Minimizing Content Reorganization and Tolerating Imperfect Workload Prediction for Cloud-Based Video-on-Demand Services

Chen Tian, *Member, IEEE*, Yi Wang, Yan Luo, *Member, IEEE*, Hongbo Jiang, *Member, IEEE*, Wenyu Liu, *Member, IEEE*, Jie Wu, *Member, IEEE*, and Hao Yin, *Member, IEEE*

Abstract—Video-on-demand (VoD) services historically rely on commercial content distribution networks (CDNs) for on-demand capacity provisioning. Content providers gradually prefer a self-managed content infrastructure because of its full control and customization. However, such a dedicated physical infrastructure could be costly in initial capital investment, and complex in management. It has become a promising alternative to host VoD services on pay-as-you-go cloud platforms, on which using dynamic server provisioning to reduce server rental cost is the key objective of content providers. In this paper we address two major challenges to reducing cost: to minimize content reorganization and to tolerate imperfect workload prediction. We first present a practical VoD servicing system design based on a pay-as-you-go cloud. We prove that previous works, focusing exclusively on cost savings, cause significant content reorganization and are vulnerable to imperfect workload prediction. To address such issues, we propose a novel idea called *workload absorber*, and design a provisioning algorithm called *Absorb Window* based on the idea. Workload absorbers eliminate the bandwidth wastage and significantly reduce content reorganization. We conduct extensive evaluations with real VoD access traces, and demonstrate the superior scalability of the proposed algorithm by producing highly optimized provisioning in seconds for thousands of servers.

Index Terms—Dynamic server provisioning, video-on-demand, cloud computing

1 Introduction

Online streaming has already dominated nowadays Internet traffic [1], [2], [3], [4], and its main form is video-on-demand (VoD). There are dedicated online VoD providers (e.g., NetFlix, Hulu, Youku); there also exists large websites which need to service a significant amount of VoD requests everyday (e.g., MSN, SINA). Historically, most VoD services rely on commercial content distribution networks (CDNs) (e.g., Akamai, Limelight) for on-demand capacity provisioning [5].

However, from a provider's perspective, a self-managed content infrastructure is more preferable than commercial CDNs. For example, link stealing is a serious problem in China [6]: a pirate VoD software could first stream an advertisement to a user, then redirect all subsequent VoD requests to the official provider's video links; in this way, the pirate software could get the advertisement income, while without the burden of video servicing. Our conversation with a major provider reveals that: even though its technical team manages to change the URL of each video segment every 10 minutes, link stealing would only be temporarily disabled, but not completely resolved. As a result, many video content providers in China (e.g., Youku, Sohu, Tencent) choose to build their own hosting infrastructures [7].

A dedicated *physical* infrastructure comes with costly capital investment and complexity in operation and management. An alternative is to build a dedicated *virtual* infrastructure based on a pay-as-you-go cloud platform. Infrastructure-as-a-service (IaaS) cloud platforms, such as Amazon AWS, enable on-demand virtual server provisioning billed hourly. The management operations of the VoD platform become turning on/off (virtual) servers through APIs of the Cloud platform. An example in this trend is Netflix: Netflix used to host contents in commercial CDNs [8]; recently, it shifts most of its traffic from Akamai and Limelight to its own AWS-based infrastructure [9], [10].

Minimizing servicing cost (i.e., resource cost paid to Cloud provider) is essential to such a Cloud-based VoD system. It is well-known that the workload of a typical web system exhibits a periodical access pattern on a daily basis [11]. For a VoD service, the bandwidth cost is almost fixed: every user request should be fulfilled, otherwise the quality-of-service (QoS) is compromised. The major opportunity of cost reduction is to save server rental cost through *dynamic server provisioning* or DSP, that is, to dynamically configure the number of active servers in response to the temporal fluctuation of VoD requests [12], [13], [14], [15].

Previous works only focus on minimize the cost [16], [17], however overlook two major challenges to cost

- *C. Tian is with State Key Laboratory for Novel Software Technology, Nanjing University, China. E-mail: tianchen@nju.edu.cn.*
- *Y. Wang, H. Jiang, and W. Liu are with the School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan, China. E-mail: {ywang, hongbojiang, liuwy}@hust.edu.cn.*
- *Y. Luo is with the Department of Electrical and Computer Engineering, University of Massachusetts Lowell, Lowell, MA 01854. E-mail: yan_luo@uml.edu.*
- *J. Wu is with the Department of Computer Science, Fudan University, Shanghai, China. E-mail: jwu@fudan.edu.cn.*
- *H. Yin is with the Department of Computer Science, Tsinghua University, Beijing, China. E-mail: h-yin@mail.tsinghua.edu.cn.*

Manuscript received 14 July 2014; revised 9 Feb. 2015; accepted 17 Mar. 2015. Date of publication 26 Mar. 2015; date of current version 9 Dec. 2016.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSC.2015.2416733

minimization. First, the cloud based VoD system should *minimize content reorganization*, which refers to the change of stored content in servers. As the content requests change dynamically, it is inevitable to adjust the storage organization to serve the new requests. Moving content around causes management overhead: (a) a daemon process needs to guarantee the completion of the reorganization operations; (b) the service could be temporally impaired due to content changes; and (c) the network bandwidth is consumed to copy the content. Dynamic server provisioning should also strive to minimize the number of content storage organization changes. Second, the system should *tolerate imperfect workload prediction*. Despite extensive analysis of past workloads patterns, the uncertainty of new requests cannot be completely eliminated. A common approach is "safety margin": overprovisioning servers beyond the expected workload to absorb unpredictable surges. However, providing margins blindly would unnecessarily increase the number of active servers hence the cost. The dynamic server provisioningshould strive to minimize the degree of overprovisioning under imperfect prediction scenarios.

To the best of our knowledge, we are the first to give a practical solution to cloud based VoD by minimizing the servicing cost and content reorganization, and tolerating imperfect workload prediction. The contributions of this paper include:

- We propose a practical VoD servicing system based on a pay-as-you-go Cloud. Specifically, we present an industrial practice of video content consolidation scheme, which significantly improves the scalability of management (Section 2).
- We mathematically define dynamic server provisioning and solve a *two-dimensional-splittable* packing problem, with respect to the requirements of *minimizing content reorganization* and *tolerating imperfect workload prediction* (Section 3).
- We prove that previous works, although could achieve near optimal cost saving, cause significant content reorganization and are vulnerable to imperfect workload prediction (Section 4).
- We propose a novel *workload absorber* idea (Section 5), and design an *absorb window (AW)* approach based it. We extend the design to handle important practical issues such as heterogeneous servers, merging newly added contents, etc (Section 6).
- We conduct extensive evaluations with real trace data and demonstrate that the approach is highly scalable: producing high-quality provisioning in seconds for thousands of servers (Section 7).

The rest of paper is organized as follows. Section 8 provides the algorithm extension to support heterogeneous cloud servers. Section 9 discusses the related work. Section 10 concludes the paper.

2 A Cloud-Based VoD System

VoD content distribution platforms follow different architecture. There are primarily two CDN design philosophies [18]. The first is to *enter deep into ISPs*: by deploying surrogate servers inside ISP point of presence (PoP), the cached content is close to users. The leading CDN provider Akamai is of this type. Another philosophy is to *bring ISPs to home*, which builds large data centers and peering these centers extensively with ISPs. The cloud-based VoD system focused on in this paper belongs to the latter type as VoD streaming is sensitive to throughput, instead of latency, between servers and users. Although there exist commercially available Cloud-based content delivery services [19] such as Amazon CloudFront, we study a self-managed, highly customizable, cloud-based content delivery infrastructure for VoD due to the content ownership and needs to protect revenues from "link stealing".

TABLE 1
EC2 Pricing per Machine-Hour

Type	Small	Medium	Large
Linux/UNIX Price	$0.060	$0.120	$0.240
Virtual CPU	1	2	4
Memory (GiB)	1.7	3.75	7.5
Storage (GB)	1×160	1×410	2×420
Bandwidth (Mbps)	300	900	1,200

This section presents an overview of our target cloud-based VoD System. Section 2.1 gives the background of Cloud pricing and resource limitation of each virtual server type. In Section 2.2, we discuss the design choice of content storage. Section 2.3 defines the dynamic server provisioning problem. Section 2.4 presents the industrial practice of content consolidation. The system diagram is shown in Section 2.5.

2.1 Cloud Pricing

A pay-as-you-go cloud platform provides a collection of on-demand computing, networking and storage services. Computing, together with networking, is provided in the granularity of a virtual server. Table 1 is a partial list of the current Amazon elastic compute cloud (EC2) pricing for on-demand general purpose virtual servers in US East region [20], [21]. A virtual server is charged based on per-whole-hour usage; each server has a resource limitation in computing, memory, instance storage and bandwidth. Note that the bandwidth capacity metrics are not published by Amazon; one of our recent work obtains these values via large-scale measurements [22].

There are two networked storage options in Amazon: S3 [23] and EBS [24]. Amazon S3 is designed to make web access easier for developers via a simple web-service interface that can be used to store and retrieve data. Amazon EBS provides persistent storage volumes for use with Amazon EC2 instances in the AWS Cloud. EBS is superior compared with S3, in terms of latency and throughput, since it targets intra-datacenter usage [25]. There are three EBS volume types: General Purpose (SSD), Provisioned IOPS (SSD), and Magnetic. Their pricing is listed in Table 2. Among them, only Provisioned IOPS is designed for I/O-intensive applications with workloads.

2.2 Design Choice of Content Storage

To service the requests to our cloud based VoD system, there are two options: one is that servers directly fetch the content from the networked storage (i.e., EBS Provisioned

TABLE 2
EBS Pricing

Type	Storage	I/O requests
General Purpose	\$0.100 per GB-month	0
Provisioned IOPS	\$0.125 per GB-month	\$0.065 per IOPS-month
Magnetic Volumes	\$0.05 per GB-month	0.05 per million

IOPS volume) to serve; another is that a server preloads the content into its local instance storage, and fetches from the instance storage when requests come.

Directly using the networked storage has two major shortcomings. First, such networked storage incurs unnecessary cost: every I/O request counts for networked storage. As a comparison, the access to instance storage of a virtual server is essentially free. Second, the network throughput between networked storage and servers is not guaranteed. For intra-datacenter networks [26], [27], [28], [29], [30], [31], [32], currently there is no guarantee for bandwidth [33], [34], [35]. It is reported that the throughput of both S3 and EBS could be compromised during heavy intra-datacenter network load period [25]. In contrast, the instance storage is physically collocated with the virtual server and is free from intra-datacenter network contention. The drawbacks of instance storage are: (1) it is limited in capacity on a single server and (2) it is not persistent or robust against failures.

To achieve robustness, our VoD system chooses to store all original video contents in persistent networked storage EBS inside exactly the same data center. To achieve performance guarantee and cost reduction, the content is served from instance storage instead of from networked storage directly. Specifically, when a server is scheduled to service a specific content, the assigned contents would be copied from EBS to the server's instance storage ahead of the service. Then the server could serve subsequent requests to the same content.

2.3 Dynamic Server Provisioning

Minimizing servicing cost is central to a Cloud-based system. The major chance is to save server rental cost via dynamic server provisioning, which dynamically configures the number of active servers in response to the fluctuating VoD workload. For a virtual server in Cloud, two resources are limited for servicing streaming requests: disk storage and network bandwidth. Storage usage depends on the content stored in the server; bandwidth usage depends on the assigned requests for those stored contents. Each server has a pre-defined instance storage upper-bound simply because it is impossible to store all video contents within every single server. Each server also has a pre-defined bandwidth upper bound as it is impossible to serve all the requests from a single server. Server provisioning hence can be transformed to a two-dimensional bin packing problem: each server is a bin with two dimensions; each content is an item with two dimensions; the objective is to pack all the items with the minimum number of bins.

An optimized dynamic server provisioning should, in every schedule slot, minimize the number of active servers. As mentioned in Section 1, for a practical dynamic server provisioning approach in a production VoD system, there are two major challenges need to be considered together with cost minimization: minimize content reorganization and tolerate imperfect workload prediction. Besides, dynamic server provisioning should also consider other practical issues, for example, incrementally merge newly added content is a daily job; reach acceptable algorithm execution time etc. Note the assumption here is that all servers have the same capacity. We notice the possibility of cases where a group of heterogeneous server instances are required. Therefore we have an extension of the approach to handle heterogeneous servers in later part of the paper (Section 8).

2.4 Industrial Practice of Content Consolidation

It is unscalable to manage content at a per-video granularity. First, the number of videos is too large for scheduling algorithms. Second, it causes difficulties for management operations such as replacing one video with another, since the file sizes of two videos are very likely to be different. Third, it is hard to have a stable popularity prediction for every single video.

The industrial practice we learned from the pre-Cloud era is to group videos into separate *channels*, and use a channel as the finest management granularity. All channels have the same nominal size, usually several hundred gigabytes; each channel could contain hundreds, sometimes thousands, of videos. The advantages are three-fold:

- The number of managed units is greatly reduced.
- The management is scalable since one can directly replace one channel in the storage with another.
- The workload prediction at channel level is more accurate and stable due to the effect of aggregation.

However, the disadvantage of channel level management is that server provisioning is now even more challenging. Assume that the bandwidth demand of every channel is smaller than the minimum value of each server's bandwidth constraint. Combined with the storage constraint, the cost optimization can be formulated as a two-dimensional bin-packing problem, and a simple two-dimensional First-Fit algorithm could be used to approximate the optimal solution [36]. However, this assumption is hardly true for large-scale VoD systems: a single *hot* channel alone could require multiple servers together to serve its peak workload. In other word, the same channel might have multiple copies of videos in multiple servers; the bandwidth demand scheduling must be *splittable* among servers.

To summarize, after content consolidation to channels, we formally define dynamic server provisioning to solve a *two-dimensional-splittable* packing problem, with respect to both *minimizing content reorganization* and *tolerating imperfect workload prediction*.

Fig. 1 gives an illustration of how the cloud based VoD system works. Fig. 1a shows the configuration during an peak hour: instances of four different channels (x, y, z and w, labeled as "CH x", "CH y", "CH z" and "CH w", respectively) are consolidated across four servers (A, B, C and D). Assume that the total workload greatly decreases at the non-peak hour; more specifically, the workloads of channel y and z decrease significantly. Then dynamic server provisioning schedules the instance of channel z in server B being replaced by an instance of channel w, as shown in Fig. 1b. Meanwhile server D is stopped hence the cost in the non-peak hours is minimized.

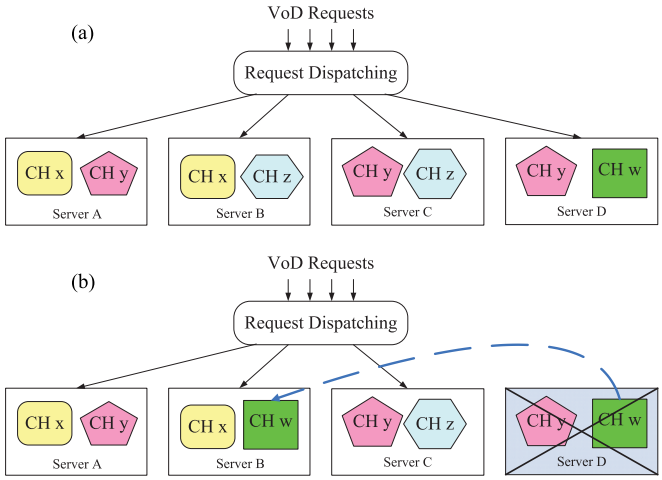


Fig. 1. Dynamic server provisioning by content consolidation (a) peak (b) non-peak.

2.5 System Overview

Fig. 2 shows our dynamic server provisioning control loop in a Cloud based VoD system. The system is composed of the following components: *Request Router*, *Workload Estimator*, *Instance Placement Controller*, *Placement Executor*, as well as back-end *Servers*.

According to the predicted workload of each channel provided by Load Estimator, the Instance Placement Controller periodically makes decisions on how many servers of the whole system should be active in the next timeslot (a timeslot here is the time unit of scheduling, e.g., one hour), outputs a placement solution which optimizes certain objective functions, and then passes the solution to the Placement Executor to start/stop servers and/or migrate channel instances accordingly. The Request Router receives external requests and forwards them to the instances based on the calculated workload dispatching decisions.

This paper focuses on the approach used by the Instance Placement Controller, which has significant influence on server costs. The specific workload estimation methodology is out of the scope of this paper. Instead, the workload prediction values, as well as their variances, are assumed to be given as system inputs in our work, as many previous studies use real-time profiling and data regression to dynamically estimate the workload [11], [37], [38].

3 Problem Formulation

We formulate the problem by modeling the resource constraints of individual servers and subsequently the cloud environment consisting of a set of servers. Table 3 lists symbols used in this paper. The inputs to the placement controller include: the current placement matrix I^*, the bandwidth and storage capacities of each server (B_n and C_n), and the predicted bandwidth demands of each channel (ω_m) in the next time slot. The outputs of the placement controller are the updated server active matrix J, placement matrix I and the workload distribution matrix Q.

The optimization objectives are prioritized as below:

$$\begin{aligned} &(i)\ \text{Minimize: } \sum_{t\in T}\sum_{n\in N} J_{n,t}E_n \\ &(ii)\ \text{Minimize: } c(t) = \sum_{m\in M}\sum_{n\in N}\left|I_{m,n,t} - I_{m,n,(t-1)}\right|, \forall t \in T. \end{aligned} \tag{1}$$

TABLE 3
Symbols Used

N	The set of customized video servers.
n	One server in the set N.
M	The set of content channels.
m	One channel in the set M.
T	The sequential set of time slots.
t	One time slot in the set T.
I	The content instance placement output of the algorithm. $I_{m,n,t} = 1$ if an instance of channel m is running on machine n in timeslot t; $I_{m,n,t} = 0$ otherwise. $I^* = I(t-1)$ is the placement matrix in the previous slot.
Q	The workload dispatching output of the algorithm. $Q_{m,n,t}$ is the bandwidth resource allocated on server n for channel m in timeslot t.
J	The dynamic provisioning output of the algorithm. $J_{n,t} = 1$ if server n is active in time slot t; $J_{n,t} = 0$ otherwise.
C_n	The storage resource capacity of server n.
B_n	The bandwidth resource capacity of server n.
E_n	The per server per hour cost of server n.
ω_m	The predicted bandwidth demand of channel m.
c	the calculated number of needed placement change.

Objective (i) specifies the minimize cost goal; (ii) specifies the minimum reconfiguration rule.

We start from a homogeneous system where all servers have the same bandwidth capacity B and storage capacity C. This is consistent with our baseline cloud VoD scenarios. The objectives and constraints are as follows:

$$\begin{aligned} &(i)\ \text{Minimize: } \sum_{t\in T}\sum_{n\in N} J_{n,t} \\ &(ii)\ \text{Minimize: } c(t) = \sum_{m\in M}\sum_{n\in N}\left|I_{m,n,t} - I_{m,n,(t-1)}\right|,\ \forall t \in T \\ &(a)\ J_{n,t} = 0 \Rightarrow Q_{m,n,t} = 0,\ \forall m \in M,\ \forall n \in N,\ \forall t \in T \\ &(b)\ I_{m,n,t} = 0 \Rightarrow Q_{m,n,t} = 0,\ \forall m \in M,\ \forall n \in N,\ \forall t \in T \\ &(c)\ \sum_{m\in M} I_{m,n,t} \le C, \forall n \in N,\ \forall t \in T \\ &(d)\ \sum_{m\in M} Q_{m,n,t} \le B,\ \forall n \in N,\ \forall t \in T \\ &(e)\ \sum_{n\in N} Q_{m,n,t} = \omega_m,\ \forall m \in M,\ \forall t \in T \\ &(f)\ Q_{m,n,t} \ge 0, I_{m,n,t} \in \{0,1\}, J_{n,t} \in \{0,1\}, \\ &\forall m \in M,\ \forall n \in N,\ \forall t \in T. \end{aligned} \tag{2}$$

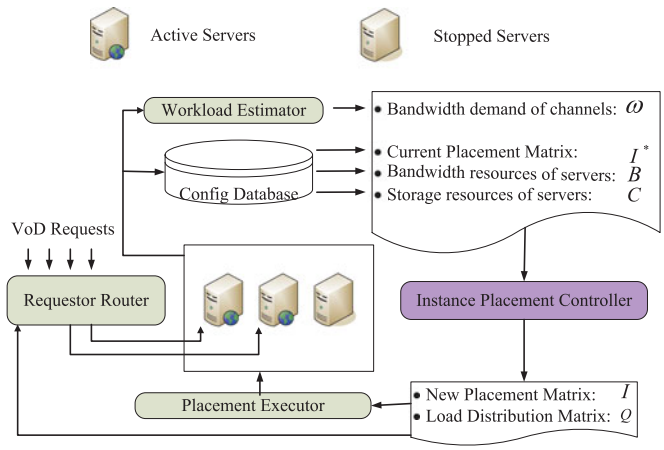


Fig. 2. Control loop.

4 Limitations of Existing Algorithms

Similar problems related to VoD dynamic server provisioning have been investigated in theoretical computer science community as the class constrained bin packing problem (CCBP) [16], [17]. Specifically, the most common CCBP formulation is that: there are unit-sized items of M distinct classes, which have to be packed into N knapsacks; the objective is to find an allocation which minimizes N. Here in our context, a *class* is semantically equivalent to a channel, and a *knapsack* is semantically equivalent to a server. It is proved that CCBP is NP-hard [16].

A near-optimal polynomial time approximation scheme called *moving window* (MW) has been proposed in [16]. However, this algorithm only minimizes the cost by minimizing the number of servers; it does not consider minimizing content reorganization or tolerating imperfect workload prediction. The details of MW will be given in Section 4.1. In Section 4.2, we demonstrate the shortcomings of this approach to motivate the necessity of a new algorithm. In Section 7, we compare with MW algorithm to demonstrate the superior performance of our proposed algorithm.

4.1 Class Constrained Bin Packing

Regards the CCBP form of dynamic server provisioning, the input is the total workload ω, which consists of M distinct classes, given as the subsets $\omega_1, \ldots, \omega_M$; there are ω_m items of class m, $1 \leq m \leq M$, and $\omega = \omega_1 \cup \omega_2 \cup \ldots \cup \omega_M$. There are N knapsacks, each having a limited volume B and a limited number C of compartments, in which the items can be placed. Accordingly, in each server we can place items of at most C different channels.

The output is a placement strategy, which specifies how to allocate compartments for each server n, and how many items of each class m are placed in n. A placement is called *legal* if n is allocated at most C compartments, and the overall size of the items placed in n does not exceed B, for all $1 \leq n \leq N$. The objective is to find a *legal* placement, thereby minimizing the number of knapsacks.

There exists approximation algorithms for CCBP problem. In [16], Shachnai and Tamir presented the moving window approximation scheme for CCBP. The algorithm keeps a vector $R = (R[1], R[2], \ldots, R[M])$ where $R[m]$ is the number of remaining items to be packed of a class m. The vector is maintained in non-decreasing order of the values $R[m]$ during the execution of the algorithm. At any given moment, it tries to pack C different classes to obtain a packing of a new bin. To that end, the algorithm keeps a window of C classes. At first, the window encompasses items from $R[1]$ to $R[C]$. If $\sum_{m=1}^{C} R[m] \geq B$, the algorithm packs the corresponding classes of $R = (R[1], R[2], \ldots, R[j])$, where $j \leq C$ is the first index such that $\sum_{m=1}^{j} R[m] \geq B$. Notice that $R[j]$ may be partially packed. The totally packed classes are removed from the vector. If $\sum_{m=1}^{C} R[m] < B$, the algorithm moves the window to the right, until that for the first time the window has C classes such that their sizes are greater or equal than B. If this is the case, the C classes are packed and the vector R is reordered (if the last considered set was partially packed). Then the algorithm restarts. If in some iterations, the window reaches the end of the vector R, i.e., the C largest classes have total size smaller than B, the algorithm generates bins by packing entirely C classes in each bin, except that the last bin may have less than C classes.

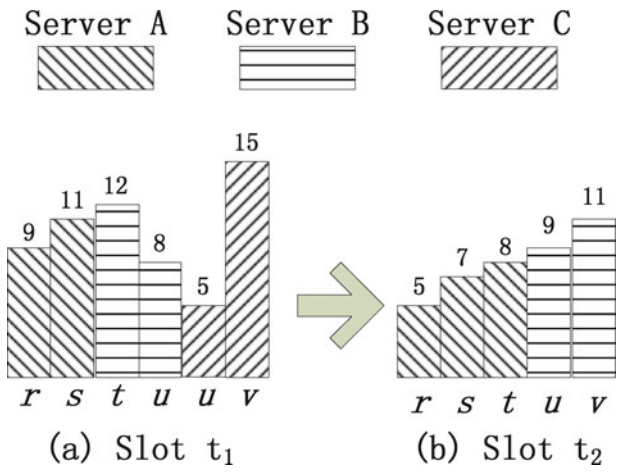


Fig. 3. Moving window results over two consecutive time slots.

4.2 Shortcomings of CCBP

However, MW algorithm incurs significant instance reconfiguration between consecutive time slots although it can obtain a near optimal channel consolidation. An example is shown in Fig. 3a, where each column denotes a channel and its height (the number above the column) denotes the workload. There are five channels $< r, s, t, u, v >$ in the figure, and every server has the same capacity of $B = 20, C = 3$. Suppose in timeslot t_1, the workloads of channels are $< 9, 11, 12, 13, 15 >$ respectively; all workloads are decreased at the next timeslot t_2 (shown in Fig. 3b) to $< 5, 7, 8, 9, 11 >$ respectively.

Shown in Fig. 3a, the MW algorithm is applied to t_1 first: the first two channels $< r, s >$ are allocated to server A and the bandwidth resources are fully used $(9 + 11 = 20)$; next two channels $< t, u >$ are allocated to server B and u still has unfulfilled demand; and then $< u, v >$ are allocated to server C. Then in t_2 (Fig. 3b), the MW algorithm packs channels $< r, s, t >$ to server A and $< u, v >$ to B respectively; server C can be stopped to save cost.

As we can observe from the figure, in both timeslots, all active servers are fully utilized; in t_2 the cost saving is maximized. Also in t_2, compared with t_1, channel $< t >$ is migrated to server A, and channels $< v >$ are migrated to server B. It is obvious that even one channel migration in a prior server can subsequently cause significant changes in instance placement on the succeeding servers.

Now let's consider the *tolerance of imperfect prediction*. Assume that different from the prediction, channel r's workload is 8 in slot t_1; also assume channel v's real workload is 16. Although the total workload maintains the same, channels $< u, v >$ together have 21 workload, which is larger than the capacity of server A. To maintain QoS, the only choice is to add an extra server to provide the safety margin.

In summary, MW algorithm is subject to content reorganization in large scale systems because the Objective (ii) is not taken into consideration at all. In addition, MW could be sub-optimal when the workload prediction is imperfect.

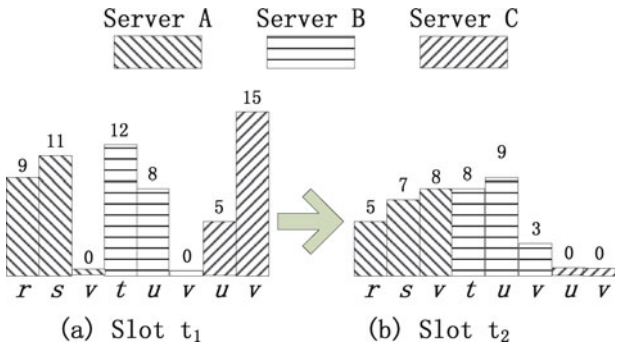


Fig. 4. Intuition of workload absorbing.

5 Intuition

5.1 Observation

Let's observe the motivating examples in Fig. 3 again: in slot t_1, $< r, s >$ are allocated to server A and $< t, u >$ are allocated to server B; in both A and B, there is one extra compartment unallocated. What if we exploit these compartments by adding channel v?

The workload dispatching in slot t_1 can be maintained the same as that in Fig. 4a. While in slot t_2, we could allocate workload 5, 7, 8 to $< r, s, v >$ on server A and workload 8, 9, 3 to $< t, u, v >$ on server B. We can still stop server C in slot t_2 and save the cost.

The advantage is to eliminate the changes to channel placement. That is, the variability of "original" channels $< r, s >$ from t_1 to t_2 can be naturally "absorbed" by assigning the workload of channel v in server A; the same happens in server B. Additionally if in slot t_1, channel r's workload is 8 and channel v's real workload is 16, the imperfect prediction could be tolerated without the need of an extra server for safety margin: simply allocate the extra 1 workload of v to server A.

In server A, channel v acts like an "absorber" which absorbs the unused bandwidth (if there is any). The same is the role of channel $< v >$ in server B.

5.2 Workload Absorbing

Patterns of web service workloads introduce opportunities for cost saving. Previous measurement works have proved that the hourly workload pattern of a typical web service takes a day as the period [11], [13]. Assume the peak workload of a channel in the next 24 hours can be predicted, and we allocate servers according to this prediction, then there could have two consequences: (a) there is no need to add more instances for this channel, since all demands in the next 24 hours should be satisfied and (b) since most of the hours in a 24-hour phase are off-peak, there could be a significant portion of "wasted" bandwidth resources unless we fully exploit the usage of server instances.

The basic idea of *workload absorbing* is to use a small set of channels as "absorbers" to consume unused bandwidth to minimize the waste in bandwidth allocation preset according to the access predictions. Specifically, we choose a special group of channels to form an *Absorber set*, and they are called *Absorber channels*; the rest of the channels are put into the *Main set*, and they are called *Main channels*. In forming the Absorber set, we prefer to select Absorber channels with large peak bandwidth demand because of their role of "absorbing" unused bandwidth by the Main channels. We mix channels from both the Main set and the Absorber set and divide them to a number of *Main groups*. For each Main group, the bandwidth capacity allocated is only based on the 24-hour peak workload prediction of member channels from the Main set. In each timeslot, the bandwidth in an active server is allocated to the Main channels with priority; the extra bandwidth, if any, is allocated to the Absorber channels assigned to the server. Absorber channels also form extra *Absorber groups*, in case that the extra bandwidth allocated to them in Main groups is not sufficient for their demand.

The benefits of workload absorbing are three-folded. First, the bandwidth of Main group servers are fully utilized since all extra bandwidth not used by Main channels are "absorbed" by the Absorber channels in the same server. Second, the instance reorganization is minimized: changes to instance placement happens only at the beginning of a 24-hour cycle; at each time slot only server start/stop operations are needed. Third, it provides sufficient safety margin for imperfect workload prediction because the variability inside *Main groups* could be covered; only a small set of *Absorber groups* may need extra servers.

Based on this intuition of *workload absorbing*, we design an Absorber Window approach which solves the *two-dimensional-splittable* packing problem in dynamic server provisioning, for both minimizing content reorganization and tolerating imperfect workload prediction.

6 Dynamic Server Provisioning with Absorb Window

6.1 Framework

Exploiting the nature of workload daily cycles, we design the scheduling framework to be two-level: *cycle* (e.g., every 24 hours) level and *time slot* (e.g., every hour) level. After each cycle level scheduling, servers are divided to the *Active set* and the *Inactive set*. As in the context of Amazon EC2, an instance in *Active set* is running whereas an instance in *Inactive set* is stopped but can resume operation with little time. Active servers' total resource capability should satisfy the peak workload demand within a cycle; all servers in the Inactive set can be stopped throughout the cycle.

The formal description of the two-level scheduling is as follows:

- In each cycle, we find a packing of service instances (i.e., channels) into the minimum number of servers, which is the Active set; the current instance placement should be taken into consideration to minimize the number of instance changes between two consecutive cycles. The instance placements of all timeslots in the new cycle are fixed in cycle level scheduling.
- In each time slot within a cycle, we determine how many servers to assign to the Active set (i.e., $J_{n,t}$) and how to dispatch the channel workloads to these servers (i.e., $Q_{m,n,t}$).

6.2 Absorb Window Algorithm

At the center of our design is the *absorb window* algorithm. Recall that the moving window algorithm is based on a

heuristic which tries to pack C different classes in each bin, but in the process MW tends to pack small and large classes in *different* bins. In contrast, our heuristic is to pack $C-1$ less demanded channels (in the Main set) and 1 busy channel (in the Absorber set) together; they together make a group of C channels in a bin. The Absorber set has L channels with the largest workloads; the Main set has P slices of $C-1$ small channels; the deduction of L and P will be presented later, and their relationship satisfies $|M| = P*(C-1)+L$.

Following the heuristic, we pack the channels to a server instance as follows. For each slice of $C-1$ channels in the Main set, we add 1 channel from the Absorber set to form a *Main group* of size C. The number of servers provisioned for this group is dynamically calculated only based on the workloads of the $C-1$ Main channels; the provisioning principle is that the total resource provided should be no less than total peak demands of $C-1$ Main channels. Channels in Absorber set are directly grouped to a *Absorber group* of size C, and their server provisioning is calculated after Main channels are dispatched due to their absorbing role.

We illustrate the algorithm following the motivation example in Fig. 3. There are four channels in two Main sets: $< r, s >$ and $< t, u >$. The only channel in Absorber set is $< v >$; then two Main groups are $< r, s, v >$ and $< t, u, v >$. We get exactly the same result as shown in Fig. 4. at timeslot t_1, the total workload of $C-1$ small channels of Main group $< r, s, v >$ is $\sum_{m=r}^{s} R[m] = B$; so we use server A to host this group. Similarly, the total load of $C-1$ Main channels of Main group $< t, u, v >$ is $\sum_{m=t}^{u} R[m] \geq B$ but $\sum_{m=t}^{u} R[m] \leq 2B$; so we use server B and C to host this group, and all surplus capacity is allocated to the large absorber group $< v >$.

6.3 Grouping Channels

In this subsection, we present the details of constructing groups of channels, which is done at the first cycle of the provisioning procedure.

The determination of P and L. Because servers allocated to the Absorber set may not be fully utilized, we should minimize the number of Absorber channels (i.e., L) in this set. Keep a vector $R = (R[1], R[2], \ldots, R[M])$ that is maintained in non-decreasing order of the peak workload values $R[m]$. The determination of P and L is shown in Algorithm 1.

Algorithm 1. GetPandL()

```
1: P = 0;
2: while Σ_{m=P*(C-1)+1}^{M} R[m] ≥ P * B do
3:     P = P + 1;
4: end while
5: L = M − P * (C − 1);
```

The intuition behind this algorithm is: each Main group may have at most B free capacity at any given timeslot; this determination method ensures that all surplus capacity of Main groups can be absorbed so that potential under-utilization are limited to Absorber groups.

Group construction. The construction of Main channels in each Main Group is relatively straightforward. Following Algorithm 1, we put every consecutive $C-1$ channels into the next group. The remaining problem is how to choose the Absorber channel for each group. As shown in Fig. 5, we iteratively add the next biggest class in Absorber set to the next smallest Main group. If the Absorber set is exhausted, the algorithm restarts from the beginning of the set. The intuition behind this policy is that: larger channels have more chances to be chosen as absorber candidates, hence makes the remaining workloads of Absorber channels more balanced.

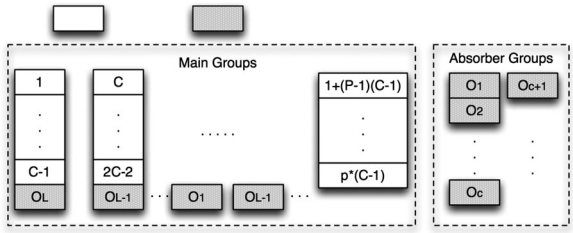


Fig. 5. Grouping rules.

For Absorber group, we keep a window of size C over the Absorber channels. All generated groups would have C different channels each, except perhaps the last one.

Add new channels. For a running system, new channels could be continuously added into the system. To minimize the disturbance, we prefer maintaining the existing group constructions: new channels are directly added to the tail of Absorber set.

However, operators may re-run the grouping algorithm from ground up to obtain better results. For example, they can apply Algorithm 1, after every new $C-1$ channels have been added, to get a new pair of P and L values. If P does not change, no new Main group is needed; otherwise, we apply group construction algorithm to construct new Main groups.

6.4 Cycle Level Scheduling

Cycle level scheduling determines the Active set of servers, based on the current grouping and the predicted peak workloads of all channels.

Step 1: Calculates the number of servers in active set

Suppose the Absorber channels for all Main groups are determined as $AbsorberChanIndex[p], 1 \leq p \leq P$, then the number of servers in each Main group is calculated by Algorithm 2.

Algorithm 2. NumOfServersForEachMainGroup()

```
1: for p = 1 ··· P do
2:     GroupHead = (p − 1) * (C − 1) + 1
3:     GroupTail = (p − 1) * (C − 1) + C − 1
4:     GroupPeakLoad = Σ_{m=GroupHead}^{GroupTail} R[m]
5:     NumberServer[p] = ceil(GroupPeakLoad/B)
6:     AbsorberChannel = AbsorberChanIndex[p]
7:     AbsorberCapacity = NumberServer[p]*
        B − GroupPeakLoad
8:     R[AbsorberChannel] = R[AbsorberChannel]−
       AbsorberCapacity
9: end for
```

After the execution of Algorithm 2, the number of Active servers for each Absorber group can be calculated directly by their remain workloads.

Step 2: Choose appropriate servers for each group

Let g denotes the group index. For each group g, the number of servers in the new cycle $NumberServer[g](t)$ is

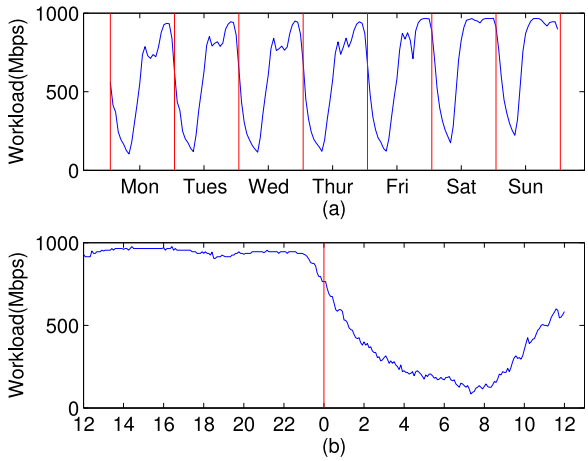


Fig. 6. (a) One week of workload pattern (Monday to Sunday) of a typical channel (b) Detailed 24 hours workload pattern.

compared with the current numbers $NumberServer[g](t-1)$. If

$$NumberServer[g](t) < NumberServer[g](t-1)$$

then $NumberServer[g](t-1) - NumberServer[g](t)$ servers are stopped and added to the Inactive set. Their instance placements are kept unchanged because this lazy clean-up policy might reduce instance changes in the future if the same group's workload increases again.

Instead, if $NumberServer[g](t)$ is bigger, then $NumberServer[g](t) - NumberServer[g](t-1)$ servers should be added to group g from the Inactive set. The algorithm first selects servers that used to serve the same group (that's how lazy policy benefits); after that other servers in the Inactive set are selected based on the Least-Frequently Used principal due to temporal locality in channel accesses.

6.5 Time Slot Level Scheduling

Server provisioning. For each time slot, we still use Algorithm 2 to get the number of required active servers of each group. The task of server provisioning in timeslot level is to stop or start necessary number of servers.

Workload dispatching. For a channel in Main set, its workload is evenly divided among the instances hosting by servers of its Main group. For a channel in Absorber set, after all instances in its corresponding Main groups have been allocated, the remaining workload is also evenly divided among the instances hosting the Absorber group.

6.6 Tolerate Imperfect Workload Prediction

As mentioned above, a safety margin (in percentage) can be added to every workload prediction to tolerate possible imperfections. Apparently, if the prediction has a variance of at most 30 percent, then provides a 30 percent safety margin is enough for any imperfect prediction. However, this policy is too conservative since it might provide many unnecessary servers.

Thanks to the nature of workload absorber, our approach can provide a much lower percentage of safety margin compared with oblivious method. We demonstrate this advantage in evaluations.

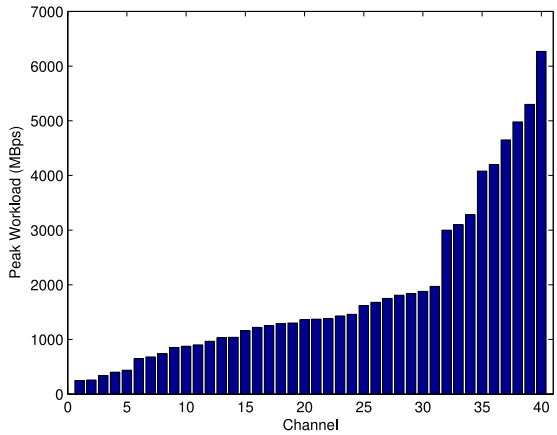


Fig. 7. Peak Workload of 40 channels.

There could exist more fine-grained per-channel margin provisioning. More specifically, the margin may be load-dependent: the margin required at a low workload channel may be higher (in percentage) than the margin required at a high workload one [39]. We prefer to leave this to future work. Also noted that we only aim at an acceptable level of imperfect workload prediction; flash-crowd traffic (such as those caused by sudden public events) is an exception to any prediction-based scheduling scheme.

7 Performance Evaluation

7.1 Evaluation Experiments Setup

We choose the *Large* type Amazon EC2 server as our evaluation example (Table 1). Each server's disk can be divided to six parts with 140 GB each: one for system and the other five for videos. So every 140 GB video content is aggregated as an individual channel. We set 200 Mbps aside for system, and use 1 Gbps for VoD servicing. That is, following our problem formulation, all servers have $C = 5$ and $B = 1$ Gbps.

The evaluations are driven by a realistic trace data from a leading VoD provider in China. All data are collected directly from the CDN of the provider. Shown in Fig. 6a is an one-week workload pattern of a typical channel in January 2008; a vertical line marks the starting of a new day in the week; the y-axis represents the changing workload. From a daily point of view, the workload distribution varies on a daily basis due to changing client numbers. To make a finer observation, we show detailed workloads of selected 24 hours in Fig. 6b: the x-axis from 12:00 noon to 12:00 the second day. As we can observe, the workload maintains high from noon to night, starts to decline after around 23:00, reaches the minimum load around 7:00, then rises again.

We use system logs for seven days from Jan 8 (Tuesday) to Jan 14 (Monday), 2008 in Shenyang data center. We have collected the records of 41 channels. Among them, 40 channels exist from the very start (their peak bandwidth is shown in Fig. 7); one new channel is added from Jan 10. We set full 24 hours as the length of a cycle from 23:00 to 22:59 next day.

The evaluations are based on numerical investigation of real trace data. All simulations are done in Matlab on a machine with a 3.0 Ghz Intel Xeon CPU. We will first thoroughly evaluate the performance of our algorithm in

TABLE 4
Group Construction

Group	Channel	Peak Workload(Mbps)
	1	248
	2	258
1(Main)	3	340
	4	400
	40	*
	5	437
	6	650
2(Main)	7	680
	8	739
	39	*
⋮	⋮	⋮
	33	3,100
	34	3,280
9(Main)	35	4,080
	36	4,200
	40	*
	37	4,650
10(Absorber)	38	4,980
	39	5,300
	40	6,270

homogenous systems. Our AW algorithm is compared with two algorithms: the first is a strawman provisioning approach, which always provides the servers to meet the predicated peak load of each channel; we also compare with MW algorithm (In Section 4) to demonstrate the superior performance of our proposed algorithm. Then the scalability of the proposed algorithm is demonstrated. When imperfect workload prediction is introduced, the tradeoff between safety margin and service performance is investigated. At last, we evaluate the generalized approach for heterogeneous systems.

7.2 Cycle Level Provisioning

We use the "Number of Active Servers" metric to denote system provisioning, as the provided network resource capacity is the total number of active server of that time slot multiplies with the network bandwidth (1 Gbps).

In the first experiment, real trace is used as prediction inputs to drive the numerical investigation. The group construction (Section 6) procedure is applied to the data of Jan 8. Shown in Table 4 is the result of initial group construction with $P = 9$ and $L = 4$. Four Absorber channels (i.e., 37 to 40) are selected and there is exactly one of them in each Main group. Then on Jan 10, the new channel 41 is added into group 10 (Absorber group).

Shown in Table 5 is the result of the number of Duty set servers for the first cycle. Totally 75 servers are allocated to groups and are placed with corresponding service instances. Note that although the aggregated workload of Absorber

TABLE 5
Cycle Level Scheduled Group Servers

Group	1	2	3	4	5	6	7	8	9	10	Sum
Number	2	3	4	5	6	6	7	9	15	18	75

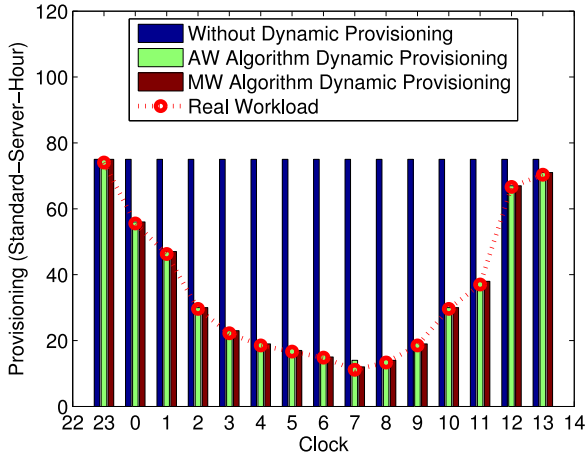


Fig. 8. Provisioning versus real workload.

group 10 almost doubles that of Main group 9, its number of allocated servers is just a little more than group 9: the reason is that a large portion of the workload can be dispatched to corresponding Absorber instances in other Main groups. We can compare the real workload with the provided workload by the servers provisioned: the maximum service capacity can provided is 75,000 Gbps, compared with 74,045 Gbps real peak workload in this cycle. These numbers are quite close to each other, which implies that our approach performs extraordinary well in minimizing the cost too.

7.3 Performance of Cost Saving

The time-slot level server provisioning from 23:00, Jan 8 to 13:00 next day is shown in Fig. 8. At each hour, Both AW algorithm and MW algorithm are applied to the access traces.

As shown in Fig. 8, without dynamic provisioning, the number of active servers always maintains at the maximum value. On the contrary, the provided servicing capacity of both AW and MW algorithms are always close to the real workloads and the number of active servers is minimized, because both algorithms make good use of full capacity of allocated servers.

To measure the effectiveness of cost saving, We use a standard-server-hour (SSH) metric. As shown in Fig. 8, without dynamic provisioning, the cost is *75 Servers* $\times$ *15 Hours = 1,125* SSHs in the period shown in Fig. 8. With AW/MW dynamic provisioning, only around 540 SSHs are needed and over 50 percent cost is saved in both cases. As both AW and MW are heuristic based appoaches, we conclude that both dynamic provisioning algorithms can achieve close-to-optimal cost saving. This is because both algorithms leverage the full capacity of allocated servers.

However, AW doesn't incur any instance placement change inside a cycle. As a comparison, the instance changes of MW algorithm are shown in Fig. 9: there are significant instance changes in every timeslot; this is also by no means any incidence, but because MW algorithm doesn't take into account the minimum reconfiguration rule. Therefore, AW has significant benefits over MW in practical deployment as reorgnization of cotents are prohibitive in cost.

In Fig. 10, we depict the number of servers of each group produced by AW algorithm in each time slot. As we have expected, the number of servers allocated to each Main group exactly corresponds to its workload fluctuation. The

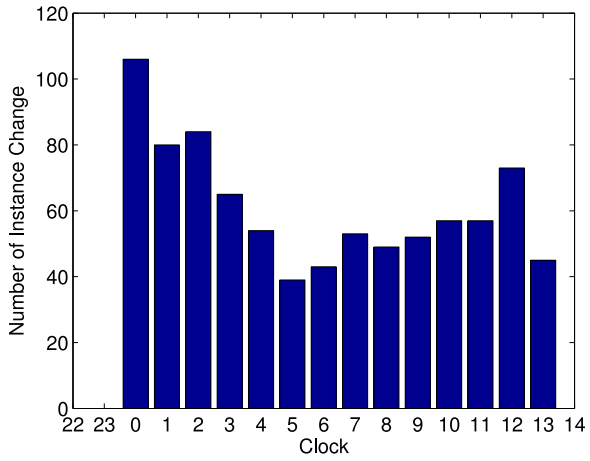


Fig. 9. Instance change of MW algorithm.

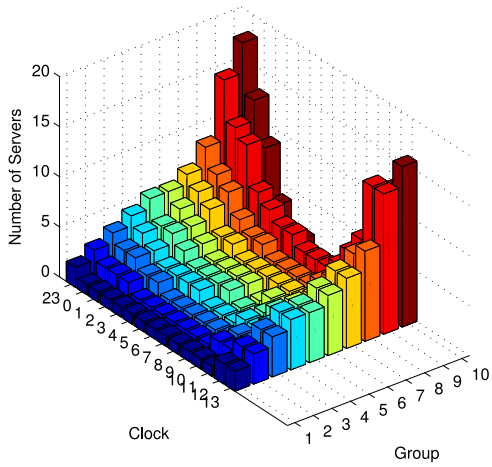


Fig. 10. Number of active servers of each group.

number of servers of the Absorber group 10 exhibits larger fluctuation than its real workload due to the existence of absorb effects; it is observed that there are only 1 server left during the bottom hours. To sum up, we can conclude that with perfect workload prediction (such as the real trace), our server provisioning approach can minimize both cost and instance reorganization.

7.4 Minimize Content Reorganization

We evaluate the number of content reorganizations on a server at cycle level. The peak workloads of 7 days are shown in Fig. 11a. The number of servers in both Active set and Inactive set are shown in the figure too. Trace analysis shows that 90 servers will meet the worst case peak demand. As a start, we allocate 75 servers for Main set and 15 servers for the Absorber set in the first cycle.

It is obvious that the Active and Inactive sets will change according to the daily peak workloads. Everyday, there could be new servers needed in one group; there also could be servers that are unnecessary and should be removed from the Active set. At the start of each cycle, the following lazy reconfiguration rules are executed:

- For groups that need fewer servers, the removed servers are put into the Inactive set; the content stored in their storage are preserved. If no other groups choose them, these servers get into hibernation mode (stopped).
- For a group needing more servers, the scheduler first looks into the Inactive set, check if there are already some hibernating servers that belong to this group before hibernation. If such servers are found, they are woken up without the need of content reconfiguration.
- If no such servers are found, the server with the oldest hibernation time is chosen (earliest-first-replacement) and a content reconfiguration is performed on this server.

With this heuristic procedure, we fully utilize previous servers' contents, thus further eliminate unnecessary reconfiguration at the cycle level.

The number of servers removed and/or added servers and reconfigured instances at each cycle are shown in Fig. 11b. As we have expected, the numbers of server removed and/or added are directly related to the workload variations. For example, at the beginning of Wednesday, two servers are removed from a group; also, two servers are added to another group. There is no hibernating servers that contain the same content, hence all two servers need to be reconfigured with channel contents. The same happens on Thursday: one server hibernates; one server is added and reconfigured. On Friday, the workload increases, thus nine servers need to be added. At this moment, three hibernating servers could be exploited. As a result, only six new added servers need reconfiguration. It is clear that due to the lazy reconfiguration policy, the number of reconfiguration is greatly reduced. As the workload decreases, there is no configuration changes on Sunday and Monday.

7.5 Tolerate Imperfect Workload Prediction

Imperfection of load prediction is inevitable in real scenarios. One common technique is to provide safety margins for the

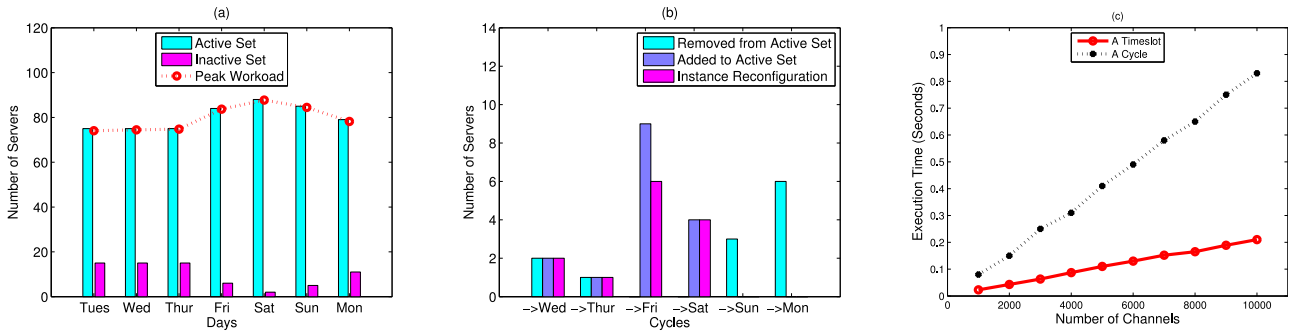


Fig. 11. (a) Peak workload in each day (b) Server reconfigurations in cycle level (c) Execution time of algorithm.

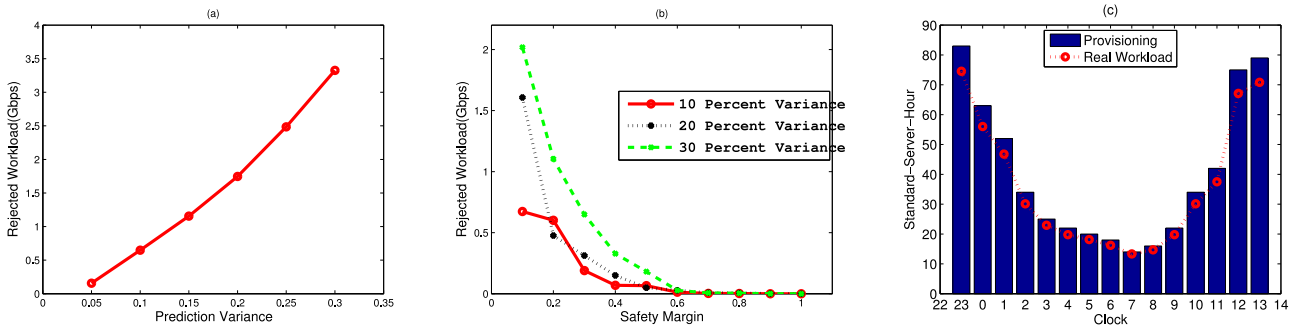


Fig. 12. (a) No safety margin (b) With partial safety margin (c) 20 percent Variance and 60 percent safety margin.

predicted results. However, the level of safety margin largely affects the level of cost saving. Hence we need to control the cost incurred by the safety margin. Next we demonstrate that with our AM algorithm, the safety-margin can be significantly reduced, without impairing the performance.

We first evaluate the performance of our algorithm without safety margin provided. The access traces on the first day from 23:00 to 13:00 the next day are used. Rejected workloads of all channels are summed up to illustrate the performance deterioration of the system. The results are shown in Fig. 12a. It is clear that with the increase of the prediction variance, the rejected workloads increases almost linearly. This result illustrates the necessity of a safety margin for tolerating workload prediction errors.

Then we evaluate the performance of partial safety margin policy: we assume prediction variance range is known a priori, and only partial safety margin is provided to the predicted values. The safety margin is increased from 10 to 100 percent of the prediction variance. The results are shown in Fig. 12b with three different values of prediction variance range. As we can see from the Figure, when the safety margin approximates 60 percent of the maximal variance, the performance lost is nearly negligible. These results show that with only partial safety margin, our approach is tolerant to certain degree of prediction errors.

In Fig. 12c we compare the provided capacity with the real workloads, where the prediction variance range is set to 20 percent and safety margin is 60 percent of the prediction variance. We calculate the standard-server-hour from the results, and conclude that, with 60 percent safety margin, the cost saving is over 45 percent in this period.

7.6 Computational Scalability

Based on real traces, a channel workload generator is developed to simulate a large number of channel inputs (following the distribution of channel peak workloads in Fig. 7). With the generated channels, we evaluate the algorithm's computational scalability. The computation time of our provisioning algorithms is shown in Fig. 11c. For 10,000 channels, the execution times at both cycle level and time-slot level are less than 1 second. We conclude that our algorithm is computationally scalable.

8 Heterogeneous Cloud

There exist heterogeneous instances in cloud computing platforms such as Amazon EC2. The performance and cost of server instances differ based on machine configuration (e.g., CPU cores and memory capacity). VoD providers may choose different servers to further reduce operation costs. We extend the problem formulation to accommodate heterogeneous server instances as follows. The optimization are subjected to the constraints shown in Equation (3): set (a) specifies that a channel can be served in a server if and only if the server is active in that time slot; set (b) specifies that a channel can be served in a server if and only if its instance is stored on that server; set (c) specifies that storage requirements of all channels in server n should not exceed its storage capacity; set (d) specifies that the aggregation of bandwidth allocated to the channels in each server should not exceed the server's bandwidth capacity; set (e) specifies that the total allocated bandwidth to a channel should meet its demand; set (f) gives the value range of decision variables.

$$
\begin{aligned}
&(a)\ J_{n,t} = 0 \Rightarrow Q_{m,n,t} = 0,\ \forall m \in M, \forall n \in N,\ \forall t \in T\\
&(b)\ I_{m,n,t} = 0 \Rightarrow Q_{m,n,t} = 0,\ \forall m \in M,\ \forall n \in N,\ \forall t \in T\\
&(c)\ \sum_{m \in M} I_{m,n,t} \le C_n,\ \forall n \in N,\ \forall t \in T\\
&(d)\ \sum_{m \in M} Q_{m,n,t} \le B_n,\ \forall n \in N,\ \forall t \in T\\
&(e)\ \sum_{n \in N} Q_{m,n,t} = \omega_m,\ \forall m \in M,\ \forall t \in T\\
&(f)\ Q_{m,n,t} \ge 0, I_{m,n,t} \in \{0,1\}, J_{n,t} \in \{0,1\},\\
&\quad \forall m \in M,\ \forall n \in N,\ \forall t \in T.
\end{aligned}
\tag{3}
$$

8.1 Support to Heterogeneous Servers

AW algorithm can be generalized to support heterogeneous systems. A key observation is that generally the number of server types is limited. We put all servers of the same type into one server type group; there may exist different types that have the same storage capacity; we further aggregate those type groups together. Assume there are K type groups, a server in group k has storage capacity C_k and this type group can provide channel capacity up to V_k. For type groups, a vector $V = (V[1], V[2], \ldots, V[K])$ is maintained in non-decreasing order of the values $V[k]$. Again the channels are grouped in non-decreasing order of the values $R[m]$.

The intuition is that we should match server type groups that have higher available channel capacity with channel groups that have higher workloads. Our heterogeneity-friendly absorber window extension (AW') iteratively matches the server type groups to channel groups. At first, the window goes from $R[1]$ to $R[C_1 - 1]$. If $\sum_{m=1}^{C_1-1} R[m] \le V_1$ then the algorithm packs the corresponding sets of $R' = (R[1], R[2], \ldots, R[C_1 - 1])$; a completely packed class

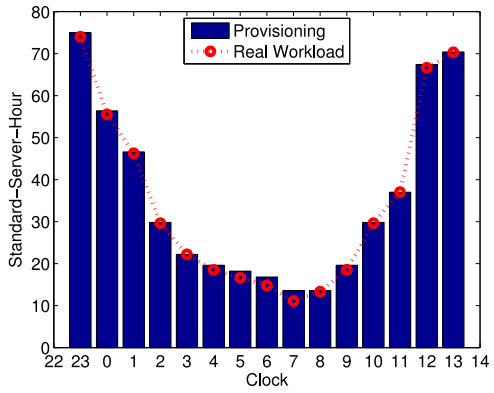


Fig. 13. Performance in heterogeneous systems.

group is removed from vector R and its corresponding capacity is subtracted from V_1. If $\sum_{m=1}^{C_1-1} R[m] > V_1$, then the algorithm moves to the next server type group with C_k and V_k, until that for the first time the server group has C_k storage capacity such that its network capacity are greater or equal than V_k. Then the algorithm iterates from the beginning until all the channels are assigned to servers.

All other auxiliary algorithms, such as the P and L calculation, can be easily generalized to support heterogeneous systems. We omit their details here.

8.2 Evaluation

We suppose that the heterogeneous server farm consists of four types of servers: ($C = 5 : B = 1G$, $C = 6 : B = 1G$, $C = 8 : B = 1.4G$, $C = 8 : B = 1.6G$), each of which has 16 servers. The total service capacity is 80 Standard-Server-Hour per time slot, slightly more than the total peak workload in Jan 8. We conduct the trace driven experiments similar to those for a homogeneous cloud system.

The result is shown in Fig. 13. It is clear that the extended Absorber Window algorithm works well in heterogeneous cloud systems, where over 50 percent of server rental costs are saved.

9 Related Work

Extensive efforts of dynamic server provisioning have been made so as to achieve power efficiency in data centers [40], [41]. In [42], Pinheiro et al. present a simple policy to turn cluster servers on and off dynamically. Heath et al. [43] study web service server ON-OFF strategies in the context of heterogeneous server types, although focusing only on short transactions. Elnozahy et al. [44] employ various combinations of dynamic voltage scaling and server ON-OFF strategies to reduce the aggregate power consumption of a server cluster during non-peak hours. Chase et al. [12] allocate computing resources based on an economic approach, where services "bid" for resources as a function of required performance, and the system continuously monitors workload and allocates resources based on its utilization. Chen et al. [13] study dynamic server provisioning techniques in the context of connection servers that host a large number of long-lived TCP connections, which have little disk I/O loads. Most of these works target on server ON-OFF strategies; they focus on the system design of dynamic provisioning.

Gandhi et al. [45] propose an algorithm that can estimate the right number of required servers without prediction of future request rate. Lu et al. [46] develop online dynamic provisioning solutions with and without future workload information available. Lin et al. [47] presents a theoretical analysis of the cost saving of online dynamic server provisioning scheduling. Starts from the observation of the characteristic of the workload prediction, Hong et al. [39] propose to provide different safety margins for different services to save costs. These researches also targets server ON-OFF strategies; they focus on the impact of imperfect workload prediction. *Compared with all the above mentioned papers, our work considers multidimensional resource constraints in servers and the requirement of minimizing content reorganization.*

Also targeting power efficiency, Verma et al. [48], [49] consider multiple resource constraints. The problem is formulated as a multi-dimensional bin-packing problem and the simple First-Fit algorithm is used to schedule services. A recent work by Xiao et al. [50] considers packing virtual machines (with multi-dimensional resource requirements) into a minimum number of physical machines (with multi-dimensional resource limitations). However, these works can only handle services with relatively small resource demands, where one server's resource capacity is sufficient in all dimensions. In other word, each resource constraint is *un-splittable*. However, this assumption is hardly true for large-scale web service systems as we described in Section 1. *In contrast, our work needs to handle practical services with splittable workloads Section 2.4). Our dynamic server provisioning solves a two-dimensional-splittable packing problem, with respect to the requirements of minimize content reorganization and tolerate imperfect workload prediction (Section 3).* There are also some less related papers, such as the work of Tian et al. [51], which improves load balancing via service consolidation.

10 Conclusion

Minimizing service cost is critical to a cloud-based VoD system. At the same time, two major challenges need to be addressed: minimizing content reorganization and tolerating imperfect workload prediction. Previous works only focus on minimizing the cost. To the best of our knowledge, we are the first to give practical solutions to the challenges. Based on the novel "workload absorber" idea, we design the Absorb Window approach to take advantage of residual bandwidth for reducing resource waste. Extensive evaluations with real trace data demonstrate that (1) the content reorganization is minimized when achieving the same level of server rental cost; (2) the safety margin in server provisioning is reduced to further lower the costs and (3) the approach can produce high-quality provisioning in seconds even with thousands of servers and channels.

Acknowledgments

The authors would like to thank anonymous reviewers for their valuable comments. This work was supported in part by the China National Basic Research Program (973 Program) under Grant 2011CB302601 and Grant 2012CB315801; by the National Natural Science Foundation of China under Grant 61202107, Grant 61271226, Grant 61371141, Grant

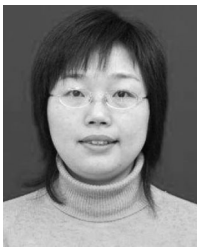
61170290, and Grant 61222213; by the National High Technology Research and Development Program of China (863 Program) under Grant 2014AA01A702; by the National Science Foundation of USA under Grant ACI-1440737; by the EU FP7 CLIMBER project under Grant Agreement No. PIRSES-GA-2012-318939; by the Jiangsu International Cooperation Program of Science and Technology under Grant No. BZ2013018; by the Natural Science Foundation of Hubei Province under Grant 2014CFB1007; and by the National Key Technology Research and Development Program of China under Grant 2012BAH46F03. Yi Wang is the Corresponding author.

REFERENCES

[1] "Cisco Visual Networking Index: Forecast and Methodology, 2011-2016,"
[2] C. Tian, R. Alimi, Y. R. Yang, and D. Zhang, "Shadowstream: Performance evaluation as a capability in production internet live streaming networks," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2012, pp. 347–358.
[3] J. Yan, C. Tian, J. Sun, and H. Mao, "Improve distributed client lifecycle control in shadowstream," *Int. J. Web Services Res.*, vol. 11, no. 4, pp. 62–78, 2014.
[4] H. Mao, C. Tian, J. Sun, J. Yan, W. Wu, and B. Huang, "Shadowvod: Performance evaluation as a capability in production p2p-cdn hybrid vod networks," in *Proc. Int. Symp. Ubiquitous Syst. Data Eng.*, 2014, pp. 1–6.
[5] H. H. Liu, Y. Wang, Y. R. Yang, H. Wang, and C. Tian, "Optimizing cost and performance for content multihoming," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 371–382, 2012.
[6] Web video firms accuse baidu of stealing content [Online]. Available: http://www.scmp.com/business/china-business/article/1355409/web-video-f irms-accuse-baidu-stealing-content
[7] Tencent video spending hundreds of millions of rmb to build network [Online]. Available: http://technode.com/2011/08/29/tencent-video-spending-hundres-of-millio ns-of-rmb-to-build-network/, 2011.
[8] V. K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z.-L. Zhang, "Unreeling netflix: Understanding and improving multi-cdn movie delivery," in *Proc. IEEE INFOCOM*, 2012, pp. 1620–1628.
[9] Limelight, akamai shares fall as netflix shifts traffic [Online]. Available: http://www.reuters.com/article/2012/06/05/us-limelight-shares-netflix-i dUSBRE8540S220120605, 2012.
[10] Chaos kong is coming: A look at the global cloud and CDN powering netflix [Online]. Available: http://www.datacenterknowledge.com/archives/2013/10/17/chaos-kong-is-co ming-a-look-at-the-global-cloud-and-cdn-powering-netflix/, 2013.
[11] G. Szabo and B. A. Huberman, "Predicting the popularity of online content," Soc. Comput. Lab., HP Labs, Palo Alto, CA, USA, 2009.
[12] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyley, "Managing energy and server resources in hosting centers," in *Proc. 18th ACM Symp. Oper. Syst. Principles*, 2001, pp. 103–116.
[13] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy-aware server provisioning and load dispatching for connection-intensive internet services," in *Proc. 5th USENIX Symp. Netw. Syst. Des. Implementation*, 2008, pp. 337–350.
[14] R. P. Doyle, J. S. Chase, O. M. Asad, W. Jin, and A. M. Vahdat, "Model-based resource provisioning in a web service utility," in *Proc. 4th USENIX Symp. Internet Technol. Syst.*, 2003, p. 5.
[15] V. Sharma, A. Thomas, T. Abdelzaher, and K. Skadron, "Power-aware QoS management in web servers," in *Proc. 24th IEEE Real-Time Syst. Symp.*, 2003, pp. 63–72.
[16] H. Shachnai and T. Tamir, "Polynomial time approximation schemes for class-constrained packing problems," *J. Scheduling*, vol. 4, no. 6, pp. 313–338, 2001.
[17] E. C. Xavier and F. K. Miyazawa, "The class constrained bin packing problem with applications to video-on-demand," *Theor. Comput. Sci.*, vol. 393, no. 1, pp. 240–259, 2008.
[18] C. Huang, A. Wang, J. Li, and K. W. Ross, "Measuring and evaluating large-scale CDNs," in *Proc. ACM 8th SIGCOMM Conf. Internet Meas.*, 2008, vol. 8, pp. 15–28.
[19] J. Barr, A. Tetlaw, and L. Simoneau, *Host Your Web Site in the Cloud: Amazon Web Services Made Easy*. Melbourne, Australia: SitePoint, 2010.
[20] Amazon EC2 pricing [Online]. Available: http://aws.amazon.com/ec2/pricing/, 2014.
[21] Amazon EC2 instances [Online]. Available: http://aws.amazon.com/ec2/instance-types/, 2014.
[22] H. Zhang, K. Chen, W. Bai, D. Han, C. Tian, H. Wang, H. Guan, and M. Zhang, "Guaranteeing deadlines for inter-datacenter transfers," in *Proc. ACM Eurosys*, 2015, To Appear.
[23] Amazon s3 [Online]. Available: http://aws.amazon.com/s3/, 2014.
[24] Amazon elastic block store [Online]. Available: http://aws.amazon.com/ebs/, 2014.
[25] Differences between s3 and ebs [Online]. Available: http://www.cloudiquity.com/2009/03/differences-between-s3-and-ebs/, 2009.
[26] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: A high performance, server-centric network architecture for modular data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 63–74, 2009.
[27] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. Maltz, "zupdate: Updating data center networks with zero loss," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 411–422, 2013.
[28] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic engineering with forward fault correction," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 527–538.
[29] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 539–550.
[30] R. Gandhi, H. H. Liu, Y. C. Hu, G. Lu, J. Padhye, L. Yuan, and M. Zhang, "Duet: Cloud scale load balancing with hardware and software," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 27–38.
[31] Y. Zhao, K. Chen, W. Bai, M. Y. USC, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang, "Rapier: Integrating routing and scheduling for coflow-aware data center networks," in *Proc. IEEE INFOCOM*, 2015, To Appear.
[32] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *Proc. 12th USENIX Symp. Netw. Syst. Des. Implementation*, 2015, To Appear.
[33] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: A data center network virtualization architecture with bandwidth guarantees," in *Proc. 6th Int. Conf*, 2010, p. 15.
[34] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, 2011, pp. 242–253.
[35] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The only constant is change: Incorporating time-varying network reservations in data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 199–210, 2012.
[36] D. S. Johnson, "Near-optimal bin packing algorithms," Ph.D. dissertation, Massachusetts Inst. Technol., Cambridge, MA, USA, 1973.
[37] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi, "Dynamic estimation of cpu demand of web traffic," in *Proc. 1st Int. Conf. Perform. Eval. Methodol. Tools*, 2006.
[38] H. Kimiyama and S. Itoh, "Method of predicting number of on-demand video requests using time series data for video cache system," in *Proc. 6th Int. Conf. Adv. Mobile Comput. Multimedia*, 2008, pp. 200–205.
[39] Y.-J. Hong, M. Thottethodi, and J. Xue. (2011). Dynamic server provisioning to minimize cost in an IaaS cloud. Tech. Rep. [Online]. Available: http://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1413&context=ecetr
[40] A.-C. Orgerie, M. D. d. Assuncao, and L. Lefevre, "A survey on techniques for improving the energy efficiency of large-scale distributed systems," *ACM Comput. Surv.*, vol. 46, no. 4, p. 47, 2014.
[41] R. Alimi, L. Chen, D. Kutscher, H. H. Liu, A. Rahman, H. Song, Y. R. Yang, D. Zhang, and N. Zong, "An open content delivery infrastructure using data lockers," in *Proc. 2nd Ed. ICN Workshop Inf.-Centric Netw.*, 2012, pp. 25–30.

[42] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, *Dynamic Cluster Reconfiguration for Power and Performance*. Norwell, MA, USA: Kluwer, 2002.
[43] T. Heath, W. M. Jr, B. Diniz, R. Bianchini, and E. V. Carrera, "Energy conservation in heterogeneous server clusters," in *Proc. ACM SIGPLAN Symp. Principles Pract. Parallel Programm.*, 2005, pp. 186–195.
[44] M. Elnozahy, M. Kistler, and R. Rajamony, "Energy efficient server clusters," in *Proc. 2nd Int. Conf. Power-Aware Comput. Syst.*, 2003, pp. 179–197.
[45] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. A. Kozuch, "Autoscale: Dynamic, robust capacity management for multi-tier data centers," *ACM Trans. Comput. Syst.*, vol. 30, no. 4, p. 14, 2012.
[46] T. Lu, M. Chen, and L. L. Andrew, "Simple and effective dynamic provisioning for power-proportional data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1161–1171, Jun. 2013.
[47] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," *IEEE/ACM Trans. Netw.*, vol. 21, no. 5, pp. 1378–1391, Oct. 2013.
[48] A. Verma, P. Ahuja, and A. Neogi, "Power-aware dynamic placement of HPC applications," in *Proc. 22nd Annu. Int. Conf. Supercomput.*, 2008, pp. 175–184.
[49] A. Verma, G. Dasgupta, T. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," in *Proc. Usenix Annu. Tech. Conf.*, 2009, p. 28.
[50] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1107–1117, Jun. 2013.
[51] C. Tian, H. Jiang, A. Iyengar, X. Liu, Z. Wu, J. Chen, W. Liu, and C. Wang, "Improving application placement for cluster-based web applications," *IEEE Trans. Netw. Serv. Manag.*, vol. 8, no. 2, pp. 104–115, Jun. 2011.

Chen Tian received the BS, MS, and the PhD degrees from the Department of Electronics and Information Engineering, Huazhong University of Science and Technology, China, in 2000, 2003, and 2008, respectively. He is an associate professor of the State Key Laboratory for Novel Software Technology, Nanjing University, China. From 2012 to 2013, he was a postdoctoral researcher with the Department of Computer Science, Yale University. From 2013 to 2015, he was an associate professor in the School of Electronic Information and Communications, Huazhong University of Science and Technology, China. His research interests include network function virtualization, data center networks, distributed systems, Internet streaming and big data processing for smart city.



Yi Wang received the BS, MS, and the PhD degrees from the Department of Electronics and Information Engineering, Huazhong University of Science and Technology, China, in 2000, 2003, and 2009, respectively. She is currently a lecturer in the School of Electronics Information and Communications, Huazhong University of Science and Technology, China. Her research interest is Cloud computing.

Yan Luo the BE and ME degrees from the Huazhong University of Science and Technology, and the PhD degree in computer science from the University of California Riverside in 2005. He is an associate professor of the Department of Electrical and Computer Engineering, University of Massachusetts Lowell. His research spans broadly computer architecture and network systems. His current projects focus on heterogeneous architecture and systems, software defined networks and deep learning. He has served on the program committee of numerous international conferences and as a guest editor and referee of premier journals. He is a member of the IEEE and ACM.

Hongbo Jiang received the BS and MS degrees from the Huazhong University of Science and Technology, China. He received the PhD degree from Case Western Reserve University in 2008. After that he joined the faculty of Huazhong University of Science and Technology where he is now a full professor. His research concerns computer networking, especially algorithms and architectures for wireless networks and mobile computing. He is a senior member of the IEEE.

Wenyu Liu received the BS degree in computer science from Tsinghua University, Beijing, China, in 1986, and the MS and PhD degrees, both in electronics and information engineering, from Huazhong University of Science and Technology (HUST), Wuhan, China, in 1991 and 2001, respectively. He is currently a professor and associate dean of the School of Electronic Information and Communications, HUST. His current research areas include sensor network, multimedia information processing, and computer vision. He is a senior member of the IEEE.

Jie Wu received the BS, MS, and PhD degrees from the School of Computer Science at Fudan University, China, in 1996, 1999, and 2008, respectively. He is a professor at the School of Computer Science, Fudan University. His research interests include computer networks, cloud computing, and P2P streaming. He is a member of ISO/IEC JTC1 SC38 on behalf of China.

Hao Yin is a professor in the Research Institute of Information Technology (RIIT), Tsinghua University. He was elected as the New Century Excellent Talent of the Chinese Ministry of Education in 2009, and won the Chinese National Science Foundation for Excellent Young Scholars in 2012. His research interests span broad aspects of Multimedia Communication and Computer Networks. He is also the vice-director of Industry Innovation Center for Future Network, China, and the secretary-general of Industry Innovation Alliance of Future Internet, China.