2015 IEEE 8th International Conference on Cloud Computing

QoE Driven Server Selection for VoD in the Cloud

Chen Wang*†, Hyong Kim*, Ricardo Morla†
chenw@cmu.edu, kim@ece.cmu.edu, ricardo.morla@fe.up.pt
*Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, USA
†INESC Porto and Faculty of Engineering, University of Porto, Porto, Portugal

***Abstract*—In commercial Video-on-Demand (VoD) systems, user's Quality of Experience (QoE) is the key factor for user satisfaction. In order to improve user's QoE, VoD providers replicate popular videos in geo-distributed Cloud and deploy cache servers close to users. Generally, the VoD provider selects a server for the user request according to the user's location. Usually geographically closely located servers would provide lower network delay. However, the performance of VoD servers deployed in cloud virtual machines (VM) depends not only on the network delay but also resource contention due to other VMs and highly dynamic user demands. Thus, QoE offered by the server varies greatly over time as user demands and network traffic fluctuate regardless of the location. Selecting a server close to users sometimes reduces the network delay but cannot guarantee QoE in general. We believe that end-users have the best perception of server performance in terms of their QoE rather than the servers themselves. What user perceives incorporate performance of all elements, such as network delay and server response time in VoD service. We propose VoD server selection schemes that dynamically select servers according to user's QoE feedback. We integrate our server selection schemes with Dynamic Adaptive Streaming over HTTP (DASH) clients and evaluate our system both in simulation and in Google Cloud. Results show our system improves user QoE up to 20% compared to existing solutions[1].**

***Keywords*-Video on Demand, Quality of Experience, Server Selection, Cloud applications**

I. Introduction

Popular VoD providers, Netflix, YouTube, Amazon Video and Hulu together contribute over 52% of downstream traffic in North America in 2014 [1]. Improving users' Quality of Experience (QoE) for user satisfaction is crucial to gain the market share.

In order to meet the high demand of video services from billions of users, VoD providers either build their Content Delivery Network (CDN) on their own data centers or they resort to commercial Cloud and CDN providers [2] for content delivery. Many video contents are generally replicated over multiple locations [3]. Upon user request, VoD provider assigns a particular server to a user. Server selection scheme significantly impacts user's QoE. A measurement study on YouTube shows that network delay between a user and data center plays an important role in YouTube video selection process [2]. Other factors influence its server selection as well, such as load-balancing, diurnal effects, availability of videos, and hot-spots. Vijay et al. find that Netflix replicates content across multiple commercial CDNs and assigns its users a particular CDN. This assignment remains unchanged over many days even if other CDNs can offer better quality of experience [4]. They also show that the network bandwidth to users can be improved by 12% if a CDN server is adaptively selected thus improving the user QoE. However, optimal assignment is difficult as network status cannot be obtained in real-time.

When VoD systems are deployed in Cloud, the server and network performance vary dynamically as the resources are shared and Cloud users generally neither have control nor visibility of the resources. The VoD provider generally selects a server for the user request according to the user's location. Usually geographically closely located servers would provide lower network delay. However, the performance of VoD servers deployed in Cloud virtual machines (VM) depends not only on the network delay but also resource contention due to other VMs and highly dynamic user demands. Thus, QoE offered by the server varies greatly over time as user demands and network traffic fluctuate regardless of the location. Selecting a server close to users sometimes reduces the network delay but cannot guarantee QoE in general. Furthermore, even if the servers can monitor all system parameters such as CPU, memory, I/O and storage utilization as well as network metrics, it would be extremely difficult to formulate a proper algorithm to determine the best server for the user in terms of user QoE. Complex interactions of server VMs and network are quite challenging.

We believe that end-users have the best perception of server performance in terms of their QoE rather than the servers themselves. What users perceive incorporate performance of all elements, such as network delay and server response time in VoD service. We propose two VoD server selection schemes that dynamically select servers according to user's QoE feedback. The first one is called QoE driven Adaptive Server Selection (QAS). In QAS, each user monitors its real-time QoE for all available servers assigned to the user and adaptively selects a server according to its own perception of QoE. The second one, called Cooperative QoE

[1]This work was supported in part by the FCT (Portuguese Foundation for Science and Technology) through the Carnegie Mellon Portugal Program under Grant SFRH/BD/51150/2010, CMU-SYSU CIRC and SYSU-CMU IJRI.

2159-6190/15 $31.00 © 2015 IEEE
DOI 10.1109/CLOUD.2015.125

driven Adaptive Server Selection (CQAS), users cooperate to obtain other users' view on additional servers. What users perceive as a good server is much more informative than a lot of system and network metrics.

We integrate our server selection schemes with Dynamic Adaptive Streaming over HTTP (DASH) clients and evaluate our system in production cloud environment. We evaluate our system both in simulation and in Google Cloud. Results show our system improves user QoE up to 9% and 20% compared to existing solutions for QAS and CQAS respectively.

The remainder of the paper is organized as follows. Section II shows related works. Section III presents our system design. In section IV, we address a practical content discovery issue that facilitates QAS in VoD system. Section V describes how QAS and CQAS can be integrated into a DASH client. Section VI provides evaluations of QAS and CQAS for DASH clients in both simulation and real-world experiments in Google Cloud. Conclusions are presented in section VII.

II. Related Works

There are several works in the area of server selection in large-scale VoD systems. YouTube study [2] concludes that the network delay between server and user is an important factor that impacts YouTube's server selection. [4] suggests that instantaneous bandwidth between a user and Netflix CDN could be used for server selection. Nevertheless, obtaining real-time network delay or instantaneous bandwidth is extremely difficult. [6] proposes to use QoE as a motive to select paths in multi-path streaming. However, their work needs additional servers that act as intermediate nodes to choose among multi-path. Dynamic Adaptive Streaming over HTTP (DASH) improves user QoE under dynamic network environment [7]. DASH adapts video quality based on the available bandwidth of a connection between user and video server. In order to achieve the rate-adaption, DASH moves the control of bit-rate selection entirely to the client side and encodes videos into multiple bit-rates beforehand. DASH client periodically probes the available bandwidth and buffer condition to decide the quality level of each video chunk.

III. Overview of QAS and CQAS

The main concept in QAS and CQAS is to make each client aware of several candidate servers and evaluate these candidate servers in real time according to its own QoE. The client controls the server selection in real-time so it can adaptively select the best server to satisfy its QoE. We deploy an agent (client agent) running in each client to collect user's experience on different servers. The server runs an agent (cache agent) that provides an initial set of candidate servers to the client as shown in figure 1.

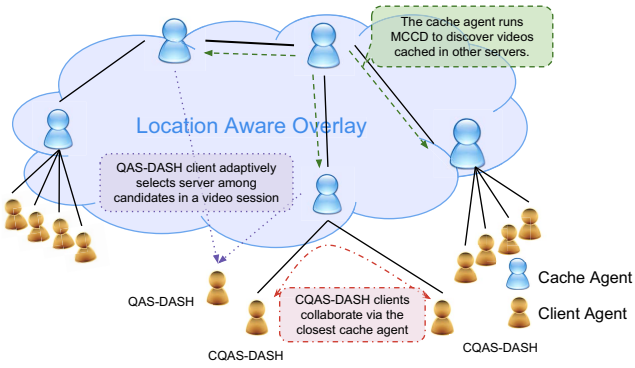


Figure 1: System Design

The client agent first connects to the cache agent to query a list of candidate servers that are geographically close with possibly low network delay. We propose a distributed algorithm, Multi-Candidate Content Discovery (MCCD), for cache agents to autonomously discover K closest candidate servers for all available videos. VoD systems generally cache videos dynamically according to the popularity of videos. Existing content discovery schemes such as DHT does not adapt to dynamic changes in content placement. Simple content discovery algorithms in unstructured P2P network scale poorly due to flooding and do not guarantee to find the content. In order to reduce the lookup time of K candidate servers, we cache a Candidate Server Table (CST)[2] in each cache agent. It includes K closest candidate server for popular videos. As content placement and server topology change dynamically, CST is updated according to these changes. After obtaining a list of candidate servers, the client agent initiates multiple connections to these candidate servers to stream the video. During streaming, the client agent evaluates candidate servers with its QoE in real time and chooses the one that offers the best QoE. In DASH and HTTP progressive downloading, video chunk is a segment of video that usually lasts several seconds. We propose a real-time QoE model that computes user QoE in terms of video chunk. We integrate QAS and CQAS with Dynamic Adaptive Streaming over HTTP (DASH) client and compare them in production Cloud.

In summary, QAS and CQAS operate as follows:
1. Each client agent picks up the closest VoD server as its cache agent (via DNS based server selection).
2. Cache agents run MCCD to discover K closest servers in terms of network latency for all available contents.
3. Client agents query their cache agents to obtain a candidate server list for the video request.
4. The client agents run QAS in DASH to stream the video.

[2]The CST can be cached for all videos in commercial VoD systems like Netflix or Hulu whose collections range from tens to hundreds of thousands of videos. Considering, there are totally $M = 100,000$ videos available and $K = 4$ candidate servers, the size of CST is $M \cdot K \cdot 32Bytes = 12.8MB$. For VoD system like YouTube whose unique reference file ID counts up to 25 million and keeps increasing, the CST can be maintained only for popular videos.

5. The client agents that run CQAS to share their QoE on candidate servers to other clients.

IV. Multi-Candidate Content Discovery

We describe MCCD that discover K candidate servers that are close to the client for one video request.

A. Location Aware Server Overlay

An easy way to discover videos from adjacent servers is to make each server flood their cached video list to servers nearby. To run flooding-like algorithms, we need an overlay network to connect cache agents close to each other. To prevent duplication, the overlay network has to be built without loops. We construct a simple location-aware overlay network using Algorithm 1. It builds a tree graph, denoted as G. $A = \{A_1, A_2, \cdots, A_M\}$ are M cache agents to be organized. The algorithm first finds two nodes with the minimum RTT among all pairwise RTTs and connects them to initialize G. The pairwise RTTs are the average of RTTs obtained via 10 ICMP pings between all pairs of servers at the time of overlay construction. It then iteratively adds a node that has the minimum RTT to the closest node in G, as shown in line 4 to line 7. To delete an existing node A_d, all child nodes of A_d need to be reconnected to nodes that are not in A_d's branch. Though the overlay construction introduces ping traffic between all pairs of servers, we believe it is acceptable as it is a one-time cost.

Algorithm 1: Construction of Location Aware Overlay

Data: All cache agents, $A = \{A_1, A_2, \cdots, A_M\}$
Pairwise RTTs between agents, $T = \{t_{ij}, 1 \le i \le M\}$
1 **Initialize overlay graph:** $G = \{V, E\}, V = \{A_x, A_y\}, E = \{(A_x, A_y)\}$ **where** $t_{xy} = \min T$
2 **Denote** $B = A - \{A_x, A_y\}$
3 **while** $B \neq \emptyset$ **do**
4 **Denote** $T_{BV} = \{t_{bv} | A_b \ in B, A_v \in V\}$
5 **Find** $t_{xy} = \min T_{BV}$
6 **Update** G **with** $V = V \cup \{A_y\}$ **and** $E = E \cup \{(A_x, A_y)\}$
7 $B = B - \{A_y\}$

B. MCCD

Unlike unstructured P2P network, we run the content discovery once at bootstrapping stage and cache the *Candidate Server Table (CST)* for popular videos in all cache agents. We propose a distributed content discovery algorithm, MCCD, to build CST on each cache agent as shown in algorithm 2. For each cache agent A_i, $V_i = \{v_j\} \subset V$ is the list of videos cached locally in A_i , where $V = \{v_t | t = 1, \cdots, T\}$ denotes all available videos in VoD system. In MCCD, each agent builds its own CST at the bootstrapping stage by flooding its locally cached video list to its directly connected neighbors. The neighbors receiving the list will merge list items into their own CSTs and iteratively forward the newly added items to their neighbors until each agent's CST is complete. Different from flooding algorithms, our cache agent stops forwarding messages once K candidate slots have been fully filled. We prove in theorem 1 that the amount of MCCD traffic is constant. The proof is omitted due to the page limit.

Theorem 1: The total outbound traffic for an agent A to build $\mathbf{CST}_A$ is proportional to $T \cdot K$, where T is the total number of videos and K is the number of candidate servers.

The outbound traffic in MCCD is independent of the size of the server overlay network. It increases linearly with the number of videos. It is acceptable as the number of popular videos is finite. Once CST on a cache agent are complete, the cache agent can look up candidate servers from its local CST and answer the client's request directly.

Algorithm 2: Multi-Candidate Content Discovery

Data: Local cached contents for agent $A_i, V_i = \{v_t\} \subset V$
1
2 **begin**
3 Initialize **CST** for A_i as $\mathbf{CST}_{A_i} = \{L(j) | 1 \le j \le T\}$
4 **for** $v_j \in V_i$ **do**
5 $L(j) = \{< A_i, 0 >\}$
6 Initialize **content update** message for A_i as $U_i \leftarrow \emptyset$
7 $n_{\text{forward}} = 1$ **for** $v_j \in V_i$ **do**
8 Add $< v_j, A_i, n_{\text{forward}} >$ into U_i
9 Send U_i to all neighbors of A_i
11 **while** ***Receive*** U_n ***from node*** A_n **do**
12 $U_i \longleftarrow \emptyset$
13 **for** $< v_j, A, n_{forward} > \in U$ **do**
14 Sort items in $L(j)$ by their values ascendingly
15 **if** $\|L_{ij}\|_0 \le K$ **then**
16 $L(j) = L(j) \cup \{< A, n_{\text{forward}} >\}$
17 $U_i = U_i \cup \{< v_j, A, n_{\text{forward}} + 1 >\}$
18 **else**
19 **if** $L(j)[k].value > n_{forward}$ **then**
20 Delete $L_j[K]$
21 $L_j = L(j) \cup \{A, n_{\text{forward}}\}$
22 $U_i = U_i \cup \{< v_j, A, n_{\text{forward}} + 1 >\}$
23 **if** $U_i \neq \emptyset$ **then**
24 Send U_i to all A_i's neighbors except A_n

C. CST Maintenance

Cache agents need a mechanism to update CST to handle the changes in server overlay network and content placement. We develop following mechanisms to update CST.

Video Deletion: When agent A deletes a video, agent A notifies all other agents about the deletion. Agent A floods a message "DELETE v_d on A" to all neighbors and lets the neighbors update their CSTs and forward the message to their neighbors iteratively.
Video Addition: Agent A caching a new video does not need to notify all other agents. Only servers nearby are interested in what agent A is caching. Agent A sends an update message with the content to all its neighbors.
Cache Agent Leaving: Before performing overlay changes as section IV-A describes, the leaving agent sends out deletion messages for all its cached videos.
Cache Agent Joining: When an agent joins to the cache agent overlay, the agent adds all its cached items to an update message and floods the update message in the updated overlay network.
Periodic Maintenances: An agent's CST can be corrupted or missing items. In order to complete the agent's CST, each cache agent periodically runs MCCD to fix corrupted CSTs. The period can be set to a relatively long period (i.e. several hours or one day).

V. QAS AND CQAS

In this section, we describe QAS and CQAS in detail. QAS is a QoE driven Adaptive server Selection scheme in which the client selects servers adaptively according to its QoE on candidate servers. CQAS is a group of QAS clients that connect to the same cache agent, cooperating to know each other's QoE on candidate servers.

A. QoE Model

Measuring user Quality of Experience (QoE) for video streaming is difficult. It involves a lot of factors such as, the quality of the video, the frequency of freezing events, the duration of each freezing event, the join time and so on. Besides, there are other factors that influence QoE such as the genre and type of videos, the popularity of videos and the user device resolution [5]. In this paper, we ignore factors that do not change once the streaming starts, such as the join time, the genre and the type of video.

To compute real-time QoE, we propose an approximate chunk QoE model that evaluates video quality and freezing events for every chunk within a DASH streaming session. In DASH, video quality of a chunk is determined only by its bit-rate. Study in [11] claims that video quality follows a logarithmic law over bit-rates, so they propose a video quality model for DASH as shown in Equation (1), where r_i is the selected bit-rate for chunk i and r is the possibly maximum bit-rate. a_1 and a_2 are positive adjustable constants.

$$Q_{\text{video\_quality}}(r_i) = a_1 \ln \frac{a_2 r_i}{r} \tag{1}$$

We also consider freezing events. There are only two possible scenarios for freezing. The streaming either freezes before the chunk is received or does not freeze until the chunk is received. We regard no freezing as the best case. The QoE deteriorates as the freezing time gets longer. Kester et al. [12] find that user experience deteriorates with the freezing time as shown in Equation (2). t denotes the length of freezing time and $c_1 \sim c_3$ are positive fitted coefficients. When $t = 0$, Q_{freezing} is 5, which is the maximum value in MOS score system [13].

$$Q_{\text{freezing}} = \begin{cases} 5 - \frac{c_1}{1+(\frac{c_2}{t})^{c_3}} & t > 0 \\ 5 & t = 0 \end{cases} \tag{2}$$

Video quality and delivery effect are both important for user QoE. We simply combine above two models to measure the chunk QoE. However, any other QoE model can be used in our work without difficulty.

$$Q(t, r_i) = \delta \cdot Q_{\text{freezing}}(t) + (1 - \delta) \cdot Q_{\text{video\_quality}}(r_i) \tag{3}$$

B. DASH Streaming

In order to validate the effectiveness of QAS scheme, we integrate QAS in popular DASH streaming clients, referred to as QAS-DASH. Before describing how QAS works in DASH, we briefly introduce DASH streaming. DASH is the MPEG adaptive streaming standard, which includes a description of Media Presentation Description (MPD) manifest file and video chunks. The MPD manifest file describes the available bit-rates of video tracks and audio tracks, their URL addresses, chunk lengths and other characteristics. DASH client downloads MPD files beforehand to retrieve a complete list of video and audio chunks. DASH client then determines the preferred bit-rate of chunk to download based on its available buffer size and network bandwidth. We denote the bit-rate selection in DASH as a function of instantaneous bandwidth bw_i and the available buffer size B_i at previous chunk, as shown in Equation(4).

$$r_{i+1} = DASH_{\text{selection}}(bw_i, B_i) \tag{4}$$

Details of commercial players like Microsoft Smooth Streaming and Adobe OSMF can be found in [14]. We use this module in QAS-DASH and CQAS-DASH and it can be instantiated by any existing bit-rate selection logic.

C. QAS-DASH

QAS-DASH gives the client agent the flexibility of changing servers dynamically to achieve desirable QoE for every chunk. The client agent starts with the closest server and initializes each candidate server with an empirical QoE evaluation. As the video plays, the client agent updates candidate servers' evaluations with its own experience. When the client agent requests a video chunk, the client agent chooses a server according to its real-time QoE feedback. Details of QAS-DASH are described in algorithm 3. $f_{QoE}(S)$ is the real-time QoE evaluations for candidate servers S. It is initialized as an empirical QoE value, denoted as $\tilde{Q}$. In line

3, client agent pings all candidate servers and selects the closest one to stream the initial chunk. In line 4, the smallest bit-rate is selected as the bit-rate of initial chunk. Each iteration in the WHILE loop is the server selection process for one chunk. Line 9 to line 11 shows the QoE evaluation on candidate servers. The algorithm uses an exponential moving average to update $f_{QoE}(S)$ in line 11 and selects the best server based on $f_{QoE}(S)$ in line 14. T_{chunk} is the playback time of a chunk. $t_i^{\text{buffering}}$, bw_i, r^i and B^i are buffering time, bandwidth capacity, bit-rate and buffer size measured at the time receiving chunk i. The client only switches to the selected server probabilistically. This prevents server overload and synchronization of server changes.

Algorithm 3: QAS-DASH

Data: Candidate servers, $\mathbf{L} = \{S_1, S_2, \cdots, S_k\}$
Available Bitrates, $\mathbf{R} = \{r_1, r_2, \cdots, r_{\max}\}$
1 **Initialize:**
2 $f_{QoE}(S) \leftarrow \tilde{Q}$ for all $S \in \mathbf{L}$;
3 Initialize the selected server to download chunk $i = 0$ as the closest server to current client $S^0 \leftarrow find\_closest(\mathbf{L})$;
4 Initialize $r^0 \leftarrow \min(\mathbf{R})$;
5 $B^0 \leftarrow T_{chunk}$;
6 Downloading chunk $i = 0$;
7 **while** *i is not the last chunk* **do**
8 **Record chunk metrics:** $t_i^{\text{buffering}}, bw_i, r^i, B^i$;
9 **Get QoE for current chunk:**$Q_i = Q(r^i, t_i^{\text{buffering}})$;
10 **Update QoE assessment and bit-rate selection:**
11 $f_{QoE}(S_i) \leftarrow \alpha \cdot Q_i + (1-\alpha) \cdot f_{QoE}(S_i)$;
12 **Adaptively select bit-rate for next chunk:** $r^{i+1} \leftarrow DASH_{\text{selection}}(bw_i, B^i)$;
13 **Find the server with the best QoE evaluation:**
14 $S_{\max} = \textbf{argmax}_{S \in \mathbf{L}}\{f_{QoE}(S)\}$;
15 **Probabilistically select S^{i+1} from $S_{\max}$ and S^i:**
16 $S^{i+1} = probSelect(S^i, S_{\max})$;
17 **if** $S^{i+1} \neq S^i$ **then**
18 $r^{i+1} \leftarrow \min(r^{i+1} + 1, r_{\max})$;
19 **Download chunk $i+1$ with r_{i+1} from S^{i+1}** ;
20 $i \leftarrow i + 1$;

D. CQAS

In algorithm 3, QAS-DASH client evaluates $f_{QoE}(S)$ based on its own experience. It is possible to avoid selecting some servers that gave bad QoE in the past. Besides, the QAS-DASH client initialize $f_{QoE}(S)$ with an empirical value $\tilde{Q}$. If the initially selected server gives the client a better QoE than the empirical value $\tilde{Q}$, the QAS-DASH client will not try any other servers. Thus the client will not even have a chance to explore other candidate servers.

We believe the cooperation among client agents would be beneficial in finding the better server. An agent cooperatively learns about servers' performance experienced by others. The cooperative learning of server performance based on QoE masks the complexity of network, server and user behaviors. Clients in neighboring network or data center would find the better server without understanding details of network or server status. We assume the client agents that are associated to the same cache agent are close to each other. The cache agent can be a rendezvous point for client agents to exchange their QoE on candidate servers. The cooperation among clients is detailed in Algorithms 4 and 5. Algorithm 4 shows how a client agent A^u reports its QoE on candidate servers to its cache agent A^c. Algorithm 5 shows how the cache agent A^c evaluates server performance according to clients' QoE. It also illustrates how the cache agent replies to the client with the server performance evaluated by latest user QoE.

Algorithm 4: CQAS-DASH on Client Agent

Data: Client Agent: A^u
Candidate servers, $\mathbf{L} = \{S_1, S_2, \cdots, S_k\}$
QoE Experienced: $f^u_{QoE}(s), s \in L^u$
1 **while** *i is not the last chunk* **do**
2 **Run QAS-DASH algorithm line 8 to 12**;
3 **if** $i\%W == 0$ **then**
4 Send $f^u_{QoE}(s)$ for $s \in L^u$ to the closest cache agent A^c;
5 **if** *Receive $F^c_{QoE}(L^u)$ from the cache agent A^c* **then**
6 $f^u_{QoE} \longleftarrow F^c_{QoE}$;
7 **Run QAS-DASH algorithm line 13 to 20**;

Algorithm 5: CQAS-DASH on Cache Agent

Data: Cache Agent: A^c
QoE Evaluation Table:
$F^c_{QoE}(S), S \in A^c = \{A^c_1, A^c_2, \cdots, A^c_M\}$
1 **while** *TRUE* **do**
2 **if** *Receive $f^u_{QoE}(s)$ from client agent A^u for candidate servers $s \in L^u$* **then**
3 $F^c_{QoE}(s) \leftarrow (1-\lambda) \cdot F^c_{QoE}(s) + \lambda \cdot f^u_{QoE}(s)$ for $s \in L^u$;
4 Send $F^c_{QoE}(s), s \in L^u$ to client agent A^u;

Our cooperation algorithm works similarly as ant in ant colony [15]. Ants in the colony leave pheromone on paths they pass by to help others to find a shorter path to resources. The shorter the path is to the resources, the stronger the pheromone is left on the path. Other ants that have no experience with these paths always choose the path according to the strength of the pheromone on these paths. However, the QoE based server evaluations are not exactly the same as the pheromones. The pheromone on a path

evaporates as time elapses if there are no ants passing the path. In server selection, clients do not converge to one best server. When users adaptively select servers with the best QoE in real time, load would be balanced among servers. It rarely happens that one server is not selected by any client. The evaporation of pheromone is reflected as the updating process of server evaluations in algorithm 5. Client agent sends its local QoE evaluation for candidate servers to the cache agent periodically and the cache agent builds up an evaluation table $F^c_{QoE}(S)$ for all servers. It accepts client agents' periodic updates and updates its $F^c_{QoE}(S)$ with a forgetting factor $(1-\lambda)$. The forgetting factor helps evaporating the outdated server evaluations. The latest server QoE for the client's candidate servers L^u is then sent back to the client agent. Therefore, the better QoE a client experiences with a server, the better evaluation of the server is updated in the cache agent. Even clients that do not have experience with that particular server learn other neighboring clients' latest experience with the server.

E. Overhead Analysis of QAS and CQAS

QAS and CQAS-DASH clients monitor QoE on candidate servers and select servers adaptively within a streaming session, thus introduce communication and computation overhead. The QAS-DASH client communicates with its closest cache agent only once, before the streaming session starts, to obtain K candidate servers for the requested video. The CQAS-DASH client periodically reports its evaluations of K candidate servers to its closest cache agent. Considering there are N clients reporting K values to M cache agents periodically, on average each cache agent receives $K \cdot N/M$ values per period. In large-scale VoD system, the number of servers M is supposed to be provisioned proportional to user demand N, namely $N/M = c$ is a constant. In this paper, we assume K is a small integer between $2 \leq K \leq 5$, so $K \cdot N/M$ is a constant. As long as each cache agent can afford $K \cdot N/M$ traffic in period T, we believe the constant communication overhead is worthy for the QoE improvement described later in Section VI. About computational overhead, each QAS/CQAS-DASH client computes QoE for each chunk and maintains $F^c_{QoE}(S)$ value for K candidate servers, yet the incurred overhead should not be noticed on user side.

VI. Evaluations

A. Google Cloud Experiments

We evaluate the performance of QAS-DASH and CQAS-DASH in Google Cloud. We implement both client and cache agents in Python and deploy cache agents in Google Cloud. We deploy 8 cache agents in 8 different zones in Google Cloud and use 9 clients, in Netherlands, Ireland, Virginia, Iowa, Texas, California, Hong Kong, Singapore, and Japan. Videos are encoded into 9 bit-rates varying from 2Kb to 6Mb. To emulate content placement algorithm, the most popular video is cached in all servers and the least popular video is only cached in two servers randomly chosen. The number of servers caching a particular video is proportional to the video's popularity. We assume the video popularity follows Zipf distribution. We force standard DASH client to always pick up the closest server (with minimum RTT) as the DNS based server selection implemented in many production systems. We set the number of candidate servers as 2 in QAS-DASH and CQAS-DASH.

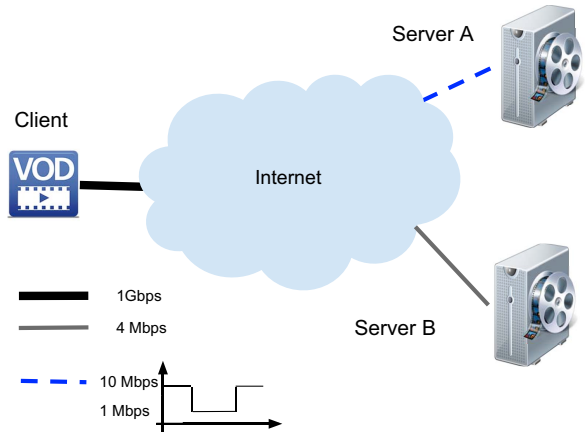
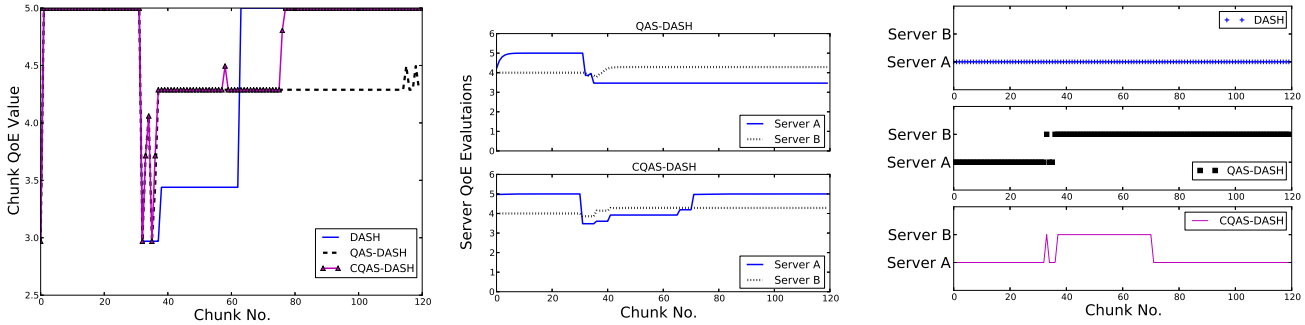


Figure 2: Single Session Experiment Setup

1) Single Session Experiment: We show how QAS-DASH client adaptively switches servers in figure 3 when the server capacity is throttled. Figure 2 shows the experiment setup. We deploy cache agents in two zones that belong to the same region in Google Cloud. We do not have control on the commercial Cloud and the network between the client and two cache agents. We use Wonder Shaper [16] to throttle server capacity to emulate the changes in the bandwidth between client and server. Server A is provisioned with 10Mbps capacity initially. We later throttle its bandwidth capacity to 1Mbps after the video played 30 chunks. The capacity in server A then recovers to 10Mbps after the video played 60 chunks. Server B is provisioned 4Mbps all the time. Figure 3 (a) compares chunk QoE values for DASH, QAS-DASH and CQAS-DASH clients. DASH client's chunk QoE drops when server A is throttled. QAS-DASH client drops when it switches to server B, but with a less degradation. CQAS-DASH explores two servers in real time and always chooses the one offering better QoE. Figure 3 (b) illustrates the benefit of agent cooperation in CQAS-DASH. QAS-DASH client does not know A when A's capacity is recovered. CQAS-DASH client learns increasing QoE with server A when A's capacity is recovered. QAS-DASH client evaluates servers only based on its own experiences so it is not aware of server A's recovery. On the other hand, CQAS-DASH client dynamically finds others' evaluation and selects servers according to the updated evaluation. Figure 3 (c) draws which server is selected for all chunks in various clients. DASH client selects server A and remains



(a) Chunk QoEs (b) Server QoE Evaluations (c) Server Selections

Figure 3: Single Session Experiment

with it. QAS-client switches to server B once it suffered bad QoE with server A. When A's capacity is recovered, QAS-DASH client relies on its own experience only and remains with B. On the contrary, CQAS-client retrieves QoE evaluations from its cache agent and becomes aware of server A's recovery quickly.

2) Multiple Session Experiment: We test 54 video sessions with 9 clients in figure 4. Figure 4 (a) shows the deployment[3] and the overlay network for all agents[4]. Figure 4 (b) shows the cumulative distribution of session QoE for all user requests. Session QoE is the average of all chunks' QoE in a single video steaming session. Considering the client with the worst QoE, CQAS's worst client has better QoE than QAS's worst client, and better than DASH's worst client. The user QoE in QAS and CQAS at every percentiles improves over DASH. In figure 4 (b), when we observe the percentage of users around 0.2, it shows that DASH guarantees 80% users with QoE above 3.5 and CQAS guarantees 80% users with QoE above 3.78. In our experiments, we set the weight of freezing in QoE model in Equation (3) as 0.5, which means if there is no freezing, the chunk QoE is above 2.5. An increase of session QoE from 3.5 to 3.78 is equivalent to guarantee 80% users streaming videos with doubled bit-rate. In figure 4 (c), we show the cases where QoE can be improved most. It plots the QoE improvement for QAS-client over DASH on a ratio of RTTs to two candidate servers. The size of the circle increases as the QoE improves. The x-axis and y-axis are the RTTs from the client to two candidate servers. RTTs to candidate servers are obtained by an average of 10 pings before the experiment starts. Figure 4 (c) illustrates that QoE driven server selection is more effective in improving QoE if two candidate servers have similar RTTs to the client.

[3]Because we did not have enough resources to emulate a large number of clients, we throttle the maximum bandwidth of each server to 4 $Mbps$ to emulate the server overloading that would happen in real systems.

[4]Because we cannot find out exactly where each zone is located in Google Cloud, we inferred locations of all zones based on their RTTs to all clients and available locations of Google data centers when we draw figure 4 (a). Google datacenter locations can be found at http://www.google.com/about/datacenters/inside/locations/.

B. Large-scale Experiment in Simulation

To evaluate large scale QAS-DASH and CQAS-DASH, we simulate both client agents and cache agents in a 3-level hierarchical network in Simgrid [17]. The network consists of 8 ASes. Each AS router connects 8 access networks and a cache server. We deploy 8 clients in each access network, totally 512 clients. Root cache server is co-located with the backbone router connecting 8 ASes. We assume each cache cluster is under-provisioned to accommodate concurrent streaming of all 64 users in the AS. Available capacity of cache servers is set to 50 $Mbps$. Capacity of backbone links are set to 250 $Mbps$. Users send requests following Poisson process with arrival rate randomly chosen from $\{0.1, 0.01, 0.001\}$. Each client agent find $K = 3$ closest cache servers as candidate servers. The cumulative distribution of session QoE for all clients is compared in figure 5. Results show that QAS-DASH improves user QoE over DASH at almost all percentiles. The session QoE improvement at 90th percentile for QAS-DASH is $(3.1822 - 2.9216)/2.9216 = 8.92\%$ over DASH. The CQAS-DASH can further improve $(3.5004 - 3.1822)/3.1822 = 10.13\%$ over the QAS-DASH, which equals to an improvement of QoE up to 20% over DASH. It indicates that the QoE guaranteed for 90% users has been improved up to 20%. When we map the QoE values to bit-rate levels, an increase of QoE from 2.92 to 3.5 is equivalent to a raise of bit-rate from $280kbps$ to $660kbps$.

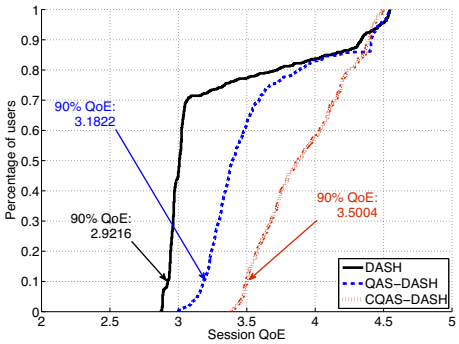
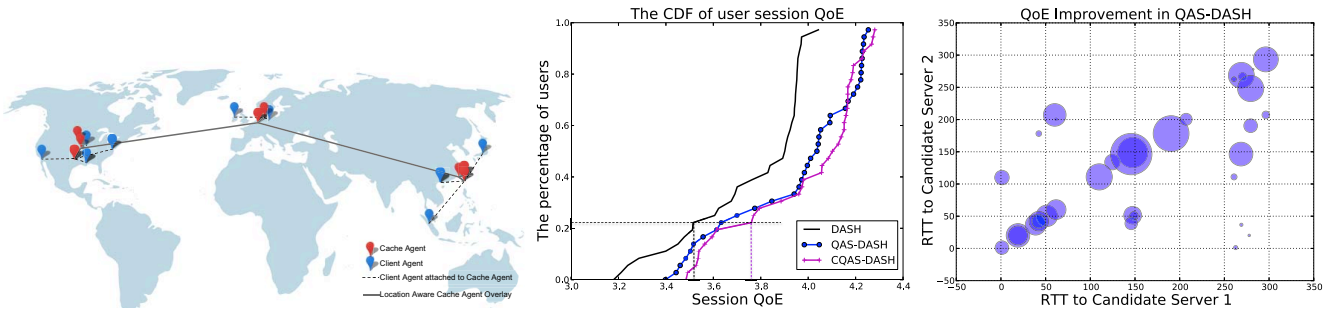


Figure 5: CDF of Session QoE in Simgrid Experiment



(a) Experiment Setup

(b) CDF of Session QoEs

(c) Scatter Plot of QoE Improvement for QAS-DASH over DASH

Figure 4: Multiple Session Experiment

VII. Conclusion

In this paper, we proposed two server selection schemes that dynamically select servers according to users' QoE feedback for VoD system in the cloud. In contrast to common practices of server selection that select servers according to network delay and server response time, we use users' real-time QoE as an evaluation of server performance. In order to validate our server selection schemes in production Cloud environment, we integrated our proposed schemes with DASH clients and evaluate the system both in Google Cloud and in simulation. Experiments in Google Cloud tested 9 clients at various locations. Results showed that by using our server selection schemes, the average bit-rate of 80% streaming sessions could be doubled. In simulation, we tested our server selection schemes in a larger scale with 512 clients. Results showed that the QoE guaranteed for 90% of users could be increased by 20%. In our future work, we aim to test the system in a large-scale setting with thousands of clients and hundreds of servers. We believe our QoE driven control can also benefit other aspects of system management, including cache management and resource provisioning.

References

[1] sandvine, "Global internet phenomena report, 2h 2014." https://www.sandvine.com/trends/global-internet-phenomena/, 2014.

[2] R. Torres, A. Finamore, J. R. Kim, M. Mellia, M. M. Munafo, and S. Rao, "Dissecting video server selection strategies in the youtube cdn," in *ICDCS*. IEEE, 2011, pp. 248–257.

[3] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan, "Optimal content placement for a large-scale vod system," in *CONEXT*. ACM, 2010, p. 4.

[4] 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 *INFOCOM*. IEEE, 2012, pp. 1620–1628.

[5] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang, "Developing a predictive model of quality of experience for internet video," in *SIGCOMM*. ACM, 2013, pp. 339–350.

[6] M. Ghareeb, C. Viho, and A. Ksentini, "An adaptive mechanism for multipath video streaming over video distribution network (vdn)," in *MMEDIA*. IEEE, 2009, pp. 6–11.

[7] T. Stockhammer, "Dynamic adaptive streaming over http–: standards and design principles," in *MMSys*. ACM, 2011, pp. 133–144.

[8] A. Shaikh, R. Tewari, and M. Agrawal, "On the effectiveness of dns-based server selection," in *INFOCOM*. IEEE, 2001.

[9] J. Pan, Y. T. Hou, and B. Li, "An overview of dns-based server selections in content distribution networks," *Computer Networks*, vol. 43, no. 6, pp. 695–711, 2003.

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

[11] W. Zhang, Y. Wen, Z. Chen, and A. Khisti, "Qoe-driven cache management for http adaptive bit rate streaming over wireless networks," *Multimedia, IEEE Transactions on*, vol. 15, no. 6, pp. 1431–1445, 2013.

[12] S. Van Kester, T. Xiao, R. Kooij, K. Brunnström, and O. Ahmed, "Estimating the impact of single and multiple freezes on video quality," in *IS&T/SPIE Electronic Imaging*, 2011.

[13] P. ITU-T RECOMMENDATION, "Subjective video quality assessment methods for multimedia applications," 1999.

[14] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http," in *MMSys*. ACM, 2011, pp. 157–168.

[15] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *Evolutionary Computation, IEEE Transactions on*, vol. 1, no. 1, pp. 53–66, 1997.

[16] B. Hubert, "The wonder shaper," 2002.

[17] H. Casanova, A. Legrand, and M. Quinson, "Simgrid: A generic framework for large-scale distributed experiments," in *UKSIM*. IEEE, 2008, pp. 126–131.