ON COMPUTATION OFFLOADING AND
VIRTUAL MACHINE SELECTION FOR
CLOUD-BASED APPLICATIONS

by

Yuan Zhao
B.Eng., Beihang University, 2006

a Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

© Yuan Zhao 2013
SIMON FRASER UNIVERSITY
Summer 2013

All rights reserved.
However, in accordance with the Copyright Act of Canada, this work may be
reproduced without authorization under the conditions for “Fair Dealing.”
Therefore, limited reproduction of this work for the purposes of private study,
research, criticism, review and news reporting is likely to be in accordance
with the law, particularly if cited appropriately.
APPROVAL

Name: Yuan Zhao

Degree: Master of Science

Title of Thesis: On Computation Offloading and Virtual Machine Selection for Cloud-based Applications

Examining Committee: Dr. Richard (Hao) Zhang

Chair

Dr. Jiangchuan Liu, Senior Supervisor

Dr. Arthur L. Liestman, Co-Senior Supervisor

Dr. Qianping Gu, SFU Examiner

Date Approved: 15 May
Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the “Institutional Repository” link of the SFU Library website (www.lib.sfu.ca) at http://summit/sfu.ca and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author's written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

While licensing SFU to permit the above uses, the author retains copyright in the thesis, project or extended essays, including the right to change the work for subsequent purposes, including editing and publishing the work in whole or in part, and licensing other parties, as the author may desire.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library  
Burnaby, British Columbia, Canada

revised Fall 2011
Abstract

Cloud computing allows elastic deployment of client applications by provisioning virtual machine (VM) instances. In practice, each VM instance is treated as a non-fractional atomic unit in provisioning, and most cloud providers offer a number of different options for VM instances. A natural question here is what are the tradeoffs of these different VM instances? Or from each client’s perspective, which VM instance(s) should it choose given its specific requirement? It is necessary to decide whether to scale up by upgrading existing VMs or scale out by adding more VMs.

We study a case of video compression computation offloading to the cloud computing and present a systematic experimental study on a state-of-the-art cloud system to understand the performance of diverse VM instances and the tradeoffs therein. Our results reveal the critical performance bottlenecks of these VM instances, particularly in the presence of resource contention, providing valuable guidelines for computation offloading opportunities and the selection of VM instances in practice.
To my family
“No great discovery was ever made without a bold guess.”

— Isaac Newton, (1642 - 1727)
Acknowledgments

First, I give my foremost gratitude to my senior supervisor Dr. Jiangchuan Liu. His significant guidance and encouragement on my study and research have led to the success of my MSc study. His insights, suggestions and vision in computing network made the road to completing this thesis smoother.

I thank my senior supervisor Dr. Arthur L. Liestman, and my thesis examiner Dr. Qianping Gu, for reviewing this thesis. I thank Dr. Richard (Hao) Zhang for taking the time to chair my thesis defense.

I also want to thank my friends, Haiyang Wang, Haitao Li, Xiaoqiang Ma and Lei Zhang, for their kind help and co-work during my master study and research.

Last but certainly not least, I thank my family for their love and support: my parents, and my wife Guiwen.
Contents

Approval ii
Partial Copyright License iii
Abstract iv
Dedication v
Quotation vi
Acknowledgments vii
Contents viii
List of Tables x
List of Figures xi

1 Introduction 1

2 Related Work 5
  2.1 Cloud-based Video Streaming ............................... 5
  2.2 Mobile Video Compression ................................. 6
  2.3 Cloud Resource Provisioning ............................... 6
  2.4 Resource Scale Up and Scale Out ......................... 7
  2.5 Cloud Computing Measurement ............................. 8
3 Cloud-Assisted Motion Estimation: A Case Study

3.1 CAME: Cloud-Assisted Motion Estimation for Mobile Devices

3.2 CAME Implementation

3.2.1 Mesh Node Selection

3.2.2 Cloud Server Motion Estimation for Mesh Nodes

3.2.3 Sub-Block Motion Estimation

3.2.4 Mobile Device Algorithm

3.2.5 Total Cost Equation

3.3 Evaluation

3.3.1 Testing Data

3.3.2 Transmission Energy Consumption

3.3.3 Video Compression Energy Consumption

3.3.4 Total Energy Consumption

4 Performance of Virtual Machine Instances

4.1 Measurement Methodology

4.1.1 Amazon EC2: The Cloud Platform

4.1.2 PlanetLab: The Client Platform

4.1.3 Experimental Data

4.1.4 Profiling Tools

4.2 Network Performance

4.3 CPU Performance

4.4 File I/O Performance

4.5 Cloud Resource Provisioning

4.5.1 The Cloud Resource Provisioning Problem

4.5.2 Scale Up and Scale Out

5 Conclusions

Bibliography
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>YUV Video Files Used in Testing</td>
<td>17</td>
</tr>
<tr>
<td>3.2</td>
<td>Transmission Energy Consumption in CPU Cycles</td>
<td>17</td>
</tr>
<tr>
<td>3.3</td>
<td>Video Compression Energy Consumption in CPU Cycles</td>
<td>19</td>
</tr>
<tr>
<td>4.1</td>
<td>Instance Types</td>
<td>22</td>
</tr>
<tr>
<td>4.2</td>
<td>CPU and Memory Usage</td>
<td>25</td>
</tr>
<tr>
<td>4.3</td>
<td>SysBench Thread Mode Parameters</td>
<td>28</td>
</tr>
<tr>
<td>4.4</td>
<td>Instance Types</td>
<td>39</td>
</tr>
</tbody>
</table>
List of Figures

3.1 CAME Illustration .................................................. 11
3.2 Mesh-based Motion Estimation ................................. 12
3.3 Mesh Nodes in Motion Estimation ............................ 12
3.4 Total Transmission Size (MB) ................................. 18
3.5 Total Energy Consumption in CPU Cycles (billion cycles) ........................................... 18

4.1 Throughput for File Downloading ............................... 24
4.2 CPU Performance ................................................... 27
4.3 Context Switches (Single Thread) .............................. 27
4.4 CPU Multi-Thread Performance Summary ................... 29
4.5 CPU Multi-Thread Context Switches Summary ................. 31
4.6 File I/O Performance .............................................. 32
4.7 File I/O Performance - Context Switches .................... 33
4.8 File I/O Performance - Page Faults ............................ 33
4.9 Normalized File I/O Throughput ............................... 34
Chapter 1

Introduction

The emergence of cloud computing [5] has been dramatically changing the landscape of services for modern computer applications. Through the Internet, it enables end users to conveniently access computing infrastructure, platforms, and software provided by remote cloud providers (e.g., Amazon, Google, and Microsoft) in a pay-as-you-go manner or with long-term lease contracts. This new generation computing paradigm, offering reliable, elastic and cost-effective resource provisioning, can significantly mitigate the overhead for enterprises to construct and maintain their own computing, storage and network infrastructures. With the aid of cloud resources, start-up companies can easily implement their ideas into real products with minimum investment in the initial stage and expand the system scale without much effort later on. A representative system is Dropbox [16], a typical cloud storage and file synchronization service provider, which largely relies on Amazon S3 servers for file storage and leverages Amazon EC2 instances to provide such key functions as synchronization and collaboration among different users. Existing content/service providers can also migrate their legacy applications to cloud platforms. For example, VoD (video-on-demand) providers have leveraged cloud resources to handle burst traffic. They pay by bytes for bandwidth and storage resources so that the long-term costs become much lower than that with over-provisioning in self-owned servers.

VoD is one of the most popular networked applications and it contributes a dominant fraction of internet traffic. Along with advances in 3G and wireless networks, mobile devices become an important end device for Internet video applications. Online statistics [46] show that YouTube mobile gets over 400M views a day, representing 13% of the overall YouTube daily views. Unfortunately, video compression on mobile devices remains a challenging task
due to limited energy. The user has to copy the video to a personal computer, then compress and upload the video. This however is not convenient and discourages people to upload and share mobile videos on Internet.

Empowered by cloud computing, mobile devices are able to fulfill more complex tasks beyond their computation and storage limitations. For example, route calculation is known to be computation intensive in map and navigation systems. With mobile Google Map or Google Navigation, a user can get an optimized route within milliseconds by offloading the route calculation to the Google server. If mobile devices can leverage Cloud Computing resources to perform video compression, the computation cost on the mobile device itself can be reduced dramatically. However, transferring large video files to the cloud server introduces huge energy consumption, contradicting the benefit. The question then becomes how to leverage cloud server computation resources without transferring the whole video file. A typical video compression consists of: motion estimation, transformation, quantization, and entropy coding. Among all these steps, the motion estimation is the most computation intensive and time consuming part accounting for 90% computation time of the whole compression process [17]. Hence it is worthwhile to transfer even part of the motion estimation computation to the cloud servers.

Motivated by computation offloading to the cloud, we propose Cloud-assisted Motion Estimation (CAME), a novel method to use cloud’s computation resources for motion estimation. We use mesh-based motion estimation which is known to be highly effective. Our method uploads anchor frames and mesh nodes to the cloud server for calculating mesh node Motion Vectors (MV). The motion vectors are pushed back to mobile devices for motion estimation of sub-blocks and other video compression steps. By carefully choosing mesh structure, video compression energy consumption can be significantly reduced on mobile devices. Our simulation results suggests that the proposed method can save up to 30% of the energy for video compression and transmission compared to All-on-Mobile method which performs the complete video compression on mobile devices.

When moving applications to cloud-based platform, careful design and planning are required to achieve best performance with minimum cost. Cloud-assisted Motion Estimation is a specific cloud-based application which offloads computation to the cloud. For general cloud-based applications, computation offloading and network throughput offloading are two of the widely used offloading paradigms. A lot of startup companies even base their whole application on the cloud. One of the key issues for cloud-based applications is resource
provisioning. Cloud computing allows elastic deployment of applications by providing provisioning services through which users can create virtual machine (VM) instances. Two kinds of provisioning plans have been widely used, namely, reservation-based and on-demand [4]. It is known that a perfect reservation plan can hardly be realized unless the future resource demand is readily available and relatively stable. Recent research has been mostly focusing on the on-demand plans, providing diverse algorithms [8, 9, 48] towards near optimal resource provisioning.

In practice, each virtual machine instance is treated as a non-fractional atomic unit in provisioning and most cloud providers offer a number of different options for virtual machine instances. For example, Amazon EC2 [2] provides more than ten types of instance [4]. They differ on such key resources as CPU and memory, and their prices differ accordingly. A natural question here is what are the tradeoffs of these different VM instances? Or from each client’s perspective, which VM instance(s) should it choose given its specific requirement for resources?

Intuitively, small instances have either fewer physical CPUs or both fewer CPU cores and smaller memories, and every instance is running on top of a basic operating system. Therefore, the fraction of usable resources that a client sees is smaller on such small instances. It is however not always preferable to choose a larger instance whenever possible. First, a larger instance is generally more expensive. Second, though a larger instance potentially supports more client access, contention for shared resources can escalate with increased accesses as well, which may in turn downgrade the system performance [24, 49]. Such shared resource contention impacts almost all components of a virtual machine, including CPU, network interface card (NIC), I/O, and main memory, with different degrees of severity depending on the types of accesses. As such, the nature of a client’s application matters here. Computation-intensive applications may require more CPUs but the memory demand might be very small; on the other hand, high network throughput applications may require both more CPUs and large memory. These lead to different resource bottlenecks that must be addressed in the VM instance selection for the particular client. The situation becomes even more complicated when the client’s application expands to a larger scale. It is necessary to decide whether to scale up by upgrading existing VMs or scale out by adding more VMs [18, 34, 44]. The answers to these questions clearly require a good understanding of the real world performance of different VM instances from diverse perspectives.
In the second part of the thesis, we present a systematic experimental study on a state-of-the-art cloud system to understand the performance of diverse VM instances and the tradeoffs therein. Specifically, we build a testing platform inside Amazon EC2, which is then accessed by geographically distributed clients from the PlanetLab platform. We devise experiments to track the capacity of different VM instances for a wide range of benchmarks. Our results reveal the critical performance bottlenecks of these VM instances, particularly in the presence of resource contention, providing valuable guidelines for the selection of VM instances in practice.

The two main parts of this thesis are closely related. First, by introducing the proposed CAME system we offer a new opportunity for computation offloading provided by cloud computing. This computation offloading paradigm can be further extended to network throughput offloading or other resource offloading for general cloud-based applications. Further, when moving current applications to cloud-based platforms, it is important to understand the performance of different VM instances for better resource provisioning. Indeed, cost is the key reason consumers moving their applications to cloud-based applications.

The rest of this thesis is organized as follows. In Chapter 2, we discuss related works in cloud-based video streaming, mobile video compression and cloud resource provisioning. In Chapter 3, we discuss the case study of computation offloading to the cloud, which is Cloud-assisted Motion Estimation. In Chapter 4, we study the Amazon EC2 cloud platform to understand the performance of diverse VM instances and the tradeoffs therein. Finally we conclude in Chapter 5.
Chapter 2

Related Work

In this chapter, we first review related studies of cloud-based video streaming and mobile video compression. Then we review related work in the area of cloud resource provisioning, resource scale up and scale out and measurement studies of cloud computing.

2.1 Cloud-based Video Streaming

With the elastic and on-demand nature of resource provisioning, cloud computing has become a promising platform for diverse applications, many of which are video related [29, 33, 42].

To reduce bandwidth reservation cost and to guarantee streaming performance, a predictive cloud bandwidth auto-scaling system is proposed by Niu et al. [36] for VoD providers. The predictable anti-correlation between the demands of video channels is exploited in the system for statistical multiplexing and for hedging the risk of under-provision. Built on a peer-to-peer storage cloud, Novasky [32] provides on-demanding streaming of cinematic-quality videos over a high-bandwidth network with two novel mechanisms: a coding-aware peer storage and replacement strategy and an adaptive server push strategy with video popularity-redundancy awareness. A cloud-based video proxy system is presented by Huang et al. [21] transcoding the original video in real time using a scalable codec based on H.264/SVC (Scalable Video Coding), which is aimed at streaming videos of various qualities.

Motivated by these previous studies, our CAME approach seeks to explore the resources
from cloud as well. Our CAME approach, however, is to utilize such resource for motion estimation, the most computation-intensive task in video compression, so as to realize realtime high quality video encoding and streaming from mobile devices.

2.2 Mobile Video Compression

Since mobile devices typically depend on a limited energy supply and video compression is computation-intensive, existing research have focused on reducing the computational cost for mobile devices [6, 10, 38]. The low complexity video compression system suggested by Jackson and Peplow [22] abandons the ME/MC paradigm and codes the difference between successive frames, making the process significantly less time consuming. A two-step algorithm is introduced by Bahari, Arslan and Erdogan [6], which is further improved by Chatterjee and Chakrabarti [10] to reduce computation and memory accesses with variable block size motion estimation. In a mobile video communication system developed by Peixoto, de Queiroz and Mukherjee [38], the transmitter uses a Wyner-Ziv (WZ) encoder while the receiver uses a traditional decoder. An efficient transcoder should be inserted in the network to convert the video stream.

Our approach differs from these earlier approaches in that we explore mesh-based motion estimation [45], which is known to be cost-effective and has yet to be examined in the mobile communication context. We demonstrate that, given the small data volume of meshes, they work well with cloud-assisted video compression to best balance computation cost, transmission overhead, and compression quality.

2.3 Cloud Resource Provisioning

Cloud computing allows elastic deployment of applications, instances can be created and terminated on-demand. In such a way, additional computing resources can be requested only when needed to save cost. Basically, the goal of cloud resource provisioning is trying to take the best advantage of the flexibility and elasticity of cloud computing.

Because of the pay-as-you-go model of cloud computing, resource provisioning should be performed to keep resource costs to a minimum, while meeting an application quality of service (QoS) needs. An optimal cloud resource provisioning (OCRP) algorithm was proposed by formulating a stochastic programming model to resolve the problem of uncertainty of
consumer’s future demand and providers’ resource prices [9]. A framework was developed which can support dynamic adaptation for applications in a cloud computing environment with a fixed time-limit as well as a resource budget [48]. The goal is to maximize the QoS with the constraints. The QoS or Service Level Agreement (SLA) are important factors during resource provisioning. Dejun, Pierre and Chi [12] studied a small instance of Amazon EC2 to measure Amazon EC2’s SLA. They found that the performance of virtual instances are relatively stable over time with fluctuations of mean response time within at most 8% of the long term average. Hu et al. [20] also studied SLA considering the processing of interactive jobs which have small processing requirements and require good response time performance.

Similarly, within data centers, power consumption is an important topic in cloud computing. Cloud machines consume significant power. Kim, Beloglazov and Buyya investigated power-aware provisioning of virtual machines for real-time services [26]. They modeled a real-time service as a virtual machine request provision virtual machines of data centers using DVFS (Dynamic Voltage Frequency Scaling) schemes.

The basic purpose of cloud computing is to serve different web applications. Juve and Deelman [25] and Dornemann, Juhnke and Freisleben [14] studied cloud resource provisioning from the view point of applications: large-scale scientific workflows and BPEL workflows.

Motivated by these cloud resource provisioning studies, we propose our studies from the view point of instance capacity. We take a different approach to build the foundation for resource provisioning decisions. The foundation is the deep understanding of different types of instance performance.

2.4 Resource Scale Up and Scale Out

Resource scale up and scale out is a relatively new topic in cloud computing and data center resource provisioning. Different implementations of scale up and scale out already exist in industry practice. Arguments between scale up and scale out exist in research field.

Using a case study of nutch/lucene [34], Michael et al. conclude that a ”pure” scale up approach is not very effective in using all the processors in a large SMP, while they were more successful in exploiting the POWER5 SMP with a ”scale-out-in-a-box” approach. In this case, multiple instances of the application run concurrently, within a single operating system. They also conclude that a scale out system can achieve about four times the
performance of a similarly priced scale up system.

Scale out architecture is also widely accepted by data centers. Vahdat et al. [44] argue that scale out architectures support flexible, incremental scalability for computing and storage. Yet they suggest that the network’s scale up approach makes it the weak link of data centers. VL2, a scalable and flexible data center network, is proposed by Greenberg et al. [18] using the scale out data center architecture.

We also support scale out architecture for most applications in cloud computing. However, our conclusion is based on the performance observation of different instances in Amazon EC2.

### 2.5 Cloud Computing Measurement

Basically, cloud computing provides a service, such as platform as a service (PaaS) or software as a service (SaaS). This service should meet certain degree of SLAs or performance standards. Some research has evaluated the SLAs and performance standards of cloud computing.

An approach for performance evaluation of cloud computing configurations is presented by Stantchev [43]. Their approach focuses on nonfunctional properties (NFPs) of individual services and provides more relevant and granular information. Their studies suggest that cloud computing providers assure certain service levels for the platform but not for a particular service instance. To understand the tradeoffs inherent in migrating to the cloud of clouds, Jackson et al. [23] evaluated the performance of HPC applications in today’s cloud environments using real applications representative of the workload at a typical super computing center.

To help customers choose a cloud service that fits their needs, CloudCmp, a systematic comparator of the performance and cost of cloud providers was developed by Li et al. [30]. CloudCmp measures the elastic computing, persistent storage, and networking services offered by a cloud. Their study on four widely used cloud providers can guide customers in selecting the best-performing provider for their applications.

Virtualization, the key technology to support cloud computing, also comes with overhead affecting every subsystem. Research has been done to measure virtualization overhead. Pu et al. [40] presented an experimental study on performance interference in parallel processing of CPU and network intensive workloads in the Xen Virtual Machine Monitors (VMMs). Their
experiments measure performance interference among VMs running network I/O workloads that are either CPU bound or network bound. Shea and Liu [41] provided an introduction to the state-of-the-art software-based virtual network interfaces. They analyzed the causes of virtualization overhead through both system analysis and real experiments.

Our work differs from earlier research in that we measure different types of VM instances systematically in large scale using PlantLab and a comprehensive benchmark SysBench. Our measurement results reveal the critical performance bottlenecks for these VM instances, particularly in the presence of resource contention. Our study provides valuable guidelines for the selection of VM instances in practice.
Chapter 3

Cloud-Assisted Motion Estimation: A Case Study

In this chapter, we present a case study of computation offloading to the cloud, specifically Cloud-assisted Motion Estimation. We discuss the architecture of CAME and its performance evaluation. This case study suggests that computation offloading to cloud is efficient.

3.1 CAME: Cloud-Assisted Motion Estimation for Mobile Devices

We consider a cloud-assisted mobile video streaming system that consists of mobile devices and cloud servers. A mobile device user captures video in realtime and expects to compress the video and then stream it to others in realtime as well. Given the high computation overhead of compression and the limited computation power and battery of the device, it is preferable that the compression operation or part of the operation is offloaded to cloud servers. Yet, simply uploading the raw video to the cloud server for compression will consume significant bandwidth and therefore energy for transmission, and is thus not suitable. To this end, our CAME approach seeks to offload motion-estimation, the most computation-intensive module in video compression to the cloud. Specifically, CAME employs a mesh-based motion estimation, which consists of two parts: mesh node motion estimation and the sub-block motion estimation. As we will show later, the mobile device can upload reference
frames and mesh data to the cloud for estimation requiring much smaller data volume. It then downloads the estimated Motion Vectors (MVs) from the cloud server and completes the remaining video compression steps.

The CAME architecture is illustrated in Figure 3.1. It includes following four key steps:

1. In the mobile device, the raw video is divided into macro blocks (MBs). For each MB, a reference frame is extracted, together with a mesh for each successive P-frame. The device then uploads the reference frame and meshes to the cloud.

2. The cloud server conducts mesh motion estimation for the uploaded reference frame and meshes. It then pushes the generated mesh MVs back to the CAME client on the mobile device.

3. The mobile devices, upon receiving the MVs for mesh nodes of each P-frame, continue to calculate sub-block MVs using block-based motion estimation as well as entry coding.

4. The compressed video is then stored in the device or streamed to other devices or servers through the wireless channel.
3.2 CAME Implementation

While the CAME architecture is intuitive, there are a number of implementation issues to be addressed. In this section, we explain the implementation details of CAME, particularly on mesh node selection strategy for cost minimization.

3.2.1 Mesh Node Selection

Mesh-based motion estimation is performed by estimating one motion vector (MV) for each mesh node. As shown in Figure 3.2 and Figure 3.3, mesh nodes are sampled on the reference frame and MVs are calculated from predictive frames (P-frame).

Generally, three different mesh structures are used: regular triangular or rectangular mesh structures, object-based mesh structure, and hierarchy mesh structure.
Regular Mesh Structures

Triangular and rectangular mesh structures are two commonly used structures in all the mesh structures. Mesh topology is simple and predefined in regular mesh.

Object-based Mesh Structure

The object-based mesh structure is also known as adaptive mesh structure. As in real world videos, usually there are discontinuities at object boundaries, so it would be more accurate to sample mesh nodes along object boundaries. In another words, mesh selection and sampling should be adapted to the video objects so that the motion within mesh nodes is smooth.

Hierarchy Mesh Structures

Hierarchy mesh structure is a combination of regular mesh structure and adaptive mesh structure. It’s based on regular mesh, but is adaptive to image objects.

Regular mesh is commonly used since it is simple and predefined. Using regular mesh, both encoder and decoder know the mesh structure, thus there is no overhead to store and send the mesh topology to the decoder. The object-based structure is more accurate because it samples mesh nodes along object boundaries, however it requires extra overhead to analyze objects and mesh topology transmission. Therefore, regular mesh is preferred in the CAME method.

Unlike standard mesh-based motion estimation, CAME exploits a smarter algorithm for mesh node selection: CAME applies a reversed mesh node selection and motion estimation algorithm in which mesh nodes are sampled on P-frames and MVs are calculated from the mesh and the reference frame. Compared to standard mesh-based motion estimation, CAME loses the advantage of tracking the same set of mesh nodes over successive P-frames. However, CAME decreases cost by uploading only the reference frame and mesh data of P-frames instead of uploading whole video frames.

For a single MB, we denote one reference frame as $R$, one P-frame as $P$, and the mesh node fraction $M_f$ from $P$ as $f$. The total transmission cost is

$$C_m = \sum_{i=1}^{m} (C_R + \sum_{i=1}^{n} C_i \cdot f)$$

(3.1)

where $m$ is the number of MB, and $n$ is number of P-frame in a single MB.
3.2.2 Cloud Server Motion Estimation for Mesh Nodes

The energy of the cloud server is not included in the total energy consumption analysis. However, total delay is crucial to the CAME system as CAME is a real-time video compression and uploading system. To reduce total delay of the CAME system, we leverage cloud parallel computation resources to compute mesh MVs. The natural separation of frames to video sequences provides a great opportunity for application level parallelism. The CAME server utilize a master-slave paradigm to coordinate mesh motion estimation for a single video. The CAME server delay can be significantly reduced thanks to cloud parallelled computation. The cloud server motion estimation is depicted in Algorithm 1.

**Algorithm 1** CloudServer-Motion-Estimation

**INPUT:** \{R, \(M_{fi}\)\}, anchor frame R and mesh nodes on P MVs = φ

for each \(M_{fj}\) in \{R, \(M_{fi}\)\} do

Full search to calculate MV from R and \(M_{fj}\)

MV\(_s\) = MV\(_s\) ∪ MV

end for

**OUTPUT:** Push motion vector MV\(_s\) back to the mobile device

3.2.3 Sub-Block Motion Estimation

The mobile device downloads estimated MVs for each P-frame mesh and we denote the downloading transmission cost as \(C_d\). The mobile device uses these MVs to calculate the sub-block MVs and finally compose a complete MV for P-frame. The cost to compute all MVs for all sub-blocks inside a single P-frame is

\[
C_f = \sum_{j=1}^{n} \sum_{i=1}^{m} C_i \cdot f
\]  

(3.2)

where \(n\) is number of P-frame in a single MB, \(m\) is sub-block number inside a single P-frame. This step is a local search which is constrained by the mesh node MV, so the computation is also restricted and relatively little energy is used. Next, the mobile device regenerates the block pixels based on the motion estimation result. We denote the motion compensation cost as

\[
C_c = \sum_{i=1}^{n} C_{comp-i}
\]  

(3.3)

where \(n\) is the number of P-frames in a single MB. All the other steps are the same as normal video compression process, and we denote their costs as \(C_o\).
3.2.4 Mobile Device Algorithm

After the whole video compression process is finished, the video is ready for uploading, and only P-frames are uploaded. We denote cost in this step as $C_u$. The complete algorithm for the mobile device video compression is shown in Algorithm 2.

**Algorithm 2 Mobile-Video-Compression**

**INPUT:** Complete raw video $V$

- Divide $V$ to MacroBlocks (MBs) denoted by $\{MB_i\}$
- for each $MB_i$ in $\{MB_i\}$ do
  - Calculate $\{R, M_{fi}\}$, anchor frame $R$ and fraction mesh nodes on $P$
  - Upload $\{R, M_{fi}\}$ to server for mesh node motion estimation
- end for

$MV_{video} = \phi$

On receiving $\{MV_{mesh}\}$ from server:

- for each $MV$ in $\{MV_{mesh}\}$ do
  - Local search to calculate sub-block $MV_{sub}$ for current frame
  - Interpolate $MV_{sub}$ into $MV$ to get $MV_f$ for frame
  - $MV_{video} = MV_{video} \cup MV_f$
  - Calculate video compensation for current frame
- end for

if $MV_{video}$ is complete then

  - Finish video compression process to get $V_{comp}$
  - Anchor frames are already deducted from $V_{comp}$
  - Upload $V_{comp}$

end if

3.2.5 Total Cost Equation

Finally, the total cost is:

$$C = C_m + C_d + C_f + C_c + C_o + C_u$$ (3.4)

where $C_m$ is the total cost for uploading all MBs, $C_d$ is MV downloading cost, $C_f$ is the total cost for sub-block motion estimation, $C_c$ is the motion compensation cost, $C_o$ is all other costs for video compression on the mobile device after motion estimation, and $C_u$ is the final step mobile video uploading cost.
3.3 Evaluation

In this section, we evaluate the implemented CAME method on a simulated system. Mobile video compression and transmission both require energy consumption. We evaluate the CAME method using a two-computer simulation system: one acts as cloud server and the other acts as the mobile device. We use a simulated system because it is more generic and the result is not specific to a mobile device.

The transmission energy consumption is measured and calculated from compressed video file size and such CAME system intermediate output as mesh motion vectors. We consider a generic energy model in terms of CPU cycle consumption.

Video compression CPU cycles can be directly monitored and measured. However, transmission measurement in CPU cycles is not straightforward, as transmission consumes energy not only on CPU, but also on wireless. Based on the research work done by Balasubramanian, Balasubramanian, Venkataramani, Yuan and Nahrstedt [7, 47], we assume the energy conversion between CPU cycle and transmission is: one byte of WiFi transmission consumes roughly the same energy as 132.3251 CPU cycles.

As mentioned previously, mesh node density impacts both transmission and computation. We choose different mesh node fractions $f$ in simulation. CAME is evaluated when mesh node is sampled from $16 \times 16$, $8 \times 8$, and $4 \times 4$ blocks respectively. Our simulation shows that $8 \times 8$ mode achieves best energy efficiency. Further, to evaluate how CAME performs over existing mobile video compression and transmission methods, we compare CAME performance with All-on-Mobile and Raw-Upload methods. In the All-on-Mobile method, standard H.264 encoding is performed completed on the mobile device and H.264 encoded video file is uploaded. In the Raw-Upload method, uncompressed video is uploaded in raw format.

In the following subsections, we first describe the video compression testing data. Then we analyze CAME transmission and computation energy, respectively. Finally, total energy consumption is evaluated by adding up transmission and computation energy consumption.

3.3.1 Testing Data

In the evaluation, we use three YUV video files which are commonly used in video compression testing. The details of the three videos are illustrated in Table 3.1. The frame size and frame number are not that important, because the total energy consumption and the
Table 3.1: YUV Video Files Used in Testing

<table>
<thead>
<tr>
<th>Video Name</th>
<th>Size(W × H)</th>
<th>Frame Num</th>
<th>YUV</th>
<th>File Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreman</td>
<td>352×288</td>
<td>300</td>
<td>420</td>
<td>43MB</td>
</tr>
<tr>
<td>Mother</td>
<td>352×288</td>
<td>300</td>
<td>420</td>
<td>43MB</td>
</tr>
<tr>
<td>Flower</td>
<td>352×288</td>
<td>250</td>
<td>420</td>
<td>36.2MB</td>
</tr>
</tbody>
</table>

Table 3.2: Transmission Energy Consumption in CPU Cycles

<table>
<thead>
<tr>
<th>Video Name</th>
<th>All on Mobile</th>
<th>CAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreman</td>
<td>6.2987 × 10^8</td>
<td>2.0429 × 10^8</td>
</tr>
<tr>
<td>Mother</td>
<td>4.6843 × 10^8</td>
<td>1.8814 × 10^8</td>
</tr>
<tr>
<td>Flower</td>
<td>8.4291 × 10^8</td>
<td>2.0201 × 10^8</td>
</tr>
</tbody>
</table>

Total energy savings increase proportionally to them. Video Mother and Daughter has low spatial detail, Foreman has medium spatial detail, and Flower has high spatial details.

3.3.2 Transmission Energy Consumption

We measure and calculate the total transmission consumption in total transmission data size, and convert it to equivalent CPU cycle consumption. There are three transmission phases in CAME: Initial reference frame and mesh node data uploading, mesh node motion vectors downloading, and compressed video data uploading.

The total transmission data size is illustrated in Figure 3.4. The total transmission energy consumption in CPU cycles is illustrated in Table 3.2. Mother and Daughter is similar to Foreman. Though Flower’s original video size is smallest, the AoM and CAME transmission size is largest, because Flower has higher spatial details. It is obvious that the AoM method’s data transmission cost is the lowest among all three. The mesh-based motion estimation interpolated to H.264/AVC encoding achieves almost 10:1 compression ratio in our testing. This result is just as expected. Compared to the AoM method, the proposed method introduces more transmission because of the extra data transmission overhead for mesh node uploading and mesh motion vectors downloading. On the other hand, compared to Raw-Upload, the CAME method still saves approximately 60% on total data transmission. This verifies our expectation that CAME can leverage cloud server resources for motion estimation and video compression without transferring the whole video data.
CHAPTER 3. CLOUD-ASSISTED MOTION ESTIMATION: A CASE STUDY

Figure 3.4: Total Transmission Size (MB)

Figure 3.5: Total Energy Consumption in CPU Cycles (billion cycles)
Table 3.3: Video Compression Energy Consumption in CPU Cycles

<table>
<thead>
<tr>
<th>Video Name</th>
<th>All on Mobile</th>
<th>CAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreman</td>
<td>$1.3225 \times 10^{10}$</td>
<td>$7.8689 \times 10^{9}$</td>
</tr>
<tr>
<td>Mother</td>
<td>$1.2650 \times 10^{10}$</td>
<td>$7.5268 \times 10^{9}$</td>
</tr>
<tr>
<td>Flower</td>
<td>$8.0500 \times 10^{9}$</td>
<td>$4.7898 \times 10^{9}$</td>
</tr>
</tbody>
</table>

3.3.3 Video Compression Energy Consumption

The measurement of video compression energy consumption in CPU cycles is straightforward. For CAME, CPU cycles are measured for part of the motion estimation and all the other steps of video compression that are performed on the mobile device. For the AoM method, complete video compression is performed solely on the mobile device, the total CPU cycle consumption is measured when video is encoded by the Java H.264 encoder. The measurement result is depicted in detail in Table 3.3.

In this measurement, we notice that CAME method saves nearly 40% compared to the AoM method. That is exactly what we are expecting. By transferring part of the motion estimation to cloud server, CAME method saves a lot on computation, approximately 50% of the motion estimation computation can be reduced on mobile devices. The test result also verifies that motion estimation accounts for 90% of total video compression time.

3.3.4 Total Energy Consumption

To evaluate the total energy consumption and calculate how much energy we can save using the CAME method, we simply add up energy consumption in transmission and video compression in Table 3.2 and Table 3.3, and the results are compared in Figure 3.5.

As shown in Figure 3.5, CAME achieves up to 30% total energy saving on video compression and transmission compared to the AoM method. This is a significant improvement towards energy saving on mobile video compression and transmission. Though the total transmission size in CAME is larger than the AoM method, CAME saves considerable energy on video compression, particularly for the motion estimation.

Our experimental result suggests that 8×8 mesh selection mode achieves the best performance compared to 4×4 and 16×16 mesh selection modes. This is because in the 4×4 mesh mode, the mesh nodes uploading and mesh motion vectors downloading data size is too large. In the 16×16 mesh mode, as the mesh node density is not enough, though the mesh uploading and mesh motion vectors downloading data size is small, the mobile device
still needs to perform a large fraction of the motion estimation.
Chapter 4

Performance of Virtual Machine Instances

We have discussed a case study of cloud-based application in Chapter 3 which benefits from computation offloading to the cloud. When moving to cloud-based platforms, careful design and planning are required to achieve the best performance with minimum cost because most cloud platforms provide different choices for VM instance types. Therefore, a natural question is what are the tradeoffs of these different VM instances? Or from each client’s perspective, which VM instance(s) should it choose given its specific requirement? The situation becomes more complicated when the client application expands to large scales. It is necessary to decide whether to scale up by upgrading existing VMs or scale out by adding more VMs. The answers to these questions clearly require a good understanding of the real world performance of different VM instances from diverse perspectives.

In this chapter, we present a systematic experimental study on Amazon EC2 cloud platform to better understand the performance of diverse VM instances and the tradeoffs therein.

4.1 Measurement Methodology

In this section we illustrate the measurement methodology used in our research. To collect testing data on a global-scale network service, we use Amazon EC2 and PlanetLab. We use PHP web server on top of Amazon EC2’s Ubuntu VM instances as our web application
CHAPTER 4. PERFORMANCE OF VIRTUAL MACHINE INSTANCES

Table 4.1: Instance Types

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>Computing Units</th>
<th>CPU Cores</th>
<th>CPU Specification</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>1</td>
<td>1</td>
<td>Intel(R) Xeon(R) E5-2650 2.00GHz</td>
<td>1.7GB</td>
</tr>
<tr>
<td>Medium</td>
<td>2</td>
<td>1</td>
<td>Intel(R) Xeon(R) E5-2650 2.00GHz</td>
<td>3.7GB</td>
</tr>
<tr>
<td>Large</td>
<td>4</td>
<td>2</td>
<td>Intel(R) Xeon(R) E5507 2.27GHz</td>
<td>7.5GB</td>
</tr>
<tr>
<td>Extra-Large</td>
<td>8</td>
<td>4</td>
<td>Intel(R) Xeon(R) E5645 2.40GHz</td>
<td>15GB</td>
</tr>
</tbody>
</table>

For single instance performance testing, we use the SysBench [27] benchmark which includes CPU, multi-threading, file I/O, OLTP and memory benchmarks.

4.1.1 Amazon EC2: The Cloud Platform

Amazon’s Elastic Compute Cloud (EC2) [2] is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers. Amazon EC2 is known to be the leading cloud computing provider. New server instances are guaranteed to be created within minutes to enable quick scaling up and down as the computing requirements change. Different instance provisioning plans are available on Amazon EC2: On-Demand Instances are paid by the hour with no long-term commitments and reserved Instances are reserved instances with lower hourly rates. For each provisioning plan, more than 10 types of server instances are available. These differ in terms of ECU number, I/O performance and memory size.

We use Amazon EC2 to build our cloud server web application. We rent different types of Amazon EC2 instances as cloud performance testing servers. Four different types of instances are tested, namely small, medium, large and extra-large. The details of different instances are illustrated in Table 4.1. The operating system used in our testing is Ubuntu version 12.

4.1.2 PlanetLab: The Client Platform

PlanetLab [39] is a global research network that supports the development of new network services. It consists of more than 1000 nodes at more than 500 sites. By creating a single
slice and assigning nodes to each slice, a large scale network testing bed can be built and run in a convenient way. When a slice is created in a node, it is populated with a minimal Fedora Core Linux installation inside a virtual server (VServer). The client side application is deployed using PlanetLab CoDeploy and simultaneous testing performed using MultiQuery.

There are many different ways to perform large scale network performance testing. Spawning multiple requests from a single client or a few clients is convenient yet constrained by the CPU resource and the network bandwidth, and only one location is tested. Testing inside a LAN is also commonly used; however, the testing results do not reflect the real world of the internet because of the bandwidth difference between the internet and a local area network.

Using PlanetLab, we build a real network testing platform with geographically distributed clients. We select 100 nodes from different locations around the world. We run simultaneous network performance tests using the PlanetLab’s CoDeploy and MultiQuery applications [28].

4.1.3 Experimental Data

To test the network performance, we build a PHP web server on top of Apache in Amazon EC2’s Ubuntu VM instances. We use multiple large files for file downloading testing.

To test the CPU, multi-threading, file I/O and memory performance, we use SysBench benchmark. SysBench is a modular, cross-platform and multi-threaded benchmark tool for evaluating OS system performance. SysBench runs a specified number of threads in parallel. The actual workload produced by requests depends on the specified test mode. For example, command sysbench --test=cpu --cpu-max-prime=20000 run tests cpu performance by calculating 20000 prime numbers.

4.1.4 Profiling Tools

To profile the network throughput, we use the ifstat tool [13] which collects the network throughput for specified time interval. Though we try to start the downloading simultaneously across PlanetLab nodes, it is not guaranteed that we can synchronize the download operations. We take the largest 5 sampled download bandwidths recorded by ifstat and calculate the average.

To analyze the network throughput performance and determine the bottle neck resource,
we also profile the Ubuntu server’s system performance using vmstat [19], a simple lightweight system performance monitoring tool that traces the system processes, memory, swap, IO and CPU’s performance information.

Perf [37] is used to analyze virtual instance performance. We use Perf together with SysBench benchmark to collect critical system parameters such as total running time of SysBench benchmarks, context switches and page faults. Perf also supports other hardware events such as CPU cycles, instructions and cache misses, and cache prefetches. However, these hardware events are not supported inside Amazon EC2 virtual instances. The lack of hardware events create a challenges for our system performance evaluation.

4.2 Network Performance

In this section, we analyze the Amazon EC2 network throughput. We conduct our experiment by initiating simultaneous access to the multiple different file downloading services via PlanetLab platform. We analyze the network performance by analyzing network throughput, CPU and memory performance on different types of instances.

To understand the network throughput performance, we create a file downloading service in our web server. The downloading file size is 100 MB. We compare the network throughput of different Amazon EC2’s instances. Four different types of instances are tested. Since PlanetLab’s CoDeploy can only deploy the same file in each directory at the same time, and MultiQuery initiates the same command to run the downloading process simultaneously, we create a different file for each client on web server named as hostname.zip, where the
hostname represents each client’s host name. We program our client application to download its own file from the same server.

With an increased number of clients downloading multiple files, the testing results are illustrated in Figure 4.1. With single client downloading, the network throughput is 15MB/S to 17MB/S for all four instances. This throughput is limited by the client network bandwidth. In our test, the bandwidth between all of the PlanetLab nodes and Amazon EC2 us-east-1a zone is between 3MB/S and 20MB/S. With the increase in the number of clients, we see a near linear increase for network throughput on all four instances which suggests that the network throughput can increases linearly with the number of clients before reaching the bottleneck if all client bandwidths are similar. Apparently there exists a network throughput bottleneck for all of the four instances. For small and medium instances the bottleneck is about 30MB/S, for large instances it is about 55MB/S and for extra-large instances it is about 80MB/S. After reaching the bottleneck, network throughput starts to degrade with the increased number of clients. We speculate that the degradation caused by increased resource contentions. Therefore, we further examine the CPU and memory usage.

**CPU and Memory Usage**

We analyze the Amazon EC2 large instance’s CPU usage, I/O interrupts, and memory usage during client file downloading. We try to find out the cause of the network throughput decreasing. The profiling results is illustrated in Table 4.2.

<table>
<thead>
<tr>
<th>Client Num</th>
<th>CPU</th>
<th>Memory</th>
<th>Interrupts</th>
<th>Context Switches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10%</td>
<td>160MB</td>
<td>4489</td>
<td>109/S</td>
</tr>
<tr>
<td>2</td>
<td>12%</td>
<td>270MB</td>
<td>5764</td>
<td>305/S</td>
</tr>
<tr>
<td>4</td>
<td>22%</td>
<td>465MB</td>
<td>6095</td>
<td>546/S</td>
</tr>
<tr>
<td>10</td>
<td>28%</td>
<td>1070MB</td>
<td>6774</td>
<td>980/S</td>
</tr>
<tr>
<td>20</td>
<td>30%</td>
<td>2.5GB</td>
<td>7827</td>
<td>1285/S</td>
</tr>
<tr>
<td>50</td>
<td>39%</td>
<td>6GB</td>
<td>7845</td>
<td>1733/S</td>
</tr>
</tbody>
</table>

As can be seen, the CPU usage increases from 10% to 46% as the number of clients increases from 1 to 50. This result is expected because when the number of clients increases and when the number of threads increases and the CPU usage increases as well. Though, as our web service is not computation-intensive but bandwidth-intensive, we do not see
significant CPU usage increase.

The memory usage results are expected. The system has a total memory of 7.5GB. After initialization, about 60MB of memory is used by the system. Our single downloading file size is about 100MB. Each client downloads a different file from the server and the system memory is large enough to hold all files in memory without paging out to disk. This tells us that the memory is not the resource bottleneck. Also we do not see a significant increase in system interrupts measured per second.

Finally, we analyze context switches per second. The number of context switches increases from 109 per second to 1733 per second with the number of clients increasing from 1 to 50. It is known that context switches consume large amount of system resources [1] [31]: to a finish single context switch, the current process state should be saved for later restoration; the new process state is then restored and starts to use the CPU time slice. Therefore, we believe that the large cost of context switching and the significant increase in the number of context switches contribute most to the network throughput decreasing as the number of clients increases. To improve the performance, we can use loading balancing so that the number of clients served by a single server is decreased. For example, Dropbox [16] which built on top of Amazon EC2, uses more than 500 distinct domain names pointing to Amazon EC2 servers [15] to balance the workload of downloading.

Our network performance testing results suggest that simply upgrading to the ”next larger level” instance is not a good practice because the network throughput on each type of instance has a hard limit ranging from 250Mbps to 10Gbps [3]. For example, small and medium instances have similar network throughput, extra-large instances have similar network throughput. Also, after reaching the network bottleneck, increasing the number of clients will degrade network throughput performance.

For the tested instances types, the ideal network throughput is are 250Mbps, 250Mbps, 500Mbps and 1000Mbps, respectively. The actual network throughput is a fraction of $N_{\text{ideal}}$ related to the total number of clients. The estimated formulation for actual network throughput is:

$$N_i = N_{\text{ideal}} f'_i \cdot \left(1 - \frac{t}{T_i}\right), \text{if } t \text{ is large enough}$$  \hspace{1cm} (4.1)

where $i$ is the instance type, $f'_i$ is a fraction between 0 and 1, and $t$ is the number of concurrent clients or the number of threads. $T_i$ is a constant integer related to the instance type which can be estimated based on measurement.
4.3 CPU Performance

In SysBench CPU mode, each request consists of the calculation of prime numbers. All calculations are performed using 64-bit integers. We evaluate virtual instance CPU performance based on the execution time. We request SysBench to calculate 100, 1000, 10000, 20000, 40000 prime numbers respectively on each instance. The results are depicted in Figure 4.2.

Figure 4.2 suggests that medium, large and extra-large instances have much better performance than small instances. However, medium, large and extra-large instances have similar performance. This is not surprising after we examine the CPU information on each
CHAPTER 4. PERFORMANCE OF VIRTUAL MACHINE INSTANCES

Table 4.3: SysBench Thread Mode Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread-yields</td>
<td>Number of lock/yield/unlock loops to execute per each request</td>
</tr>
<tr>
<td>thread-locks</td>
<td>Number of mutexes to create</td>
</tr>
<tr>
<td>num-threads</td>
<td>Number of Threads</td>
</tr>
</tbody>
</table>

instance and context switches in Figure 4.3. A medium instance has 2 computing units, which decreases CPU competition compared to small instance. Though SysBench CPU mode is single threaded, there are still many system and background services competing for CPU resources. There are multiple computing units and cores on medium, large and extra large instances. Therefore, the running time on these instances depends on CPU frequency and instructions per cycle (IPC). The CPU frequencies are 2.00 GHz, 2.27 GHz and 2.40 GHz, respectively, and they have similar IPCs.

Most web applications are multi-threaded to support a large volume of concurrent client requests. It is important to benchmark the multi-threading performance of each VM instance. We use SysBench threads mode to evaluate the multi-threading performance. In SysBench threads mode, a large number of threads compete for some set of mutexes. The different input parameters and their descriptions are given in Table 4.3. In the ideal case, the total running time of SysBench should increase linearly with the increase of thread-yields or num-threads parameters. Also, the running time should decrease linearly with the increase of thread-locks when there are enough CPU units or CPU cores.

We first test the CPU multi-threading performance differences with the increase of the number of threads. We test four different cases of thread locks on all four VM instances. The results are illustrated in Figure 4.3.

The result in Figure 4.3 is interesting. When the number of thread locks is 1, all the threads compete for single mutex. The performance is similar with single thread testing because single mutex determines the sequential execution of threads. With the increase of the number of thread locks, the next larger instance shows its power with more computing units or CPU cores. When the number of thread locks is 2, medium, large and extra-large instances perform similarly because they can all accommodate 2 locks to different CPUs or CPU cores. Once the number of thread locks exceeds 2, the performance of medium instances drops compared to large and extra-large instances because at least two locks will
Figure 4.4: CPU Multi-Thread Performance Summary
compete for one CPU. Similarly, when the number of thread locks exceeds 4, extra-large instances show better performance than large instance.

Small instances require about twice as much running time as medium instances and medium instances require about twice as much running time as large instances. This is due to the increased number of CPU units in next larger instance. With multi CPU cores, the workload is balanced across the CPU cores. On all instances, when the number of threads exceeds a certain number the total running time keeps stable. This is contradicted with our expectation. We expect that the total running time increases linearly with the number of threads.

To analyze this problem, we analyze the number of context switches in Figure 4.3. Figure 4.3 explains why the running time keeps stable once the number of threads exceeds a certain number. The number of context switches fluctuates around a specific number once the number of thread exceeds a certain limit. This result is interesting. On one hand, it suggests that context switch time dominates the total running time. Compared to context switches, the fraction of per request lock/yield/unlock running time is small. On the other hand, it suggests that the scheduler does a very good job on these instances. The scheduler reduces the number of context switches when the number of threads increases. This can obviously improve system performance. Also the randomness of context switches explains the randomness of total running time.

According to previous CPU performance studies, CPU resource scales up well and a good scheduler is very important to the performance improvement. Also shared resources should be minimized to reduce resource contention.

Based on the measurement results of single threading CPU intensive benchmark and multi-threading benchmark, we can formulate CPU performance on each VM instance. We first define $C_0$ as the baseline of the CPU capacity, which is capacity of a small instance in our testing. $C_0$ can be described by: $f_0$, the CPU frequency in GHz; $I_0$, the instructions per cycle(IPC); $U_0$, the number of computing units. Then the actual CPU capacity $C_i$ of an VM instance $i$ is defined as:

$$C_i = C_0 \cdot \frac{f_i \cdot I_i \cdot U_i}{f_0 \cdot I_0 \cdot U_0}$$

(4.2)

where $i$ is the instance type. This equation confirms the fact that the CPU resource scales up well.
CHAPTER 4. PERFORMANCE OF VIRTUAL MACHINE INSTANCES

Figure 4.5: CPU Multi-Thread Context Switches Summary
4.4 File I/O Performance

In this section, we test the file I/O performance using SysBench fileio mode. At first, SysBench creates a specified number of files with a specified total size. Then at the run stage, each thread performs specified I/O operations on these files. We also tell SysBench to perform checksums validation on all of data reading from and writing to the disk. We test file I/O performance using random read and write testing. The test result is illustrated in Figure 4.6.

Our file I/O performance testing suggests that doubling total file I/O workload does not double the total running time. The system gains benefits from prefetching which predicts future data access and prefetches the data from disk to memory to reduce page faults. We also notice the next larger instance shows better performance on file I/O, yet the performance is not doubled when the amount of resources is doubled.

We further examine the underlying context switches in Figure 4.7 and page faults in Figure 4.8. The normalized file I/O throughput is illustrated in Figure 4.9. Figure 4.7 and Figure 4.8 show that both the number of context switches and the number of page faults start to increase once total file size exceeds total available memory. The I/O block size and the same for memory page size are both 16 KB, so 1 GB is 65536 pages. The cost of 1 page fault is usually less than 100 microseconds which is not significant. Therefore, it is really abnormal that the increase of the number of context switches is large which leads to the degrading of I/O performance. We call this the Memory Running Out Phenomenon.
CHAPTER 4. PERFORMANCE OF VIRTUAL MACHINE INSTANCES

Figure 4.7: File I/O Performance - Context Switches

Figure 4.8: File I/O Performance - Page Faults
The memory sizes of each VM instance can help to analyze the sudden increase of the number of context switches. When there is enough available memory, page faults only introduce the cost of page loading into memory (paging in). However, when there is not enough available memory, page faults introduce both paging out and paging in, which in turn increases the number of context switches. If the total size of files is large enough, the I/O throughput converges to a stable throughput, which is related to the theoretical instance file I/O performance.

According to the measurement and discussion in file I/O performance testing. We define the file I/O throughput of a VM instance as:

\[ O_i = f_i \cdot O_t, \quad 0 < f_i < 1 \]  \hspace{1cm} (4.3)

where \( f_i \) is a fraction of documented I/O throughput \( O_t \).

### 4.5 Cloud Resource Provisioning

Based on the performance study of the different types of VM instances, we formulate the cloud resource provisioning as a mathematical optimization problem in this section, with the objective of minimizing the cost for specific resource requirements. We also discuss the pros and cons of two different cloud resource scaling paradigm, namely, *scale up* and *scale out*. Finally we provide some guidelines for *scale up* and *scale out*.
4.5.1 The Cloud Resource Provisioning Problem

We consider both the ideal case and the practical case for the cloud resource provisioning problem.

**Ideal Case**

In the ideal case, each VM instance $I_i$ can provide exactly the theoretical amount of resources $(N_i, C_i, O_i, M_i)$, where $N_i$ is the network throughput in $Mbps/s$, $C_i$ is the computing unit number which can be represented by $n \cdot C_0$ ($C_0$ is the CPU unit number of smallest VM instance), $O_i$ is the file I/O throughput in $Mbps/s$, $M_i$ is the memory size in $MB$.

The problem of the ideal case is defined as:

"Given the requirement of resources $(N_R, C_R, O_R, M_R)$, the number of concurrent clients $T$, the resource matrix of VM instances $I$ and the price vector of VM instances $P$, what is the minimum cost denoted by $C$?"

The resource provisioning problem of ideal case can be described as an Integer Linear Programming (ILP) problem. We define the resource of the smallest VM instance as $(1, 1, 1, 1)$. Then, the resource of an arbitrary VM instance can also be defined. Assume the required number of VM instances in the optimal solution is $N = (x_1...x_n)$. The price of the VM instance $I_i$ is $p_i$, then the ideal case is formulated as:

$$\min \sum_{i \in N} p_i x_i$$

s.t. $\sum_{i \in N} N_i x_i \geq N_R$

$\sum_{i \in N} C_i x_i \geq C_R$

$\sum_{i \in N} O_i x_i \geq O_R$

$\sum_{i \in N} M_i x_i \geq M_R$

$x_i \geq 0$, and $x_i$ are integers, $i \in N$

The ILP problem is known to be NP-hard. The last constraint $x_i \geq 0, i \in N$ can be relaxed to convert the problem to a linear programming (LP) with efficient solutions [11], which, after rounding to integers, serve as approximations to the original problem.


**Practical Case**

In practice, the problem is more complex considering the resource contention; that said, a VM instance is not always able to provide the ideal amount of resources. According to our experimental results, we formulate the practical resource planning problem based on the following assumptions:

- The total workload is balanced using Weighted Round Robin (WRR) paradigm. The weight of a VM instance is the sum of the total resources it can provide. For example, the smallest VM instance has a weight of 4, which is the sum of (1, 1, 1, 1). An arbitrary VM instance \( I_i \) has a weight of \( w_i \). Therefore, the number of concurrent clients served by a VM instance \( I_i \) is:

\[
T \cdot \frac{w_i}{\sum_{i \in N} w_i x_i} 
\]  

(4.4)

- The number of contention mutexes is more than the number of computing units on every VM instance. In such cases, the resource contention between threads for CPU tasks can be ignored. Indeed, reducing contention of shared resources is one of the basic design principals hold by web applications to improve performance.

- The memory resource scales linearly to total memory size.

- The network throughput scales according to Equation 4.1.

- The file I/O throughput scales according to Equation 4.3.

Then this practical case of resource provisioning problem can be formulated as a Non-Linear Programming (NLP) problem:

\[
\begin{align*}
\min & \sum_{i \in N} p_i x_i \\
\text{s.t.} & \sum_{i \in N} N_i f_i (1 - \frac{T_i}{\sum_{i \in N} w_i x_i}) x_i \geq N_R \\
& \sum_{i \in N} C_i x_i \geq C_R \\
& \sum_{i \in N} O_i f_i x_i \geq O_R \\
& \sum_{i \in N} M_i x_i \geq M_R \\
x_i \geq 0, \text{ and } x_i \text{ are integers, } i \in N
\end{align*}
\]
In this equation, $T_i$, $f'_i$ and $f_i$ can be estimated based on measurements. The first constraint can be transformed to the following equivalent format:

\[
\left(\sum_{i \in N} N_i f'_i x_i\right) \cdot \left(\sum_{i \in N} w_i x_i\right) - \sum_{i \in N} \frac{T}{T_i} N_i f'_i w_i x_i - N_R \sum_{i \in N} w_i x_i \geq 0
\]

To solve this NLP, we again apply LP relaxation to the last constraint as $x_i \geq 0, i \in N$, and the problem then becomes a quadratically constrained programming problem which is known to be NP-hard [35]. Note that the quadratic constraint contains only positive coefficients for quadratic terms, and hence the problem can be translated to a semidefinite programming (SDP) which can be effectively solved using the interior point method [11]. To further reduce the computation cost, we can first assume only the smallest instances are used, which offers an upper bound $U$ for each $x_i$. In practice, considering the scale of about 10 types of VM instances and multiple thousands of VM instances in state-of-the-art real systems, the search space is limited and this problem can be solved with reasonable computation time.

### 4.5.2 Scale Up and Scale Out

The practical model discussed in the previous section can be used to guide the plan of resource provisioning though, in real systems there are even more factors to consider. In this section, we discuss two different resource provisioning paradigms: *scale up* and *scale out*, their pros, cons and tradeoffs. *Scale up* is also called vertical scaling, which means increasing the amount of resources in VM instances without increasing the number of total instances. The key resources are: CPU, memory, network throughput, file I/O throughput and storage. Storage is not a significant issue in resource provisioning because large storage could be mounted to any type of instance at any time. The network throughput, the CPU, the file I/O throughput and the main memory are the resources to be increased. Usually by doubling the resources the prices are also doubled. Intuitively *scale up* is the natural way a user will consider when resources running out.

*Scale out* or horizontal scaling is to provide more small instances. By adding more small instances, workloads are balanced across multiple small instances. The large clustered computing systems apply *scale out* scheme. For example, without clusters, the e-commerce web applications are not able to serve tens of millions of concurrent client accesses even using the most powerful mainframe machines. It is not easy to implement a *scale out* application...
system. If a user decides to use *scale out* scheme, the application should be capable of managing the cluster of instances. This in turn increases application complexity. However, if the cluster and load balancing is already supported by system architecture, *scale out* is the best approach to improve performance.

**Pros and Cons**

The pros and cons should be considered carefully before making decisions. The pros of *scale up* is as follows. *Scale up* is easier to administer as the total number of instances is stable, which means that the architecture of the application stays unchanged. There is no extra maintenance and development overhead. In some special cases, *scale up* might be the only solution. For example, if a load balancing server encounters resource limits and this single server is not clustered, then it must *scale up* to a more powerful server with more resources.

The main downside of *scale up* is that the instance is supposed to accept more workloads or more concurrent client accesses which can in turn increase resource contentions. Resource contention may introduce significant performance degrading. For example, the increased resource contention for LLC, BUS and I/O increases process waiting time and introduces a large number of context switches. Further, *scale up* introduces the extra overhead of moving one application system to a new system.

The pros of *scale out* are as follows. It is particularly effective in high-throughput web-centric applications [34]. By carefully partitioning the workload and reducing the inter-server communications, the performance of the application system can be improved linearly with the number of instances. *Scale out* typically distributes data across many storage controllers to avoid heavy I/O contentions. Furthermore, the clustered architecture ensures high availability. *Scale out* is the widely accepted commercial resource scaling paradigm.

However, *scale out* can introduce substantial software licensing fees by deploying server softwares to each scaled out system. Also the it scale out scheme requires a mature cluster system architecture which can be a significant development overhead.

A key consideration of *scale up* or *scale out* is the cost. The prices of the four tested instance type are given in Table 4.4. According to Table 4.1 we know that the resources of an instance are doubled in CPU, memory and I/O performance when *scale up* to next level instance. The only exception is small and medium instances have the same I/O performance.

According to the measurement and analysis in the previous section we know that for CPU intensive applications it is a good practice to *scale up*. The advantage is obvious:
when there are enough threads and they do not compete for a small set of shared resources, upgrading to next level will double the capacity (sometimes even more). Though the cost is the same, the underlying structure is simpler and the administration cost is reduced.

For network intensive or file I/O intensive applications, scale out is the way to go. This is because each VM instance has an upper limit on network throughput and the network throughput drops when the current number of clients is large.

For general cloud-based applications, we provide the following principals for scale up or scale out based on our experimental results and problem formulation:

- Small instances should never be used in production because small instance has only 1 computing unit. Performance is impacted seriously by system processes and services. Simple scale up to medium instance doubles the performance.

- It is a good practice to partition the web application to different modules. For different modules, apply different strategies for VM instance selection, scale up or scale out resource provisioning. In Dropbox, for example, the load balancing servers and security servers are good candidates for scale up: use high-CPU VM instances to serve large concurrent client requests and logins. In such cases, the load balancing server and the security server is centralized, which is easy to administer. On the other hand, downloading service servers are good candidates for scale out because of the heavy network throughput and I/O throughput requirements. By load balancing the downloading requests to different small servers, total network throughput is multiplied by the number of small servers and the per-server number of clients is reduced. This further reduces the performance degrading effect when there are many concurrent client requests.

- Before deciding between scale up and scale out, the first consideration is to understand the specific application requirements by profiling the application. CPU intensive only applications or simple file downloading applications are relatively easy to understand.
For a complex web application, it is really not easy to understand the performance requirements without profiling.

- The advantages of cloud computing are: no long-term commitments, pay-as-you-go, scalable and flexible. To take advantage of cloud computing, VM instances should be created or terminated on-demand dynamically. The dynamic planning of resource provisioning can even be based on daily curves of client requests. Smaller instances are easier and cheaper to manage. Yet larger instances are expensive and careful system designs are required to take the best usage of larger instances.

- Though our measurement and analysis are based on only four types of Amazon EC2’s instances our methodology and *scale up* and *scale out* guidelines can be extended to other instance types or other cloud computing platforms built using the VM instance concept.
Chapter 5

Conclusions

In this thesis, we proposed Cloud-assisted Motion Estimation (CAME) motivated by the idea of computation offloading. By offloading part of the most computation-intensive task to the cloud server, CAME significantly reduces the complexity of video compression on mobile devices which leads to considerable energy saving. CAME employs mesh-based motion estimation for video compression which is known to be highly effective. This computation offloading paradigm can be extended to the offloading of other resources to the cloud for general web-based applications, such as network throughput offloading to reduce bandwidth cost.

As a further step, we formulated a resource provisioning problem and provided guidelines on resource scale up and scale out for general cloud-based applications from a measurement approach. We performed a systematic experimental study on four types of Amazon EC2’s VM instances. Our results revealed the critical performance bottlenecks of these VM instances, particularly in the presence of resource contention.

We also observed the abnormal the Memory Running Out Phenomenon which results in a significant decrease of the file I/O throughput. There is an opportunity here to optimize memory management to improve the file I/O throughput without increasing the size of memory. We plan to do more performance testing regarding this issue and to research to mitigate the Memory Running Out Phenomenon in future.
Bibliography


[23] Keith R Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J Wasserman, and Nicholas J Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In


