool for evaluating OS parameters that are important for a system running a database under intensive load. The test options includes file I/O, scheduler, memory allocation and transfer speed, POSIX threads and database server

performance.

Redis is an in-memory key-value store. It includes the redis-benchmark utility that simulates SETs, GETs, INCRs, LPUSHs, LPOPs, PINGs, LRANGEs done by N clients at the same time sending M total queries (it is similar to the Apache's ab utility).

Osdb is an open source database benchmark for measuring performance of a database system.

BlogBench simulates the workload of a real-world blog by creating blogs with content and pictures, modifying blog posts, adding comments to these blogs, and then reading the content of the blogs. It will give final scores on the disk write and read.

Httperf [7] is a tool for measuring web server performance. It provides a flexible facility for generating various HTTP workloads and for measuring server performance.

ab is a tool for benchmarking the Apache HTTP server. It can find out that how many requests per second Apache server is capable of serving.

QStat [4] is a command-line utility for collecting real-time statistics from on-line game servers. The games supported are generally limited to the first-person-shooter genre (Quake, Half-Life, Unreal,etc).

6) Xen virtualization

XenMon [15] is a performance monitoring tool built for Xen, which reports a variety of metrics (e.g. CPU usage, block time and waiting time) that are accumulated over the execution period.

Xenoprof [29] is a system-wide profiler for Xen virtual machine environments, capable of profiling the Xen virtual machine monitor, multiple Linux guest operating systems, and applications running on them.

Xentrace [15] is a light weight event generation facility present in Xen. Using xentrace it is possible to raise events at arbitrary control points in the hypervisor.

7) Other Tools

Stress is a deliberately simple workload generator for POSIX systems. It imposes a configurable amount of CPU, memory, I/O, and disk stress on the system.

iostat reports instantaneous CPU statistics and input/output statistics for devices, partitions and network filesystems.

sar collects, reports and saves system activity information (CPU, memory,

disks, interrupts, network interfaces, TTY, kernel tables,etc.)

top shows how much processing power and memory are being used, as well as other information about the running processes.

dstat allows users to view all of your system resources instantly, compare disk usage in combination with interrupts from IDE controller, or compare the network bandwidth numbers directly with the disk throughput

vmstat reports information about processes, memory, paging, block IO, traps, and CPU activity.

tc,[4] shows/manipulates traffic control settings.

Chapter 3

Methodology

3.1 Environment Setup

3.1.1 Amazon EC2

In Amazon EC2, our measurement experiments are mainly based on standard small and large instances. We take the **m1.large** instance as the example to evaluate the performance because this instance has a relatively large amount of memory and can be used in various general applications. However, we also carry out some parts of experiments on high-cpu medium and xlarge instances to support our evaluation. All the instances selected are in us-east-1c zone of US East (Virginia) region. To eliminate the differentiation that might originate from different OS, we launch the instances from our customized Amazon EC2 Machine Image (AMI) with Linux kernel 2.6.18, Cent OS 5.6 distribution. An AMI is a special type of virtual appliance which is used to instantiate (create) a virtual machine in Amazon EC2. We create two AMIs for both Cent OS 5.6 32-bit and 64-bit.

3.1.2 Local Cloud

To explore the internal mechanisms of Amazon EC2, we have established local platforms based on Xen hypervisor to emulate the characteristics of the Amazon instances. In local data center, we use two servers to set up different versions of Xen hypervisor and the associated VMs. The hardware of the two servers has the same configurations: HP ProLiant BL280c G6 Server Blade with one Quad-Core Intel Xeon E5640 2667MHz processor, 8 GB DDR3 1333

MHz memory, one HPNC362i dual port Gigabit server adapter. The virtualization hypervisors for the two platforms are: Xen 3.4.3 (based on SEDF scheduler), and Xen 4.1.1 (Credit scheduler). The Linux kernel for Dom0 is CentOS 5.6 distribution. Various numbers of VMs with different number of virtual CPUs are running atop the two platforms to emulate different instance types and different neighboring interference environments.

3.2 Measurement Metrics and Tools

3.2.1 CPU Performance

In CPU performance measurement, we firstly obtain the hardware information of CPU with *cpuid* command, a non-trapping instruction that can be used in user mode without triggering trap to the underlying processor. Thus, the hypervisor does not capture the instruction and return modified results. Furthermore, we run *cat /proc/cpuinfo* command to verify the results from *cpuid*.

Then, *UnixBench* is adopted to run multiple tests to measure a wide range of the system performance, primarily the CPU performance. The test results are compared to the baseline system to produce an index value. The entire set of index values are then combined to make a composite index for the system which helps us evaluate the overall performance of CPU.

Finally, we develop a CPU micro-benchmark, which we name as *CPUben*, to measure the CPU utilization rate of one VM. The CPUben measures the elapsed time by making gettimeofday() system call continuously for one million times, the same approach as was employed in [38]. In a regular call, the approach has a resolution in the order of magnitude of one microsecond. On contemporary server processors, one such system call usually costs less than one microsecond, which means several gettimeofday() calls can be completed within one microsecond. In some systems, the efficiency of the system call can be enhanced by utilizing certain optimizations, e.g. using shared library to allow applications in user space to perform such calls with less overhead than a system call. However, as the primary purpose of the CPU utilization rate measurements is to capture the CPU activity as precise as possible, and the gettimeofday() process is to a large extent the only process running in the VM, the overhead is not a critical issue. Thus, no speedup for gettimeofday() system call is used in this paper. When one VM is scheduled off, the gap in the acquired system time is, in most circumstances, in the order of

milliseconds, significantly larger than the time spent on a normal system call. When the gap is larger than one millisecond, we consider that a scheduling occurs. By analyzing the CPU running time and off time intervals, we can figure out the CPU utilization rate of one VM, and further induce the VM scheduling mechanisms of the underlying VMM, at least for computationally intensive applications. In VMs with multiple virtual CPUs (VCPUs), multiple processes running gettimeofday() system calls are introduced to make full use of the CPU capability where appropriate.

3.2.2 Memory Performance

We use Redis from the application level to test the memory performance. Redis is an in-memory key-value store that has the benchmark utility to simulate multiple concurrent clients to send requests at the same time. In our measurement, we measure the PUT and GET operations corresponding to the write and read data in NoSQL in-memory databases.

3.2.3 Disk Performance

Throughput of disk performance is measured by Dbench which could generate I/O workloads with different clients. We compare the throughput variation with different clients running simultaneously.

3.2.4 Network Throughput

To investigate the impact of different CPU utilization rates on the performance of network traffic, we develop two fine-grained micro-benchmarks to measure the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) throughput. They are referred to as *TCPben* and *UDPben*, respectively, in this paper. Each micro-benchmark has a client and a server component. The server side of TCPben calculates the TCP throughput upon receiving per 256 KB data, whilst the UDPben calculates the throughput upon receiving per 128 KB data. The same approach was adopted in [38]. For TCP throughput, we deploy TCPServer and TCPClient program on two instances which are launched in the us-east-1c zone. TCP client sends data from memory to the server while TCP Server receives data and calculate the approximate throughput at the application level. The data is read from a file with a size of 828MB and the whole transmission lasts approximately 10 seconds. Both sending and receiving buffer sizes are set to 256KB. Based on

our experience, most RTTs between instances are less than 0.5ms and therefore if the bandwidth allowed the TCP throughput is at least 4Gbps. As for UDP throughput, we deploy the UDPServer and UDPClient which are similar to the TCP server and client. However, from the Xen virtualization point of view, all the packets are sent to Dom0 and then to the destination IPs. The communication between Dom0 and DomUs is simply copying data from memory to memory. If the packets are sent as fast as possible, it will burst data at very high rate to the Dom0. A lot of packets will loss when Dom0 cannot send them out in time. Thus, in our TCPben and UDPben, we need to control the sending rate by adding small idle intervals between every 256KB data to avoid the network congestion.

3.2.5 Network Latency

We measure the latency to the instance under test using ping utility. To capture the variations of the latency (also referred to as the latency jitter) during scheduled-on and scheduled-off time periods, which are usually in the order of tens of milliseconds from our experience, we deliver the ping requests at a rate of 1000 requests/second.

To evaluate the network latency from the application level, we set up Counter Strike 1.6 game server, a first-person-shooter(FPS) game, in Amazon. We measure the latency using *qstat* from both inside and outside the Amazon. The latency measured by *qstat* is more useful and reliable than that tested by ping.

3.2.6 Web Server Performance

We measure the Web server performance of various instances using *httperf* benchmark tool. Apache Web server is selected and evaluated in terms of static and dynamic requests. For static measurement, we send a http request to get a file from the Web server. Different file sizes are used, including 30KB, 50KB, 70KB, and 100 KB. The response time means the time between sending the first byte of the request and receiving the first byte of the reply. 95% confidence level is used where appropriate.

Dynamic HTTP request is used to make the processor busy. Dynamic request means after receiving a request from a client, the Web server performs a mathematical summation from 1 through 100, and then returns the result to the client. Thus, the dynamic Web test is CPU-bound rather than network-bound. To try to avoid potential bottleneck from client machine, we use a

high-CPU medium instance from the same zone acting as the client.

3.2.7 Energy Efficiency of Mobile Clients

To examine the mobile clients connecting to the cloud data center, we use Samsung Google Nexus S as the testing mobile client to request a test video in Apache Web server in Amazon EC2. The mobile phone access Internet through Wi-Fi network. *Realplay* player[1] for Android platform is adopted to play the test video. The model of Wi-Fi AP is wrt 160nl, Linksys Cisco. During the test, the mobile phone is the only device connected to the Wi-Fi AP, to avoid contention and interference within the same AP. Tcpdump[2] is used to capture the streaming packets, and Monsoon Power Monitor[3] is used to measure the energy consumption of the mobile phone. The goal of the test is to exploit traffic scheduling mechanisms to save the energy for mobile clients.

[1] `https://play.google.com/store/apps/details?id=com.real.RealPlayer&hl=en`

[2] `http://www.tcpdump.org/`

[3] `http://www.msoon.com/LabEquipment/PowerMonitor/`

Chapter 4

Evaluation

4.1 Amazon EC2 API Tools

Amazon EC2 allows users to create and terminate an instance, manipulate the security group through API tools. After registration, users could immediately obtain the security credentials including AWS account ID, Access Keys, X.509 Certificates, and Key Pairs. Firstly, Amazon EC2 users need to download and extract the Amazon AMI tools[1] into the a directory, i.e, /var/ec2tools/. Then, users need to download the security files and configure proper path to EC2 resource in the *.bashrc* file.

```
1 export EC2_HOME= /var/ec2tools/
2 export EC2_PRIVATE_KEY=~/.ec2/pk-yourprivatekey.pem
3 export EC2_CERT=~/.ec2/cert-yourcert.pem
```

After enable *.bashrc* settings, users could use the EC2 commands to interact with EC2 cloud data center. The API functions include creating AMI, getting the information about instances and available zones, creating security keypair and configuring the route tables, to name but a few. For example, the following commands control the whole life cycle of an instance.

- *ec2-run-instances*: launch an instance which means that the first time users create a instance with specifications such as OS, AMI, available zones, security group and access key pairs.

- *ec2-stop-instances*: stop an instance which means shutting down the instance. When the instance is stopped, the data will not get lost and Amazon

[1] `http://aws.amazon.com/developertools/351`

does not charge anything.

- *ec2-start-instances*: start an instance which means resuming instances from stop status. All the data will come back and Amazon starts to charge.

- *ec2-reboot-instances*: reboot an instance is similar to that we restart a computer.

- *ec2-terminate-instances*: terminate an instance which means destroying it and all the data will lost.

To connect to the instance, Amazon offers SSH connection for Linux and remote desktop for Windows instance. Recently, Amazon releases an new way to connect Linux instances through browser using the Java SSH Client.

4.2 CPU Performance

4.2.1 Detect CPU Type

The immediate way to detect the CPU type is to *cat /proc/cpuinfo*. The */proc* file system provides an interface to access kernel data structures through common file operations such as *cat* and *more*, which is often considered to be a *pseudo-filesystem*. */proc/cpuinfo* contains information about the CPU including vendor id, model name, CPU frequency. But there is a possibility that the Amazon EC2 might modify the information getting from the psedo-filesystem. Thus, we further use *cpuid* instruction, which is a non-trapping instruction that can be used in user mode without triggering trap to the underlying processor. In the Xen paravirtualized hypervisor (what Amazon uses), it means that the hypervisor would not be able to intercept the instruction, and change the result that it returns. Therefore, the output from *cpuid* is the real output from the physical CPU. The *cpuid* not only provides the processor signature, but also provides the information about the features supported by processors. For each instance, we use *cpuid* to get the reliable CPU type. Meanwhile, we also record the frequency of each CPU type. The results are shown in the table 4.1. To save space, we omit the vendor from the CPU model purposely, e.g. Intel Xeon E5430 is shorthanded as E5430, AMD Opteron 2218 HE is shorthanded as 2218 HE. As a summary, Intel Xeon processors are started with characters, whilst AMD Opteron processors are started with digits.

To make the samples representative for analysis, we collected CPU information for different instances from 2nd, April, 2011 until 25th, July, 2011.

Table 4.1: CPU type in different instances

Instance Type	CPU Model	Clock Speed(GHz)	Percentage(%)
m1.small	E5430	2.66	38
	E5507	2.26	12
	E5645	2.0	30
	2218HE	2.6	20
m1.large	E5430	2.66	17
	E5507	2.26	40
	E5645	2.0	42
	2218HE	2.6	1
m1.xlarge	E5430	2.66	46
	E5507	2.26	6
	E5645	2.4	48
t1.micro	E5430	2.66	95
	2218HE	2.6	5
m2.xlarge	X5550	2.66	100
m2.2xlarge	X5550	2.66	100
m2.4xlarge	X5550	2.66	100
c1.medium	E5410	2.33	82
	E5345	2.33	10
	E5506	2.13	8
c1.xlarge	E5410	2.33	19
	E5345	2.33	3
	E5506	2.13	72
	E5430	2.66	6
cc1.4xlarge	X5570	2.93	N/A
cg1.4xlarge	X5570	2.93	N/A

To minimize the bias from different usage peaks on the hardware availability, samples were collected from varied time slots. Meanwhile, to eliminate the impact of different availability zones on the results, the samples were collected in all three functional availability zones within US East (Virginia) region. Still another availability zone was listed in the same region; however, we came across some error message which stated that the zone was not supported anymore in the region. The number of samples for most of the instance types is 100, except the high-memory instance category, which is 50. The slightly smaller number of samples for the high-memory instance category (including m2.xlarge, m2.2xlarge, and m2.4xlarge) is because just one type of hardware configuration, i.e. Intel Xeon X5550, was found during the

whole evaluating process. Both cluster compute and cluster GPU instances are too expensive to calculate the frequency.

It is clearly that diversified hardware are also widely used in standard (m1.small, m1.large and m1.xlarge) and high-CPU instance family (c1.medium and c1.xlarge). Furthermore, we notice that the probability of a specific type of processor, e.g. E5645, significantly varies in different availability zones. In one availability zone, we can acquire 95% of instances hosted by E5645 machines,whilst in another zone, the probability of E5645 instances is as low as 10%. We conjecture that the availability zone with 95% of E5645 machines is a newly built data center within the US east region.

The interesting question to ask is whether the heterogeneous hardware configuration within the same instance type leads to diversified performance. In the following tests, we further divide m1.large instance into four sub-types by CPU model, namely E5430, E5507, E5645 and AMD. However, it is difficult to find enough AMD instances since the probability of AMD instances is only about 1%. Hence, we only focus on Intel sub-type of *m1.large* instances in Amazon EC2. The reason why we select the *m1.large* instance as main example to evaluate the performance is because this instance has a relatively large amount of memory and can be used in various general applications.

4.2.2 CPU Performance

To evaluate the overall performance of CPU, we use Unixbench which provides a basic indicator of the performance. It consists of a wide range of tests including string handling, floating point operation, execl calls throughput, file copy, pipe throughput, pipe-based context switching, process creation, shell scripts and system call overhead. If the system has more than one CPU, it will run the benchmark twice – once with one copy of each test program running at a time, and once with N copies, where N is the number of CPUs. Based on these benchmark results, a final score will be given to reflect the overall performance of CPU.

We run the Unixbench in different sub-type of m1.large instance in ten rounds. For each round, we launch two instances for each sub-type and each instance runs the Unixbench for three times to get three CPU performance scores. We only calculate the average value of three scores as a final score.

The results are shown in the figure 4.1. It is clearly that E5507 instance is of the poorest performance in both single and double process whilst E5645 remains the best performance. It is interesting to see that the performance of

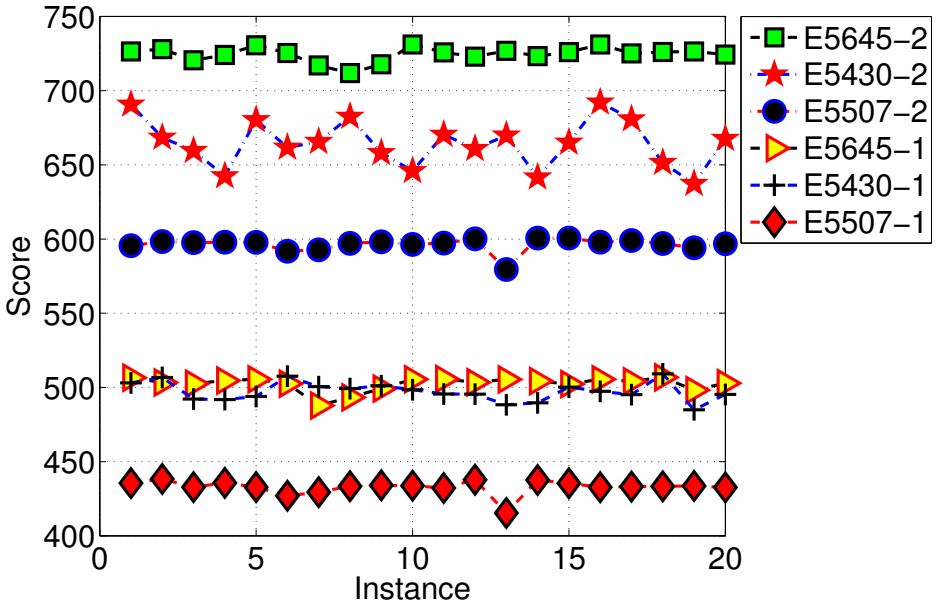

Figure 4.1: CPU performance scores for the *m1.large* instance with different cpu type

E5430 double process fluctuates greatly. If one process is running, E5430 and E5645 are comparable in terms of performance, whilst they are approximately 1.15 times of the performance of E5507. When two processes are running, E5645 outperforms E5430, whilst E5430 further outperforms E5507. The performance variation in times is 1.21, and 1.1 times for E5645, and E5430, respectively, wherein E5507 is taken as the baseline. It is well known that clock speed is one of the important factor to evaluate the CPU rate. The higher the clock speed is, the faster CPU runs. Through the table 4.1, we could find that E5430 has the fastest clock speed, following by E5507 and E5645. It is evident the results from the Unixbench are not closely related with CPU clock speed. We may conjecture that CPU performance of E5430 is degraded because of virtualization and there may be more than one instance sharing one physical CPU. However, through the measurements, we could reach the two important conclusions:

1. Virtualization has an impact on the CPU performance and there exists processor sharing in Amazon EC2.

2. CPU performance variation among the same sub-type of instances is small whilst the differences between different sub-types are significant.

4.2.3 CPU Process Sharing

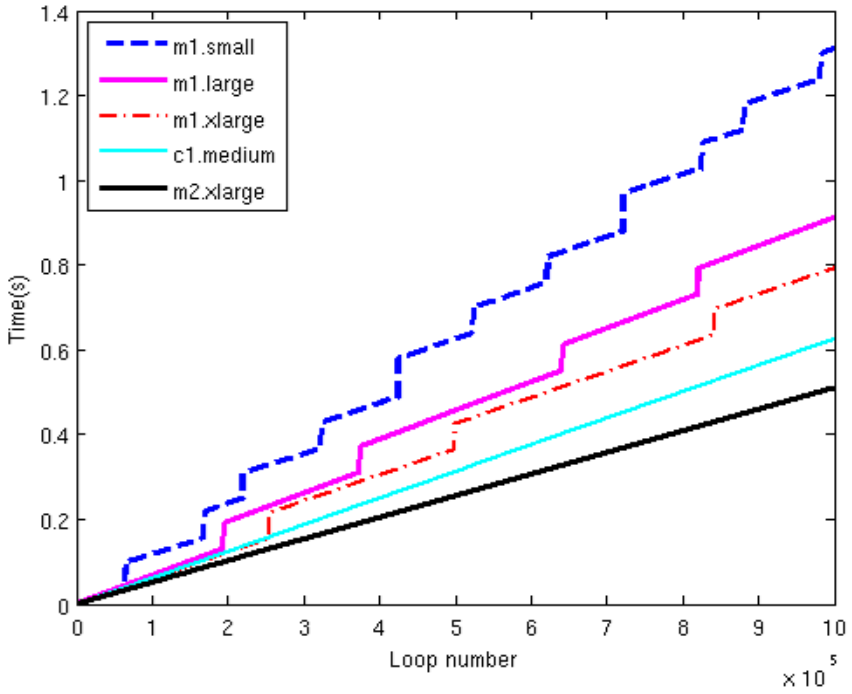

Figure 4.2: CPU process sharing

To further explore the process sharing in Amazon EC2, we develop a CPU micro-benchmark to measure the CPU utilization rate of VMs. The source code is in Appendices A.1 named CPUben. The CPUben collects the elapsed time by making *gettimeofday()* system calls continuously for one million times, and the same approach as was employed in [38]. The core of the CPUben is a loop which repeatedly calls *gettimeofday()* function to get the current system time. To avoid the effects of I/O operations, all the timestamps are saved in an array which has the same length of loops. The memory of the array is allocated by calling malloc() function before the loops start. When the loops finish, the timestamps in the array will be written to a file stored on disk. We run the CPUben in multiple instance at different time. The results are shown in the figure 4.2.

We illustrate the CPU utilization rates of various instance types, covering standard, high-CPU, and high-memory categories in figure 4.2. From the figure 4.2, when the CPUben run on the *m2.xlarge* instances, the timestamps increase smoothly. This suggests that there is no remarkable interruption

while the instances execute the CPUben program. It is the same case for high-CPU (i.e. c1.medium and c1.xlarge) and high-memory (i.e. m2.xlarge, m2.2xlarge, and m2.4xlarge) instance with more ECUs as well as a non-virtualized machine, which always produces a straight line(i.e. 100% CPU utilization rate).

However, in the instances of standard family(i.e. m1.small, m1.large and m1.xlarge), it can be seen the vertical turning points periodically. The turning points mean that CPU does not execute the *gettimeofday()* function. There is an small interval about 0.1 second between two adjacent loops. In CPUben program, it gets the timestamps and put it into a pre-allocated memory. After getting all the timestamps, the program starts to write the timestamps to the file. There is no other operation such as I/O before the loop has finished and the CPUben is also the only user process running on the instance. Thus, the only appropriate reason of execution gap is virtual machine scheduling. Furthermore, we define these execution gaps as *process delay*. During the process delay, the instances wait for Xen Hypervisor to assign CPU time to them.

To conclude, in this section, three conclusions are reached.

1. In Amazon EC2, process sharing is a common phenomenon, particularly in the instances of standard family. Such process sharing leads to *process delay*, which may have an impact on the applications.

2. It can be seen that the more ECUs an instance has, the less process delay an instance suffers. The instance with less ECUs has to contend for physical processor with other instances.

3. Amazon EC2 has their own CPU scheduling mechanisms to ensure the process fairness among different instances.

4.2.4 Amazon Scheduling Mechanisms

4.2.4.1 CPU Utilization

Based on the conclusions of previous section, the immediate follow-up questions come up : how does Amazon EC2 assign physical processor to the instance? What are the aggregate CPU utilization rates of these instances? Are there any patterns or regularities for the running time, waiting time etc? To answer these questions, we refine our CPUben to support multi-process and make a in-depth analysis on the timestamps traces. When one VM is

scheduled off, the gap in the acquired system time is, in most circumstances, in the order of milliseconds, significantly larger than the time spent on a normal system call. When the gap is larger than *one millisecond*, we consider that a scheduling occurs. By analyzing the CPU running time and off time intervals, we can infer the CPU utilization rate of one VM, and further induce the VM scheduling mechanisms of the underlying VMM, at least for computationally intensive applications. A program, written by C++, is used to process the timestamps and calculate the average CPU rate.

Table 4.2: CPU utilization rate

Instance Type	Sub-type	1-process (%)	2-process (#1, #2) (%)
m1.small	E5430-1	38.4	N/A
	E5430-2	41.1	N/A
	E5507	43.4	N/A
	2218HE	44.4	N/A
m1.large	E5430-1	75.7	(75.4, 74.9)
	E5430-2	99.4	(74.6, 74.8)
	E5507	72.5	(71.1, 71.9)
	E5645	99.9	(99.9, 99.6)
c1.medium	E5410	99.5	(95.6, 96.1)
m2.xlarge	X5550	99.9	(99.9, 99.6)

The results of CPU utilization rate are presented in table 4.2. For single virtual core *m1.small* instance, we run the single-process CPUben. For *m1.small* instance, it is clearly shown that E5430-1 obtain the least CPU time, only 38.4% while E5507 and 2218H have 43.4% and 44.4%, respectively. Although E5430 has the fastest CPU clock speed, it acquires the least CPU time. Therefore, it follows that Amazon assigns CPU time not only according to the instance type, but also depends on the different CPU type within same instance type, which try to bridge the performance gaps of different hardware. Meanwhile, during the experiments, we found a different E5430, identified with E5430-2, whose CPU utilization is obviously larger than other E5430-1 small instances by almost three percentage points. We speculate there maybe two kinds of different mechanisms employed in E5430 instances.

For *m1.large* instance, we run the CPUben in both single and double process modes. From table 4.2, we also acquire similar observations. Instances from the same E5430 processor present two diverse characteristics. On the one hand, E5430-1 can obtain around 75% CPU time when running a single-process CPUben, it acquires the same amount of CPU share, i.e. roughly

75%, for each virtual core when running two-process CPUben. On the other hand, E5430-2 is able to gain a CPU utilization rate close to 100% in one-process CPUben tests, then it decreases significantly to close to 75% on each core when running two-process CPUben tests. Meanwhile, it is interesting to find that E5645 could both close to 100% of CPU utilization, which means E5645 does not share CPU with other instances.

For high-cpu medium and high memory extra large instances, the CPU utilization of them, just as expected, are both close to 100 percent since they have more ECUs than large instance.

4.2.4.2 Scheduling Mechanisms

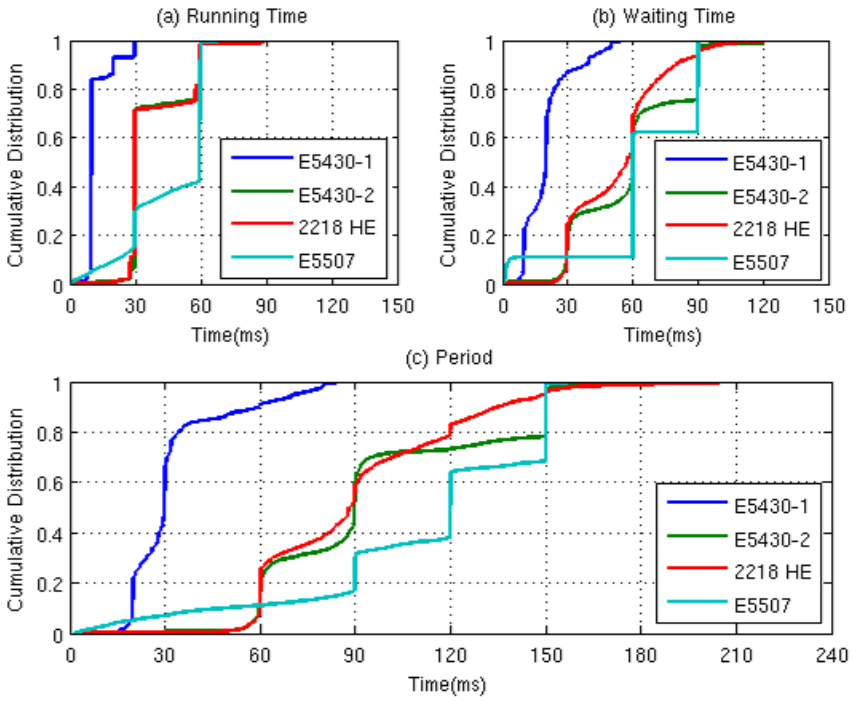

Figure 4.3: Running time, waiting time, and period of m1.small instances (single-process)

To further explore the scheduling mechanism, we analyze the timestamps in three angles, namely period, running and waiting time. Figure 4.3 clearly demonstrates that there are three slightly different subtypes within m1.small instance type, i.e. E5430-1, E5430-2 and 2218 HE, and E5507. E5430-2 functions similarly with 2218 HE. It is extremely interesting to notice that two

distinct subtypes exist with the same hardware configuration, i.e. E5430. To differentiate them, they are notated as E5430-1 and E5430-2 respectively. Another interesting interpretation from figure 4.3 is that different hardware can be configured to function similarly. E5430-2 and 2218 HE shows a good example of this phenomenon. Approximately 80% of the running time intervals of E5430-1 are 10ms (the other major intervals are 20ms and 30ms, accounting for 5% of time each), whilst roughly 60% of the running time intervals of E5430-2 and 2218 HE are 30ms (the other major interval is 60ms, accounting for 20% of time), and 60% of the running time intervals of E5507 are 60ms (the other major interval is 30ms, accounting for 20% of time).

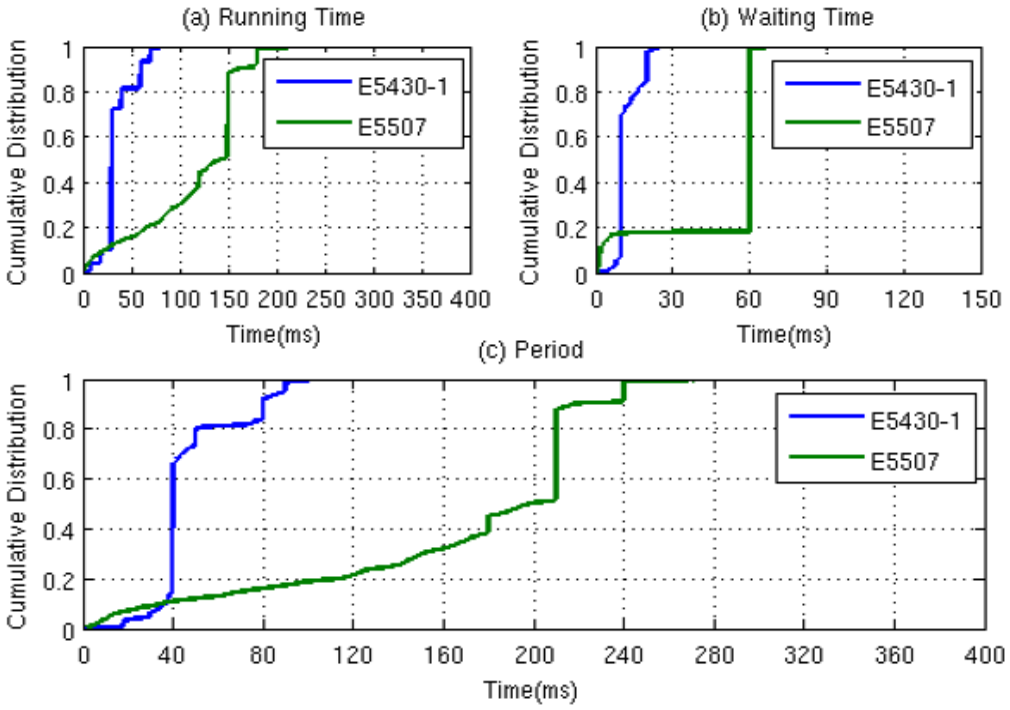

Figure 4.4: Running time, waiting time, and period of m1.large instances (single-process)

Figure 4.4 illustrates the CDF of running time, waiting time, and period of E5430-1, and E5507 in m1.large instance type (single-process). The curves from E5430-2 and E5645 are excluded from Figure 4.4 because they are straight lines, which can be indicated from their respective close to 100% CPU utilization for single-process measurements. E5645 functions dissimilarly from the rest of three m1.large instance subtypes, whilst acts similarly to m2.xlarge X5550. From the figure 4.4, we can see that compared to E5430,

E5507 has a long period, running and waiting time, which lead to bad performance for latency sensitive applications.

4.2.4.3 Local Emulation

Table 4.3: CPU utilization of emulated instances

Instance Type	Emulated instance scheduler and parameters	1-process (%)	2-process (#1, #2)(%)
m1.large	E5430-1 SEDF (30, 40, 0)	75.6	(74.0, 74.2)
	E5430-2 Credit (w=256, c=150)	99.6	(75.9, 76.0)
	E5507 SEDF (150, 210, 0)	73.6	(68.6, 70.6)
	E5645 Credit (w=256, c=200)	99.9	(99.9, 100)
c1.medium	E5410 Credit (w=512 c=190)	99.9	(92.0, 91.8)
m2.xlarge	X5550 Credit (w=512, c=200)	99.9	(99.7, 99.6)

To probe into the internal functioning mechanisms of these seemingly confusing appearances, we set up two platforms natively based on Xen hypervisor to emulate the CPU utilization rates of various instances. We tune the various parameters of the CPU schedulers, i.e. SEDF and Credit, to acquire desirable CPU utilization rates. For SEDF scheduler, the parameters tuned are slice, and period; whilst for Credit scheduler, the tuned parameters are weight, and cap.

The emulated results from local environments are shown in Table 4.3. We are able to emulate instances that are sufficiently analogous to *m1.large* instances from Amazon EC2 in local environments. SEDF scheduler can be used to emulate the instances that have the same CPU utilization rates for both single-process and dual-process CPUben tests. Credit scheduler can be used to emulate the instances that have different CPU utilization rates for single-process and dual-process CPUben test. The working principle is that SEDF scheduler allocates CPU resources per virtual core, whilst Credit scheduler allocates CPU resources per VM. It means that when the Credit scheduler sets a cap value 150 for a VM with two virtual cores, when the VM is running a single-process application, it can acquire maximum 100% of CPU share; when it is running a dual-process application, it can obtain maximum 75% for each of its virtual core. When using SEDF scheduler, if we set (75, 100, 0) for a VM with two virtual cores, each core can achieve maximum 75% of CPU time even if it is running a single-process program. SEDF scheduler is criticized for its poor performance in supporting Symmetric Multi-Processing (SMP) architecture, which is, on the contrary, the

strength of Credit scheduler. Furthermore, using the parameters listed in table 4.2, we are able to emulate similar running time, waiting time, and period as shown in figure 4.4. The aforementioned phenomena are not sufficient to prove that Amazon EC2 uses SEDF, or Credit scheduler to schedule its instances. However, based on the similarities observed from Amazon EC2 and local environments, it is safe to speculate that Amazon EC2 utilizes at least two different schedulers, one is SEDF-like, and the other one is Credit-like, to schedule its VM instances.

4.3 Memory Performance

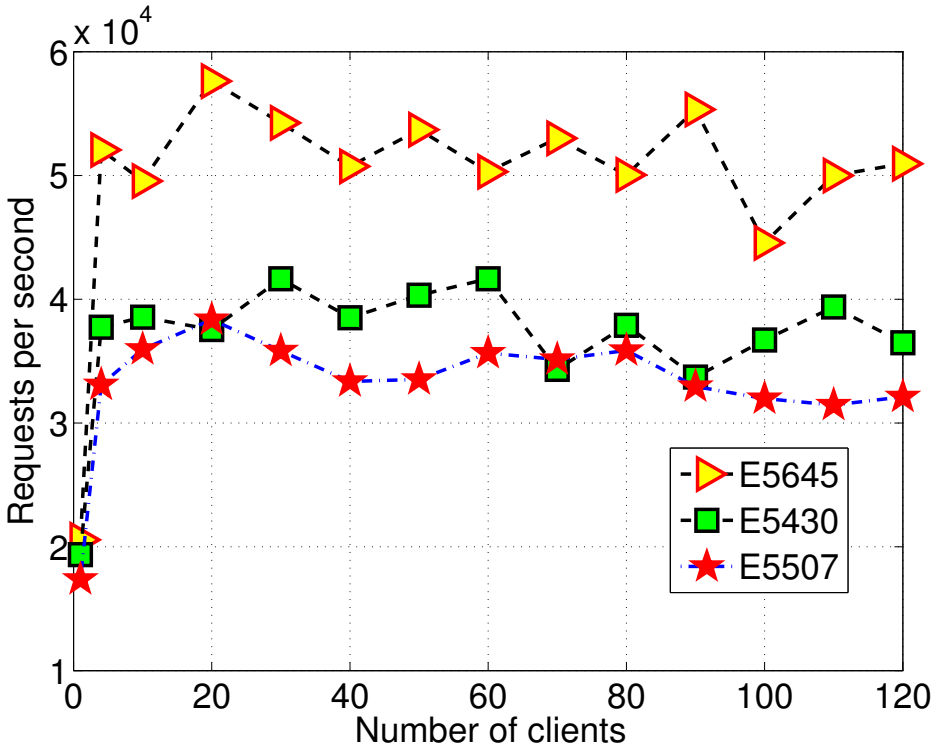

Figure 4.5: Redis GET operation

Redis is an in-memory key-value store that has the *redis-benchmark* utility to simulate multiple concurrent clients to send requests. For example, redis workload simulate SETs/GETs performed by N clients at the same time sending M total queries. In our measurements, we use random keys to perform 100,000 requests for SETs and GETs operations. We vary the number

of concurrent clients. The detailed results from GET operations are depicted in figure 4.5. The results from other operations are similar to GET operation. It is clear that the best performance of memory workload is E5645 while E5507 performs poorly.

To further investigate how the CPU process latency impact on memory performance, the handling latency of Get request is shown in the table 4.4. For E5645, all the requests are handled within 3ms, which lead to the best memory performance. We can observe the large latency as high as 60ms in E5507 while that of E5430 is less than 23ms. From the figure 4.4, we already know more than 50% amount of waiting time of E5430 and E5507 are 10ms and 60ms, respectively. Therefore, we can see the process delay lead to the handling latency of memory requests.

Table 4.4: Handling Latency of GET Request

Latency \ Instance	E5430	E5507	E5645
≤1ms	96.86%	99.27%	99.93%
≤3ms	97.83%	-	100%
≤10ms	98.41%	-	-
≤15ms	99.80%	-	-
≤23ms	100%	-	-
≤60ms	-	99.63%	-
≤61ms	-	100%	-

For the memory performance, we have two conclusions:

1. E5645 instances outperform E5430 and E5507 instances in memory operations. The memory performance of E5645 is 1.5 times of that of E5507, whilst E5430 is 1.14 times of E5507.

2. CPU process delay results in the handling latency of memory operation requests, which further have an impact on the throughput of memory requests.

4.4 Disk Performance

We use the dbench workload to take the reference numbers of disk I/O performance. This test was intended to show how different number of clients

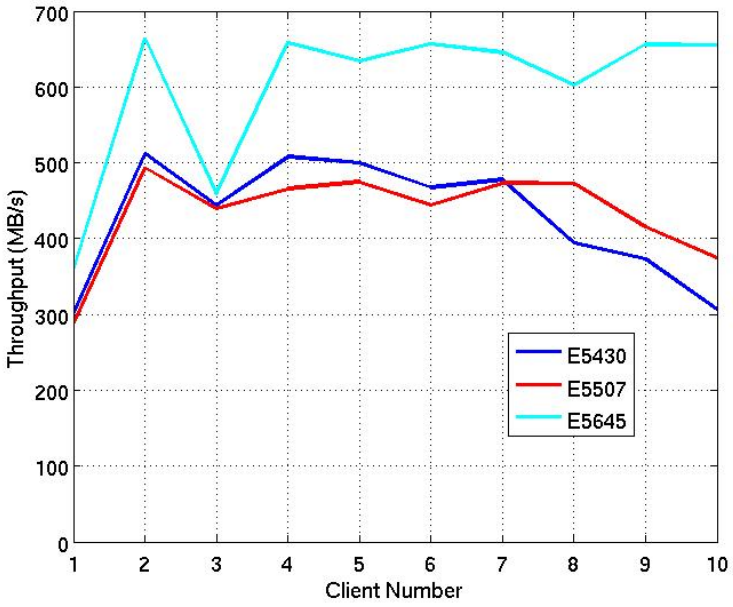

Figure 4.6: Dbench workload throughput

impact throughput. We varied the client numbers. The results are shown in the figure 4.6. The best performance for our dbench workload is seen with E5645, whose disk throughputs are always larger than others. E5430 and E5507 show a similar performance. Since all of three kinds of instances have two virtual CPUs(VCPUs), they can make full utilization of VCPUs when the number of clients is two, thereby achieving highest disk throughput. Meanwhile, it is interesting to see that the throughput of E5430 decreases dramatically when number of clients is greater than 7. All in all, we can safely achieve three conclusions for disk performance.

1. E5645 shows the best disk performance while that of other two instance are almost similar.

2. Along with the increase of concurrent clients, the disk performance of both E5430 and E5507 instances start to decline while E5645 remains the highest throughput.

3. For the peak throughput, the disk performance of E5645 is 1.4 times of that of E5507, whilst E5430 is 1.04 times of E5507.

4.5 Network Throughput

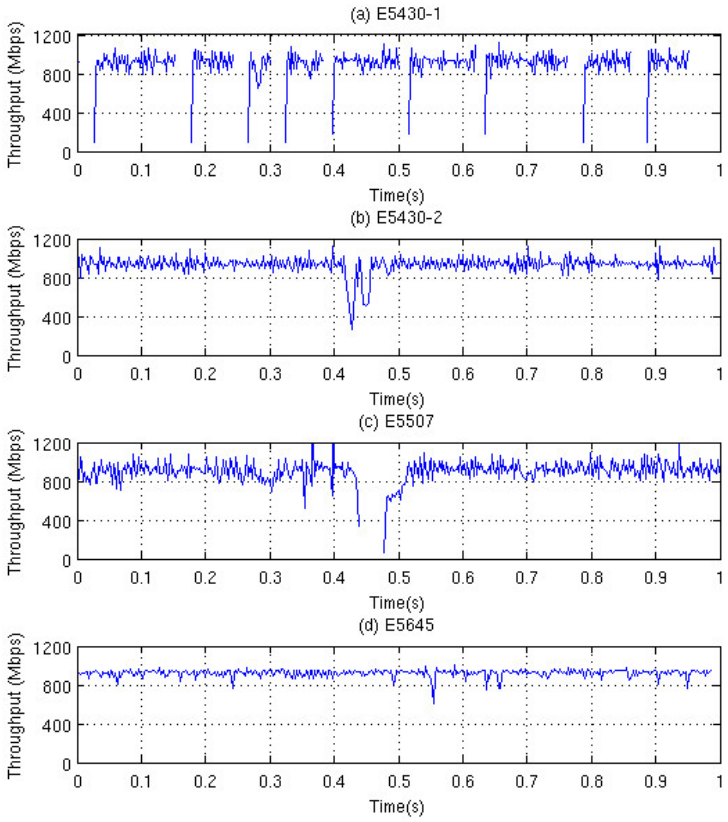

Figure 4.7: TCP throughput of m1.large instances

For network throughput measurement, we use TCPben and UDPben to measure the fine-grained TCP and UDP throughput. The client for the TCPben and UDPben measurements is located on an *m2.xlarge* instance within the same availability zone as the instances under test. The powerful client instance is chosen to minimize the impact from the client side, because *m2.xlarge* instance is shown to be able to obtain near 100% CPU utilization rate (cf. Table 4.2). The hop-count from the client to the instances under test is the same, i.e. four hops.

The measured TCP and UDP throughput is illustrated in figure 4.7 and figure 4.8 respectively. From these two figures, it is clear that E5645 shows the most stable throughput, whilst E5430-1 illustrates the least stable results, for

both TCP and UDP throughput. E5507 and E5430-2 behaves similarly, and their performance behavior is located in between E5645 and E5430-1. The difference between E5430-1 and E5507 is interesting, as they are both able to obtain roughly the same amount of CPU utilization rates. The difference originates from the distinct scheduling mechanisms they utilize, i.e. E5430-1 uses shorter intervals (30ms running time), and schedules more frequently (40 ms periods), whilst E5507 uses longer intervals (150 ms running time) and schedules less frequently (210 ms periods).

Furthermore, the most remarkable is the throughput pattern of E5430-1. Due to the frequent process sharing in E5430, the transmission will stop for a while (about 20ms) periodically. Thus, to some extent, the process delay reshapes the TCP and UDP traffic into the small bursty traffic, which might have a positive effect on the energy efficient of mobile clients. We will discuss this in Section 4.8.

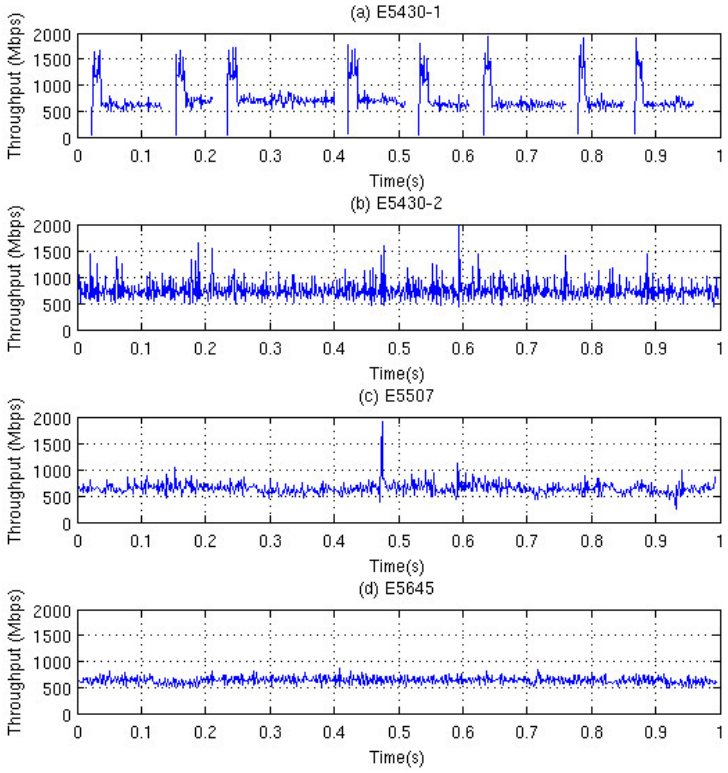

Figure 4.8: UDP throughput of m1.large instances

4.6 Network Latency

In this section, we discuss the network latency base on RTT. We use *ping* Linux utility to collect the RTTs. The same client instance as in network throughput experiments is used for the network latency measurements. The results are shown in figure 4.9. As expected, E5645 performs the best among the four instances. E5430-2 shows a clear benefit against E5430-1, which is primarily due to its close to 100% single-process CPU utilization. Basically

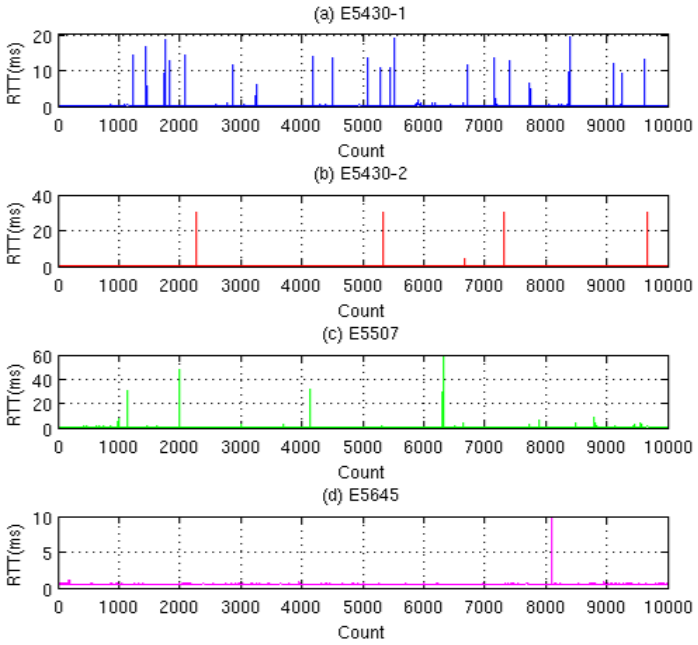

Figure 4.9: Network latency of m1.large instances

speaking, packet RTT delay is mainly caused by queuing delay on the routers. After receiving a data packet, the router will send it to a queue where all the data packets are waiting to be transferred. When the network gets busy, the speed of adding packets to the queue are much faster than that of removing packets from the queue, which cause network congestions and large delays. In Amazon EC2 Data center, we infer that the large delay among Amazon instances are caused by the long queuing delay at Xen Dom0. When the receiver VM instance is in process delay, the probe packets will be buffered at the Xen Dom0 until the receiver gets CPU time from hypervisor again. This long buffer queuing causes higher delay among large instances. For

E5430 and E5507, they are heavily sharing processors with other instances. Therefore, they suffer more network latency compared to E5645.

4.7 Web Server Performance

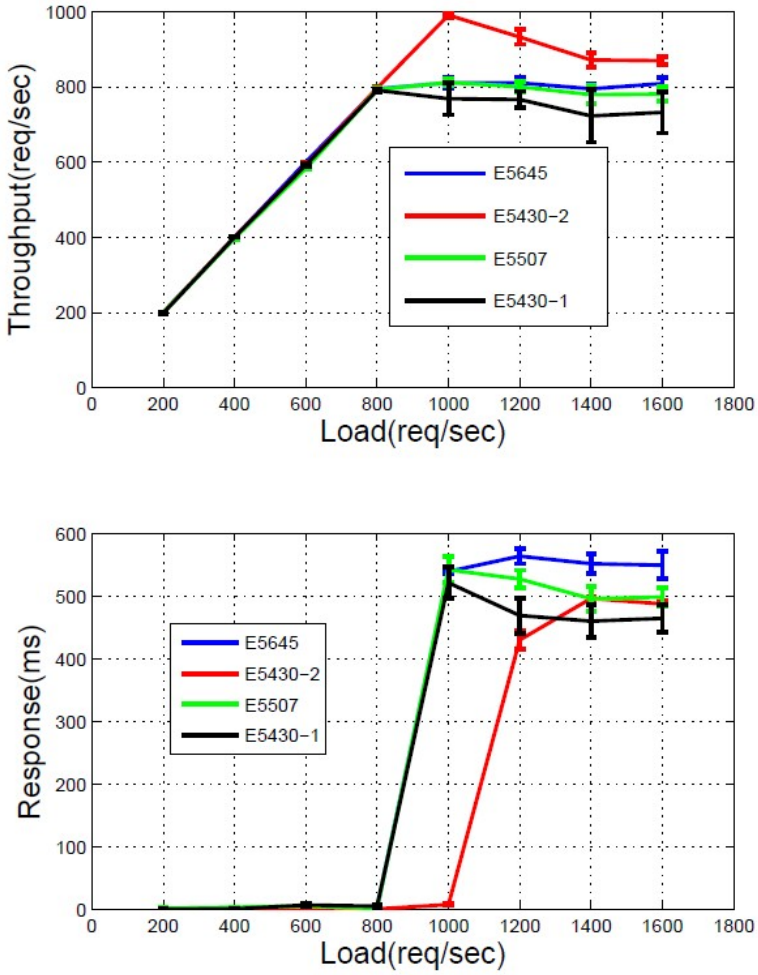

Figure 4.10: Httperf performance of static HTTP requests

4.7.1 Static HTTP Request

For static HTTP request, we use *httperf* to send request concurrently to retrieve different size of files. As they show similar trends, we only present

the throughput and response time of file size 100KB in figure 4.10. From the figure, we can easily observe E5430-2 has the best Web performance for static requests. The highest throughput is as much as 1000 req/s. It follows immediately that the network throughput is about 1000 req/s $\times$ 100KB/req = 100000KB/s = 100MB/s.

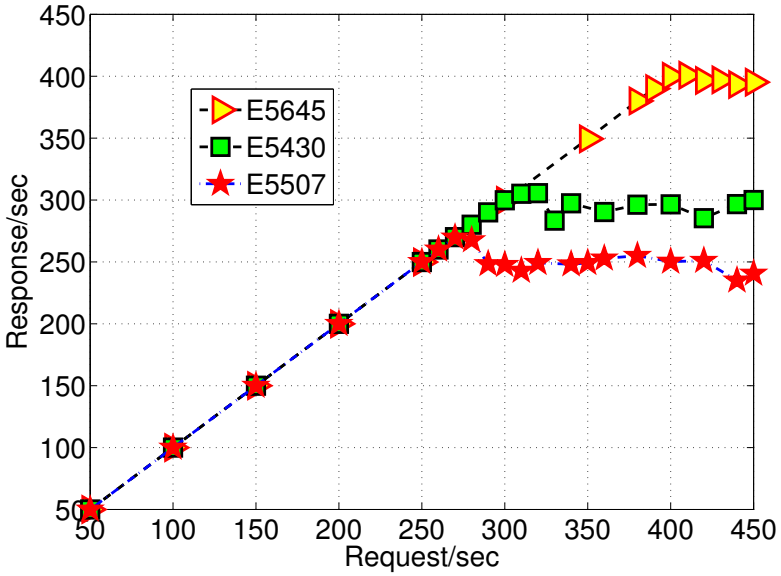

Figure 4.11: Httperf performance of dynamic HTTP requests

4.7.2 Dynamic HTTP Request

Dynamic HTTP request is used to make the processor busy. Dynamic request means after receiving a request from a client, the Web server performs a mathematical summation from 1 through 100, and then returns the result to the client. Thus, the dynamic Web test is CPU-bound rather than network-bound. To try to avoid potential bottleneck from client machine, we use a high-CPU medium instance from the same zone acting as the client. The Httperf throughput results are depicted in Fig.3. The figure demonstrates that the advantages from separate subsystems, e.g. CPU, memory and disk, are accumulated at application-level, where E5645 is 1.6 times as efficient as E5507 and E5430 is 1.2 times as E5507.

4.8 Energy Efficiency of Mobile Clients

4.8.1 Traffic Characteristic

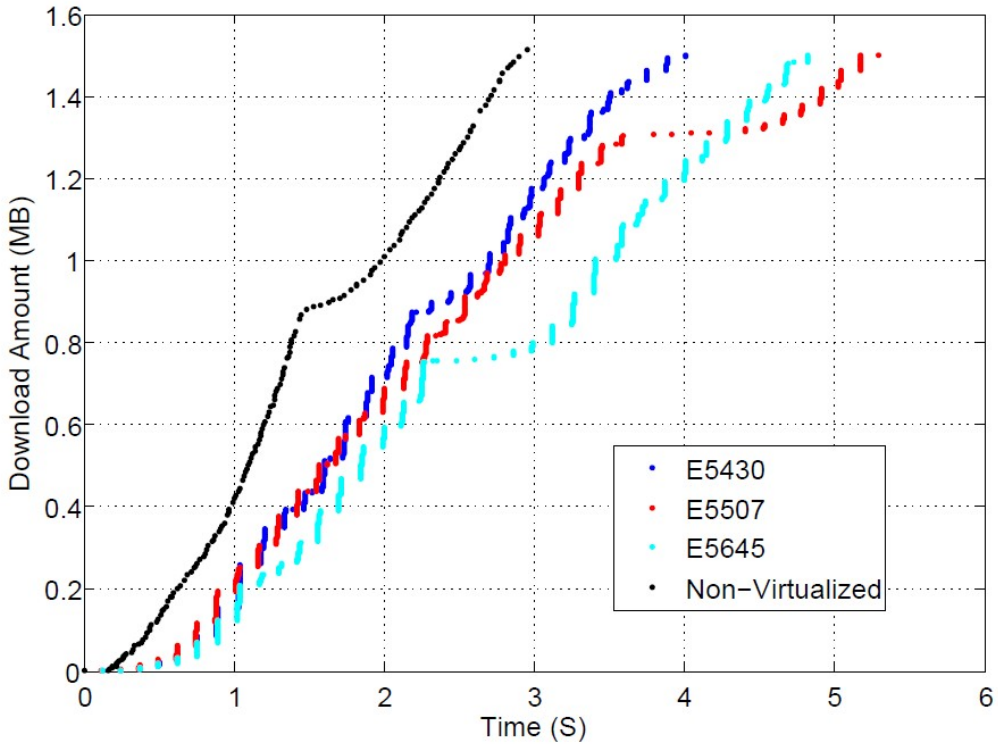

Figure 4.12: Traffic Shape

In the section 4.5, we find the process delay could reshape the TCP/UDP traffic and delay packets into consistent bursts, which may impact energy efficiency of mobile clients that are using Amazon EC2 services. It is well known that the basic power saving method for mobile clients is to reshape the traffic into bursts, which make it possible to put the wireless network interface(WNI) into the sleep mode. For example, IEEE 802.11 power saving mode(PSM) can reduce the power consumption by putting the WNI into sleep as much as possible. Thus, in our measurements, we may have the following questions: Does the process sharing in Virtualization improve the energy efficiency?

Thus, we use Samsung Google Nexus S as mobile clients to access a test video which are put in the Apache Web Server in m1.large instance. Tcpdump is

used to capture the packets sending from the Amazon to mobile clients. The results are shown in the figure 4.12. It is evident that the traffic become bursty not only in E5430 and E5507 but also in E5645 instances. Compared to them, the traffic from a non-virtualized machine is smooth.

From the figure 4.12, we can define two metrics to characterize the traffic. One is *burst size* which refers to the size of packets sending to mobile clients; the other one is *burst interval* which is the gap between the neighboring bursts. To minimize the energy consumption of data transmission, it is essential to select appropriate value for burst size and interval. If the burst size is too large, the traffic may suffer congestion; if the burst size is too small, the energy consumption is not that efficient. However, in this thesis, we only present our idea to customize the Xen virtualization to control the values of burst size and interval, which make it energy efficient for mobile clients. This is one of our future work.

4.8.2 Energy Efficiency Discussion

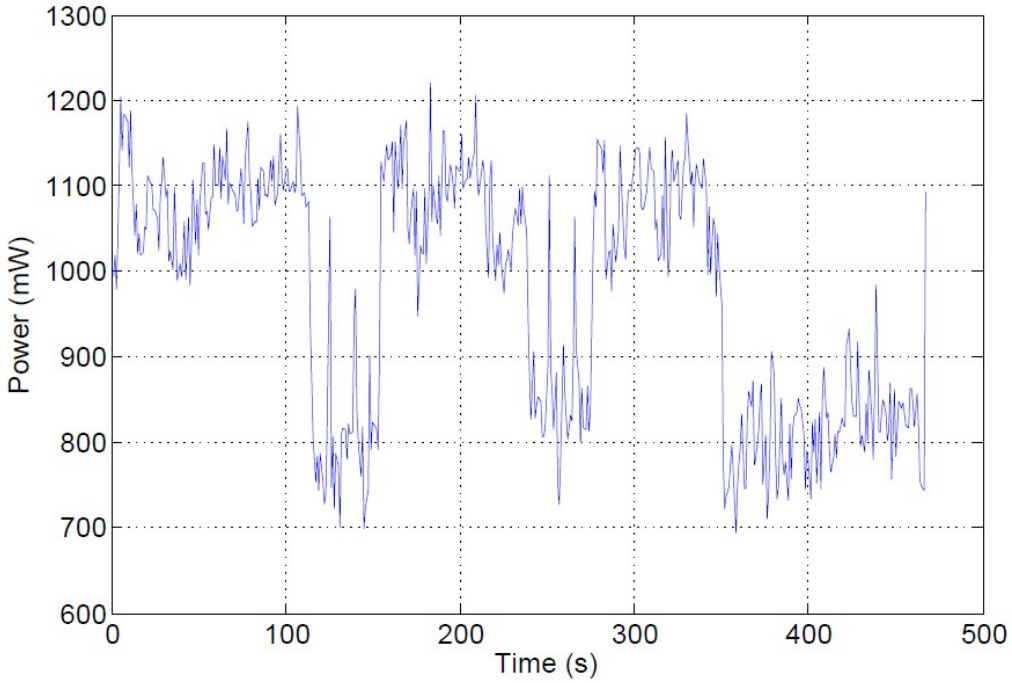

Figure 4.13: Energy Efficiency

We use Monsoon power monitor to collect the power consumption data of three *m1.large* instances. However, the results of E5430, E5645 and E5507

are similar, which are 34133μAh, 34030μAh and 34058μAh. Hence, figure 4.13 only plots the power consumption of E5430. From the figure 4.13, we can see that there are three phases of video playback.

1. *Initial Buffering* phase. The power is the highest at the beginning because mobile clients need to initialize the player environment, establish the connection and load the video.

2. *Play and Buffering* phase. At this phase, the power should support the video playback as well as buffering the video content. Thus, the energy consumption consists of two main parts: transmission and playback energy.

3. *Playing* phase. After the cache is full, the player will stop buffering. At this moment, power is mostly consumed by video playback.

Phase 2 and 3 will be repeated until all the video contents are buffered. In our cases, the energy of playback is the same since we play the same video with the same mobile player. Thus, the difference of measured energy consumption is actually the difference of transmission energy. However, we only observe a minor difference, which we can regard as a margin of error.

Chapter 5

Discussion

5.1 Performance Review

Based on the results of performance evaluation, we conclude the *performance variations* in table 5.1 with E5507 as a baseline server. From table 5.1, it is clearly that the best-performance instance in *m1.large* instance is E5645 while the worst is E5507 as the baseline.

Table 5.1: Performance variations

Performance	E5430	E5507	E5645
CPU single	1.11	1	1.15
CPU double	1.10	1	1.21
Memory	1.14	1	1.5
Disk	1.04	1	1.38
Web Server Static	1.25	1	1
Web Server Dynamic	1.33	1	1.60

We firstly analyze the CPU performance variations. According to Amazon EC2, they evaluate the CPU capacity using ECU which is equivalent a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. From the table 5.1, the CPU performance variations for both single and double processes are also ranging from the 1.0 to 1.21. Thus, our results remain consistent with that of Amazon. However, we also find other aspects of instance performance variations are much larger than that of CPU. For example, for dynamic Web server performance, E5645 is 1.60 times than E5507 while E5430 1.33 times than E5507.

For cloud users, they are only informed that there is a discrepancy in CPU performance from 1.0 GHz to 1.2GHz. But the fact is the gap is growing in the performance of other aspects, which the cloud users may suffer economical loss. In next section, we will make a cost analysis from cloud users points of view.

5.2 Cost Analysis

Table 5.2: Notations

Notation	Definition
f	Hourly cost of an instance
h	Number of hours to run
m	Number of different instances
n	Number of instances needed with worst performance
p_i	Probability of instances hosted with a specific hardware
x_i	Performance variation compared to the baseline instance
C	The total cost

In this section, we will present our strategy for cloud users to seek the best-performing instances in the same instance type, which achieves the cost saving. In the table 5.1, we select the worst-performing instance (E5507) as the baseline and other instances are x (larger than 1) times better than the baseline. In our model, we define x as performance variation and we define other notations in table 5.2. Suppose the same amount of tasks (e.g, computation, communication etc), with better-performing instances, there are two alternatives to complete the task.

1. Smaller the number of instances running for the amount of time;
2. Same number of instances running for shorter period of time.

From the cost perspective, these two ways are the same. We take the first alternative as the example. The expected value of the performance of a random instance is defined as follows:

$$E(X) = \sum_{i=1}^{m} x_i * p_i \tag{5.1}$$

The total cost of completing the task, equivalent to n*h hours work, using random instances can be deduced as follows:

$$C_{random} = n * h * f/E(X) \tag{5.2}$$

If we aim to select the best-performing instances to complete the task, the cost of this optimized scenario is:

$$C_{opt} = n * h * f/x_{opt} \tag{5.3}$$

Furthermore, the ″trial and error″ testing process results in extra cost for the optimized scenario. As in Amazon EC2, the less than one hour usage is rounded up to and charged as one hour. Thus, the extra cost of finding n best-performing instances is:

$$C_{extra} = n * f/p_{opt} \tag{5.4}$$

During the process, we assume that the selection of finding one fast instance takes no more than one hour and the jobs are relatively small to the population of available servers. As a matter of fact, we can simply launch one instance from Amazon, then check its CPU type with *cpuid*. If the instance is not the best-performing one, we simply terminate it and launch another one. The potential cost saving is:

$$C_{saving} = C_{random} - C_{opt} - C_{extra} \tag{5.5}$$

Put Eq. 1, Eq. 2, Eq. 3, and Eq. 4 in Eq. 5, we can deduce the following equation:

$$C_{saving} = (h/(\sum_{i=1}^{m} x_i * p_i) - h/x_{opt} - 1/p_{opt}) * n * f \tag{5.6}$$

For example, there are three different sub-types of instances, E5430, E5507, and E5645 in m1.large instance. From the table 4.1, the probability of each subtype of instance is 17%, 40%, and 42%, respectively. The unit cost of a regular m1.large instance (excluding reserved instances and spot instances) is $0.34/hour. The worst performing instance is E5507, thus it is taken as the baseline. On average, E5430 and E5645 is 1.1 and 1.4 times, respectively, as fast as E5507. Put all these values in Eq.6, we can acquire the following equation:

$$C_{saving} = 0.34 * n * (0.1368 * h - 2.38)$$

In order to achieve cost saving, the requirement is $C_{saving} > 0$. It follows immediately that we get h>17.4. Put it another way, only when the complete time of the task is at least grater than 17.4 hours, cam we reach the cost savings with our selection strategy. Specifically, if we have a task requires 100 E5507 comparable m1.large instances to complete in a year (24hours/-day*365days/year=8760 hours), the potential cost saving for the whole year is $40664, a 16% cost saving in percentage.

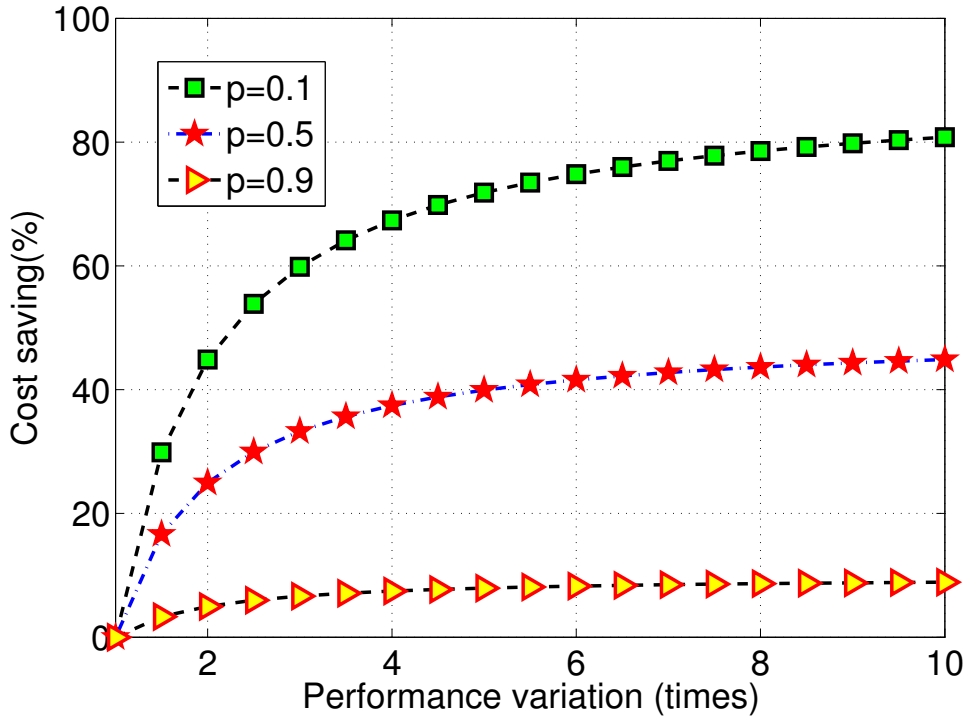

Figure 5.1: Cost saving analysis

Furthermore, we find out that different hardware is not distributed uniformly among all the availability zones. In some zones, the majority instances is one type of hardware while another type dominates in another zone. Thus, it would also be interesting to analyze two types of hardware (e.g. E5507 and E5645) and investigate the maximum cost saving achievable. The result is depicted in figure 5.1, wherein p stands for the probability of the fast instances (e.g. E5645), and x-axis stands for the performance variation in times.

If the fast instances account for the majority of the overall instances, e.g. p = 0.9, without a selection strategy, the probability of obtaining a fast instance

is very high. Thus, the performance is close to the optimal situation with the selection strategy, and the cost saving achievable is trivial. However, as the fast instances account for less proportion of the overall instances, the cost saving achievable is becoming significant. In the case of p=0.1, if the fast instance is 10 times as fast as the slow instance, the cost saving is as high as 80%. Obviously, this is an unrealistic situation with all the efforts Amazon contributes to make the same type of instances function closely. From table 5.1, we know that 1.1-1.6 times variation is highly possible. With 1.5 times variation, the achievable cost saving can reach 30%. For medium and small companies, which are the major customers of Amazon EC2 platform, they could make a large saving.

5.3 Implications

5.3.1 High Performance Computing

High performance computing is to run many parallel computational intensive workloads. It is therefore the best choice for these applications are High-CPU instances. For the instances in standard family, we observe the widespread process delay and unstable network throughput and latency, which make them unstable for the CPU-intensive workloads. For example, in a typical scenario of Map-Reduce application, the worker nodes process the sub-problems and pass the results back to its master nodes. The master nodes collect all the answers to each sub-problem from the work nodes and combine them in some way to form the output. If the network connections between master nodes and work nodes are suffered high latency and low throughput, the master nodes have to wait for the answers, which lead to the performance degradation. To improve the performance for Map-Reduce application, more dedicated work should conduct in cloud data center, such as refining resource allocation mechanisms as well as customized Xen CPU schedulers for Map-Reduce workloads.

5.3.2 Latency Sensitive Application

During the evaluation, we observe the large RTTs in E5507 and also frequent small RTTs in E5430. The results could give some useful implications for latency sensitive application providers. It is well advised that latency sensitive applications are not suitable to deploy in the instances of standard

family. We further deploy the Counter Strike server, which is also a first-person-shooter(FPS) game, on the Amazon EU-west region in Ireland and measure the latency using Qstat from Aalto university in Finland. The duration of test lasted about 1 hour. The results show that, during the 1 hour, the maximum latencies of E5430, E5507 and E5645 are 1138ms, 1241ms and 1152ms, respectively while the average latencies about 62ms. To achieve better game experience, the latency should be less than 150ms. Although the average latencies are less than 150ms, there are quite many latencies larger than 200ms. Thus, we suggest the latency sensitive application should avoid to deploy on the Standard instances. In our case, it would be much better to select High-Memory or High-CPU instances to deploy the Counter Strike game server.

Chapter 6

Conclusions and Future Work

6.1 Summary

In this thesis, we focus on the Amazon EC2 cloud data center and Xen virtualization. We adopt a series of scientific benchmarks to carry out the experiments in both Amazon EC2 and local data center. In this section, we conclude our work in following five aspects.

Firstly, we find out Amazon EC2 uses widely diverse hardware configurations to build up its platform. For example, they use Intel E5410 for high CPU instance and X5550 for High Memory Extra large instance. Moreover, we also find the diversity not only exists among different instance types, but also occurs within the same instance type. For m1.large instance, there are three mainly CPU types, namely Intel E5430, E5507 and E5645.

Secondly, to decrease or eliminate the diversity, distinct scheduling mechanisms have been employed to assure the fair-share of CPU processing capability of the same instance type. We find for E5430, there are two distinct scheduling mechanisms. Based on the similarities observed from Amazon EC2 and local environments, it is safe to speculate that Amazon EC2 utilizes at least two different schedulers, one is SEDF-like, and the other one is Credit-like. Meanwhile, we find out the different scheduling intervals of E5430 and E5507. For E5430, the scheduling intervals are small but frequent while that of E5507 large but less frequent. It is noticeable that E5645 can achieve almost 100% CPU utilization.

Thirdly, the distinct scheduling mechanisms incur unpredictable performance on the other aspects of the VMs, e.g. memory, disk, network throughput, network latency, and Web server performance. The different scheduling mecha-

nisms lead to the memory latency of handling requests and unstable network throughput. From the results of network latency, we do not recommend that latency-sensitive applications are deployed in virtualization environment. For Web server performance, the m1.large instance with E5645 CPU is the best choice for cloud users.

Fourthly, we further analyze that the performance variation of the same sub-type of instances, i.e. hosted by identical hardware, is relatively small, whilst the variation among different sub-types of instances , i.e. hosted by heterogeneous hardware, can reach up to 60%. Through the results of a set of detailed micro-benchmark and application-level benchmark, we could model the performance variations. Moreover, we launch a batch of instances to obtain the probability of each kind of sub-type instance.

Finally, we make a hot discussion on the implications of our performance evaluation. Firstly, we find the performance variation caused by CPU are growing largely in other aspects such as memory, disk as well as Web server applications. Secondly, we present an instance selection strategy for cloud users, which makes a large savings for the companies. Finally, we discuss the potential impact of virtualization on high performance computing workloads as well as latency sensitive applications

6.2 Future Work

Future work can be conducted in the following aspects. The First and immediate work is to find out how to minimize or even eliminate the hardware heterogeneous in cloud data center. It is common for cloud data centers to replace their outdated hardware and upgrade with better one. How to construct a relatively fair platform to cloud-users is challenging. The future work could focus on building such a model, which provides a fair resource allocation mechanisms to balance the performance variation in CPU capacity, network capability, etc, of the hardware.

Also, it is interesting to further explore the energy efficiency of mobile clients connecting to cloud data center. The process sharing reshapes the traffic and delay the packets into burst. Our idea is to design a mechanisms to control the burst size of packets as well as burst intervals from virtualization point of view, which achieve the energy saving on the mobile clients. For example, we could optimize Xen CPU scheduler to make it more suitable for mobile clients. At present, we indeed observe the burst traffic from Amazon but there is no noticeable difference in their power consumption. Thus, more

dedicated and careful work should be conducted in the future.

In the Chapter 5, we discuss the impact of virtualization on high performance computing and latency sensitive applications. How to minimize the effects of virtualization from application developers points of view? We may refine the Xen CPU schedulers as well as network virtualization architecture to low the latencies of both processor and network.

Last but not least, we also plain to extend our research to other cloud data center such as Microsoft Azure, Google AppEngine. We try to find out the difference and also make a performance comparison among different cloud data centers. Thus, more valuable implications can be found to enable enterprise to customize their applications to achieve good performance in different cloud data center.

Bibliography

[1] Aggarwal, B., Chitnis, P., Dey, A., Jain, K., Navda, V., Padmanabhan, V., Ramjee, R., Schulman, A., and Spring, N. Stratus: energy-efficient mobile communication using cloud support. In *ACM SIGCOMM Computer Communication Review* (2010), vol. 40, ACM, pp. 477–478.

[2] Balasubramanian, N., Balasubramanian, A., and Venkataramani, A. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference* (2009), ACM, pp. 280–293.

[3] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev. 37*, 5 (Oct. 2003), 164–177.

[4] Barker, S. K., and Shenoy, P. Empirical evaluation of latency-sensitive application performance in the cloud. In *Proceedings of the first annual ACM SIGMM conference on Multimedia systems* (New York, NY, USA, 2010), MMSys '10, ACM, pp. 35–46.

[5] Bjorkqvist, M., Chen, L., Vukolic, M., and Zhang, X. Minimizing retrieval latency for content cloud. In *INFOCOM, 2011 Proceedings IEEE* (april 2011), pp. 1080 –1088.

[6] Cherkasova, L., and Gardner, R. Measuring cpu overhead for i/o processing in the xen virtual machine monitor. In *Proceedings of the annual conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2005), ATEC '05, USENIX Association, pp. 24–24.

[7] Cherkasova, L., Gupta, D., and Vahdat, A. Comparison of the three cpu schedulers in xen. *SIGMETRICS Perform. Eval. Rev. 35*, 2 (Sept. 2007), 42–51.

[8] Chonglei Mei, Daniel Taylor, C. W. A. C., and Weissman, J. Mobilizing the cloud: Enabling multi-user mobile outsourcing in the cloud. Tech. rep., Center for, 2011.

[9] Christensen, J. H. Using restful web-services and cloud computing to create next generation mobile applications. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications* (New York, NY, USA, 2009), OOPSLA '09, ACM, pp. 627–634.

[10] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing* (New York, NY, USA, 2010), SoCC '10, ACM, pp. 143–154.

[11] Dogar, F., Steenkiste, P., and Papagiannaki, K. Catnap: exploiting high bandwidth wireless interfaces to save energy for mobile devices. In *Proceedings of the 8th international conference on Mobile systems, applications, and services* (2010), ACM, pp. 107–122.

[12] Du, J., Sehrawat, N., and Zwaenepoel, W. Performance profiling of virtual machines. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments* (New York, NY, USA, 2011), VEE '11, ACM, pp. 3–14.

[13] Duda, K. J., and Cheriton, D. R. Borrowed-virtual-time (bvt) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. *SIGOPS Oper. Syst. Rev. 33*, 5 (Dec. 1999), 261–276.

[14] Garfinkel, S. L. An evaluation of amazon grid computing services: Ec2, s3 and sqs. Tech. rep., Center for, 2007.

[15] Gupta, D., Cherkasova, L., Gardner, R., and Vahdat, A. Enforcing performance isolation across virtual machines in xen. In *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware* (New York, NY, USA, 2006), Middleware '06, Springer-Verlag New York, Inc., pp. 342–362.

[16] Hoque, M., Siekkinen, M., and Nurminen, J. On the energy efficiency of proxy-based traffic shaping for mobile audio streaming. In

Consumer Communications and Networking Conference (CCNC), 2011 IEEE (jan. 2011), pp. 891 –895.

[17] Jackson, K., Ramakrishnan, L., Muriki, K., Canon, S., Cholia, S., Shalf, J., Wasserman, H., and Wright, N. Performance analysis of high performance computing applications on the amazon web services cloud. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on* (30 2010-dec. 3 2010), pp. 159 –168.

[18] Korhonen, J., and Wang, Y. Power-efficient streaming for mobile terminals. In *Proceedings of the international workshop on Network and operating systems support for digital audio and video* (2005), ACM, pp. 39–44.

[19] Kosta, S., Aucinas, A., Hui, P., Mortier, R., and Zhang, X. Thinkair: Dynamic resource allocation and parallel execution in cloud for mobile code offloading.

[20] Kosta, S., Aucinas, A., Hui, P., Mortier, R., and Zhang, X. Unleashing the power of mobile cloud computing using thinkair. *Arxiv preprint arXiv:1105.3232* (2011).

[21] Kumar, K., and Lu, Y.-H. Cloud computing for mobile users: Can offloading computation save energy? *Computer 43*, 4 (april 2010), 51 –56.

[22] Lagerspetz, E., and Tarkoma, S. Mobile search and the cloud: The benefits of offloading. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on* (march 2011), pp. 117 –122.

[23] Lenk, A., Menzel, M., Lipsky, J., Tai, S., and Offermann, P. What are you paying for? performance benchmarking for infrastructure-as-a-service offerings. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on* (july 2011), pp. 484 –491.

[24] Li, A., Yang, X., Kandula, S., and Zhang, M. Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th annual conference on Internet measurement* (New York, NY, USA, 2010), IMC '10, ACM, pp. 1–14.

[25] Li, A., Zong, X., Kandula, S., Yang, X., and Zhang, M. Cloudprophet: towards application performance prediction in cloud. In *Proceedings of the ACM SIGCOMM 2011 conference* (New York, NY, USA, 2011), SIGCOMM '11, ACM, pp. 426–427.

[26] Looga, V., Xiao, Y., Ou, Z., and Yla-Jaaski, A. Exploiting traffic scheduling mechanisms to reduce transmission cost on mobile devices. In *Wireless Communications and Networking Conference (WCNC), 2012 IEEE* (april 2012), pp. 1766 –1770.

[27] Manweiler, J., and Roy Choudhury, R. Avoiding the rush hours: Wifi energy management via traffic isolation. In *Proceedings of the 9th international conference on Mobile systems, applications, and services* (2011), ACM, pp. 253–266.

[28] Mei, Y., Liu, L., Pu, X., Sivathanu, S., and Dong, X. Performance analysis of network i/o workloads in virtualized data centers. *Services Computing, IEEE Transactions on PP*, 99 (2011), 1.

[29] Menon, A., Santos, J. R., Turner, Y., Janakiraman, G. J., and Zwaenepoel, W. Diagnosing performance overheads in the xen virtual machine environment. In *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments* (New York, NY, USA, 2005), VEE '05, ACM, pp. 13–23.

[30] Miettinen, A. P., and Nurminen, J. K. Energy efficiency of mobile clients in cloud computing. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing* (Berkeley, CA, USA, 2010), HotCloud'10, USENIX Association, pp. 4–4.

[31] Ongaro, D., Cox, A. L., and Rixner, S. 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, ACM, pp. 1–10.

[32] Palankar, M. R., Iamnitchi, A., Ripeanu, M., and Garfinkel, S. Amazon s3 for science grids: a viable solution? In *Proceedings of the 2008 international workshop on Data-aware distributed computing* (New York, NY, USA, 2008), DADC '08, ACM, pp. 55–64.

[33] Rao, A., Legout, A., Lim, Y., Towsley, D., Barakat, C., and Dabbous, W. Network characteristics of video streaming traffic. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies* (2011), ACM, p. 25.

[34] Schad, J., Dittrich, J., and Quiané-Ruiz, J.-A. Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proc. VLDB Endow. 3*, 1-2 (Sept. 2010), 460–471.

[35] Schulman, A., Schmid, T., Dutta, P., and Spring, N. Demo: Phone power monitoring with battor.

[36] Turcu, G., Foster, I., and Nestorov, S. Reshaping text data for efficient processing on amazon ec2. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing* (New York, NY, USA, 2010), HPDC '10, ACM, pp. 435–444.

[37] Walker, E. Benchmarking amazon ec2 for high-performance scientific computing. Tech. rep., login: magzine, 2008, vol. 33, no. 5, pp. 18-23.

[38] Wang, G., and Ng, T. The impact of virtualization on network performance of amazon ec2 data center. In *INFOCOM, 2010 Proceedings IEEE* (march 2010), pp. 1 –9.

[39] Wang, X., Vasilakos, A. V., Chen, M., Liu, Y., and Kwon, T. T. A survey of green mobile networks: Opportunities and challenges. *Mob. Netw. Appl. 17*, 1 (Feb. 2012), 4–20.

[40] Wee, S., Apostolopoulos, J., tian Tan, W., and Roy, S. Research and design of a mobile streaming media content delivery network. In *Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on* (july 2003), vol. 1, pp. I – 5–8 vol.1.

[41] Wood, T., Cherkasova, L., Ozonat, K., and Shenoy, P. Profiling and modeling resource usage of virtualized applications. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware* (New York, NY, USA, 2008), Middleware '08, Springer-Verlag New York, Inc., pp. 366–387.

[42] Xiao, Y., Hui, P., Savolainen, P., and Ylä-Jääski, A. Cascap: cloud-assisted context-aware power management for mobile devices. In *Proceedings of the second international workshop on Mobile cloud computing and services* (2011), ACM, pp. 13–18.

[43] Yeo, S., and Lee, H.-H. Using mathematical modeling in provisioning a heterogeneous cloud computing environment. *Computer 44*, 8 (aug. 2011), 55 –62.

[44] Zhai, Y., Liu, M., Zhai, J., Ma, X., and Chen, W. Cloud versus in-house cluster: evaluating amazon cluster compute instances for running mpi applications. In *State of the Practice Reports* (New York, NY, USA, 2011), SC '11, ACM, pp. 11:1–11:10.

Appendix A

Implementation Issues

A.1 CPUBen

A.1.1 Single Process

```
1  /*
2  This is a CPUBenchmark for single process
3  int gettimeofday (struct timeval *tp, struct timezone *tzp):
       returns the current calendar time
4  struct timeval represents an elapsed time. It has the two members
       :
5    long int tv_sec
6    This represents the number of whole seconds of elapsed time.
7    long int tv_usec
8    This is the rest of the elapsed time (a fraction of a second),
         represented as the number of microseconds. It is always
         less than one million.
9  Compile:  gcc -o CPUben CPUben.c
10 Usage: ./CPUben <Process Number>
11 */
12 #include "stdio.h"
13 #include "stdlib.h"
14 #include "malloc.h"
15 #include "sys/time.h"
16 #define MAX_LOOP 1000000  //finish in 1 second
17 int main(int argc, char* argv[]){
18   if (argc != 2){
19           printf("Usage:CPUBen <file name>\n");
20           exit(1);
21     }
22
23   double *timestamps = NULL;
```

```
24    timestamps = (double *)malloc(MAX_LOOP * sizeof(double));
25    int i;
26    for(i = 0; i < MAX_LOOP; i++) {
27      struct timeval now;
28      gettimeofday(&now, NULL);//get current time
29      timestamps[i] = now.tv_sec+(double)now.tv_usec/1000000;
30    }
31    //write all the timestamps to the file;
32    FILE *fp;
33    if((fp=fopen(argv[1],"a+"))==0){
34      printf("Cannot open file..\n");
35      exit(1);
36    }
37    i=0;
38    while(i<MAX_LOOP){
39      fprintf(fp,"%f\n",timestamps[i]);
40      i++;
41    }
42      printf("File saves successfully!\n");
43      free(timestamps);
44    fclose(fp);
45    return 0;
46
47  }
```

A.1.2 Multiple Processes

```
1  /*
2  This is a CPUBenchmark for multiple process.
3  Compile:  g++ -o MutltipleCPUben MutltipleCPUben.cpp
4  Usage: ./MutltipleCPUben <Process Number>
5
6  */
7  #include <unistd.h>     /* Symbolic Constants */
8  #include <sys/types.h>  /* Primitive System Data Types */
9  #include <errno.h>      /* Errors */
10 #include <stdio.h>      /* Input/Output */
11 #include <sys/wait.h>   /* Wait for Process Termination */
12 #include <stdlib.h>     /* General Utilities */
13 #include "malloc.h"
14 #include "sys/time.h"
15 #include <string.h>
16
17 #define MAX_LOOP 1000000  //finish in 1 second
18
```

```
19
20 int  ChildProcess(const char *);                /* child process
       prototype  */
21
22 int  main(int argc, char* argv[])
23 {
24         if (argc != 2){
25            printf("Usage:MutltipleCPUben <Processor Number>\n");
26            exit(1);
27         }
28         int NUM = atoi(argv[1]); // Get the process number
29         pid_t  pid;
30         int  children = 0;
31         char temp[16];
32         const char *format=".txt\0";
33         while (NUM)
34         {
35           switch (pid = fork ())
36             {
37             case 0:             /* child */
38                sprintf(temp,"%d",NUM);
39                memcpy(&temp[1],format,5);
40                exit ( ChildProcess(temp));
41             case -1:            /* error */
42               printf("fork failed\n");
43               break;
44             default:            /* parent */
45               ++children;
46             }
47           --NUM;
48         }
49
50         while (children)
51         {
52              int status, ret;
53
54               if ((pid = wait (&status)) > 0)
55               {
56                    --children;
57               }
58         }
59
60      return 0;
61 }
62
63 int  ChildProcess(const char *fileName)
64 {
65
66         double *timestamps = NULL;
```

```
67          timestamps = (double *)malloc(MAX_LOOP * sizeof(double));
68          int i;
69          for(i = 0; i < MAX_LOOP; i++) {
70                  struct timeval now;
71                  gettimeofday(&now, NULL);//get current time
72                  timestamps[i] = now.tv_sec+(double)now.tv_usec
                        /1000000;
73          }
74          //dump timestamps to the file;
75          FILE *fp;
76          if((fp=fopen(fileName,"a+"))==0){
77                  printf("Cannot open file..\n");
78                  exit(1);
79          }
80          i=0;
81          while(i<MAX_LOOP){
82                  fprintf(fp,"%f\n",timestamps[i]);
83                  i++;
84          }
85          printf("File saves successfully!\n");
86          free(timestamps);
87          fclose(fp);
88
89          return 0;
90  }
```

A.2 TCP Benchmark

A.2.1 TCPBenServer.cpp

```
1  /*
2  This is the server of the TCPBenServer to calculate the
       throughput.
3  Compile:  g++ -o TCPBenServer TCPBenServer.cpp
4  Usage: ./TCPBenServer
5  */
6  #include <stdio.h>
7  #include <cstdlib>
8  #include <cerrno>
9  #include <sys/types.h>
10 #include <sys/socket.h>
11 #include <netinet/in.h>
12 #include <arpa/inet.h>
13 #include <strings.h>
```

```
14 #include <string.h>
15 #include <iostream>
16 #include <errno.h>
17 #include "malloc.h"
18 #include "sys/time.h"
19 #include "fstream"
20 using namespace std;
21 #define BACKLOG 10
22 //the function to subtract the time structure
23 void tv_sub(struct timeval *out,struct timeval *in)
24 {       if( (out->tv_usec-=in->tv_usec)<0)
25         {       --out->tv_sec;
26                 out->tv_usec+=1000000;
27         }
28         out->tv_sec-=in->tv_sec;
29 }
30
31 int main(int argc, char* argv[])
32 {
33    int connfd, sockfd;
34    struct sockaddr_in servaddr;
35    struct sockaddr_in cliaddr;
36    socklen_t clilen;
37    ssize_t number_bytes;
38    int ret_val;
39
40    char ip_str[INET_ADDRSTRLEN];
41
42    sockfd = socket(AF_INET, SOCK_STREAM,0);
43    if (sockfd == -1) {
44       perror("socket");
45       exit(1);
46    }
47
48    bzero(&servaddr,sizeof(servaddr));
49    servaddr.sin_family = AF_INET;
50    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
51    servaddr.sin_port = htons(6666);
52
53    int BUFFER_SIZE = 256*1024;
54    int ret = setsockopt( sockfd, SOL_SOCKET, SO_RCVBUF, (const
55 char*)&BUFFER_SIZE, sizeof(BUFFER_SIZE) );
56
57    int MSG_LEN = 1024;
58    char recvbuf[MSG_LEN];
59
60    ret_val = bind(sockfd,(struct sockaddr *)&servaddr,sizeof(
          servaddr));
61
```

```
62    if (ret_val == -1)
63    {
64       perror("bind");
65       exit(1);
66    }
67    ret_val = listen(sockfd,BACKLOG);
68
69    if (ret_val  == -1) {
70       perror("listen");
71       exit(1);
72    }
73
74    ofstream o_file;
75    o_file.open("tcpout.txt");
76    while(1){
77         cout << "Server is listening on port "<<
78 ntohs(servaddr.sin_port) << endl;
79     clilen = sizeof(cliaddr);
80         connfd = accept(sockfd,(struct sockaddr *)&cliaddr,&
               clilen);
81         if(connfd == -1){
82            perror("accept");
83            continue;
84         }
85         struct timeval starttime;
86         struct timeval endtime;
87         char t[16];
88         double rtt;
89         int total=0;
90         while(1){
91       if(total==0){
92         gettimeofday(&starttime,NULL);  // record the
93       }
94
95              number_bytes = read(connfd,recvbuf,MSG_LEN);
96            if (number_bytes == 0){
97               break;
98            }
99              total += number_bytes;
100             //calculate the throughput every BUFFER_SIZE bytes
101       if(total >= BUFFER_SIZE){
102         gettimeofday(&endtime,NULL);
103         tv_sub(&endtime,&starttime);
104         rtt=endtime.tv_sec*1000000+endtime.tv_usec;
105         o_file << total*8/rtt << endl;
106         total=0;
107       }
108         }
109       close(connfd);
```

```
110     }
111     o_file.close();
112      return 0;
113 }
```

A.2.2 TCPBenClient.cpp

```
1  #include <stdio.h>
2  #include <strings.h>
3  #include <string.h>
4  #include <cstdlib>
5  #include <sys/types.h>
6  #include <sys/socket.h>
7  #include <netinet/in.h>
8  #include <arpa/inet.h>
9  #include <unistd.h>
10 #include <cerrno>
11 #include "sys/time.h"
12 #include "malloc.h"
13 #include <iostream>
14 /*
15 This is the client of the TCP Benchmark to send the file.
16 Compile:  g++ -o TCPBenServer TCPBenClient.cpp
17 Usage: ./TCPBenClient <address> <port>
18 */
19 using namespace std;
20 int main(int argc, char *argv[])
21 {
22     int sockfd;
23     int conn_ret;
24     struct sockaddr_in servaddr;
25
26     ssize_t number_bytes;
27
28     if(argc != 3) {
29        printf("Usage:TCPBenClient <address> <port> \n");
30        return 1;
31     }
32
33     sockfd = socket(AF_INET,SOCK_STREAM,0);
34     if (sockfd == -1) {
35          perror("sock");
36          exit(1);
37     }
38
```

```
39      bzero(&servaddr,sizeof(servaddr));
40      servaddr.sin_family = AF_INET;
41      servaddr.sin_port = htons(atoi(argv[2]));
42      inet_pton(AF_INET,argv[1],&servaddr.sin_addr);
43
44      int BUFFER_SIZE = 256*1024;
45      int ret = setsockopt( sockfd, SOL_SOCKET, SO_SNDBUF, (const
            char*)&BUFFER_SIZE, sizeof(BUFFER_SIZE) );
46
47
48      int MSG_LEN = 1024;
49      char sendbuf[MSG_LEN];
50
51      conn_ret = connect(sockfd,(struct sockaddr *)&servaddr,sizeof(
            servaddr));
52      if(conn_ret == -1) {
53        perror("connect");
54        exit(1);
55      }
56      //send a testfile of 828MB which the duration of transimission
            is about 10 seconds
57      FILE *fp;
58      if((fp=fopen("testfile.txt","r"))==0){
59     printf("Cannot open file..\n");
60     exit(1);
61      }
62
63      int total=0;
64      while(!feof(fp))
65      {
66        fread(sendbuf, sizeof(char),MSG_LEN, fp);
67        number_bytes = write(sockfd,sendbuf,MSG_LEN);
68        total += number_bytes;
69        if(total == BUFFER_SIZE)
70        {
71         total = 0;
72        }
73      }
74      fclose(fp);
75      close(sockfd);
76      return 0;
77  }
```