CALMS: Cloud-Assisted Live Media Streaming for Globalized Demands with Time/Region Diversities

Feng Wang
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada
Email: fwa1@cs.sfu.ca

Jiangchuan Liu
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada
Email: jcliu@cs.sfu.ca

Minghua Chen
Department of Information Engineering
The Chinese University of Hong Kong
Shatin, NT, Hong Kong
Email: minghua@ie.cuhk.edu.hk

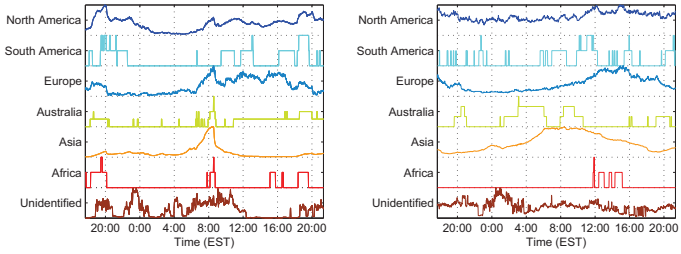
***Abstract*—Live media streaming has become one of the most popular applications over the Internet. We have witnessed the successful deployment of commercial systems with CDN- or peer-to-peer based engines. While each being effective in certain aspects, having an all-round scalable, reliable, responsive and cost-effective solution remains an illusive goal. Moreover, today's live streaming services have become highly globalized, with subscribers from all over the world. Such a globalization makes user behaviors and demands even more diverse and dynamic, further challenging state-of-the-art system designs.**

The emergence of *cloud computing* however sheds new lights into this dilemma. Leveraging the elastic resource provisioning from cloud, we present CALMS (Cloud-Assisted Live Media Streaming), a generic framework that facilitates a migration to the cloud. CALMS adaptively leases and adjusts cloud server resources in a fine granularity to accommodate temporal and spatial dynamics of demands from live streaming users. We present optimal solutions to deal with cloud servers with diverse capacities and lease prices, as well as the potential latencies in initiating and terminating leases in real world cloud platforms. Our solution well accommodates location heterogeneity, mitigating the impact from user globalization. It also enables seamless migration for existing streaming systems, e.g., peer-to-peer, and fully explores their potentials. Simulations with data traces from both cloud service provider (Amazon EC2) and live media streaming service provider (PPTV) demonstrate that CALMS effectively mitigates the overall system deployment costs and yet provides users with satisfactory streaming latency and rate.

I. Introduction

In the past decade, live media streaming has become one of the most popular applications over the Internet [12][5]. We have witnessed a number of successful commercial deployments with CDN (Content Delivery Network)- or peer-to-peer based engines. The former achieves high availability and short startup latencies, but suffers from excessive costs for deploying dedicated servers. This is particularly severe if the user demand fluctuates significantly and the servers have to be over-provisioned for peak loads. The peer-to-peer solution generally incurs lower deployment cost and is more scalable, but the reliability and hence service quality can hardly be guaranteed. There have also been efforts toward synergizing dedicated servers with peer-to-peer [6]. Unfortunately, having an all-round scalable, reliable, responsive, and cost-effectiveness solution remains an illusive goal.

To make it even worse, today's live streaming applications have become highly globalized, with subscribers from all over the world. Such a globalization makes user behaviors and demands even more diverse and dynamic. For illustration, we examine the user demand distribution of PPTV [1], one of the most popular live media streaming systems in China with multi-million users from a trace analysis [21][24]. Fig. 1 shows the results of two representative channels (CCTV3 and DragonBall) during one day[2]. It is easy to see that although PPTV is from China, it has attracted users from all over the world, and the peak time therefore shifts from region to region, depending on the timezone. For example, on the CCTV3 channel, the peak time of North America is around 20:00, while for Asian users, it is around 8:00. During the period of 12:00 to 20:00, the Asian users have very low demands while the European users generate most of their demands and the North American users also have moderate demands. Similar observations can also be found from the DragonBall channel, despite that the streaming contents delivered on the two channels are completely different. The impact of such globalized demand turbulence has yet to be addressed in existing systems that largely focus on regional services only.



(a) CCTV3 (b) DragonBall

Fig. 1: Illustration of the user demand distributions and variations of a popular live media streaming system (PPTV) on its two typical channels (CCTV3 and DragonBall) during one day.

The emergence of *cloud computing* however sheds new lights into this dilemma. A cloud platform offers reliable, elastic and cost-effective resource provisioning, which has been dramatically changing the way of enabling scalable and dynamic network services [10] [2][19][4]. There have been studies on demand-driven resource provision [22][13][11][25]; there have been also initial attempts leveraging cloud service

[1]www.pptv.com – formerly known as PPLive.

[2]For ease of comparison, the user demands have been normalized by the corresponding maximum demand of each day. And the time shown on x-axis is based on EST.

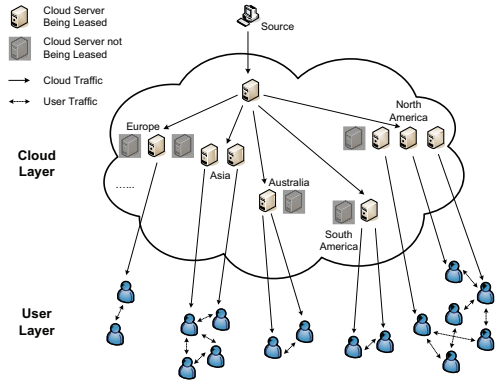


Fig. 2: An overview of CALMS.

to support VoD (Video-on-Demand) applications, from both industry (e.g. NetFlix) [15] and academia [23][8]. Live media streaming however has more stringent playback delay constraints with content being updated in realtime. The larger, dynamic, and non-uniform client population further aggravates the problem, calling for new solutions toward a successful migration to the cloud.

In this paper, we present CALMS (Cloud-Assisted Live Media Streaming), a generic framework that facilitates a cost-effective migration to the cloud. CALMS adaptively leases and adjusts cloud servers in a fine granularity to accommodate temporal and spatial dynamics of user demands. We present optimal solutions to deal with cloud servers with diverse capacities and lease prices, as well as the potential latencies in initiating and terminating leases in real world cloud platforms. Our solution well accommodates location diversity, mitigating the impact from user globalization. It also enables seamless migration for existing streaming systems, e.g., peer-to-peer, and fully explores their potentials. We further develop a set of practical solutions for dynamic adjustment of lease schedules, smart user redirection, and cloud server organization. To understand the performance of CALMS, extensive simulations have been carried out with real data traces from both cloud service provider (Amazon EC2) and live media streaming service provider (PPTV). The results demonstrate that the proposed CALMS effectively mitigates the overall system deployment costs and yet provides users with satisfactory streaming latency and rate.

The remainder of this paper proceeds as follows: Section II presents an overview of the CALMS framework. In Section III, we first investigate the basic problem of leasing cloud service and provide an optimal solution, which is then extended by integrating locality-awareness and user assistance. The implementation issues and further optimization are discussed in Section IV. We then evaluate CALMS through trace-driven simulations in Section V. Finally, Section VI concludes the paper and discusses potential future directions.

II. Cloud-Assisted Live Media Streaming (CALMS): An Overview

Our CALMS intends to provide a generic framework that facilitates the migration of existing live media streaming services to a cloud-assisted solution. Fig. 2 shows an illustration of CALMS, which is divided into two layers, namely, *Cloud Layer* and *User Layer*. The Cloud Layer consists of the live media source and dynamically leased cloud servers. Upon receiving a user's subscription request, the Cloud Layer will redirect this user to a properly selected cloud server. Such a re-direction however is transparent to the user, i.e., the whole Cloud Layer is deemed as a single source server from a user's perspective. Since the user demands change over time, which are also location-dependent, the Cloud Layer will accordingly dynamically adjust the amount and location distribution of the leased servers. Intuitively, it will lease more server resources upon demand increase during peak times, and terminate leases upon decrease.

There are however a number of critical theoretical and practical issues to be addressed in this generic CALMS framework. First, the cloud servers have diverse capacities and lease prices; the lease duration is not infinitesimally short, either, e.g. 1 hour for Amazon EC2. As such, when being leased, the server and the pricing cannot be simply terminated at anytime. In addition, for a newly leased server, the configuration and boot up takes time, too, e.g., 10-30 minutes for EC2. Though the cloud services are improving, given the hardware, software, and network limits, such latencies can hardly be avoided in the near future. Therefore, CALMS must well predict when to lease a new servers to meet the ever-changing demands and when to terminate a server to minimize the lease costs. These problems are further complicated given the global heterogeneous distributions of the cloud servers and that of the user demands.

Note that we do not assume any particular implementation of the User Layer in this study. They can be individual users purely relying on the Cloud Layer, or served by peer-to-peer or a CDN infrastructure, but seeking for extra assistance from the Cloud Layer during load surges. In other words, our CALMS framework can smoothly migrate diverse existing live streaming system, and we will also explore the potential assistance from user peers in our study.

III. Framework Design and Solution

In this section, we discuss the detailed design issues and their solutions for migrating the live media streaming application to the cloud service. Our discussions first start from modeling the basic form of the leasing cloud service problem, and then extend to integrate with locality-awareness and user-assistance, respectively. We will also present centralized algorithms that yield optimized solutions for addressing these issues, which will further motivate the practical solutions for implementation and optimization in next section.

A. Basic Problem: Leasing Cloud Service

Denote the set of cloud servers that can be leased from the cloud service providers as $C = \{c_1, c_2, \cdots, c_m\}$. In practice, most cloud providers have a minimum unit time for the duration of leasing a server (e.g. 1 hour for Amazon EC2) and when being leased, a cloud server must spend some

time in setup and initialization before ready to use. We use D_m to denote this required minimum unit duration and T_s as the latency for preparing the cloud server. For simplicity, we assume there is always a cloud server directly connecting to the live media source to act as the live media server and use c_0 to denote it. Let R be the rate of the live media streaming. We assume that there is a demand forecast algorithm (such as the algorithms proposed in [16]) to predict the demand in the next period T, where the demand may contain the estimated online population of users, their distributions at different areas or ISPs, and other type of information. At current stage, we only consider the online population of users and denote it as $\mathbb{P}(t)$ for a given time t ($t \leq T$). The discussions for utilizing other forcasted demand information will be deferred to later subsections. Define a cloud service lease schedule as $S = \{(x_1, t_1, d_1), (x_2, t_2, d_2), \cdots, (x_k, t_k, d_k)\}$ ($t_1 \leq t_2 \leq \cdots \leq t_k \leq T$), where a tuple (x_i, t_i, d_i) ($x_i \in C$, $d_i > 0$ and $d_i \mod D_m = 0$ for $i = 1, 2, \cdots, k$) means at time t_i, we start to lease cloud server x_i for the duration d_i. Our problem is thus to find a proper cloud lease schedule S, subjecting to the following constraints:

(1) Service Availability Constraint:

$$\forall (x_i, t_i, d_i) \in S, \text{ if } \exists (x_j, t_j, d_j) \in S \text{ and } x_i = x_j,$$
$$\text{then } [t_i, t_i + d_i) \cap [t_j, t_j + d_j) = \emptyset \ ;$$

(2) Streaming Quality Constraint:

$$\forall T_s \leq t \leq T,$$
$$\mathbb{U}(c_0) + \sum_{(x_i, t_i, d_i) \in S} \big(\mathbb{U}(x_i) - R\big) \cdot I_{\left[t \in [t_i + T_s, t_i + d_i)\right]} \geq R \cdot \mathbb{P}(t) \ ;$$

where $\mathbb{U}(\cdot)$ is the upload capacity and $I_{[\cdot]}$ is the indicator function. The service availability constraint asks that at any given time, a cloud server can only appear in one schedule. And the streaming quality constraint asks that at any given time t, the streaming rate demands of $\mathbb{P}(t)$ users have to be satisfied. Our objective is thus to minimize the lease costs:

$$\mathbb{C}_l(c_0) + \sum_{(x_i, t_i, d_i) \in S} \mathbb{C}_l(x_i) \cdot d_i + \mathbb{C}_b(R \cdot \sum_{t \leq T} \mathbb{P}(t)) \ ,$$

where $\mathbb{C}_l(\cdot)$ and $\mathbb{C}_b(\cdot)$ are the costs for leasing a cloud server and out-cloud bandwidth usage, respectively[3]. As the first and last part can not be reduced, we focus on minimizing the middle part of the total costs, which we denote as $Cost_{lease}$:

$$Cost_{lease} = \sum_{(x_i, t_i, d_i) \in S} \mathbb{C}_l(x_i) \cdot d_i \ .$$

This problem is challenging as the cloud servers are scheduled both along the time dimension and at each time instance, along the user demand dimension. By exhaustively searching along both dimensions, the optimal solution can be achieved. However, simply using a naive searching algorithm can be quite inefficient as the solution space increases very fast with the combination of both dimensions. To this end, we proposed an enhanced DFS algorithm, which can skip most parts of the solution space and find the optimal solution efficiently. The proposed algorithm is summarized in Fig. 3. To improve the search efficiency, we first sort the cloud servers in C by the ascendant order of the lease cost per unit upload bandwidth (line 1). This allows the servers with cheaper upload bandwidth being explored first and near-optimal solutions can then be quickly found. With such solutions, we can further cut other search branches with equal or higher costs (line 6-11) and greatly reduce the search space for the optimal solution. In addition, we use variable $renew$ to distinguish whether a server is renewed or newly leased. And when all renewable servers have been considered, we also check if any new server needs to be leased (line 15). If so, we will reset k and $renew$ to further explore the branch (line 16). Every time a server c_k is selected (line 20-21), it will be leased for the time of D_m and during this period, it will also be updated as leased in C. $Load[time]$ (or $Load[time + T_s]$, depending on whether c_k is renewed or newly leased) to $Load[time + D_m]$ will be reduced by $\mathbb{U}(c_k) - R$. After that, the algorithm checks if any other server needs to be leased or can be renewed (line 22-25). If not, $time$ will be increased to another position and both k and $renew$ will be reset accordingly. When the search finishes, the optimal cloud lease schedule S will be generated and returned (line 32-33), where a tuple (c_i, t_i, D_m) will be created and added into S for a server c_i newly leased at t_i, and another D_m will be added to the tuple's lease duration if it is renewed afterwards. We then have the following theorem.

Theorem 1: The algorithm proposed in Fig. 3 returns the optimal cloud lease schedule.

The proof can be found in [20].

B. Integrating with Locality-Awareness

As the users may be from various locations or time zones over the world, registering to different ISPs and in different ASes, a further optimization to the basic problem is thus to integrate with the locality-awareness, i.e., to maximize the number of users that connect to the local cloud servers. By achieving this, the delay between users and cloud servers can be effectively reduced. Another advantage is that such locality-awareness can also help to reduce the cross-boundary traffic (e.g. cross-ISP traffic), which is especially important when considering user assistance as discussed in the following subsections. To this end, we use an abstract notation "region" to represent the locality that we care about, which can be interpreted into different meanings in different contexts (e.g. an area with some extent of physical closeness or a group of ASes belonging to one ISP). Let $\mathbb{A} = \{A_1, A_2, \cdots, A_n\}$ be the set of regions that the cloud servers and users may be in. For a cloud server c_i, we use $c_i \in A_j$ to denote that it is in the region A_j. In addition, we further extend $\mathbb{P}(t)$ to $\mathbb{P}(A, t)$ to denote the online population of users in region A at time t. The basic problem thus can be rewritten as to find a proper cloud lease schedule S, subjecting to the following constraints:

[3]At current stage, Amazon EC2 has no charges on the traffic into the cloud as well as the traffic within the cloud.

Algorithm OptimalCloudLeaseSchedule()

Sort C by ascendant order of $\mathbb{C}_l(\cdot)/\mathbb{U}(\cdot)$;
for $\forall t \leq T$, $Load[t] \leftarrow R \cdot \mathbb{P}(t) - \mathbb{U}(c_0)$; **end for**
Set $Stack$ empty; $k \leftarrow 0$; $time \leftarrow 0$; $renew \leftarrow$ **false**;
$Cost \leftarrow 0$; $Cost_{min} \leftarrow \infty$; Set $Stack^*$ empty;
while true,
  if $time > T$ **or** $Cost \geq Cost_{min}$,
    if $Cost < Cost_{min}$,
      $Cost_{min} \leftarrow Cost$; $Stack^* \leftarrow Stack$;
    end if
    goto 27;
  end if
  $k \leftarrow k + 1$;
  if $k < |C| + 1$ **and** (k-th server in C is leased **or** ($renew$ **and** k-th server cannot be renewed)),
    continue;
  else if $renew$ **and** $k \geq |C| + 1$ **and** need to lease new server at $time$ based on $Load$,
    $k \leftarrow 0$; $renew \leftarrow$ **false**; **continue**;
  else if $k \geq |C| + 1$,
    goto 27;
  end if
  Push $\{k, time, renew, C\}$ in $Stack$;
  Update $Load$, C and $Cost$ accordingly;
  if no server needs to be leased and no server can be renewed at $time$ based on $Load$,
    Increase $time$ until a server can be renewed or new servers need to be leased or $time > T$;
    $k \leftarrow 0$; $renew \leftarrow$ if any server can be renewed;
  end if
  continue;
  if $Stack$ is empty, **break**;
  else
    Pop $Stack$; Update $Load$ and $Cost$ accordingly;
  end if
end while;
Generate optimal cloud lease schedule S from $Stack^*$;
return S;

Fig. 3: Algorithm to compute the optimal cloud lease schedule.

(1) Service Availability Constraint:

$$\forall (x_i, t_i, d_i) \in S, \text{ if } \exists (x_j, t_j, d_j) \in S \text{ and } x_i = x_j,$$
$$\text{then } [t_i, t_i + d_i) \cap [t_j, t_j + d_j) = \emptyset \ ;$$

(2) Streaming Quality Constraint:

$$\forall T_s \leq t \leq T,$$
$$\mathbb{U}(c_0) + \sum_{(x_i,t_i,d_i)\in S} \big(\mathbb{U}(x_i) - R\big) \cdot I_{\left[t\in[t_i+T_s, t_i+d_i)\right]} \geq R \cdot \sum_{A\in\mathbb{A}} \mathbb{P}(A,t) \ .$$

Our objective is now to minimize the lease costs:

$$Cost_{lease} = \sum_{(x_i,t_i,d_i)\in S} \mathbb{C}_l(x_i) \cdot d_i \ ,$$

as well as maximize the locality, which is defined as:

$$Locality = \frac{1}{R \cdot \sum_{t\leq T}\sum_{A\in\mathbb{A}} \mathbb{P}(A,t)} \cdot \sum_{t\leq T}\sum_{A\in\mathbb{A}} \min\left(R \cdot \mathbb{P}(A,t), \sum_{(x_i,t_i,d_i)\in S} \mathbb{U}(x_i) \cdot I_{\left[t\in[t_i+T_s, t_i+d_i), x_i\in A\right]}\right) .$$

It is easy to see that these two objectives may contradict with each other, as leasing more servers in each region can improve the locality but also increase the lease cost. We thus adopt the following linear combination form to align them together with different weights:

$$p \cdot \frac{Cost_{lease}}{Cost_{max}} + q \cdot (1 - Locality) \ ,$$

where p and q are two parameters that can assign different weights to the two goals, and $Cost_{max}$ is the minimum lease costs of the case where the demand is constantly set to be the maximum demand within T. As $Locality$ is a ratio of the intra-region streaming traffic over the total streaming traffic, $(1 - Locality)$ is thus the ratio of the cross-region traffic over the total traffic, which should also be minimized like $Cost_{lease}$. To make the lease cost part also a ratio ranged between $[0, 1]$, we further divide $Cost_{lease}$ by $Cost_{max}$ and then use parameters p and q to linearly combine the two parts together. This new problem can also be solved by the algorithm proposed in Fig. 3, with some proper but simple modifications. We defer the details to [20] due to space limitation.

C. Exploring User Resources

It is known that peer-to-peer streaming is highly scalable through exploring user contributed resources. However, in a pure peer-to-peer system, users may join or leave at their own wills, and their available upload bandwidth may vary significantly from time to time and from user to user. Even if the aggregate user contributed upload bandwidth is equal to or greater than the total demand, a pure peer-to-peer design may still suffer from content bottlenecks [14], where users have upload bandwidth but no expected streaming content available for sharing. An extreme is a flashcrowd; that is, during a peak time, many fresh users join the system with a vast amount of ready-to-share upload bandwidth but no content available.

Our CALMS could also benefit from such readily available resources in the User Layer, and yet it can elegantly mitigate the problems mentioned above. To this end, we introduce a group of parameters (α, β), which we call as *user assistance index*. Both of the two parameters range from 0 to 1. The parameter α determines the ratio of the user contributed upload bandwidth that can be effectively used to assist the live streaming. The parameter β denotes the minimum ratio of cloud servers to be reserved to deal with the content bottleneck (since the cloud servers always have the updated streaming content to share). The full version of our problem thus can be written as to find a proper cloud lease schedule S, subjecting to the following constraints:

(1) Service Availability Constraint:

$$\forall (x_i, t_i, d_i) \in S, \text{ if } \exists (x_j, t_j, d_j) \in S \text{ and } x_i = x_j,$$
$$\text{then } [t_i, t_i + d_i) \cap [t_j, t_j + d_j) = \emptyset \ ;$$

(2) Streaming Quality Constraint:

$$\forall T_s \le t \le T,$$
$$\mathbb{U}(c_0) + \sum_{(x_i,t_i,d_i)\in S} \big(\mathbb{U}(x_i) - R\big) \cdot I_{\left[t\in[t_i+T_s,t_i+d_i)\right]}$$
$$+ \min\left(\alpha \cdot \sum_{A\in\mathbb{A}} \mathbb{B}(A,t), \big(1-\beta\big)\cdot R \cdot \sum_{A\in\mathbb{A}} \mathbb{P}(A,t)\right)$$
$$\ge R \cdot \sum_{A\in\mathbb{A}} \mathbb{P}(A,t)\ ;$$

where $\mathbb{B}(A,t)$ is the aggregate user upload bandwidth in region A at time t. Our objective is still to minimize

$$p \cdot \frac{Cost_{lease}}{Cost_{max}} + q \cdot (1 - Locality)\ ,$$

while with the user-assistance taken into account, the locality measure is now calculated as

$$Locality = \frac{1}{R \cdot \sum_{t\le T}\sum_{A\in\mathbb{A}} \mathbb{P}(A,t)} \cdot \sum_{t\le T}\sum_{A\in\mathbb{A}} \min\Bigg(R\cdot\mathbb{P}(A,t),$$
$$\sum_{(x_i,t_i,d_i)\in S} \mathbb{U}(x_i)\cdot I_{\left[t\in[t_i+T_s,t_i+d_i),x_i\in A\right]}$$
$$+ \min\big(\alpha\cdot\mathbb{B}(A,t), (1-\beta)\cdot R\cdot\mathbb{P}(A,t)\big)\Bigg)\ .$$

To address this problem, some modifications need to be applied to the algorithm proposed in Fig. 3. First, the $Cost$ computation (line 21 and 29) now needs to integrate with the locality. In addition, $Load$ now needs to have a second dimension to distinguish the demands from different regions and when initialized (line 2), it needs to further subtract the user contributed upload bandwidth. We omit more details here due to the space limitation, which can be found in [20].

IV. Migration Implementation and Optimization

Previous section addresses the major design problems for CALMS. However, in practice, there still remain some issues that need to be considered carefully. First, the user demand and capacity may not be forecasted precisely, where the accuracy often degrades as the predication time get more forward than the current time. This renders that a statically computed optimal cloud lease schedule for a long period may become less useful due to the prediction error. Another issue is how to organize the leased servers in the Cloud Layer. Since they may be from different regions, a careless organization may introduce unnecessary cross-boundary and cross-region traffics and also degrade the performance. Similar situation also happens to the User Layer, where users need to be carefully organized to enable assistances among each other as well as enforce locality and good QoS. In this section, we present our solutions to address these issues.

A. Cloud Layer Organization and Evolution

As the Cloud Layer is the core part of the CALMS framework, its organization is thus very important to the performance of the migrated live media streaming application. As the tree structure is well known for its efficiency for organization, in our implementation, we adopt a tree structure for the Cloud Layer. In particular, we let server c_0 be the root of the tree and in charge of the whole Cloud Layer. The servers in the interior part of the tree relay the live streaming content to other servers to amplify the upload capacity. And the remaining servers in outskirts then transmit the live streaming content to the User Layer.

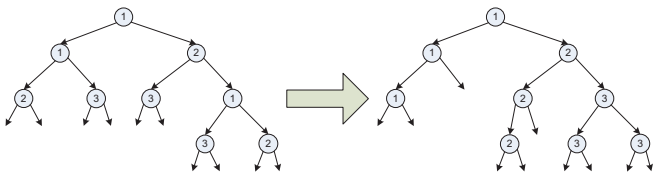


Fig. 4: Illustration of Cloud Layer organization.

Since the leased cloud servers may be from different regions, naively organizing servers into a tree may still cause poor performance. Fig. 4 shows an example, where each circle denotes a cloud server and the number inside denotes the region that the server belongs to. In this figure, there are 3 cloud servers of 2 unit upload capacities leased from each of the regions of 1 to 3, which are expected to provide similar upload capacities to each region. By a naive organization shown on the left part, the upload capacities provided by cloud servers to each region is 0, 4 and 6, which is greatly different from the expectation; while by handling carefully as shown on the right part, the upload capacities for each region becomes 3, 3 and 4. In addition, by the naive organization, to arrive at the User Layer, the live media streaming traffics must cross at least one region boundary and may cross as many as 3 region boundaries; while the carefully handled organization can effectively reduce such cross-boundary traffics.

To deal with these issues, we let server c_0 select and keep tracking a root server for each region from the leased servers. When a cloud server is newly leased, it will first be redirected to the root server in the same region, and the root server will be responsible to help the newly leased server join the subtree rooted at itself. If a region currently has no root server, server c_0 will temporarily take the role until a server from that region is leased. And if a root server is stopped due to the decreased demand in its region, the root server will select another server in the same region from its descendants to take its role. If no such server exists, server c_0 will take the role temporarily. In addition, the root server of each region will also help to optimize the server organization within its region, such as allocating the servers with higher upload capacities closer to itself as well as moving a server to a new parent closer to the root server than the current parent, so that the height of the subtree rooted at the root server can be minimized.

Through our simulation results in Section V, we observe this design, combined with the locality-aware scheduling proposed in the previous section, can substantially reduce the cross-boundary traffic as compared to a straightforward approach without locality-awareness.

B. User Layer Organization and Evolution

The main task of the User Layer organization is to enforce locality and good QoS as well as enable user-assistance. When a user joins the live media streaming session, it will first be redirected to the root server at the same region if there is available upload capacity within that region, otherwise the user will be redirected to the root server of other region with available upload capacity. The root server then decides where the user should go next. If there is available upload capacity directly from the servers in the region, the user will then be redirected to the server that can provide the upload capacity. If not, the user will be redirected to a server that contains information about users that can provide the upload capacity. This server will then randomly choose some users with enough aggregate upload capacities and send their IPs to the newly joined user. The user will then add these users as neighbors and exchange the live media streaming content with them by a peer-to-peer protocol.

To enforce good QoS and handle the content bottleneck problem that may exist, when a server randomly picks users for the newly joined user, only those with high playback buffer levels will be selected and sent out by the server. In addition, we also let a user with high upload capacity to preempt a user that directly downloads from a server but with low upload capacity. And When the playback buffer level of a user becomes low, either due to user dynamics or network bandwidth fluctuation, the user will request more neighbors from its server to maintain good live streaming quality.

To track such user information at the Cloud Layer while keeping good scalability, we adopt a hierarchical structure that is naturally provided by the tree organization. We let the last server that the user has been redirected to keep the full information of that user. Such information will then be aggregated and reported to this server's parent. This process will continue on level by level until reaching the root server at that region. The root server of each region will then aggregate and report the user information directly to server c_0. By this means, when a user is being redirected, the current server that issues the redirection can then choose the next stop for the user based on the collected user information.

To deal with user dynamics, each user will periodically report its updated information to its server. The server will also check whether the user has left if a report has not been received recently. And when a server needs to be stopped due to the decreased demands from its region, it will first redirect all its users one by one to server c_0 to redo the join process, then stop and leave the Cloud Layer.

C. Dynamic Lease Scheduling

In practice, the user demand and capacity forecast may not be 100% correct, and such inaccuracies may cause a statically computed cloud lease schedule for a long future period becomes less useful or even invalid. To overcome this problem, we use dynamic lease scheduling instead for the migration implementation. In particular, with the collected user information, server c_0 can know current user demands and upload capacities in each region. If current user demands are greater than the prediction or current user upload capacities are less than the prediction, server c_0 will dynamically recalculate the cloud lease schedule with the updated information and then lease additional cloud servers by the new schedule. Besides, when a cloud server has been leased for the multiple times of D_m, server c_0 will check if current user demands in that region are less than the half of current upload capacities even with this server stopped. If so, it will stop this server to reduce costs and recompute the cloud lease schedule accordingly.

In addition, due to the content bottleneck issue, sometimes the QoS perceived at users may degrade even the total upload capacities are still greater than the total user demands. To make the Cloud Layer responsive to such situations, we also collect the playback buffer level of each user when tracking other information. Server c_0 will then also check the minimum buffer level of the users who have already started playing the media streaming. If the minimum level is below a threshold, which indicates potential content bottleneck may happen, server c_0 will lease a new server to increase the content availability and recompute a new lease schedule accordingly.

Another conceivable approach is to adopt the results from classic ski-rental problems. In such approach, upon satisfying certain conditions that indicates system supply sufficiently exceeding user demand, the lease controller decides whether to end the idle cloud server's lease or keep leasing it to avoid abrupt cost upon unexpected demand surging. Such decision problem shares the same "rent-or-buy" structure as the classic ski-rental problem [9]. There are quite a number of solutions available for the ski-rental problem. For example, the optimal deterministic strategy for deciding when to end the lease of an idle server (i.e., to stop "renting" and "buy") is the 2-competitive [9], i.e., the worst-case overall cost is at most twice of the optimal computed with full future knowledge. However, two important difference between the ski-rental problem and our cloud leasing problem are that first, the cost of the "buy" and "rent" actions may change arbitrarily over time; second, we still have access to fairly-accurate estimates of near-future demand. These difference make the competitive ratio of classic ski-rental algorithms not directly applicable to our scenario. As a future work, it would be interesting to explore how we can exploit the insights from the online problem literature to further improve the performance of our dynamic lease scheduling.

V. Performance Evaluation

We run extensive simulations to evaluate CALMS. The simulations are conducted on the packet level and driven by real traces and data sets collected from Amazon EC2 and a popular live media streaming system (PPTV). We first briefly introduce the collected traces and data sets, and then continue on to present the methodology as well as the evaluation results.

A. Data Sets and Traces

1) Amazon EC2 Measurement: Our measurement on Amazon EC2 is mainly on the bandwidth distribution between

Different Region	Instance Lease (per hour)			Data Transfer Out (per month per GB)			
	Small	Large	Extra Large	First 1GB	Up to 10TB	Next 40TB	Next 100TB
US East (Virginia)	$0.085	$0.34	$0.68	$0.000	$0.120	$0.090	$0.070
US West (Northern California)	$0.095	$0.38	$0.76	$0.000	$0.120	$0.090	$0.070
EU (Ireland)	$0.095	$0.38	$0.76	$0.000	$0.120	$0.090	$0.070
Asia Pacific (Singapore)	$0.095	$0.38	$0.76	$0.000	$0.190	$0.150	$0.130
Asia Pacific (Tokyo)	$0.10	$0.40	$0.80	$0.000	$0.201	$0.158	$0.137

TABLE I: Location distribution and pricing policies of Amazon EC2 servers.

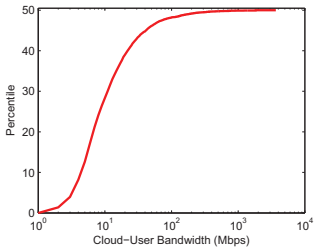


Fig. 5: Measured bandwidth distribution (CDF) between cloud servers and users.

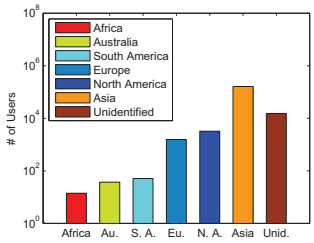


Fig. 6: Region distribution of PPTV users in one day's traces.

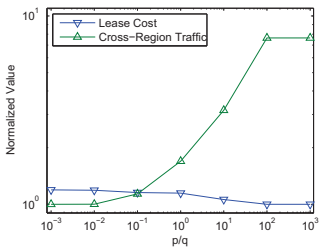


Fig. 7: Impacts of different p/q values on lease cost and cross-region traffic.

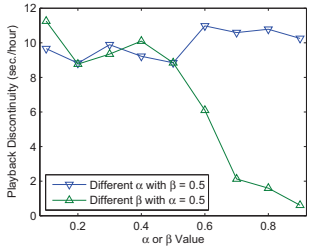


Fig. 8: Impacts of different α or β values on playback discontinuity.

cloud servers and users. In particular, we first send DNS requests to the EC2 domains to find the IP addresses of EC2 servers. The DNS server replies with a list of unique IP prefixes which are reserved by Amazon for their EC2 instances. Based on this IP list, we further probe these IP prefixes with ICMP, TCP and UDP packets from different locations, identifying active instances and then measuring their bandwidth accordingly. Fig. 5 shows the measured bandwidth distributions between different cloud servers and users. Other settings on Amazon EC2 in our simulations are adopted from previous measurement and evaluation works [3][10][17] as well as the official web site of Amazon EC2 [1]. Tab. I further summarizes the location distribution and pricing policies of Amazon EC2 that we use in our simulations.

2) PPTV Traces: PPTV is one of the largest commercial peer-to-peer live streaming systems to date, attracting over $100,000$ online users during peak times, and is also one of the mostly examined systems in academia [5][7][24]. Our simulations are based on traces from two of its popular channels, namely, CCTV3 and DragonBall. These traces are gathered by an online crawler that continuously collects information from each channel [21]. Fig. 6 shows the region distribution of PPTV users in one day's traces and Fig. 1 shows the user demands distributions and variations. Other settings about the live media streaming users are adopted from [5][24].

B. Methodology

With the data sets and traces from Amazon EC2 and PPTV, we then conduct extensive simulations to evaluate our migration framework. For comparison, we implement other three approaches: *Max-Central* statically provisions servers all from one region by the maximum user demand in the corresponding trace, which emulates the solution that uses centralized server clusters to provide the live media streaming service. *Max-CDN* also provisions servers by the maximum user demand, but the provisioned servers may be selected from different regions based on the average user demands from each region. This approach emulates the solution that uses CDNs to provide the streaming service. *P2P-Locality* only uses the server c_0 as the streaming server and delivers streaming content by peer-to-peer technology and with locality-awareness, which emulates the solution for peer-to-peer live media streaming.

In addition, we use the following four metrics in our simulation: *Lease costs* is the costs for leasing cloud servers, which represents the major concern from the live media streaming service providers; *Cross-region traffic* is the amount of traffic that crosses different regions. This metric shows the locality-awareness of an approach and is also an indicator to the general performance as lower cross-region traffic means that users are closer to their servers and the Cloud Layer are well organized; *Playback discontinuity* is the average duration that a user may experience discontinued playback per hour, which is mainly caused by that the streaming data packets fail to arrive at a user before its playback deadline; *Startup latency* is the latency taken by a user between its requesting to join the session and receiving enough data to start playback.

C. Impacts of Different Parameter Settings

We first conduct simulations to investigate the impacts of different parameter settings. Fig. 7 demonstrates how the lease cost and cross-region traffic change with different p/q values. For ease of comparison, the results are normalized by the corresponding minimum values. When p/q is small ($\leq 10^{-2}$), the cross-region traffic is minimized while introducing the highest lease costs. On the other hand, when p/q grows large ($\geq 10^2$), it results in the minimum lease cost but at the expenses of excessive cross-region traffic. Moreover, within the region near 10^{-1}, both the lease cost and cross-region traffic stay relatively low. We thus pick up this value $p/q = 0.1$ as the default for the remaining evaluations.

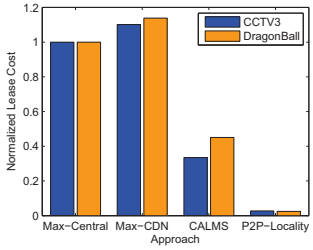


Fig. 9: Lease costs of different approaches.

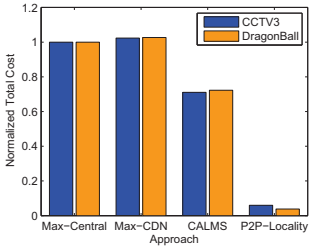


Fig. 10: Total costs of different approaches.

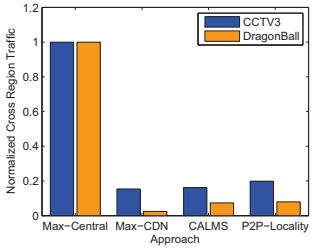


Fig. 11: Cross-region traffics of different approaches.

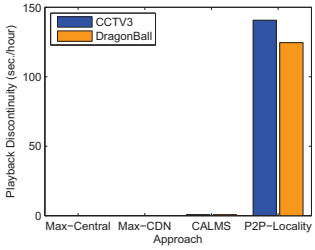


Fig. 12: Playback discontinuity of different approaches.

We next investigate the impacts of different α and β values on the playback discontinuity. The results are shown in Fig. 8. It is easy to see that the impact of β is more significant than that of α, while the results are different. When α becomes larger, the playback discontinuity slightly increases, which matches the definition of α since more user-assistances are involved, resulting in that more content bottlenecks may happen and degrade the playback quality. On the other hand, the playback discontinuity changes inversely with the value of β. This also matches the definition of β as reserving more cloud servers to compensate the content bottlenecks will improve the playback quality. More interestingly, the playback discontinuity will change more dramatically as either of the two parameters changes within the region of $(0.2, 0.7)$. We thus choose $\alpha = 0.2$ and $\beta = 0.7$ as the default setting.

D. Performance Observed at Service Providers

With the default parameter setting, we then conduct simulations to see how CALMS performs with both CCTV3 and DragonBall traces. Fig. 9 shows the lease costs of different approaches. For ease of comparison, the lease costs in each trace are normalized by the corresponding cost of the Max-Central approach. It is not surprising that P2P-Locality has the lowest costs. At the same time, our CALMS also has much lower lease costs, achieving 33.5%-45.1% of the Max-Central approach and 30.5%-39.6% of the Max-CDN approach, respectively. Another observation is that the lease costs in the DragonBall trace are generally higher than those in the CCTV3 trace. This is because the content provided in the DragonBall channel attracts much more Asian user demands than those from the other regions and the lease prices of Amazon EC2 in Asia are relatively higher, which further verifies the locality-awareness of both the Max-CDN and CALMS approaches. Since Amazon EC2 also charges for the data traffic transferred out of the cloud boundary, we also examine the total costs of different approaches, which is shown in Fig. 10. It is easy to see that even with the data transfer costs, CALMS can still reduce a great amount of the total costs by roughly 28%-30%, which further demonstrates the effectiveness of migrating the live media streaming application to the cloud service.

Fig. 11 shows the cross-region traffics generated by different approaches, which are also normalized by the corresponding cross-region traffic of the Max-Central approach. As the Max-CDN, CALMS and P2P-Locality approaches are all locality-aware, it is thus not surprising that their cross-region traffics are much lower than that of the Max-Central approach. Yet an interesting thing is that the cross-region traffics of the three approaches in the DragonBall trace are much lower than those in the CCTV3 trace. This is also because the DragonBall channel attracts much more Asian user demands than from the other regions, which allows the server/peer selections to be more dedicated in Asia and results in more intra-region traffics instead of cross-region traffics. Also, by comparing among the three approaches, we can find that by both dynamically provisioning enough cloud servers and exploiting user-assistance, CALMS can actually achieve a balance of the locality provided by either mechanism, always avoid the worst case – having the highest cross-region traffic, and stays close to the best one.

E. QoS Perceived by Users

Previous subsection has shown that migrating the live media streaming application to the cloud service can achieve good performance as observed at the service providers. Next we go on to explore possible benefits that may be brought to users.

We first investigate the playback discontinuity, which is shown in Fig. 12. We can see that CALMS performs similar to both Max- approaches and achieves very low playback discontinuity. On the other hand, due to the peer dynamics and content bottlenecks, P2P-Locality suffers relatively high playback discontinuity with the average as many as over 120 seconds per hour. To better understand the QoS perceived by each user, we also draw the CDF of the playback discontinuity for both CALMS and P2P-Locality in Fig. 13. It is clear to see that for P2P-Locality, some users may suffer playback discontinuity up to near 800 seconds per hour, which means that these users cannot watch the live media for more than 20% of the time. On the other hand, by provisioning enough cloud servers to provide both upload contents and capacities, CALMS significantly reduces the worst case playback discontinuity to around 1 minutes and afford more than 95% of the users to enjoy zero playback discontinuity.

Besides watching live media fluently, a user also prefers to short waiting time before the live media can start to play. Fig. 14 shows the startup latency of different approaches. Due to the delay of being redirected to other servers and filling up the playback buffers, both Max- approaches have the startup latency of about 2 seconds. CALMS, by provisioning enough

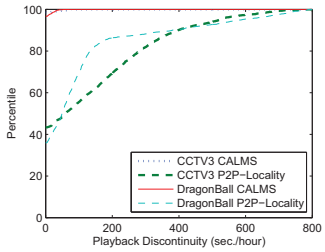


Fig. 13: CDF of playback discontinuity of CALMS and P2P-Locality.

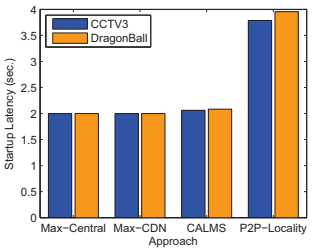


Fig. 14: Startup latency of different approaches.

cloud servers with upload contents and capacities, can also achieve a similar startup latency. On the other hand, the startup latency of P2P-Locality is almost of doubled length of the other three approaches due to the long latency for identifying enough peers with both the live media streaming contents and available upload capacities.

VI. Conclusion and Future Work

This paper presented CALMS, a generic framework that facilitates migrating live media streaming to a cloud platform. Being the very first paper towards this direction, we strived to offer fundamental understandings on the practical feasibility and theoretical constraints in the migration. We first modeled the basic problem of leasing cloud services to support time-varying user demands and developed an optimal algorithm. We then extended our solution to integrate locality-awareness and user-assistance, alleviating the impact from service globalization. We further designed a series of practical solutions for both cloud and user layer organization and optimization, as well as dynamic lease scheduling that accommodates inaccuracy in demand and capacity forecast. Extensive simulations driven by traces from Amazon EC2 and PPTV demonstrated the cost-effectiveness and superior streaming quality of CALMS, even with highly dynamic and globalized demands.

We are currently conducting more simulations to further evaluate and improve the migration framework with data sets and traces from other cloud service providers and live media streaming service providers. In addition, we expect to develop a prototype system to further verify and evaluate CALMS. We are also interested in exploring other open issues along this direction, such as designing better user demand and capacity forecast mechanisms as well as extending CALMS to integrate other type of cloud service platform, e.g., SpotCloud [18], a platform that allows the users to contribute/sell their idle computing resources and build their own cloud services.

Acknowledgment

Feng Wang and Jiangchuan Liu's research is supported by a Canadian NSERC Discovery Grant, a Discovery Accelerator Supplements Award, an NSERC Engage Grant, a MITACS Project Grant, and a China NSFC Major Program of International Cooperation Grant (61120106008). Minghua Chen's research is partially supported by the China 973 Program (Project No. 2012CB315904), and the General Research Fund grants (Project No. 411209, 411010, and 411011) and an Area of Excellence Grant (Project No. AoE/E-02/08), established under the University Grant Committee of the Hong Kong SAR, China, as well as two gift grants from Microsoft and Cisco. The authors thank Dr. Xiaojun Hei for providing PPTV traces.

References

[1] Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2/.
[2] M. Armbrust, R. G. A. Fox, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," University of California, Berkeley, Tech. Rep., 2009.
[3] S. Garfinkel, "An Evaluation of Amazon's Grid Computing Services : EC2 , S3 and SQS," Harvard University, Tech. Rep., 2008.
[4] M. Hajjat, X. Sun, Y.-W. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani, "Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud," in *ACM SIGCOMM*, 2010.
[5] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross, "A measurement study of a large-scale P2P IPTV system," *IEEE Trans. Multimedia*, vol. 9, no. 8, pp. 1672–1687, December 2007.
[6] C. Huang, J. Li, and K. W. Ross, "Peer-Assisted VoD: Making Internet Video Distribution Cheap," in *USENIX IPTPS*, 2007.
[7] Y. Huang, T. Z. J. Fu, D.-M. Chiu, J. C. S. Lui, and C. Huang, "Challenges, design and analysis of a large-scale P2P-VoD system," in *ACM SIGCOMM*, 2008.
[8] Z. Huang, C. Mei, L. E. Li, and T. Woo, "CloudStream: Delivering High-Quality Streaming Videos through a Cloud-Based SVC Proxy," in *IEEE INFOCOM Mini-Conference*, 2011.
[9] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator, "Competitive Snoopy Caching," *Algorithmica*, vol. 3, no. 1, pp. 79–119, 1988.
[10] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Comparing Public Cloud Providers," in *ACM IMC*, 2010.
[11] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska, "Dynamic Right-Sizing for Power-Proportional Data Centers," in *IEEE INFOCOM*, 2011.
[12] J. Liu, S. G. Rao, B. Li, and H. Zhang, "Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast," *Proceedings of the IEEE*, vol. 96, no. 1, pp. 11–24, January 2008.
[13] S. Liu, R. Zhang-Shen, W. Jiang, J. Rexford, and M. Chiang, "Performance Bounds for Peer-Assisted Live Streaming," in *ACM SIGMETRICS*, 2008.
[14] N. Magharei and R. Rejaie, "PRIME: Peer-to-Peer Receiver-drIven MEsh-Based Streaming," in *IEEE INFOCOM*, 2007.
[15] NetFlix, http://www.netflix.com.
[16] D. Niu, Z. Liu, B. Li, and S. Zhao, "Demand Forecast and Performance Prediction in Peer-Assisted On-Demand Streaming Systems," in *IEEE INFOCOM Mini-Conference*, 2011.
[17] Rackspace Cloud Servers versus Amazon EC2: Performance Analysis, http://www.thebitsource.com/featuredposts/rackspacecloud-servers-versus-amazonec2-performanceanalysis/.
[18] SpotCloud, http://www.spotcloud.com/.
[19] K. Sripanidkulchai, S. Sahu, Y. Ruan, A. Shaikh, and C. Dorai, "Are Clouds Ready for Large Distributed Applications?" in *SOSP LADIS Workshop*, 2009.
[20] F. Wang, J. Liu, and M. Chen, "On Cloud-Assisted Live Media Streaming," Simon Fraser University, Tech. Rep., 2011.
[21] F. Wang, Y. Xiong, and J. Liu, "mTreebone: A Collaborative Tree-Mesh Overlay Network for Multicast Video Streaming," *IEEE Trans. Parallel and Distributed Systems*, vol. 21, no. 3, pp. 379–392, March 2010.
[22] C. Wu, B. Li, and S. Zhao, "Multi-Channel Live P2P Streaming: Refocusing on Servers," in *IEEE INFOCOM*, 2008.
[23] Y. Wu, C. Wu, B. Li, X. Qiu, and F. C. Lau, "CloudMedia: When Cloud on Demand Meets Video on Demand," in *IEEE ICDCS*, 2011.
[24] K. Xu, H. Li, J. Liu, W. Zhu, and W. Wang, "PPVA: A Universal and Transparent Peer-to-Peer Accelerator for Interactive Online Video Sharing," in *IWQoS*, 2010.
[25] J. Zhu, Z. Jiang, and Z. Xiao, "Twinkle: A Fast Resource Provisioning Mechanism for Internet Services," in *IEEE INFOCOM*, 2011.