OPTIMIZING AUTO-SCALING VIRTUAL MACHINES FOR A CLOUD-BASED VOD DATA CENTER

by

KA WAI AU

A Thesis Submitted to
The Hong Kong University of Science and Technology
in Partial Fulfillment of the Requirements for
the Degree of Master of Philosophy
in Computer Science and Engineering

June 2016, Hong Kong

Copyright © by Ka Wai AU 2016
Authorization

I hereby declare that I am the sole author of the thesis.

I authorize the Hong Kong University of Science and Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the Hong Kong University of Science and Technology to reproduce the thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

[Signature]
KA WAI AU

ii
OPTIMIZING AUTO-SCALING VIRTUAL MACHINES FOR A CLOUD-BASED VOD DATA CENTER

by

KA WAI AU

This is to certify that I have examined the above M.Phil. thesis and have found that it is complete and satisfactory in all respects, and that any and all revisions required by the thesis examination committee have been made.

[Signature]
PROF. SHUENG-HAN GARY CHAN, THESIS SUPERVISOR

[Signature]
PROF. ALBERT CHI-SHING CHUNG, ACTING HEAD OF DEPARTMENT

Department of Computer Science and Engineering
5 August 2016
ACKNOWLEDGMENTS

I sincerely thank my advisor, Professor S.H. Gary Chan for his mentoring and advices on my research and personal development in these years. I have learnt a lot about the research process which is greatly beneficial to me as a student and also to my future career.

I would also like to take this opportunity to thank my thesis defence committee members, Professor Sunghun Kim and Professor Cunsheng Ding. Their valuable comments helped me very much on polishing my work.

In my years at HKUST, I was very fortunate to meet my colleagues and friends who helped me with my research and studies. I thank Mr. Zhangyu Chang, Mr. Hongzheng Xiong and Mr. Bo Zhang for their constructive suggestions and comments. This work would have been much more difficult without them.
TABLE OF CONTENTS

Title Page i

Authorization Page ii

Signature Page iii

Acknowledgments iv

Table of Contents v

List of Figures vii

List of Tables viii

Abstract ix

Chapter 1 Introduction 1

Chapter 2 Related Work 6

2.1 Cloud-based VoD architecture and resource provisioning 6

2.2 Content Replication and Traffic Dispatching 9

2.2.1 Inter-data-center 9

2.2.2 Intra-data-center 10

2.3 Auto-scaling 11

Chapter 3 System Model and Problem Formulation 13

3.1 System Model 13
LIST OF FIGURES

1.1 A cloud-based VoD data center with auto-scaling VMs. 3

2.1 A traditional cloud-based VoD data center. 6

2.2 A Cloud-based VoD data center network. 7

2.3 Architecture of AMATO. 8

5.1 Number of operating VMs against request rate. 24

5.2 Remote traffic against request rate. 25

5.3 Data served per minute of each VM. 27

6.1 Number of operating VMs against request rate. 29

6.2 Number of operating VMs against remote traffic limit. 30

6.3 Number of operating VMs against video popularity skewness. 30

6.4 Number of operating VMs against VM storage to average file size ratio. 31

6.5 Number of operating VMs against VM streaming bandwidth. 32

6.6 Load fairness against request rate. 33

6.7 Load fairness against remote traffic limit. 34

6.8 Load fairness against video popularity skewness. 34

6.9 Load fairness against VM streaming bandwidth. 35

6.10 Load fairness against VM storage to average file size ratio. 36
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Major symbols used in the paper.</td>
<td>14</td>
</tr>
<tr>
<td>5.1</td>
<td>Video traces statistics</td>
<td>23</td>
</tr>
<tr>
<td>6.1</td>
<td>Baseline of the parameters.</td>
<td>29</td>
</tr>
</tbody>
</table>
OPTIMIZING AUTO-SCALING VIRTUAL MACHINES FOR A CLOUD-BASED VOD DATA CENTER

by

KA WAI AU

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology

ABSTRACT

We consider a Netflix-like video-on-demand (VoD) system, where the video popularity remains stable over a day or week while the request traffic may vary significantly within a day (e.g., by orders of magnitude). To respond to dynamic user traffic in a timely manner, we study an auto-scaling cloud-based VoD data center, where the virtual machines (VMs) can be turned on and off according to user traffic to elastically scale system resources. Movies are stored in persistent storage as standby unit, which is attached to VMs on-demand at any time. Due to limited VM streaming capacities and persistent storage size, a movie request is dispatched either to an operating VM or to a remote repository (the so-called remote traffic). We are interested in minimizing the number of operating VMs given a certain remote traffic constraint, by jointly optimizing which movies to store in
each persistent storage unit, which storage units to be attached for online VMs, and which VMs to dispatch user requests.

We formulate the problem and show that it is NP-hard. We propose AMATO (Auto-scaling Movie Allocation and Traffic Optimization), an efficient approximation algorithm which achieves 2-approximation at peak traffic. Both experiments on cloud platform (Amazon EC2) and trace-driven simulation based on large-scale real-world data show that AMATO is closely optimal. It achieves significantly lower number of operating VMs as compared with the other state-of-the-art and traditional schemes.
CHAPTER 1

INTRODUCTION

We consider a Netflix-like cloud-based video-on-demand (VoD) data center. The data center consists of virtual machines providing blockbuster movies to a large group of audience.\(^1\) As opposed to user generated contents whose video access popularity may change in minutes or hours, in such a Netflix-like VoD system the popularity is usually rather stable over the time scale of, say, a day or a week. Despite the relatively stable popularity, the total user traffic presented to the VoD system may fluctuate significantly over hours. This has been observed in a large-scale commercially deployed system [17], where the popularity remains quite stable (varies less than 10%) over a day but the request traffic may vary by an order of magnitude.

When serving highly dynamic user demand, the traditional static provisioning approach (where the content provider owns a fixed server farm with the machines running all the time) is no longer cost-effective. To address this, content providers can employ auto-scaling virtual machines (VMs) from cloud service provider to form a data center, where the VMs can be turned on or off in an on-line manner.\(^2\) As compared with static provisioning, the auto-scaling system can adapt to the hourly demand fluctuation to achieve higher resource utilization at a lower cost. With such auto-scaling VMs, we may optimize the video replication in the cloud with the time scale of days (due to popularity change), while scaling up and down the VMs over the day to respond to the fluctuating

\(^1\)In this paper, we use interchangeably “video” and “movie,” and “user” and “request.”

\(^2\)VMs are also commonly referred to as virtual servers/instances in infrastructure as a service (IaaS) of cloud paradigm.
demand.

There are three components in a traditional cloud-based VoD data center. All movie files are placed inside the *movie repository*, which is usually the persistent storage cluster in the cloud paradigm. There is a group of *streaming servers*, which is the Auto-Scaling Group (ASG) that scales in and out the number of operating VMs according to the current demand. Upon an incoming user request, the *request dispatcher* determines one single streaming server, which is an operating VM in the ASG, to deliver the video.

Unlike general web service or scientific computing, an auto-scaling VoD data center is both storage and I/O demanding because it requires a large volume for movie storage and I/O for user requests. When a request is dispatched to a streaming server, the requested movie file may not be immediately available in that VM. In such case, the VM has to first fetch the target file from the repository. As the I/O of persistent storage (repository) is often limited, it is uneconomical to replicate all the videos on-the-fly at the VMs. Therefore, traditional and state-of-the-art cache-based solution may not perform well on VoD services, even achieving a small miss ratio may still lead to a huge remote traffic towards the repository. Moreover, users of VoD services are likely to watch videos at some specific time of the day, say, after work. Content replacements during such peak hours where a large number of concurrent users are watching videos may also lead to severe service disruptions. To overcome this, the videos should be pre-stored or “replicated” in some persistent (and usually homogeneous) cloud storage volumes in advance as standby units, which are attached to the VMs upon demand to serve users. Indeed, such option has been commercially available nowadays (e.g., Amazon Elastic Block Store [22]). As the VMs may be turned on and off any time, their short-lived storage such as RAM or local disk can serve as cache to accelerate the I/O performance. In such a cloud-based VoD data center, the cost mainly comes from the number of operating VMs at a time.
Figure 1.1: A cloud-based VoD data center with auto-scaling VMs.

We consider an auto-scaling cloud-based VoD data center. In this system, the content provider sets up an *Auto-Scaling Group* consisting of a number of VMs as the streaming servers to deliver the video contents. A VM is associated with a persistent storage with a certain (homogeneous) video storage capacity. Each VM has a certain processing and streaming capacity (homogeneous) to serve user demand. These VM instances can be turned on or off any time (but the persistent storage shall always be on to be attached to the VMs), and hence not all of them are operating at the same time.

Figure 1.1 shows a typical cloud-based VoD data center we are considering. A repository stores all the movies. The VM has minimal caching capacity (the work on VM caching is orthogonal to our current work [24, 25], and may be used to further reduce the remote traffic to the repository.) A traffic distributor dispatches each movie request to an operating VM (attached with a persistent storage) or to the repository (the remote traffic). We regard a movie as the smallest unit for storage in a VM, i.e., it is either entirely stored or not at all. Note that this does not sacrifice generality, as we can still
accommodate movie segmentation (as used in DASH or other VoD system) by treating a movie segment as a single “movie” in this paper.

Due to the I/O limit of the repository, there is a certain bandwidth constraint between the streaming servers and the repository. Given that, we are interested in minimizing the deployment cost in terms of the number of operating VMs. Such number depends on the following three factors:

- **Movie allocation**: Given heterogeneous movie popularity and traffic requirement, the movie allocation problem is to decide which movie should be replicated in which persistent storage attached to a VM, or simply, which movies in which VM. The decision affects what movies can be served by a VM when the VM is turned on. As mentioned before, due to relatively stable popularity, movies can be re-allocated in the VMs with a much longer time scale (e.g., daily or weekly) as compared with state switching interval for the VMs due to fluctuating traffic (e.g., with time scale of hours).

- **VM management**: The VM management problem is to decide which VM should be turned on when the remote traffic approaches the remote bandwidth constraint. As each VM may store different movies, the set of VMs to be turned on should be optimized to minimize the VM number.

- **Traffic dispatching**: The distributor dispatches the traffic of a movie to the operating VMs to minimize VM number. As a movie may be stored in multiple operating VMs, which VMs to dispatch the movie requests is critical to the performance of the system.

Clearly movie allocation, VM management and traffic dispatching are inter-related. Due to such inter-dependence, they need to be jointly optimized. We study in this pa-
per their joint optimization to minimize the operating VM number. To the best of our knowledge, this is the first piece of work considering such optimization for an auto-scaling cloud-based VoD data center.

We first formulate the joint problem as a Mixed-integer Linear Programming (MILP) termed MAVMTDA (Movie Allocation, VM Management and Traffic Dispatching in Auto-scaling Cloud Data Center) and show it is NP-hard. To address it, we propose AMATO (Auto-scaling Movie Allocation and Traffic Optimization), a novel and efficient joint approximation algorithm with performance guarantee at peak traffic level to achieve low operating VMs for large movie pool. We prove that AMATO is highly scalable with low computational complexity and 2-approximation at peak traffic.

We evaluate AMATO by conducting experiments on Amazon EC2 cloud platform and extensive trace-based simulation based on large-scale real-world data (as obtained from a leading telecommunication company in China). Our results show that AMATO is closely optimal and efficient. Compared with other state-of-the-art and traditional scheme, AMATO achieves significantly lower VM number given a certain remote traffic constraint (often by more than 50%).

The remainder of this paper is organized as follows. We first briefly review related work in Chapter 2. In Chapter 3 we describe our system model and formulate our problem. We present AMATO in Chapter 4 and prove its approximation ratio. We discuss our experiment result in Chapter 5 and trace-driven simulation results in Chapter 6. We discuss the practicability of AMATO and conclude in Chapter 7.
CHAPTER 2

RELATED WORK

2.1 Cloud-based VoD architecture and resource provisioning

Cloud-based VoD system has attracted much attention recently due to its highly flexible resource provisioning capability. Wu et al. [27] are one of the pioneers who bring VoD system on the cloud platform. From the viewpoint of a cloud service provider (CSP), they propose an extensive cloud-based VoD model consisting of a VM cluster, storage cluster, monitor, broker, negotiator, and scheduler. Figure 2.1 shows a simplified traditional

![Figure 2.1: A traditional cloud-based VoD data center.](image)
cloud-based VoD data center with the CSP components (monitor, broker, negotiator and scheduler) hidden, this would be a good starting point for designing a cloud VoD system from the viewpoint of a content provider. Their work also presents a general idea on how to achieve system cost optimization through resource negotiation between the cloud service provider and the content provider.

![Image of Cloud-based VoD data center network](image)

**Figure 2.2:** A Cloud-based VoD data center network.

The work in [7] considers a cloud-based VoD data-center-network architecture (see Figure 2.2) across multiple regions and proposes an ARIMA demand prediction scheme and a locality-aware resource booking algorithm. The works of [19, 20] focus on the same architecture and presents a demand predicting system to understand bandwidth usage in an on-line fashion, which helps us make better bandwidth reservation among
the data centers. The authors then propose a pricing strategy that achieves significantly lower market price for cloud bandwidth reservation.

A hybrid deployment has been proposed in [14], where the author suggests migrating popular movies into a cloud with usage-based pricing, in order to improve resource utilization and address burst traffic issues. The work uses the cloud for Content-Delivery-Network (CDN) purpose and focuses on the video delivery from the original servers to all other regions across the cloud CDN, which outlines the interactions of the original servers and the cloud. The architecture further extends to P2P and hence the bandwidth consumption of the cloud can be minimized [15, 16].

![Figure 2.3: Architecture of AMATO.](image)

Regardless of the architecture, these works often either treat the VMs as only vertically scalable or use it for streaming purpose alone. As distinct from these works, ours
studies the architecture (dispatcher, ASG and repository) as shown in Figure 2.3 and the dynamics (scaling in and out) inside a cloud VoD data center from the viewpoint of a VoD content provider. We take advantage of the scalable infrastructure and better utilize the persistent VM local storage to avoid huge remote traffic being directed to the storage cluster. To the best of our knowledge, this is the first work in literature to address the joint optimization in such setting.

2.2 Content Replication and Traffic Dispatching

2.2.1 Inter-data-center

Content replication over a cloud has been widely studied from a macroscopic point of view. The work in [21] elevates a traditional CDN to cloud paradigm and decomposes the problem into graph partitioning and replica placement problems. Other work includes user access pattern detection at different geographical region [12], collaborative cache strategy [23], and social User-Generated-Contents (UGC) propagation over a cloud CDN [26, 11]. Content placement for a cloud-based VoD system has been discussed in [19, 20, 7, 5]. The work has provided sophisticated cost models and proposed replication schemes that achieve low operational cost with QoS guarantee. All the above work focuses on the coordination of the nodes (data centers) in cloud CDN as shown in Figure 2.2, with the details inside each of the nodes remaining abstract or unconsidered. As a result of this abstraction, some of the important features of cloud computing actually take place inside the node, such as auto-scaling, has not been considered. Our work, in contrast, investigates content replication, VM management and traffic dispatching from a more microscopic view, which is a timely complement to the studies mentioned above.
2.2.2 Intra-data-center

There has been some previous work to address content replication problem in both traditional and cloud-based VoD data center. On one side, before everybody moves to the cloud paradigm, such problem has already been widely studied in the traditional VoD data center where resources are provisioned according to the well-known 95 percentile rule. Applegate et al. [1] present a content placement algorithm which provides a near-optimal solution, which is more efficient than the traditional LP-relaxation method by orders of magnitude. Borst et al. [3] treat each server as a cache node and develop a cooperative cache management algorithm that maximizes the traffic volume from the cache and minimizes the bandwidth costs. Both works assume there are no dynamics within the data center, in which the server configurations and bandwidth reservation are rarely changed. However, on a cloud platform, any operating VM could be shut down in the next few minutes which makes the server configurations and the total available bandwidth highly dynamic. In such setting, we will have to plan the content replication for every possible VM configuration, while at the same time, adjust the traffic dispatching scheme to adapt to the changing environment. Given the dynamics, this problem is much harder than it was in the traditional data center setting.

On the other side, Gao et al. in [9] consider an adaptive VoD streaming service and suggest selectively caching multiple bit-rate versions of popular contents, but transcoding the unpopular video on-the-fly to save storage resource. Their work also determine whether the transcoding task should be dispatched to a dedicated computing cluster or to a streaming server to save CPU costs. De Cicco et al. [6] present a resource allocation controller which determines the right number of VMs to support the highest quality of video streams for adaptive streaming. They further propose a load-balancing algorithm to provide a fair load distribution among the servers at any time. These works are proven
to be scalable in terms of the number of requests. However, neither of these works have considered the extra accessible VM storage space gained or lost during the auto-scaling, which leads to an underutilization of the scalable cloud infrastructure. Therefore there is still room for optimization by a proper content replication in those storage.

Finally Cai et al. [4] propose a cloud-based scalable cache replacement and load balancing scheme that maximizes the total cache hit ratio for Zipf workloads. Their cache-scaling algorithm also determines the number of active VMs which should be used at a specific time. The work is not cost-effective for a VoD data center with large volume of movie files, where content replacement on-the-fly is costly, there is a remote traffic constraint among the active VMs, and workload does not necessarily distributed as Zipf. We show in our experiment and trace-driven simulation that indeed AMATO performs much better.

### 2.3 Auto-scaling

How to efficiently auto-scale resources in the cloud has been studied by many researchers. In [8], the authors take advantage of the learning automata to offer an approach for the scalability of the web applications, in order to provide the best possible way for scaling up and down of the VMs. As cloud service provider, how to efficiently provide auto-scaling property. How other works tackle the instance startup time overhead. The work of [2] studies how to efficiently support the auto-scaling feature from the cloud provider’s perspective in order to dynamically scale the service level. The scheme proposed in [18] seeks to meet the deadline of all VM tasks by minimizing the cost through cloud resource allocation and work flow scheduling. The VM auto-scaling scheme proposed in [13] predicts the requests and tries to balance cost and latency with optimal cloud resource re-allocation. With extensive work making auto-scaling more responsive to the real-time traffic, it is promising to bring the technology to practical commercial uses,
for example, a VoD system. However, unlike most studies in which VMs are performing general computation tasks, applying auto-scaling to a VoD system is quite different and imposes new challenges. Each VM instance stores different content and hence scaling in and out will also requires changing the traffic dispatching schemes based on the content stored in the currently operating VMs. In this work we introduce a novel auto-scaling VoD system. Then our work studies how to better utilize the resource of an auto-scaling VoD cloud.
CHAPTER 3

SYSTEM MODEL AND PROBLEM FORMULATION

3.1 System Model

A cloud VoD data center is composed of a number of VMs, which can store movies and stream them to users. To achieve high resource utilization, the number of running VMs can adapt to the change of traffic on the cloud data center, i.e., when the traffic increases, more VMs are automatically started to ensure high performance; when the traffic decreases, some running VMs are shut down to ensure high resource utilization. We denote the set of all VMs in ASG as \( V \). VM \( v \in V \) has the storage capacity \( c \) (bits), and streaming capacity \( u \) (bits/second).

Let \( M \) denote the set of all movies. For each movie \( m \in M \), it has a corresponding file size \( f^m \) and a time-varying traffic requirement \( r^m(t) \). The traffic requirement of a movie is the minimum required server bandwidth which keeps the movie playback smooth. All movies are available in the storage cluster, or the repository. The repository has certain I/O limit \( B \), which is essentially the remote traffic constraint in our problem.

Denote \( t_a \) and \( t_b \) as 2 instances of time where we are operating VMs between \( t_a \) and \( t_b \). This period is relatively short (e.g., 24 hours) so that the movie popularity does not change and we do not update our persistent storage due to its I/O limitation. However, this period is much longer than the time scale of the changes in the request rate. Therefore for any time \( t \in [t_a, t_b] \), the total traffic \( \Phi(t) \) can be highly fluctuating. As we cannot
change the movie allocation whenever the traffic changes, the binary movie allocation variable \( A^m_v \) is time-invariant within the time-range \([t_a, t_b]\), indicating that the movie allocation should be fixed in this period.

### 3.2 Problem Formulation

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>An instance of time for VM operation</td>
</tr>
<tr>
<td>( M )</td>
<td>The set of movies.</td>
</tr>
<tr>
<td>( r^m(t) )</td>
<td>The traffic requirement of movie ( m ) at time ( t ) (bits/s).</td>
</tr>
<tr>
<td>( f^m )</td>
<td>The file size of movie ( m ) (bits).</td>
</tr>
<tr>
<td>( V )</td>
<td>The set of all VMs in the ASG.</td>
</tr>
<tr>
<td>( V_{on}(t) )</td>
<td>The set of operating VMs in the ASG at time ( t ).</td>
</tr>
<tr>
<td>( u )</td>
<td>The streaming capacity of a VM (bits/s).</td>
</tr>
<tr>
<td>( c )</td>
<td>The storage capacity of a VM (bits).</td>
</tr>
<tr>
<td>( M_v )</td>
<td>The set of all movies stored in VM ( v ).</td>
</tr>
<tr>
<td>( \Phi(t) )</td>
<td>The total traffic at time ( t ) (bits/s).</td>
</tr>
<tr>
<td>( \Phi(t) = \sum_{m \in M} r^m(t) )</td>
<td></td>
</tr>
<tr>
<td>( B )</td>
<td>The remote traffic constraint (bits/s).</td>
</tr>
<tr>
<td>( A^m_v )</td>
<td>Binary movie allocation variable. ( A^m_v = 1 ) if VM ( v ) stores movie ( m ); ( A^m_v = 0 ) otherwise.</td>
</tr>
<tr>
<td>( U_v(t) )</td>
<td>The total traffic served by VM ( v ) at time ( t ) (bits/s).</td>
</tr>
<tr>
<td>( r^m_v(t) )</td>
<td>The traffic of movie ( m ) served by VM ( v ) at time ( t ) (bits/s).</td>
</tr>
</tbody>
</table>

Firstly, we should allocate movies to each VM \( v \), but a VM cannot store movie files beyond the storage capacity limit \( c \). Denote \( M_v \) as the set of all movies allocated to VM \( v \), we have the following constraint

\[
\sum_{m \in M_v} A^m_v \times f^m \leq c. \tag{3.1}
\]
Having stored the movie files in $M_v$, a VM $v$ has to serve the corresponding traffic. Regardless of the request dispatching, the total traffic served by a VM should never exceed the streaming capacity $u$, i.e.

$$U_v(t) = \sum_{m \in M_v} r_v^m(t) \leq u. \quad (3.2)$$

A VM $v$ can serve the traffic of movie $m$ if and only if it stores movie $m$, i.e.

$$r_v^m(t) \leq A_v^m \times r^m(t). \quad (3.3)$$

For any movie $m$, we do not need to serve traffic more than it requires, i.e.

$$\sum_{v \in V_{on}(t)} r_v^m(t) \leq r^m(t). \quad (3.4)$$

Lastly, the remote traffic of the system must not exceed the specified constraint $B$, i.e.

$$\Phi(t) - \sum_{v \in V_{on}(t)} U_v(t) \leq B. \quad (3.5)$$

The cost of the whole system consists of the operating cost of VMs, storage cost of movies in both repository and ASG and the data transmission cost from the cloud to the users. Movie storage cost of the repository and the data transmission cost are fixed given that we are not rejecting any of the movie requests. The VM operating cost is composed of the unit price of VM and the number of running VMs at that time, i.e.

$$\int_t \text{cost}(u, c) * |V_{on}(t)| dt. \quad (3.6)$$

It is trivial that these two components are inter-dependent of each other. Also note that the unit price of the VM depends on the streaming and storage capacity provisioned to each VM. These two factors will affect the granularity of the auto-scaling system, which is beyond the scope of this thesis.
The objective of this study is hence to minimize the number of operating VMs at any time given a fixed VM resource provision and the remote traffic constraint. In this thesis, we will simply use the number of operating VMs as the cost of the VoD system, i.e.

$$\int_{t_a}^{t_b} |V_{on}(t)| dt.$$  \hspace{1cm} (3.7)

In other words, we want to minimize the power-on VMs during a period while the remote traffic does not violate the constraint. Formally, MAVMTDA is formulated as follows: Minimize (3.7) subject to (3.1) - (3.5).

### 3.3 The nature of the MAVMTDA problem

Although MAVMTDA is formulated as a Mixed-integer Linear Programming problem in the previous section, a more natural perspective to the problem is to treat it as a variant of the 2D Vector Bin Packing Problem (2D-VBPP)[10]. Consider the VMs as bins and the movies as items. Each bin has 2 constraints which are the storage and streaming capacities. The objective of the problem is to pack items into the minimum number of bins such that the remote traffic constraint is not violated.

Unlike the original 2D-VBPP in which each item is packed into any of the bins exactly once. A movie file in MAVMTDA may be replicated in many VMs at the same time. Moreover, items cannot be fragmented in the original problem, while replicating a movie file means sharing the traffic requirement $r^m(t)$ among all the replicas.

Formally, let file $i$ denote a replica of a original movie file 1 for $i = 2...n$, where $n$ is the total number of copies including the original file. Let $r^1(t)^*$ denote the traffic requirement of movie file 1 before the replication. After the replication, we have

$$\sum_{m=1}^{n} r^m(t) = r^1(t)^*.$$ \hspace{1cm} (3.8)
Although each of these replicas may share a different portion of the original total traffic requirement, all of them have the same file size $f^m$. Hence, in the MAVMTDA problem, replication can be considered as a special fragmentation where only the traffic requirement part is fragmented.

### 3.4 The hardness of the MAVMTDA problem

As MAVMTDA is similar to 2D-VBPP, it is intuitive to think that the problem might have a comparable hardness. But since 2D-VBPP is more complicated, we will simply show that MAVMTDA is NP-Hard by reducing the classic NP-Complete Bin Packing Problem (BPP).

Consider an instance $l$ of MAVMTDA in which each VM has sufficiently large streaming capacity $u$ such that no replication of movies are needed, i.e.

$$\sum_v A^m_v = 1, \forall m.$$  \(3.9\)

Furthermore, as long as the storage constraint (3.1) is satisfied, every VM can serve all of the total traffic requirements of its stored movies without any residual traffic, regardless of the content placement, i.e. for any possible values of $A^m_v$,

$$\sum_m A^m_v \times r^m(t) \leq u, \forall v.$$  \(3.10\)

Lastly, instance $l$ has a remote traffic constraint $B = 0$, which means all movies have to be placed in the ASG according to (3.5). Now instance $l$ is essentially any instances of BPP as the VM storage becomes the only dimension and we want to pack all the movie files into minimum number of VMs. Hence we may conclude that MAVMTDA is NP-Hard.
We observe that in a cloud-based auto-scaling server, there are some VMs who are always operating and there are other VMs which will be active only if request rate rises. The base VMs (who are always “on”) should have their content placement targeting a low traffic level, while the additional VMs (who are “on” when the service scales out) should have a movie allocation targeting a high traffic level. Therefore we may consider the auto-scaling group as a stack of VMs. When the request rate rises, we will need to replicate hot contents and start allocating some cold contents whose traffic is getting larger. Hence we push a new VM containing a good mix of them on the stack. When request rate drops, we no longer need replication for hot contents and the traffic of the cold contents will become insignificant. We may just pop the top VM off the stack. After the push-pop operations, we should update the request dispatching scheme as there are more(fewer) VMs when the system scales out(in). From these considerations, we propose AMATO and proved its performance at peak traffic. We also prove that the push-pop operation is optimal to decide which VM to turn on.

4.1 Algorithm Description

AMATO consists of three different components, which are movie allocation, VM management and traffic distribution. In the off-line phase, we will first allocate the movies for all the VMs according to our Movie Allocation Algorithm. In the on-line phase, and
based on the remote traffic level, we will turn the VMs on and off according to the VM Management Algorithm. Finally the Traffic Distribution Algorithm determines the dispatching probability of each movie request given the set of currently operating VMs and the current request rate.

### 4.1.1 Movie Allocation

In the ASG, we have \( n \) VMs indexed from \( i = 1 \) to \( n \). These VMs will be filled one by one at each iteration of the algorithm until the remote traffic requirement is satisfied. Let \( r_{\text{max}}^m, \hat{r}_i^m \) and \( \tilde{r}_i^m \) denote the maximum traffic of movie \( m \), the residual traffic of movie \( m \) at the iteration of VM \( i \) and the distributed traffic of movie \( m \) in VM \( i \) respectively.

Define residual traffic \( \hat{r}_1^m = r_{\text{max}}^m \) for all \( m \in M \) initially, we have the following algorithm:

1. Sort the movies in decreasing order of streaming density \( \hat{r}_i^m / f^m \)

2. For each new empty VM \( i \), fill videos that first-fit into this VM’s storage capacity according to the sorted order, until it can no longer store any videos

3. Distribute movie traffics proportionally according to the residual traffic,

\[
\tilde{r}_i^m = \begin{cases} 
\hat{r}_i^m \times \frac{u}{\sum_{m \in M_i} \hat{r}_i^m}, & \text{if } \sum_{m \in M_i} \hat{r}_i^m \geq u, \\
\hat{r}_i^m, & \text{otherwise.}
\end{cases}
\]

4. Update the residual traffic of each movie \( \hat{r}_{i+1}^m = \hat{r}_i^m - \tilde{r}_i^m \) for all \( m \in M_i \)

5. If \( \sum_{m \in M} \hat{r}_{i+1}^m > B \), goto step 1 for \( i + 1 \)th VM, exit otherwise.

### 4.1.2 VM Management

We should turn on new VMs when the remote constraint \( B \) is going to be violated as the request rate increases. Let \( R(t) \) denote the total remote traffic at time \( t \). Theoretically
we will turn on new VMs when \( R(t) \geq B \). However, due to the time-delay in remote traffic monitoring and the warm-up time required for a VM at its startup, we should always scale out at a threshold lower than the theoretical value. i.e. Scale out occurs when \( R(t) \geq B \times h \) for some scaling out threshold factor \( h \leq 1 \). Similarly, we turn off VMs when \( R(t) \leq B \times l \) for some scaling in threshold factor \( l \leq 1 \).

4.1.3 Traffic Distribution

We can derive the request dispatching probabilities of movie \( m \) to VM \( i \) at peak traffic level as \( \hat{p}^m_i = \hat{r}^m_i / r^m_{\text{max}} \). Therefore at any time \( t \) with request rate \( \lambda(t) \), we will have the dispatching probability of movie \( m \) to VM \( i \),

\[
p^m_i(t) = \min\left(\frac{\hat{p}^m_i \times \lambda_{\text{max}}}{\lambda(t)}, 1\right)
\]

, where \( \lambda_{\text{max}} \) is the request rate at peak traffic level.

On one side, \( p^m_i(t) \) rises when the request rate \( \lambda(t) \) is low. The load of a particular movie \( m \) is shared among a small set of operating VMs and each of the VM will serve a large portion of the total requests. On the other side, when the request rate is high, \( p^m_i(t) \) drops and the load of the movie will be shared among a large set of operating VMs, with each VM only serve a small portion of the total requests.

4.2 Proof of 2-Approximation at Peak Traffic

**Theorem** AMATO is 2-Approximation at peak traffic.

Let \( D \) denotes the set of all MAVMTDA problems. Let \( AMATO(d) \) and \( OPT(d) \) denote the AMATO and optimal solution for problem \( d \in D \) at peak traffic respectively.

Suppose \( \exists d \in D \), whose minimum number of VMs used are \( l \) and \( k \) respectively in \( AMATO(d) \) and \( OPT(d) \), and \( l > 2k \). Let \( OPT^*(d) \) be the solution that for each VM
in $OPT(d)$, its videos contents are splitted into 2 VMs. i.e. $OPT^*(d)$ stores exactly the same set of videos as $OPT(d)$ but it uses $2k$ VMs.

On average, $OPT^*(d)$ utilizes at most half of a VM’s storage capacity but each of the VM serves more traffic than those in $AMATO(d)$ as $l > 2k$. If on average $AMATO(d)$ utilizes more than half of a VM’s storage capacity, videos stored in $OPT^*(d)$ must be either also picked in $AMATO(d)$ due to their high streaming density and small file size, or are replaced by movies of even higher streaming densities which probably better utilizes the VM storage capacity as according to the greedy nature of the method. In both cases $AMATO(d)$ must perform at least as good as $OPT^*(d)$. If on average $AMATO(d)$ utilizes less than half of a VM’s storage capacity, then obviously some of the VMs can be merged to give even better utilization. Thus we can see that $AMATO(d)$ can never be worse than $OPT^*(d)$.

Therefore it is impossible for such problem instance $d$ to exist. Hence we may conclude $\forall d \in D, l \leq 2k$.

In reality the worse case only occurs when we have large movies whose file size is comparable to the storage of the VM. Although this performance guarantee no longer holds when we scales down the VMs, we will show that AMATO still outperformed the state-of-the-art LRU-based caching strategy at different traffic levels below the peak in the next chapter.

### 4.3 Proof of the Optimality of Push-Pop Operation

**Theorem** The Push-Pop Operation in AMATO is Optimal. Note that we always fill the video with the largest user traffic into new VM first. Therefore, when the user request increases so we need to turn on a new VM, the VM that has the movies which can
accommodate most user traffic shall be turn on to most effectively reduce the remote traffic. Therefore, the push-pop operation according to AMATO is the optimal solution to turn on/off the VM.
CHAPTER 5

EXPERIMENT RESULTS ON AMAZON EC2

In this experiment, we had implemented the request dispatcher and the auto-scaling program to verify the performance of our algorithm against the modified-LRU (Least Recently Used) heuristic in [4]. The modified-LRU heuristic aims to maximize hit-ratio of the VM cache servers by selectively replicating popular contents into more VMs and avoiding caching very unpopular contents. We will use “LRU” to address this schemes hereafter in both Chapter 5 and 6.

5.1 Data and system settings

We had collected video request traces from a VoD provider in China as the input data of our experiment. These traces include 48817 requests over 15114 different videos. The statistics of the traces are shown in Table 5.1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Average</th>
<th>Std.Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests per video</td>
<td>3.23</td>
<td>9.59</td>
</tr>
<tr>
<td>Video file size(KB)</td>
<td>356177.64</td>
<td>434509.99</td>
</tr>
<tr>
<td>Video bit-rate(kbps)</td>
<td>1645.18</td>
<td>372.93</td>
</tr>
<tr>
<td>Average view time(s)</td>
<td>979.53</td>
<td>2238.72</td>
</tr>
</tbody>
</table>

We considered the EC2 instance are homogeneous in their streaming bandwidth, which is 250mbps. We then allocate EBS (Elastic Block Store) instances with 100GB storage capacity. We generated random video requests according to the access probabilities obtained from the video traces. The request rate rose from 0.25 to 0.75 over 4 hours. The
request dispatcher then distributed the requests to the corresponding VM according to our algorithm. The remote traffic was recorded and the auto-scaling program would start new VMs when it exceeded our predefined threshold for this experiment, which was 250mbps. The system would turn on at most 11 VMs at the same time. Normally, the EC2 instances would take around 2 minutes to warm-up before they could serve requests. We had then recorded the number of operating VMs and the remote traffic at different request rate as the performance metrics for this experiment.

5.2 Illustrative Results

We can see from Figure 5.1 that LRU used significantly more VMs than our scheme AMATO. It is also worth noting that starting new VMs in AMATO would effectively keep the remote traffic under control as shown in Figure 5.2. In particular, the remote traffic is fluctuating around 250mbps, which is the threshold of the experiment, until the request
rate rose beyond 0.6 requests per second. While on the other side, the same operations could only slow down the growth of remote traffic a little bit in LRU. This suggests that VoD system can be highly benefited from a more persistent content placement when the traffic is predictable.

There is a sharp increase of the remote traffic of the LRU scheme as the request rate goes from 0.25 to 0.35 requests per second. To deal with such high remote traffic, many VMs are turned on and a maximum of 11 VMs are operating in the LRU case when the request rate is beyond 0.35 requests per second. The increase in remote traffic then slows down and stop increasing after 0.5 requests per second. This shows the remote traffic in LRU is not sensitive to the increase in the number of operating VMs, which indicates a low utilization on the resources. Therefore given a fixed upper bound on the number of VMs in the ASG, which is what is done in practice, LRU will fail to keep the remote traffic under a small limit.
In contrast, the remote traffic of AMATO slowly increases from 0.35 to 0.65 requests per second. The number of operating VMs also increases at a lower pace than that in the LRU case. This indicates the additional resources during auto-scaling are effectively utilized to keep the remote traffic low. In particular, the effect is progressive, i.e. each new operating VM has a significant effect on the remote traffic of the system. The 250mbps remote traffic constraint is violated beyond 0.65 requests per second, at which both schemes have reached the upper bound of the VM number limit. This suggests that the system has exhausted the resource capacity of the ASG and the remote traffic will start increasing at a higher pace.

Beyond 0.7 requests per second, all three schemes have turned on 11 VMs. The remote traffic of AMATO keep increasing while that of LRU remains steady at around 4 kbits per second. This shows that in a highly under-provisioned environment, LRU might start outperforming AMATO. This is because the off-line movie allocation in AMATO are based on a predicted peak traffic. Once the actual traffic exceeds the original peak traffic predicted in the off-line phase of the algorithm, the performance of the system will deteriorate. While in the LRU case, the system can slowly adapt to the change of movie traffic by replacing contents in the VMs. However, despite LRU performs better when all resources are exhausted, there is a huge underutilization before reaching this point. Moreover, the low Quality of Experience will be an issue in such case, for all 3 schemes.

Figure 5.3 presents a bar chart of the per minute served data of each VM in AMATO and LRU. The per minute served data is calculated as the total served data of each VM divided by its operating time in minutes. This measures the average load distributed to each VM while they are in the running state. It can be seen that each VM in the LRU scheme shares a fair amount of load except the first one, which is probably due to the slow reaction of the auto-scaling system. In AMATO, the load distribution among the VMs
are not as fair as it is in the LRU’s case. We can see that the first three VMs have rather similar values of per minute data served, but the value decreases gradually afterwards. The experiment result aligns well to our algorithm design in which the lately-launched VMs are serving the residual traffic of the early-launched VMs. As a result, AMATO has a higher resource utilization but a less fair load distribution than LRU. However, in the next chapter, we will soon see that by fragmenting movie files, AMATO can still achieve high resource utilization without sacrificing load distribution fairness.
CHAPTER 6

TRACE-DRIVEN SIMULATION RESULTS

We have carried out extensive simulations to evaluate our algorithm's performance against LRU[4], which is also evaluated in the experiment in last chapter, and a persistent allocation scheme MPF (Most-Popular-First heuristic). In MPF, movie files are allocated according to their access popularity. For a fixed number of VMs, the more popular movies will always be allocated and replicated before the less popular movies. Traffic of a movie will be distributed randomly to any VMs who own a replica. Given this comparison, we can see how AMATO perform as compared with other persistent allocation scheme.

In this simulation, we would use the same real data traces as we did in the experiment except for the analysis of video popularity skewness and VM storage to average file size ratio, which we would use generated video traces to explore a wider range of values. The baseline of the parameters are given in Table. 6.1.

We use the number of operating VMs as the major performance metric of this study. Since we are also interested in the load distribution among the operating VMs, we would evaluate this metric by Jain's Fairness Index. In the remainder of this paper, we will use “VM number” and “load fairness” to address these 2 metrics. For simplicity, we assume a zero VM warm-up time for this simulation.

Firstly, we investigate the performance of AMATO, the VM number, against the variation of different parameters. As shown in Figure 6.1, AMATO uses significantly fewer VMs than LRU and MPF at higher request rates beyond 1 request per second. This shows that AMATO is highly scalable at large request rate compared to MPF and
Table 6.1: Baseline of the parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Baseline(*from real data traces)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request rate(requests/s)</td>
<td>1.0</td>
</tr>
<tr>
<td>Remote traffic limit(mbps)</td>
<td>500</td>
</tr>
<tr>
<td>Video popularity skewness</td>
<td>0.55576*</td>
</tr>
<tr>
<td>VM storage to average file size ratio</td>
<td>294.4*</td>
</tr>
<tr>
<td>VM streaming bandwidth(mbps)</td>
<td>250</td>
</tr>
</tbody>
</table>

LRU as it only turns around one-third of the VM number of MPF, which is less than half of the number of VMs as LRU does. In Figure 6.2, we can see that AMATO and MPF only use a small number of VMs even when the remote traffic limit is strict, which is below 500mbps. In the LRU simulation, many more VMs have to be turned on in order to cope with such tight remote traffic limit. This is because the LRU heuristic does not consider the movie traffic statistics when it replicates and replaces movie files. As a result, simply aiming for higher content hit-ratio as in LRU does not effectively relieve the remote traffic as AMATO does.
We then further investigate how the skewness factor of the video popularities affect the performance of the three schemes. In this simulation, movie popularities are gen-
erated and normalized according to the power-law $x^{-\alpha}$ with $x$ as the rank and $\alpha$ as the skewness factor. The larger is the skewness factor, the wider is the gap between the movie popularities of different ranks. Figure 6.3 shows that AMATO is very insensitive to video popularity skewness. The number of VMs turned on in AMATO are rather stable over the whole range of the skewness factor in this simulation. The result aligns well with our model where we do not assume any ranges or distributions of the movie popularity. In contrast, both LRU and MPF rely on a highly skewed popularity distribution in order to have a performance close to AMATO. For a skewness factor less than 1, AMATO outperforms the other two by a large margin.

![Number of operating VMs against VM storage to average file size ratio](image)

Figure 6.4: Number of operating VMs against VM storage to average file size ratio.

Next we would like to examine the effect of movie file size on the number of operating VMs. We alter the VM storage to average file size ratio from 100 to 1000, i.e. the average movie file size changes from one hundredth of the VM storage size to one thousandth in the simulation. While keeping the VM storage size constant at 100GB, the movie file sizes are generated randomly in a normal distribution with mean ranging from 100MB to
1GB and a constant standard deviation. In figure 6.4, which is similar to the result of the skewness factor simulation, AMATO is less sensitive to the movie file sizes as compared with the other two schemes. The number of operating VMs in AMATO and LRU differ by a margin of 10 VMs until the average file size becomes small enough at a VM storage to average file size ratio of 700. All three schemes tend to have bad performance when the average movie file size is large (more than one hundredth of the VM’s storage size), which in such case much VM storage space is wasted.

Figure 6.5: Number of operating VMs against VM streaming bandwidth.

Figure 6.5 provides a good evidence that, with the knowledge of content popularity, persistent placement is likely to have a higher utilization of the VM resource. This is because higher VM streaming bandwidth removes the need of content replication in AMATO and MPF, thus requires fewer VMs to meet the remote traffic limit. The curve becomes flatter towards high VM streaming capacity as the VM storage now becomes the bottleneck of the system. However, the LRU curve seems to be flat over the whole VM streaming capacity range. This is because the increase in VM streaming capacity will not
raise the hit-ratio of the LRU scheme, which only depends on the movie popularity and the VM storage size.

Besides the number of operating VMs, we are also interested in the load fairness of the system. We use the Jain’s Fairness Index to show how the load is distributed among the operating VMs. The higher is the index, the fairer is the load share. As shown in the figure 6.6, load fairness of LRU remains high throughout all the ranges. AMATO, while does not achieve the same high fairness as LRU, gives reasonably fair load to the VMs with fairness index above 0.6 cross different request rates. The index is slowly increasing with the request rate in AMATO and MPF as many of the lower stack VMs will be fully load at higher request rates.

With increasing remote traffic limit in figure 6.7, VMs tend to have fairer load in AMATO and MPF as most of the requests become remote traffic. The fairness index of MPF drops at high remote traffic limit as it has a highly uneven mix of video contents among the VMs, thus the VM with more popular contents will dominate the load share.
In figure 6.8, AMATO achieves low fairness at small skewness factor regardless of its better utilization of VM resources against the other two. It is because AMATO will try to
serve everything from the firstly launched VMs whenever possible, leaving a small amount of residual traffic to the latter launched VMs. When the skewness factor is high, AMATO can replicate hot contents at the latter launched VMs which will share a significant amount of load, hence lead to higher load fairness. A similar situation happens in figure 6.9. With high streaming bandwidth, AMATO tries to serve the majority of the incoming traffic from the first few launched VMs, leaving an insignificant amount of traffic to the latter launched VMs which lead to the low load fairness. Despite this, AMATO has better overall resource utilization as shown in Figure 6.3 and 6.5.

Figure 6.10 shows that AMATO can actually achieve a close-to-LRU fairness by fragmenting videos into smaller files, which is a common practice in state-of-the-art streaming method such as HLS and MPEG-DASH.

From the simulation results we can see that AMATO is highly scalable and flexible under different situations. Not only it maintains low VM number which is insensitive to
content popularity skewness and file size, but also outperform the modified-LRU heuristic by a wide margin in many cases. By fragmenting videos, AMATO can even achieve high load fairness which makes the system much more resilient to overloading than the traditional MPF scheme.

Figure 6.10: Load fairness against VM storage to average file size ratio.
CHAPTER 7

DISCUSSIONS AND CONCLUSION

In order to respond to VoD traffic with highly dynamic daily request variation, we have considered an auto-scaling cloud-based video-on-demand (VoD) data center where the virtual machines (VMs) can be turned on or off within a short period (i.e., within hours). As the VMs have limited streaming capacities and persistent storage, we study the joint optimization of movie allocation, VM management and traffic dispatching to minimize the operating VM number and ensure low remote traffic.

We have formulated the problem as an integer linear programming problem and showed its NP-hardness. We then propose a novel and efficient algorithm termed AMATO (Auto-scaling Movie Allocation and Traffic Optimization). AMATO is highly efficient with provably approximation ratio at peak traffic. We have conducted experiment on commercial cloud platform and extensive simulation based on large-scale real-world VoD traces. Our results show that AMATO under the auto-scaling model achieves much lower operating VM usage than other traditional and states-of-the-art schemes by a wide margin (often by more than 50%).

From the experiment and simulation results shown in the previous chapters, we can see that AMATO outperformed the state-of-the-art LRU-based content placement scheme in many cases. In particular, persistent content placement is likely to provide much higher resource utilization during the period in which content popularities are rather stable and predictable. On the other side, AMATO is not suitable for services with highly unpredictable traffic and rapidly changing content popularities (for example, user-generated
contents such as the videos on the YouTube platform). In practice, one should consider persistent content placement for the majority and a small cache to handle unexpected traffic by replacement, which will give a more balanced system.
REFERENCES


