An Approximation Algorithm to Maximize User Capacity for an Auto-Scaling VoD System

Zhangyu Chang and S.-H. Gary Chan, *Senior Member, IEEE*

***Abstract*—In a video-on-demand (VoD) service, blockbuster videos have stable and predictable popularity, but the traffic can vary significantly within short timescale. To efficiently serve the user pool in a geographic region, we consider a regional auto-scaling cloud-based data center consisting of multiple servers. For efficient storage, we partition the videos into fixed-size blocks. To respond to dynamic user traffic in a timely and cost-effective manner, we may activate or deactivate each server according to the traffic while keeping at least one replica for each block in the active servers. We maximize the user capacity of the active servers (and hence minimizing the number of active servers at any time) by jointly optimizing block allocation in the servers, server selection at each traffic level, and request dispatching to a server. We believe that this is the first work to study such problem for an auto-scaling cloud-based VoD data center. We first formulate the problem and show its NP-hardness. We then propose AVARDO (Auto-scaling Video Allocation and Request Distribution Optimization), a simple but efficient approximation algorithm with proven optimality. AVARDO operates the servers like a stack, with a server being pushed into or popped from the existing active server set according to some optimized traffic thresholds. We prove that AVARDO approaches the theoretical optimum as the block size reduces. Trace-driven experimental results based on large-scale real-world video data further validate that AVARDO is closely optimal. It achieves significantly higher user capacity as compared with other state-of-the-art and traditional schemes, and reduces the optimality gap by multiple times.**

***Index Terms*—Video-on-demand, auto-scaling cloud, video allocation, request dispatching, joint optimization.**

I. Introduction

We consider a video-on-demand (VoD) service (e.g., Netflix) that provides blockbuster videos to a large group of audience. In each geographic region, a cloud-based data center provides VoD service to the regional local users within its proximity. The popularity of blockbuster video is usually rather stable and predictable over days or weeks (in contrast to user generated content where popularity may be more volatile, changing drastically in minutes or hours). The total user request traffic presented to the system, on the other hand, may vary quite significantly within much shorter timescale. As shown in [1], the traffic may change by an order of magnitude over merely hours.

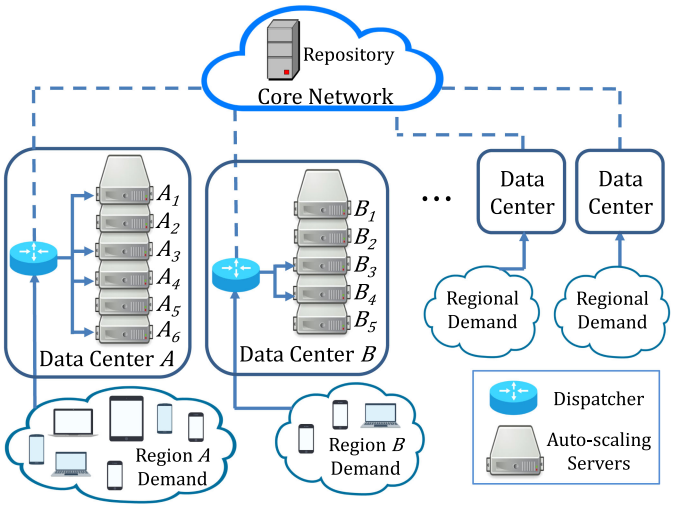


Fig. 1. A video cloud consisting of auto-scaling VoD data centers.

To serve highly dynamic user traffic in a region, the traditional infrastructure approach where the content provider statically allocates a fixed number of servers for the peak regional user demand and keeps them running all the time is no longer efficient. To meet demand in a timely and cost-effective manner, content providers may employ *auto-scaling* servers from a private cloud or a cloud service provider (e.g., Amazon AWS [2]), where standby servers may be activated or deactivated according to the user traffic.

Fig. 1 shows the system architecture of a typical video cloud, which consists of several regional auto-scaling VoD data centers[1] placed close to the user pools. (For simplicity, we show the user devices and internal architecture of data center in region A and B only.) Each data center has a *dispatcher* and a set of standby *auto-scaling servers* to serve its regional user demand. We consider the realistic and simple case of homogeneous servers in a data center. Each server has a certain limited storage and streaming capacity, and may be activated or deactivated within a short period of time (usually in tens of seconds). The dispatcher distributes the demand to the active servers or to

Manuscript received December 17, 2019; revised June 15, 2020 and August 24, 2020; accepted October 7, 2020. Date of publication October 15, 2020; date of current version October 19, 2021. This work was supported, in part, by Hong Kong General Research Fund under grant 16200120. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Mea Wang. *(Corresponding author: Zhangyu Chang.)*

The authors are with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong (e-mail: zchang@cse.ust.hk; gchan@cse.ust.hk).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TMM.2020.3031068.

Digital Object Identifier 10.1109/TMM.2020.3031068

[1]This work addresses the auto scaling feature of the VoD servers, and can be extended to CDN or edge server clusters if they support such feature.

the core network if the data center does not have the requested video.

We aim at optimizing these auto-scaling VoD data centers. In this network, a data center locally stores the content of wide interest to serving its user pool. The video lifetime is typically in the order of weeks, which is much longer than the timescale of user traffic fluctuation. By storing many videos, a data center captures most of its regional demand, and hence the core network traffic between data centers is expected to be minimal. As the data centers are operating rather independently, we focus on an arbitrary one in this work.

To operate a data center cost-effectively, we activate or deactivate the servers according to incoming traffic to elastically scale system resources. For example, data center A activates more servers than B because region A has more demand than B. Currently, A has activated server A_1, A_3, A_4 and A_6. If the demand in region A further increases, we can activate server A_2 or A_5. Conversely, if the demand decreases, we can deactivate some servers.

For efficient video storage, we consider the storage unit in our cloud as a video *block*. Each video block has the same size. If a blockbuster video has a larger file size, we partition this video into our fixed-size blocks. Note that our video block is different from a DASH segment such that our block is only for management purpose. Nevertheless, our design is amendable and extensible to adaptive streaming. A block can consist of multiple DASH segments depending on video's bitrate. When a user plays the video, the server streams the video to the user based on segments. After streaming the last segment in a block, the user moves to another block with the subsequent segments.

An incoming user demand for a video hence consists of multiple block accesses. For each block access, the dispatcher distributes it to an active server storing the requested block. To guarantee service, we clearly must have at least one replica for each video block in the active servers at any time. For example, if server B_3 and B_4 are necessary to keep the full replication in data center B, when the user demand further decreases, we still cannot deactivate any of them.

Let V be the set of all standby servers in the data center. The total block request rate λ (requests/second) is first mapped to an *auto-scaling level* i ($i = 0, 1, 2, \ldots$) such that λ is between some thresholds of λ_i and λ_{i+1} ($\lambda_i < \lambda_{i+1}$), with a corresponding predefined set of servers $V_i \subseteq V$ being activated which contains at least one replica of all video blocks. We show in Fig. 2 the mapping mechanism. For request rate less than λ_0 (the lowest auto-scaling level 0), servers in V_0 are activated. When the request rate is between λ_i and λ_{i+1}, servers in V_{i+1} are activated to serve the users. As the user request rate increases (decreases), we increase (decrease) the auto-scaling level so that more (fewer) servers are activated. Let $|V_0| = \nu$ be the number of servers in V_0, we clearly have $|V_i| = \nu + i$ for auto-scaling level i.

The deployment cost of such a system depends on the number of active servers. In order to minimize it, we hence seek to maximize *user capacity* in terms of λ_i at each auto-scaling level i. To maximize λ_i, we have to optimize the following interdependent design dimensions:

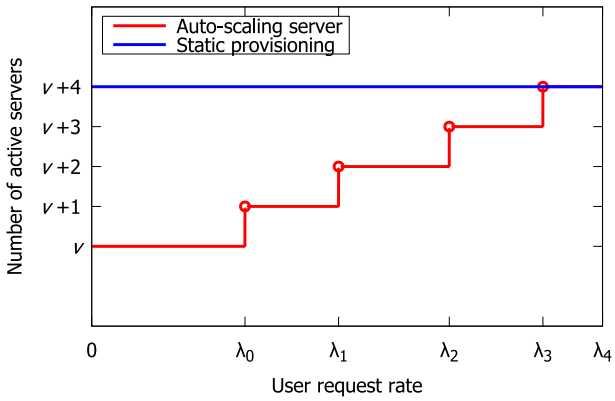


Fig. 2. Mapping mechanism of auto-scaling levels.

- *Block allocation (BA):* Due to limited storage on a server, a single server alone cannot store all the video blocks. Therefore, we need to decide which blocks should be allocated (or replicated) in each server, the so-called block allocation (BA) problem. Note that at auto-scaling level 0, we still need V_0 to be with enough total storage to cooperatively store at least one replica of every block in the whole video pool.
- *Server selection (SS):* This is to decide which servers should be activated (i.e., the set V_i) for each auto-scaling level i. Since different server may store different set of video blocks, we have to ensure that servers in V_i have enough replicas for each video block to support the request.
- *Request dispatching (RD):* Request dispatching is to decide which server to cater a block request. Since some blocks may be stored in multiple active servers, the dispatcher has to balance the load of each active server so as to minimize user delay or server utilization.

Note that, due to the relatively stable video popularity as compared with user traffic, on-the-fly BA is not necessary. We hence consider video blocks are preloaded in all the servers for SS and RD. Furthermore, BA has much longer timescale (in day or week) than SS (in hour), whose timescale is in turn much longer than RD (in second). Therefore, RD decision should be based on a given SS, while SS decision should be based on a given BA. Therefore, to maximize λ_i, we need to jointly optimize these three interdependent dimensions.

To the best of our knowledge, this is the first work to maximize user capacity for an auto-scaling cloud-based VoD data center by jointly optimizing block allocation, server selection and request dispatching. Our contributions are on the following:

- *Problem formulation and its NP-hardness:* We study the novel problem of maximizing user capacity for each auto-scaling level i (in terms of λ_i) for an auto-scaling VoD data center. We formulate the optimization problem as a multi-objective mixed-integer linear program, and prove that it is NP-hard. Our formulation is a general model such that, by allowing only a single auto-scaling level (i.e., $V_0 = V$), it becomes the optimization of the traditional fixed-server system.

- *Stack-based algorithm with proven approximation ratio:* To tackle the joint problem, we propose a novel and efficient approximation algorithm called AVARDO (**A**uto-scaling **V**ideo **A**llocation and **R**equest **D**ispatching **O**ptimization). AVARDO is a stack-based approach where the servers are arranged in a linear array and are *pushed* (activated) or *popped* (deactivated) according to the increment or decrement of auto-scaling level corresponding to the user traffic. AVARDO's overhead is low, because only one server is activated or deactivated between successive auto-scaling levels. We prove its approximation ratio to show its closely optimal performance. We show that the optimality gap can be further narrowed by reducing the block size in the system (i.e., videos are partitioned into smaller blocks).
- *Extensive trace-driven experimental study based on real-world data:* We conduct extensive trace-driven experiments with real-world VoD data (from a leading video service website in China) to evaluate AVARDO. Our results show that AVARDO's performance is close to the optimum, validating our theoretical analysis. Compared with other state-of-the-art and traditional schemes, AVARDO significantly lowers the number of active servers, and reduces the optimality gap (by multiple times).

The remainder of this paper is organized as follows. We first review related work in Section II. In Section III we describe our system model, formulate our joint problem and show its NP-hardness. We present AVARDO and prove its optimality in Section IV. We discuss illustrative trace-driven experimental results in Section V, and conclude in Section VI.

II. Related Work

Content replication over a cloud has been widely studied by abstracting a data center as a super server. The work in [3] elevates a traditional CDN to cloud paradigm and decomposes the problem into a graph partitioning and replica placement problems. Other work includes user access pattern detection at different geographical region [4], collaborative cache strategy [5]–[7], delivery through software-defined networking [8], [9] and social UGC propagation over a cloud CDN [10]–[12]. Content placement for a cloud-based VoD system has been discussed in [13]–[16]. Recent work on energy efficiency [17], femtocell networks [18] and optimization based on machine learning [19] provides sophisticated cost models and proposes impressive replication schemes to achieve low operational cost with QoS guarantees, but such research has yet to consider some of the important features of cloud computing inside the data center due to model abstraction. Our work, in contrast, complements to these studies by investigating video replication, server selection and traffic dispatching in auto-scaling data center from a more detailed point of view.

There has been previous work to address content replication problem in both traditional and cloud-based VoD data center [20]–[22]. Such work assumes that there is no dynamics within the data center, in which the server configurations and bandwidth reservation are rarely changed. Some other work [23]–[25] is scalable in terms of the number of requests, but has not considered the change of storage and video replication of the auto-scaling servers. Dynamic data replication [4] needs extra network and time cost to load the video content into the server dynamically, which is not necessary in our scenario as we preload the content due to the relatively stable video popularity. For auto-scaling servers, we have to optimize content replication for every possible auto-scaling levels, and adjust the traffic dispatching scheme to adapt to the changing environment. Furthermore, our trace-driven experimental results are based on large-scale real world data, which effectively validate the performance in the real-world settings.

How to efficiently auto-scale cloud resources has attracted much interest of researchers in recent years. The work in [26]–[30] focuses on effective predicting the user demand in order to scale up and down of the servers. The work in [31] considers the problem of reducing the response time for the auto-scaling features. Such work optimizes the auto-scaling system in the online phase such that it predict the user demand and improves the performance in the coming few hours, which is orthogonal to our work as the time scale we are considering is much longer. The related work and our work can work together to achieve better overall performance.

Various schemes have been proposed to address the cost optimization of an auto-scaling system. The work in [32]–[34] considers the general problem of jointly optimizing the resource allocation and server selection problem. The work in [35]–[39] explores how auto-scaling cloud can support live streaming video service. The work in [40] considers auto-scaling network to manage the camera surveillance network using the algorithm given in [41]. The work in [42], [43] considers management of traffic network through auto-scaling. These problems, while challenging, are different from our work because each request or task considered in the problems is served by only one server. For a VoD service, as some videos are too popular to be served by one server, we have to consider both replicating of the video files and dispatching the user requests. The approaches they are using cannot be directly applied to our problem.

Note that the research works on user demand prediction [44], [45], user start-up delay reduction, and server oscillation avoidance are orthogonal to ours. Advancement in these fields would benefit the performance of our auto-scaling VoD data center, which focuses on the maximization of the user capacity given dynamic user traffic.

III. Problem Formulation and Its NP-Hardness

In this section, we first describe the system model of a cloud-based VoD data center in Section III-A, and formulate the optimization problem in Section III-B. Then we show its NP-hardness in Section III-C. The major symbols used in the formulation are given in Table I.

A. System Model

As we partition the videos into fixed-size blocks in our system, a block is the basic storage unit (i.e., it is either entirely stored on a server or not at all). Each block has the same file size f (bits).

TABLE I
MAJOR SYMBOLS USED IN FORMULATION.

Notation	Definition
u	The streaming capacity of a server (bits/s)
c	The storage capacity of a server (bits)
f	The file size of a video block (bits)
V	The set of all standby servers in data center
V_i	The set of active servers at auto-scaling level i
ν	The number of servers in the set V_0 (i.e., $\nu = \|V_0\|$)
M	The set of all video blocks
M_v	The set of blocks stored on server v
λ	Total video block request rate (requests per second)
λ_i	Request rate threshold (requests per second) at auto-scaling level i ($i \geq 0$)
p^m	Access probability of block m
L^m	Average holding time of block m (in seconds)
$R^m(\lambda)$	Traffic of block m (bits/s) at request rate λ (i.e., $R^m(\lambda) = \lambda p^m L^m b^m$)
b^m	Streaming rate of block m (bits/s)
I_v^m	Binary variable indicating whether server v stores block m
$r_v^m(i)$	Probability of streaming a request of block m from server v at auto-scaling level i
μ	Server utilization limit to ensure quality-of-service

A smaller f leads to better optimality, but this comes with a substantial increase in management overhead of video blocks. Therefore, in practice, the block size cannot be too small to strike a good trade-off between system optimality and management complexity (though the transmission of these blocks may be in segments of very small size using, say DASH).

In adaptive streaming, a video may have different quality versions. The number of blocks of the high-quality version is more than its low-quality counterpart. In video transmission, when a change in end-to-end bandwidth is detected, the corresponding quality version of the video block is then identified, switched and streamed. As different blocks have different access probability, such adaptive streaming mechanism does not affect our problem formulation and our work can be extended to the case.

In our system, the video blocks are preloaded into the servers. Due to the much longer lifetime of blockbuster video and its relatively stable popularity as compared with auto-scaling decision interval, the interval between video update is much longer than a typical auto-scaling interval. Also, such preload is usually scheduled when the network traffic is low (e.g., early morning). Therefore, the cost to preload videos has little impact on our optimization problem, and we consider our auto-scaling optimization independently from preloading cost.

A cloud VoD data center is composed of a number of servers, which store videos and stream them to users. We denote the set of all standby servers in the data center as V. Each server $v \in V$ has the same storage capacity c (bits) and streaming capacity u (bits/second). With auto-scaling, the number of active servers can adapt to the change of user traffic. When the traffic increases, we activate more servers to ensure quality of service; when the traffic decreases, we deactivate some servers to reduce the cost.

To describe the system in different states (i.e., with different active server set), we define the auto-scaling levels by considering block request rate as thresholds. We numerate the *auto-scaling levels* as $\{0, 1, \ldots, n\}$ and the corresponding request rate threshold as $\{\lambda_0, \lambda_1, \ldots, \lambda_n\}$. We denote V_i, a subset of V (i.e., $V_i \subseteq V$), as the set of active servers when the system is at auto-scaling level i ($0 \leq i \leq n$). For V_0, we denote its number of servers as ν (i.e., $|V_0| = \nu$). Surely, for the maximum auto-scaling level n we have $V_n = V$, and for the number of active servers we have $|V_i| = \nu + i$.

At auto-scaling level 0 where the total block request rate is no more than λ_0 (i.e., $\lambda < \lambda_0$), servers in V_0 shall serve all requests, and we deactivate all the other servers. As we must ensure that we can serve the request of any video block, the servers in V_0 have to collectively store all the blocks $m \in M$ even if the request rate is minimum, namely, we must have $\nu c > |M| f$. At level i ($i \geq 1$) where the request rate is between λ_{i-1} and λ_i (i.e., $\lambda_{i-1} < \lambda < \lambda_i$), our active server set V_i has $\nu + i$ servers in total to fulfill the request. Note that, in our problem formulation, we do not enforce the stack operation (i.e., it is not necessary that $V_0 \subseteq V_1 \subseteq \ldots \subseteq V_n$).

B. Problem Formulation

Let M be the set of all video blocks. For a block $m \in M$, we denote the access probability as p^m ($\sum_{m \in M} p^m = 1$) and streaming rate as b^m (bits/second). Given the total block request rate λ (requests per second) and access probability p^m, the request rate for a block m is given as λp^m. We denote L^m as the average holding (or viewing) time for block m. As we consider the traffic at quasi-steady state, the distribution of the holding time may be different for different blocks, and we are interested in the average holding time. Denoting the traffic of a video block m at request rate λ as $R^m(\lambda)$ (bits/s), we have

$$R^m(\lambda) = \lambda p^m L^m b^m, \quad \forall m \in M. \tag{1}$$

To indicate whether server v stores block m, we denote M_v as the set of video blocks stored on server v, and the binary block allocation variable as

$$I_v^m = \begin{cases} 1, & \text{if } m \in M_v, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

As we do not change the block allocation within the timescale of optimization, the binary block allocation variables I_v^m are invariant for all the request rate λ. As a server cannot store blocks beyond its storage capacity limit c (bits), we have

$$\sum_{m \in M_v} I_v^m f \leq c, \quad \forall v \in V. \tag{3}$$

Having stored the video block set M_v, a server v has to serve the corresponding traffic. For auto-scaling level i, we denote $r_v^m(i)$ as the probability of streaming a request of block m from server v. A server $v \in V_i$ can serve the traffic of a block m only if it stores this block, i.e.,

$$r_v^m(i) \leq I_v^m, \quad \forall v \in V_i, m \in M, i = 0, 1, \ldots, n. \tag{4}$$

To ensure that we serve all the user requests for each block, we shall have $R^m(\lambda_i) \leq \sum_{v \in V_i} r_v^m(i) R^m(\lambda_i)$ for all $m \in M$, or

simply

$$\sum_{v \in V_i} r_v^m(i) \geq 1, \quad \forall m \in M, i = 0, 1, \ldots, n. \tag{5}$$

At request rate threshold λ_i, the traffic of block m served by server $v \in V_i$ is $r_v^m(i)R^m(\lambda_i)$ for all $m \in M$. The total traffic served by server v is given as $\sum_{m \in M} r_v^m(i)R^m(\lambda_i)$ for any $v \in V_i$. To ensure the video playback performance, the utilization of the streaming capacity of every server should not exceed a certain limit μ, i.e.,

$$\sum_{m \in M} r_v^m(i)R^m(\lambda_i) \leq \mu u, \quad \forall v \in V_i, i = 0, 1, \ldots, n. \tag{6}$$

The number of active servers of each auto-scaling level is another constraint, which is given as

$$|V_i| = \nu + i. \tag{7}$$

Recall that for V_0 we must have $\nu c > |M|f$ to store all the video blocks, and ν can be treated as a given parameter.

Formally, our **A**uto-scaling **V**ideo **A**llocation and **R**equest **D**ispatching (AVARD) problem is formulated as follows: given server set V, streaming capacity u, storage capacity c, video block set M, access probability $\{p^m\}$, file size f, average holding time $\{L^m\}$, streaming rate $\{b^m\}$ and server utilization limit μ, we seek to maximize all the request rate threshold $\{\lambda_i\}$, i.e.,

$$\max(\lambda_0, \lambda_1, \ldots, \lambda_n) \tag{8}$$

subject to constraints from (1) to (7). For every λ_i, the optimal solution consists of the block stored on each server (i.e., the block allocation decision $\{I_v^m\}$), the set of active servers (i.e., the server selection decision $\{V_i\}$) and the probability of streaming a request of video to a server (i.e., the request dispatching decision $\{r_v^m(i)\}$).

Note that $\{I_v^m\}$ is the same for all λ_i, but each λ_i has its own V_i and $\{r_v^m(i)\}$. For an arbitrary $\lambda < \lambda_i$, $R^m(\lambda)$ satisfies all the constraints from (1) to (7) by keeping the current $\{I_v^m\}$, V_i and $\{r_v^m(i)\}$.

C. The NP-Hardness of AVARD Problem

We prove that the *Partition Problem*, a known NP-complete problem in number theory, is polynomial-time reducible to AVARD. The partition problem is to decide whether a given multiset $S = \{s_1, s_2, \ldots, s_n\}$ of n positive integers can be divided into 2 subsets S_1 and S_2 such that the sums of the numbers in S_1 and S_2 are the same.

Given S, we construct an instance of AVARD with auto-scaling level 0 as follows. We denote the sum of all numbers in S as s. We have 2 servers v_1 and v_2 both with storage capacity $c = n$, streaming capacity $u = s/2 + n$ and utilization limit $\mu = 1$. We have $2n$ video blocks with size $f = 1$. The required traffic R^m of the first n blocks is related to the numbers in S (i.e., $R^m = R^m(\lambda_0) = s_m + 1$ for $1 \leq m \leq n$). The required traffic of the other n blocks is 1 (i.e., $R^m = 1$ for $n + 1 \leq m \leq 2n$). The decision version of AVARD is whether these 2 servers can accommodate all the required traffic.

Theorem 1: The partition problem is polynomial-time reducible to AVARD problem.

Proof: We show that we can partition the numbers in S into 2 subsets with the same sum if and only if the 2 servers can serve all the required traffic.

If there is a solution to AVARD that makes the servers to serve all the required traffic, the streaming capacity of both servers must be fully utilized (i.e., $\sum_{m \in M_{v_1}} R^m = \sum_{m \in M_{v_2}} R^m = s/2 + n$). As we have exactly $2n$ storage for $2n$ blocks, each block must have exactly one replica. To construct the solution to the partition problem, for every number in S, if its associated video block is in M_{v_1}, we put it in S_1, otherwise it is in S_2.

Conversely, if we can partition the numbers successfully, we construct the solution to AVARD as follows. For the numbers in S_1, we put the associated blocks in M_{v_1}, and the other blocks in M_{v_2}. The unused storage shall be filled with blocks with the required traffic $R^m = 1$. ■

IV. AVARDO: An Approximation Algorithm for Auto-Scaling Block Allocation and Request Dispatching

In this section, we present AVARDO (**A**uto-scaling **V**ideo **A**llocation and **R**equest **D**ispatching **O**ptimization) to jointly optimize video allocation, server selection, and request dispatching for a large video pool.

To address the auto-scaling, AVARDO has a stack-based *server selection* scheme such that, besides the servers in V_0, we arrange servers in an orderly sequence $\{v_1, v_2, \ldots, v_n\}$ and consider the set of active servers as a stack, namely, $V_{n+1} = V_n \cup \{v_{n+1}\}$. Each server stores a predefined set of video with a good mix of both hot and cold contents. We denote v_0^n $(1 \leq n \leq \nu)$ as the servers in V_0 which are always in the stack for any auto-scaling levels. When the request rate exceeds threshold λ_n, we push (activate) server v_{n+1} on the stack. When the request rate falls under threshold λ_n, we just pop (deactivate) the top server v_{n+1} off the stack. After the stack operations, we update the request dispatching accordingly.

In Section IV-A, we propose the preprocessing stages of AVARDO where we replicated the video blocks and put them into clusters for easier management. In Section IV-B, we propose the details of AVARDO for all the auto-scaling levels. In Section IV-C, we show the optimality gap by comparing λ_n given by AVARDO and the theoretical upper bound $\bar{\lambda}_n$. The major symbols used in this section are given in Table II.

A. Preprocessing: Block Replication and Clustering

To reduce the complexity of our optimization, we need to replicate popular video blocks and put the blocks into clusters. By replicating popular blocks, we avoid the situation that too many users demand the same block. By clustering the blocks, we can let each cluster be a mega video file with the same size and nearly same access probability. Then, the BA decision of AVARDO simply becomes picking clusters for each server instead of choosing numerous video blocks.

Denoting G as the set of video clusters, we aims at putting the videos into ν^2 video clusters $g \in G$ such that each server gets ν clusters. At auto-scaling level 0, servers in V_0 have all the ν^2 clusters to ensure the full replication of the video blocks.

TABLE II
MAJOR SYMBOLS USED IN AVARDO ALGORITHM

Notation	Definition
v_i	The server to activate when auto-scaling level goes from $i-1$ to i (i.e., $V_i = V_{i-1} \cup \{v_i\}$)
P^m	Streaming ratio of block m
N^m	Number of replicas for block m stored in V_0
N_{T}	Number of replicas can be stored in V_0
N_{A}	Number of surplus replicas in V_0 (i.e., $N_{\mathrm{T}} - \|M\|$)
σ^m	Average replica streaming ratio of block m
σ	Average replica streaming ratio threshold
G	The set of video clusters
$G(v)$	The set of video clusters on server v
G_k	The set of video clusters that have k replicas
$P(g)$	Total streaming ratio of replicas in cluster g
$C(g)$	Storage capacity used for cluster g
q_g^m	Probability of streaming a request of block m from cluster g at auto-scaling level 0
λ_{op}	Theoretical upper limit of λ threshold

For a server v which is not in V_0, v and any server in V_0 always have a same cluster. When a new server v is activated, its preloaded contents can effectively offload the traffic from the existing active servers.

To clarify the relation between the storage and streaming, we denote the *streaming ratio* of video block m as P^m such that

$$P^m = \frac{p^m L^m b^m}{\sum_{m \in M} p^m L^m b^m}, \quad \forall m \in M. \tag{9}$$

According to (1), P^m is proportional to the traffic of block m (e.g., a block m with $P^m = 0.1$ accounts for 10% of the total traffic). Clearly, we have $\sum_{m \in M} P^m = 1$.

Denote N^m as the number of replicas for block m stored in V_0. For each replica of block m, its average streaming ratio σ^m is defined as

$$\sigma^m = P^m / N^m, \quad \forall m \in M. \tag{10}$$

Note that σ^m is only for auto-scaling level 0, and it is not necessary to evenly distribute the request of a block m to its replicas.

We denote $P(g)$ as the streaming ratio distributed to cluster g, which has

$$P(g) = \sum_{m \in g} \sigma^m, \quad \forall g \in G. \tag{11}$$

The ideal situation is that the traffic is evenly distributed to each cluster, (i.e., $P(g) = 1/\nu^2$ for all n).

Therefore, AVARDO consists of two preprocessing stages:

- The *block replication* stage decides how many replicas are required for a video block (i.e., N^m).
- The *replica clustering* stage decides which replicas are in a cluster (i.e., g).

The *block replication* is a popularity-based (in terms of P^m) scheme with the following properties:

1) The least popular block has at least one replica in V_0 (i.e., $N^m \geq 1$).
2) For the most popular blocks m, each server has at most one replica (i.e., $N^m \leq \nu$).
3) For the other blocks, N^m is proportional to P^m.

Algorithm 1: AVARDO Replica Clustering.

```
Initialization: P(g) = 0, C(g) = 0, ∀g ∈ G;
Put all partially replicated replicas into priority queue ℚ;
while ℚ ≠ ∅ do
    Pop top ν² replicas with max σ^m from ℚ;
    Put these ν² replicas into priority queue ℚ_m;
    Put clusters g ∈ G into priority queue ℚ_g;
    while ℚ_M ≠ ∅ do
        Pop the replica m with max σ^m from ℚ_m;
        Pop the cluster g with min P(g) from ℚ_g;
        Store a replica m in g: g ← m;
        Update parameters: P(g) += σ^m, C(g) += f;
    end
end
```

Denote $N_{\mathrm{T}} = \nu c / f$ as the number of replicas can be stored in V_0. As V_0 must store all the video blocks, we must have $N_{\mathrm{T}} \geq |M|$, and the number of surplus replicas is denoted as $N_{\mathrm{A}} = N_{\mathrm{T}} - |M|$. If V_0 can only store $|M|$ replicas (i.e., $N_{\mathrm{T}} = |M|$), we skip this block replication stage. If we have extra storage (i.e., $N_{\mathrm{A}} > 0$), as the required traffic of a hot video block may be more than the streaming capacity of a server, we wish to store such block into multiple servers so that each server only needs to serve a fraction of its required traffic.

We introduce a tunable parameter called *average replica streaming ratio threshold* σ such that $\sigma^m \leq \sigma$ for all the blocks with $N^m < \nu$. In other words, besides the blocks that have replicas in all the servers, each replica is expected to accommodate at most σ of the total traffic. Therefore, we have

$$N^m = \begin{cases} \nu, & \text{if } P^m > \nu\sigma, \\ \lceil P^m/\sigma \rceil, & \text{if } \sigma < P^m \leq \nu\sigma, \\ 1, & \text{if } P^m \leq \sigma. \end{cases} \tag{12}$$

For blocks that has replicas in all the servers (i.e., $N^m = \nu$), we call them as *fully replicated* blocks. For the other blocks, we call them as *partially replicated* blocks.

A smaller σ will increase the number of replicas. To avoid wasting storage, we find the smallest possible σ through binary search. In the binary search, we can set the initial lower bound of σ as $\sigma_L = 1/N_{\mathrm{T}}$ and upper bound as $\sigma_H = 1/N_{\mathrm{A}}$. In each iteration of the search, we let $\sigma = (\sigma_L + \sigma_H)/2$. If σ makes too many replicas, we let σ be the new σ_L. If σ makes too few replicas, we let σ be the new σ_H. We keep iterating until we find the suitable σ.

For the *replica clustering*, we denote $C(g)$ as the used storage capacity for cluster g. Initially, we let $P(g) = 0$ and $C(g) = 0$. We first put all the replicas of partially replicated blocks into a priority queue $\mathbb{Q}$. We then repeatedly take top ν^2 replicas with maximum σ^m from the queue $\mathbb{Q}$ and let every cluster take a replica such that the cluster g with smaller $P(g)$ can get a replica with larger σ^m. We give the pseudocode in Algorithm 1.

The preprocessing has efficient algorithmic time complexity. It has been shown that searching for σ can be done in $O(|M|)$ [46]. The major component of clustering is to get the replicas from the priority queue, which is equivalent to sort the replicas

by their σ^m. As the number of replicas is comparable to $|M|$, the time complexity of preprocessing is $O(|M| \log |M|)$.

B. Block Allocation and Request Dispatching

After the preprocessing stage, the optimization becomes how to manage video clusters. For the auto scaling level $i > 0$, we can write $i = k\nu + j$ such that $k \geq 0$ and $1 \leq j \leq \nu$.

We make the *block allocation* decisions as follows:

- All the servers shall store fully replicated blocks.
- For the servers in V_0, we distribute the ν^2 clusters into the ν servers such that each server $v \in V_0$ stores ν clusters.
- For server v_i such that $i \leq \nu$, it shall pick one cluster from each server $v \in V_0$ where the cluster has not been picked by the other server v_l such that $l \leq \nu$.
- For server v_i such that $i = k\nu + j$ with $k \geq 1$, we simply let $G(i) = G(j)$ (i.e., server v_i and v_j have the same block replication).

As some clusters may have the same video blocks and these clusters may be put into the same server, a server can have multiple replicas of the same video block. In this case, the solution may be sub-optimal, but we still have a good approximation ratio as shown in Section IV-C.

For *traffic dispatching*, we hope to evenly distribute the traffic to each server. It can be solved as a MaxFlow or Linear Program problem, but we have a very simple closed-form solution. The traffic of the fully replicated blocks is evenly distributed to each server. For the partially replicated blocks at auto-scaling level i, we consider the $i = 0$ and $i > 0$ cases separately.

We first consider the $i = 0$ case. For all $g \in G$ and $m \in M$, we denote q_g^m as the probability of streaming a request of block m from cluster g at auto-scaling level 0, which is given as

$$q_g^m = \begin{cases} 1/N^m, & \text{if } m \in g; \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

For fully replicated blocks, the traffic is evenly distributed to each server. We denote $G(v)$ as the set of video clusters on server v. The probability of streaming request $r_v^m(0)$ for the partially replicated block is given as

$$r_v^m(0) = \sum_{g \in G(v)} q_g^m, \quad \forall m \in M, v \in V_0. \tag{14}$$

Note that our AVARDO works for any feasible V_0. We can set a larger ν for better request dispatching flexibility at the cost of more active servers at auto-scaling level 0. By simply letting $V_0 = V$, AVARDO can also be applied to the traditional static provisioning VoD data center.

We then consider the $i > 0$ case. At each level i, server v_i is added to the stack. We want v_i to evenly offload the traffic of the other servers. At auto-scaling level $i = k\nu + j$, some clusters have $k + 1$ replicas while the others have $k + 2$ ones. We denote the set of the former clusters as G_{k+1}, and the set of the latter clusters as G_{k+2}.

The basic idea for RD is that each server shall have the same traffic by adjusting the traffic distributed to the clusters, which can be formulated as a linear equation. By solving this equation, we get the following result:

- For the servers $v \in v_1, \ldots, v_i$, we have

$$r_v^m(i) = \frac{\nu}{\nu + i} \sum_{g \in G(v)} q_g^m, \quad \forall m \in M. \tag{15}$$

- For the servers in $v \in V_0$, denoting $G_x = G(v) \cap G_{k+2}$ and $G_y = G(v) \cap G_{k+1}$, we have

$$r_v^m(i) = \frac{j}{\nu + i} \sum_{g \in G_x} q_g^m + \frac{\nu + j}{\nu + i} \sum_{g \in G_y} q_g^m, \tag{16}$$

for all $m \in M$.

C. Optimality Gap

If the exact optimal solution has request rate threshold λ_{op}, and our solution has threshold λ, the approximation ratio is given as $\lambda_{\text{op}}/\lambda$. As the approximation ratio is always greater than 1, we can define the *optimality gap* as $\lambda_{\text{op}}/\lambda - 1$. A smaller optimality gap indicates a better performance of the solution.

We show the optimality gap of AVARDO by proving the following lemmas and theorem. In Lemma 1, we show the upper bound of the average replica streaming ratio threshold σ. In Lemma 2, we show the upper bound of the streaming ratio of every video cluster $g \in G$. In Theorem 2, we calculate the optimality gap of AVARDO.

Lemma 1: σ is less than $1/N_{\text{A}}$.

Proof: According to (12), a block m with P^m streaming ratio has at most $\lceil P^m/\sigma \rceil$ replicas. As $\lceil P^m/\sigma \rceil < P^m/\sigma + 1$ and $\sum_{m \in M} P^m = 1$, the total number of replicas $\sum_{m \in M} N^m = N_{\text{T}} = N_{\text{A}} + |M|$ is less than $\sum_{m \in M}(P^m/\sigma + 1) = \sum_{m \in M} P^m/\sigma + |M| = 1/\sigma + |M|$. As $N_{\text{A}} + |M| < 1/\sigma + |M|$, we have $\sigma < 1/N_{\text{A}}$. ■

Lemma 2: For every video cluster $g \in G$, its streaming ratio $P(g)$ is no more that $1/\nu^2 + \sigma$.

Proof: We prove this by mathematical induction. We show that, at each iteration of putting a replica into each cluster in our replica clustering algorithm, the difference between $\max P(g)$ and $\min P(g)$ is always no greater than σ (i.e., $\max P(g) - \min P(g) \leq \sigma$).

We first consider the base case where there is only one replica in each cluster. As the largest σ^m is no greater than σ, it is obvious that $\max P(g) - \min P(g) \leq \sigma$.

Suppose that when we have k replicas in each cluster, $\max P(g) - \min P(g) \leq \sigma$ still holds. When we put the $(k + 1)$th replica into each cluster, we always put the replica with the larger σ^m into the cluster with smaller $P(g)$. Therefore, $\max P(g) - \min P(g)$ shall not increase, and $\max P(g) - \min P(g) \leq \sigma$ holds.

As we have ν^2 clusters, on average, each cluster has $1/\nu^2$ streaming ratio. For all $g \in G$, $P(g) \leq 1/\nu^2 + \sigma$. ■

Theorem 2: The optimality gap of AVARDO, given by $\lambda_{\text{op}}/\lambda - 1$, is no more than $\nu^2\sigma$.

Proof: In the ideal case, each cluster has $P(g) = 1/\nu^2$ so the traffic can be evenly distributed to servers. Consider the worst case where a server has all clusters with $P(g) = 1/\nu^2 + \sigma$. For this server, the ratio of its traffic versus the average server traffic is $(1/\nu^2 + \sigma)/(1/\nu^2) = 1 + \nu^2\sigma$. As the most overloaded server has $1 + \nu^2\sigma$ traffic compared with the ideal case, to meet

TABLE III
BASELINE OF THE PARAMETERS

Parameter	Baseline value
Number of blocks $\|M\|$	around 3×10^6
Block request rate λ (requests/s)	$2,000$
Number of blocks in a server c/f	6×10^5
Server streaming capacity u (Gbps)	25
Server utilization limit μ	0.9

the constraint of streaming capacity utilization in (6), we have to reduce λ threshold such that $\lambda_{\text{op}}/\lambda = 1 + \nu^2\sigma$. Thus, in the worst case scenario, the approximation ratio of our algorithm is $1 + \nu^2\sigma$, and the optimality gap is $\nu^2\sigma$. ■

Consider the real-world settings, a nowadays video server can easily have over 10 TB storage, which can store more than 10^5 videos. Therefore, it is easy to have $N_A \geq 10^5$ and $\sigma \leq 10^{-5}$. For auto-scaling level 0, 30 servers are more than enough. Therefore, the optimality gap $\nu^2\sigma$ is less than 1%. We can further reduce σ by partitioning the video files into blocks with smaller size f.

Note that Theorem 2 gives the upper bound of the optimality gap, which means AVARDO performs no worse than this bound. There exists the probability that AVARDO performs as bad as this bound, but such poor performance may not happen in reality. The worst cases (i.e., AVARDO meets the upper limit) only occurs when the file size of video block f is comparable to the server storage c, or when the number of video blocks $|M|$ is comparable to the number of servers ν, which are both far from the reality.

V. ILLUSTRATIVE EXPERIMENTAL RESULTS

In this section, we present experimental results based on real-world data trace of AVARDO. In Section V-A, we first describe our experimental settings and performance metrics. In Section V-B, we present illustrative results of AVARDO.

A. Experimental Environment and Performance Metrics

The real-world data trace for the experiment is from a leading video service website in China over 2 weeks. The website has around one hundred million daily active users (DAU). To collect the user trace, the video player on the client side reports the user status (e.g., videos being played and their network traffic) every minute. We only consider the long videos where the length of the video is longer than 10 minutes. There are around 1.5 million videos in total, where the top 20,000 videos account for more than 60% of the total traffic. When a video has multiple resolutions and bit rates, we treat them as multiple video files. In our baseline parameters, we partition the videos into the blocks of the same size of 100 MB, and we have around 3×10^6 blocks. At peak hour, there are around 2,500 block requests per second. A server can store 6×10^5 such blocks. Unless otherwise stated, we used this trace data and the baseline value given in Table III in our system parameters.

As AVARDO is applicable to any block access probability distribution, we also use synthetic data in Fig. 8 and 9 where the block access probability follows the Zipf distribution with

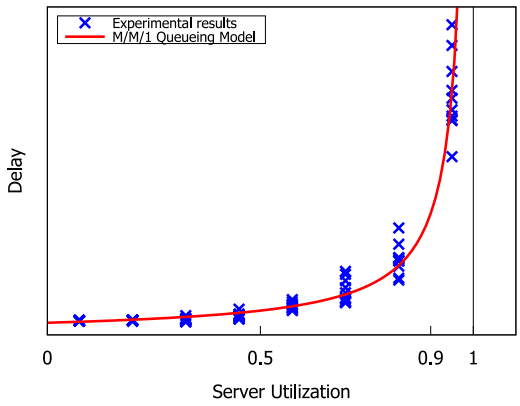


Fig. 3. Delay model for an auto-scaling server.

Zipf parameter z. For instance, the mth popular block has the access probability $p^m \propto 1/m^z$. In the experiment, the block access probability is generated and normalized according to the power-law x^{-z} with x as the rank and z as the *Zipf's parameter*. A greater Zipf's parameter indicates a wider gap of the video block access probability between different ranks.

We show in Fig. 3 the relationship between the server delay and utilization. The delay increases quite sharply when the server is getting fully utilized, and the experimental results agree with the M/M/1 queueing model. Due to the highly convex nature of the curve, the servers should be uniformly loaded to achieve overall low delay. A high fairness index indicates that no server's utilization is much higher than the other servers, and the load on all the active servers is well balanced to achieve overall low delay. As fairness is a good indicator of delay, we focus on the fairness of active server utilization in the experiment.

We compare AVARDO with the following traditional and state-of-the-art video replication schemes:

- *Uniform replication*, where every video has the same number of replicas. The videos are randomly stored in the servers.
- *Hierarchical popularity replication* [47], [48], where we have 2 types of server: repository and cache. The repository servers V_0 collaboratively store all the videos, and the cache stores the videos based on video popularity. A video with higher popularity has a higher chance to be stored in the cache.
- *Super optimum*, which serves as the theoretical performance bound (i.e., no scheme can perform better than super optimum). In super optimum, we assume that a video can be partitioned into blocks with infinitesimal size (i.e., $f \to 0$). As a smaller f indicates smaller σ, when $f \to 0$ our optimality gap $\nu^2\sigma \to 0$. The super optimum cannot be put into practice because it divides a video into too many blocks, and the user has to set up the same number of connections to fetch the whole video, which will incur a lot of overhead.

For request dispatching of the comparison schemes, we use the optimal dispatching strategy by solving the MaxFlow problem. Note that the algorithmic complexity of MaxFlow

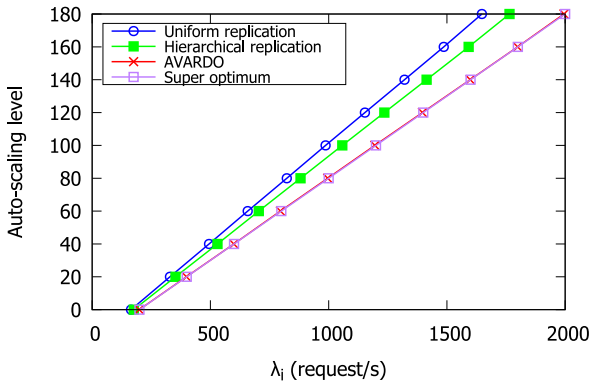


Fig. 4. Maximum request rate threshold versus auto-scaling level.

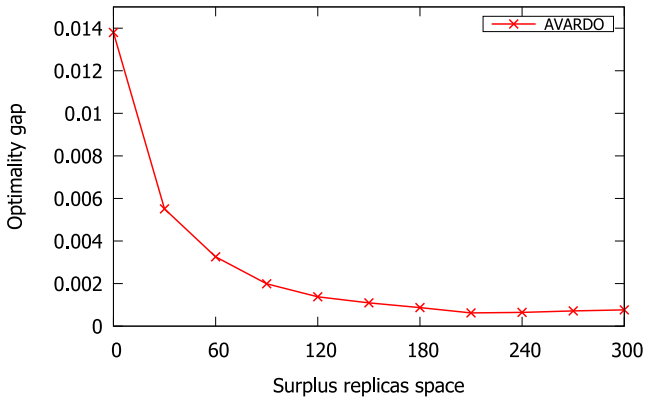


Fig. 5. Optimality gap versus number of surplus replicas.

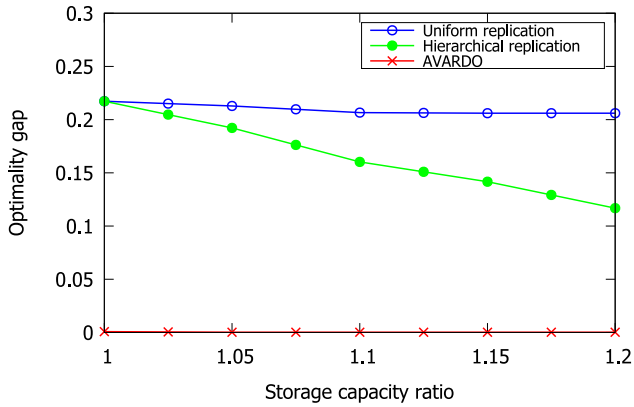


Fig. 6. Optimality gap versus storage capacity ratio.

is $O(|M|^2|V|)$, which cannot be applied in a real world system. Comparatively, the complexity of AVARDO's RD is only $O(|M||V|)$.

The performance metrics we are interested in are:

- *Request rate threshold* λ_n, which is the optimization objective of AVARDO.
- *Optimality gap* of λ_n, which reflects the difference between scheme performance and the theoretical bound. Optimality gap can be calculated as $(\lambda_{\text{op}}/\lambda_n) - 1$ where λ_{op} is the result of the *super optimum*.
- *Number of active servers*, which is used to evaluate the operation cost over a given time period.
- *Fairness of active server utilization*, which shows how the load is distributed among active servers. We use the Jain's Fairness Index to indicate the fairness, which is between 0 and 1. The higher is the index, the fairer the load is shared.

B. Illustrative Data-Driven Experimental Results

We compare in Fig. 4 the maximum request rate threshold λ_i versus the auto-scaling level for different schemes. The maximum request rate threshold increases with auto-scaling level as we increase the number of active servers. AVARDO is very close to the super optimum, and achieves significantly higher request rate threshold than the other 2 schemes. In other words, given the same request rate (i.e., concurrent users in the system), AVARDO uses much fewer number of active servers. Hierarchical replication's performance is not as good as AVARDO because it mainly relies on the repository to serve the unpopular videos. Uniform replication, due to its popularity-blind nature, stores insufficient replicas of the popular videos, leading to even poorer performance. As the optimality gap is quite stable for each auto-scaling level, for the following graphs, we use optimality gap as the y-axis to compare the schemes.

In Fig. 5, we examine the effect of the surplus replica space N_A on the optimality gap. A larger N_A allows AVARDO to have more replicas for the popular video blocks. We alter the number of extra storage N_A in the server set V_0. AVARDO performs well as it can converge to the super optimum with very little need of the extra storage. Even when $N_A = 0$ (i.e., we cannot replicate hot contents), AVARDO still has a small optimality gap. In reality, the operator does not need to care much about the extra storage issue as the performance is far better than the theoretical bound.

In Fig. 6, we examine the effect of the storage capacity ratio (given by $N_T/|M|$) on the optimality gap. We alter the server storage capacity ratio by changing the server storage capacity. AVARDO is less sensitive to the storage capacity ratio as it has already been close to the optimum. Uniform replication has constant performance because it cannot utilize the extra space due to the popularity-blindness. Hierarchical replication has better performance with a large storage capacity ratio because it has more space to store unpopular videos in both repository and cache.

In Fig. 7, we examine the effect of block size on the optimality gap. We alter the number of video blocks stored in a server by changing the block file size. All three schemes tend to have better performance with larger number of blocks in a server (i.e., smaller block size) because the traffic of popular video can be distributed to more servers if we partition it into more blocks. AVARDO is less sensitive to the video file sizes as it already has very small optimality gap for larger block size. The optimality gap of AVARDO and the comparison schemes differ by a wide margin until the block size becomes small enough. Note that our baseline parameter (i.e., 100 MB block size) is the knee point of AVARDO's performance curve in this system. When the block size further decreases, the performance gain is negligible.

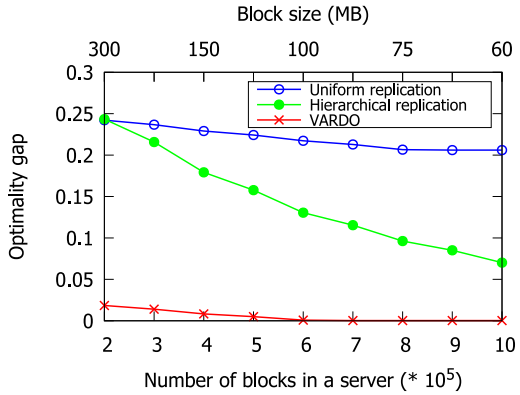


Fig. 7. Number of blocks in a server versus optimality gap.

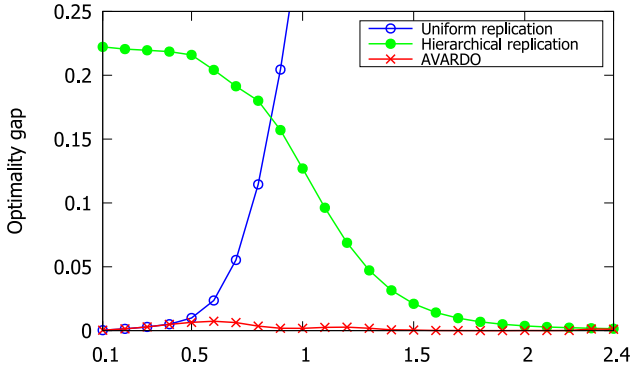


Fig. 8. Optimality gap versus Zipf's parameter of block access probability distribution.

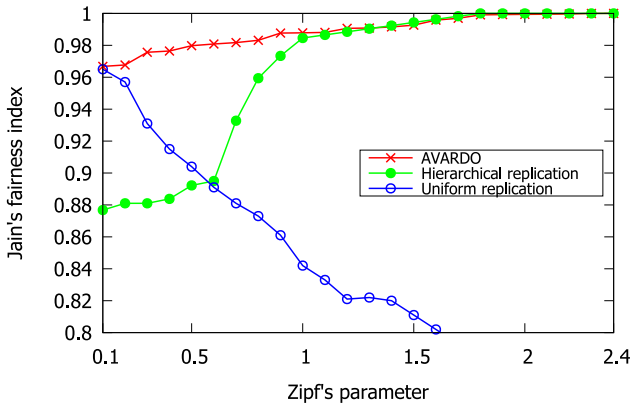


Fig. 9. Utilization fairness versus Zipf's parameter of block access probability distribution.

Fig. 8 and 9 use synthetic user trace data which follows the Zipf distribution.

We show in Fig. 8 how the skewness factor affects the performance of the three schemes. AVARDO is very insensitive to skewness of block access probability. The optimality gap of AVARDO is rather stable over the whole range of the skewness factor. With small Zipf's parameter, the difference of the blocks is small, so uniform replication is indeed the close-to-optimum solution. However, its performance degrades drastically with larger Zipf's parameter. Hierarchical replication does not perform well for small Zipf's parameter because the popular videos in its cache actually do not have so many requests.

We show in Fig. 9 the utilization fairness versus the Zipf's parameter of block access probability distribution. AVARDO has an overall good performance on fairness as we have a good mixture of hot and cold contents in every server. The high fairness index indicates that AVARDO can effectively balance the load to all the active servers to achieve overall low delay. When the Zipf's parameter is higher, hierarchical replication replicates hot contents at the newly activated servers, which will share a significant amount of load. This leads to higher utilization fairness. Uniform replication preforms worse as we increase the Zipf's parameter because it randomly put the hot contents into the servers. The fairness degrades when the hot contents have a larger fraction of user demand. Note that the comparison schemes may perform better than AVARDO in some scenario as they use the optimal dispatching algorithm, which has higher algorithmic complexity than AVARDO.

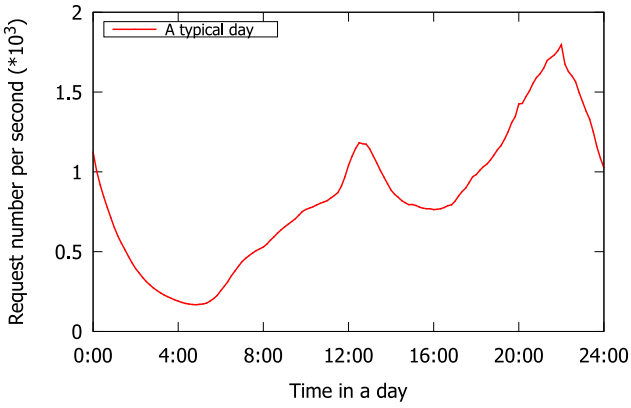


Fig. 10. The request rate over a typical day.

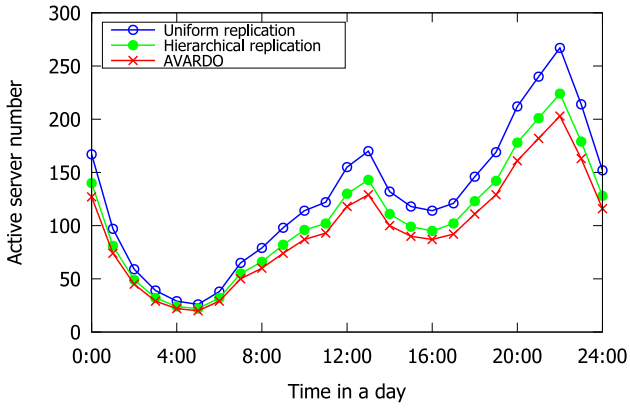


Fig. 11. Number of active servers over a typical day.

We show in Fig. 10 the trend of user request over a typical day in a large video service website. Request traffic can vary by an order of magnitude over a day. We plot in Fig. 11 the number of active servers over a typical day given different schemes. AVARDO uses significantly less number of active servers, which agrees with the result shown in Fig. 4. At the peak hour, AVARDO can save more numbers of active servers.

VI. Conclusion

In this work, we have examined the problem of providing blockbuster VoD service in a geographic region. To respond to the dynamic user traffic in a cost-effective manner, we have considered a regional auto-scaling cloud-based data center where servers may be activated or deactivated at any time. User traffic is mapped to one of the auto-scaling levels where a certain set of servers are activated. Videos are divided into fixed-size blocks. To minimize cost, we seek to maximize user capacity at each auto-scaling level by jointly optimizing block allocation at the servers, server selection and request dispatching.

We have formulated the problem as a Multi-objective Mixed-integer Linear Programming problem, and have shown that the problem is NP-hard. We have proposed AVARDO, a novel and efficient algorithm for a very large video pool with $O(|M|\log|M|)$ algorithmic complexity, where $|M|$ is the number of video blocks. AVARDO is a stack-based scheme, and has a proven optimality gap of $\nu^2\sigma$, where ν is the number of active servers at auto-scaling level 0 and σ is the average replica streaming ratio threshold. The optimality gap is less than 1% under practical settings. The approximation solution can further approach the theoretical optimum as we reduce the block file size in the optimization. We conduct extensive trace-driven experiments under real-world settings. The results agree with the theoretical proofs, and validate that AVARDO is closely optimal. It outperforms substantially the traditional and state-of-the-art schemes, narrowing the optimality gap by multiple times.

References

[1] N. Liu, H. Cui, S.-H. G. Chan, Z. Chen, and Y. Zhuang, "Dissecting user behaviors for a simultaneous live and VoD IPTV system," *ACM Trans. Multimed. Comput. Commun. Appl.*, vol. 10, no. 3, pp. 23:1–23:16, Apr. 2014.

[2] A. W. Services, "Amazon web services," accessed date October 20, 2020, [Online]. Available: http://aws.amazon.com

[3] C. Papagianni, A. Leivadeas, and S. Papavassiliou, "A cloud-oriented content delivery network paradigm: Modeling and assessment," *IEEE Trans. Dependable Secure Comput.*, vol. 10, no. 5, pp. 287–300, Sept./Oct. 2013.

[4] M. Jeon, K.-H. Lim, H. Ahn, and B.-D. Lee, "Dynamic data replication scheme in the cloud computing environment," in *Proc. IEEE 2nd Symp. Netw. Cloud Comput. Appl.*, 2012, pp. 40–47.

[5] G. Silvestre, S. Monnet, R. Krishnaswamy, and P. Sens, "AREN: A popularity aware replication scheme for cloud storage," in *Proc. IEEE 18th Int. Conf. Parallel Distrib. Syst.*, 2012, pp. 189–196.

[6] S. H. G. Chan, W. P. Yiu, and A. C. P. Lui, "Peer-to-peer interactive media-on-demand," U.S. Patent 9,325,786, Apr. 26 2016.

[7] S. H. G. Chan and W. P. Yiu, "Distributed storage to support user interactivity in peer-to-peer video streaming," U.S. Patent 7,925,781, Apr. 12 2011.

[8] J. Yang *et al.*, "Software-defined multimedia streaming system aided by variable-length interval in-network caching," *IEEE Trans. Multimedia*, vol. 21, no. 2, pp. 494–509, Feb. 2019.

[9] E. Bourtsoulatze, N. Thomos, J. Saltarin, and T. Braun, "Content-aware delivery of scalable video in network coding enabled named data networks," *IEEE Trans. Multimedia*, vol. 20, no. 6, pp. 1561–1575, Jun. 2018.

[10] Z. Wang *et al.*, "Propagation-based social-aware replication for social video contents," in *Proc. 20th ACM Int. Conf. Multimedia*, 2012, pp. 29–38.

[11] H. Hu *et al.*, "Joint content replication and request routing for social video distribution over cloud CDN: A community clustering method," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 26, no. 7, pp. 1320–1333, Jul. 2016.

[12] J. Tang, X. Tang, and J. Yuan, "Traffic-optimized data placement for social media," *IEEE Trans. Multimedia*, vol. 20, no. 4, pp. 1008–1023, Apr. 2018.

[13] D. Niu, C. Feng, and B. Li, "A theory of cloud bandwidth pricing for video-on-demand providers," in *Proc. IEEE INFOCOM*, 2012, pp. 711–719.

[14] D. Niu, H. Xu, B. Li, and S. Zhao, "Quality-assured cloud bandwidth auto-scaling for video-on-demand applications," in *Proc. IEEE INFOCOM*, 2012, pp. 460–468.

[15] D. Deng, Z. Lu, W. Fang, and J. Wu, "CloudStreamMedia: A cloud assistant global video on demand leasing scheme," in *Proc. IEEE Int. Conf. Services Comput.* , 2013, pp. 486–493.

[16] S.-H. G. Chan and Z. F. Xu, "LP-SR: Approaching optimal storage and retrieval for video-on-demand," *IEEE Trans. Multimed.*, vol. 15, no. 8, pp. 2125–2136, Dec. 2013.

[17] O. Ayoub, F. Musumeci, M. Tornatore, and A. Pattavina, "Energy-efficient video-on-demand content caching and distribution in metro area networks," *IEEE Trans. Green Commun. Netw.*, vol. 3, no. 1, pp. 159–169, Mar. 2019.

[18] Z. Hajiakhondi-Meybodi, J. Abouei, and A. H. F. Raouf, "Cache replacement schemes based on adaptive time window for video on demand services in femtocell networks," *IEEE Trans. Mobile Comput.*, vol. 18, no. 7, pp. 1476–1487, Jul. 2019.

[19] Y. Zhang *et al.*, "Geo-edge: Geographical resource allocation on edge caches for video-on-demand streaming," in *Proc. 4th Int. Conf. Big Data Comput. Commun.*, Aug. 2018, pp. 189–194.

[20] D. Apple gate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan, "Optimal content placement for a large scale VoD system," in *Proc. ACM Co NEXT*, Nov. 2010, Art. no. 4.

[21] S. Borst, V. Gupt, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.

[22] H. Zhao, Q. Zheng, W. Zhang, B. Du, and H. Li, "A segment-based storage and transcoding trade-off strategy for multi-version VoD systems in the cloud," *IEEE Trans. Multimedia*, vol. 19, no. 1, pp. 149–159, Jan. 2017.

[23] G. Gao, Y. Wen, W. Zhang, and H. Hu, "Cost-efficient and QoS-aware content management in media cloud: Implementation and evaluation," in *Proc. IEEE Int. Conf. Commun.*, 2015, pp. 6880–6886.

[24] L. De Cicco, S. Mascolo, and D. Calamita, "A resource allocation controller for cloud-based adaptive video streaming," in *Proc. IEEE Int. Conf. Commun. Workshops*, 2013, pp. 723–727.

[25] C. X. Cai, G. Liang, and U. C. Kozat, "Load balancing and dynamic scaling of cache storage against zipfian workloads," in *Proc. IEEE Int. Conf. Commun.*, 2014, pp. 4208–4214.

[26] M. Fallah, M. G. Arani, and M. Maeen, "NASLA: Novel auto scaling approach based on learning automata for web application in cloud computing environment," *Int. J. Comput. Appl.*, vol. 117, no. 2, pp. 18–23, 2015.

[27] L. De Cicco, S. Mascolo, and V. Palmisano, "QoE driven resource allocation for massive video distribution," *Ad Hoc Netw.*, vol. 89, pp. 170–176, 2019.

[28] J. Yang *et al.*, "A cost-aware auto-scaling approach using the workload prediction in service clouds," *Inf. Syst. Frontiers*, vol. 16, no. 1, pp. 7–18, Mar. 2014.

[29] W. Iqbal, A. Erradi, and A. Mahmood, "Dynamic workload patterns prediction for proactive auto-scaling of web applications," *J. Netw. Comput. Appl.*, vol. 124, pp. 94–107, 2018.

[30] F. Tseng, X. Wang, L. Chou, H. Chao, and V. C. M. Leung, "Dynamic resource prediction and allocation for cloud data center using the multi-objective genetic algorithm," *IEEE Syst. J.*, vol. 12, no. 2, pp. 1688–1699, Jun. 2018.

[31] W. Liao, S. Kuai, and Y. Leau, "Auto-scaling strategy for amazon web services in cloud computing," in *Proc. IEEE Int. Conf. Smart City/SocialCom/SustainCom (SmartCity)*, Dec. 2015, pp. 1059–1064.

[32] C. Valliyammai and R. Mythreyi, "A dynamic resource allocation strategy to minimize the operational cost in cloud," in *Emerging Technologies in Data Mining and Information Security*, A. Abraham, P. Dutta, J. K. Mandal, A. Bhattacharya, and S. Dutta, Eds. Berlin, Germany: Springer, 2019, pp. 309–317.

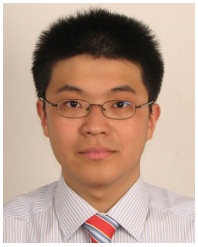
[33] S. Mousavi, A. Mosavi, and A. R. Varkonyi-Koczy, "A load balancing algorithm for resource allocation in cloud computing," in *Recent Advances in Technology Research and Education*, D. Luca, L. Sirghi, and C. Costin, Eds. Berlin, Germany: Springer, 2018, pp. 289–296.

[34] J. Niño-Mora, "Resource allocation and routing in parallel multi-server queues with abandonments for cloud profit maximization," *Comput. Operations Res.*, vol. 103, pp. 221–236, 2019.

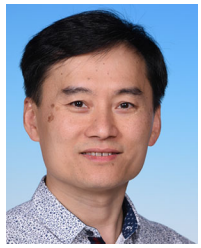
[35] H. Zhao, J. Wang, Q. Wang, and F. Liu, "Queue based and learning based dynamic resources allocation for virtual streaming media server cluster of multi version VoD system," *Multimedia Tools Appl.*, vol. 78, pp. 21827–21852, Apr. 2019.

[36] J. Du, C. Jiang, Y. Qian, Z. Han, and Y. Ren, "Resource allocation with video traffic prediction in cloud-based space systems," *IEEE Trans. Multimedia*, vol. 18, no. 5, pp. 820–830, May 2016.

[37] A. Alasaad, K. Shafiee, H. M. Behairy, and V. C. M. Leung, "Innovative schemes for resource allocation in the cloud for media streaming applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 1021–1033, Apr. 2015.
[38] K. T. Bagci and A. M. Tekalp, "Dynamic resource allocation by batch optimization for value-added video services over SDN," *IEEE Trans. Multimedia*, vol. 20, no. 11, pp. 3084–3096, Nov. 2018.
[39] Y. Li, J. Liu, B. Cao, and C. Wang, "Joint optimization of radio and virtual machine resources with uncertain user demands in mobile cloud computing," *IEEE Trans. Multimedia*, vol. 20, no. 9, pp. 2427–2438, Sep. 2018.
[40] A. Mohan, A. S. Kaseb, Y. Lu, and T. Hacker, "Adaptive resource management for analyzing video streams from globally distributed network cameras," *IEEE Trans. Cloud Comput.*, 2008, to be published, doi:10.1109/TCC.2018.2836907.
[41] T. G. Crainic, G. Perboli, W. Rei, and R. Tadei, "Efficient lower bounds and heuristics for the variable cost and size bin packing problem," *Comput. Operations Res.*, vol. 38, no. 11, pp. 1474–1482, 2011.
[42] O. Anisfeld, E. Biton, R. Milshtein, M. Shifrin, and O. Gurewitz, "Scaling of cloud resources-principal component analysis and random forest approach," in *Proc. IEEE Int. Conf. Sci. Elect. Eng. Israel*, Dec. 2018, pp. 1–5.
[43] L. Chen, J. Lingys, K. Chen, and F. Liu, "AuTO: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2018, pp. 191–205.
[44] Y. Feng, P. Zhou, J. Xu, S. Ji, and D. Wu, "Video big data retrieval over media cloud: A context-aware online learning approach," *IEEE Trans. Multimedia*, vol. 21, no. 7, pp. 1762–1777, Jul. 2019.
[45] P. Yang *et al.*, "Content popularity prediction towards location-aware mobile edge caching," *IEEE Trans. Multimedia*, vol. 21, no. 4, pp. 915–929, Apr. 2019.
[46] Z. Cheng and D. Eppstein, "Linear-time algorithms for proportional apportionment," in *Proc. Int. Symp. Algorithms Comput.*, 2014, pp. 581–592.
[47] F. Chen *et al.*, "Migrating big video data to cloud: A peer assisted approach for VoD," *Peer to Peer Netw. Appl.*, vol. 11, pp. 1060–1074, 2018.
[48] Y. Zhou, T. Z. J. Fu, and D. M. Chiu, "A unifying model and analysis of P2P VoD replication and scheduling," *IEEE/ACM Trans. Netw.*, vol. 23, no. 4, pp. 1163–1175, Aug. 2015.



Zhangyu Chang received the B.Sc. degree (Hons.) in physics and computer science (double major) and the M.Phil. degree in computer science and engineering, in 2011 and 2015, respectively, from The Hong Kong University of Science and Technology, Hong Kong, where he is currently working toward the Ph.D. degree with the Department of Computer Science and Engineering. His research interests include multimedia networking, fog/edge computing, and video/location data analytics.



S.-H. Gary Chan (Senior Member, IEEE) received the B.S.E. degree (Hons.) in electrical engineering from Princeton University, Princeton, NJ, USA, in 1993, with certificates in applied and computational mathematics, engineering physics, and engineering and management systems, and the MSE and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, USA, in 1994 and 1999, respectively, with a minor in business administration. He is currently a Professor with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology (HKUST), Hong Kong. He is also Chair of the Committee on Entrepreneurship Education Program with HKUST, and Board Director of Hong Kong Logistics and Supply Chain MultiTech R&D Center (LSCM). His research interests include smart sensing and IoT, cloud and fog/edge computing, indoor positioning and mobile computing, video/location/user/data analytics, and IT entrepreneurship. Prof. Chan has been an Associate Editor for the IEEE TRANSACTIONS ON MULTIMEDIA (2006–2011), and a Vice-Chair of Peer-to-Peer Networking and Communications Technical Sub-Committee of IEEE Comsoc Emerging Technologies Committee (2006–2013). He has been a Guest Editor of Elsevier *Computer Networks* (2017), *ACM Transactions on Multimedia Computing, Communications and Applications* (2016), IEEE TRANSACTIONS ON MULTIMEDIA (2011), *IEEE Signal Processing Magazine* (2011), *IEEE Communication Magazine* (2007), and Springer *Multimedia Tools and Applications* (2007). He was the TPC Chair of IEEE Consumer Communications and Networking Conference (IEEE CCNC) 2010, Multimedia symposium of IEEE Globecom (2007 and 2006), IEEE ICC (2007 and 2005), and Workshop on Advances in Peer-to-Peer Multimedia Streaming in ACM Multimedia Conference (2005). He has co-founded and transferred his research results to several startups. Due to their innovations and commercial impacts, his startups and research projects have received local and international awards (2012–2020). Notably, he received Hong Kong Chief Executive's Commendation for Community Service for "outstanding contribution to the fight against COVID-19" in 2020. He was the recipient of Google Mobile 2014 Award (2010 and 2011) and Silver Award of Boeing Research and Technology (2009). He was a visiting professor and researcher with Microsoft Research (2000–2011), Princeton University (2009), Stanford University (2008–2009), and University of California at Davis (1998–1999). He was the Director of Entrepreneurship Center (2016–2020), Undergraduate Programs Coordinator with the Department of Computer Science and Engineering (2013–15), Director of Sino Software Research Institute (2012–2015), Co-Director of Risk Management and Business Intelligence program (2011–2013), and Director of Computer Engineering Program (2006–2008) with HKUST. He was a William and Leila Fellow with Stanford University (1993–1994), and the recipient of the Charles Ira Young Memorial Tablet and Medal, and the POEM Newport Award of Excellence at Princeton (1993). He is a member of honor societies Tau Beta Pi, Sigma Xi and Phi Beta Kappa, and a Chartered Fellow of The Chartered Institute of Logistics and Transport (FCILT).