Users Know Better: A QoE based Adaptive Control System 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*—As VoD systems migrate to the Cloud, new challenges emerge in managing user Quality-of-Experience (QoE). The complexity of the cloud system due to virtualization and resource sharing complicates the QoE management. Operational failures in the Cloud could be challenging for QoE as well. We believe that end users have the best perception of system performance in terms of their QoE. We propose a QoE based adaptive control system for VoD in the Cloud. The system learns server performance from the user QoE and then adaptively selects servers for users accordingly. We deploy our proposed system in Google Cloud and evaluate it with hundreds of clients deployed all over the world. Results show that given the same amount of resources, our system provides 9% to 30% more users with QoE above the Mean Opinion Score (MOS) "good" level than the existing measurement based server selection systems. The system guarantees a better QoE (above 6% better) for 90% users. Additionally, our system discovers operational failures by monitoring QoE and prevents streaming session crashes. A computational overhead analysis shows that our system can easily scale to large VoD systems containing thousands of servers[1].**

Keywords*—VoD, QoE, Cloud, Adaptive Control, Server Selection

I. Introduction

The Cloud infrastructure has become an ideal platform for large-scale applications with periods of flash crowds, such as Video-on-Demand (VoD). The Cloud can provide elastic amount of resources to meet the dynamic user demand [1]. As VoD systems migrate to the Cloud, new challenges emerge for VoD providers to manage user Quality-of-Experience (QoE) [2]. Extra complexity due to virtualization and resource sharing adds to the challenge.

In large-scale VoD systems, performance issues in various components of the system, such as the server overload or the network congestion, degrade the user experience. Commercial systems (e.g. YouTube [3], Netflix and Hulu) use Content Delivery Network (CDN) to improve the user experience. CDN places video servers close to users in order to reduce their server response time. CDN replicates the same content across multiple servers to provide fault-tolerance. When one server is unavailable, other replicas can serve the user request. In order to improve the user experience, CDN providers (e.g. Akamai [4]) perform extensive network and server measurements to select an appropriate replica server and network path.

When VoD systems are deployed in the Cloud, it is more challenging for VoD providers to offer consistently good experience for users. The Cloud itself is complex. The Infrastructure as a Service (IaaS) in the Cloud is usually offered by a virtualized datacenter consisting of a cluster of physical machines (PMs). Each PM hosts multiple virtual machines (VMs) possibly belonging to different customers. The video server would be deployed in one of these VMs. Multiple VMs on the same PM are sharing physical resources such as CPU, disk, memory and network interface. Resource sharing may lead to performance interference from other VMs. The applications on the other customers VMs are not visible to the VoD provider. A VoD provider cannot easily predict its video server performance and usually would not have control over other VMs. Even if there are no other VMs in the physical host, the performance of a particular physical host in the data center could be unpredictable as the user has limited or no access to the physical host. Existing works study the impact of interference by benchmarking the Cloud using CPU/disk/memory/network intensive tasks. However, the impact of interference can only be captured but cannot be easily quantified [5]. The user QoE cannot be simply modeled by resource measurements in the data center. Thus, we believe that the best way to understand the performance impact is to observe the user QoE directly. Instead of modeling complex systems in the Cloud, we believe that the user QoE would give the best indication of the system performance.

Provisioning cloud resources on the fly may cause failures that are difficult to identify. A user request might be sent to a server that has been removed. A new server might boot with an outdated cache table and directs the user request to a wrong server. Operational failures happen often. Servers may hang due to sudden high user demands during peak hours. A system administrator could misconfigure content folders. Simple monitoring schemes like the network probing and the server load monitoring cannot identify these failures [6]. When a failure happens, the VoD provider may not know the cause of the failure but users can definitely perceive early symptoms. For example, before a streaming session crashes, the video player on the user side may observe several video Chunk request timeouts. The user would observe that the buffer is depleting, or the video simply freezes for a while. One can take advantage of these early symptoms to prevent streaming crashes by adaptively selecting a backup server for the user.

We propose a QoE based adaptive control system that can 1) Monitor individual user QoE; 2) Infer server performance directly from users' QoE; 3) Adaptively select servers according to the user QoE; and 4) Effectively respond to various failures using user QoE.

To meet demands from millions of users, our system should operate in a distributed fashion to scale. We adopt an agent based system design that deploys agents in existing video

[1]This work was supported in part by CMU-SYSU CIRC, SYSU-CMU IJRI and ICTI under Grant SFRH/BD/51150/2010.

servers and client players for distributed control. Our system runs in VoDs serving DASH streaming (Dynamic adaptive video streaming over HTTP) [7]. We deploy our system in Google Cloud [8] as our underlying cloud platform and emulate hundreds of user clients in PlanetLab [9]. We evaluate our system under various levels of interference and failures. Results show that our QoE based control system outperforms existing measurement based server selection systems in three aspects: 1) Provides more users with QoE above a predefined acceptable level; 2) Improves the worst QoE users; and 3) Prevents streaming session crashes in a timely manner. Lastly, we describe an overhead analysis that compares the scalability of our system with the existing systems.

II. Related Work

Quality-of-experience (QoE) is a subjective perception of user's acceptability of an application or a service [10]. There are several works modeling QoE by quantitative quality-of-service (QoS) metrics. Some works conduct subjective experiments for the video streaming under various network impairments. They apply machine learning methods or statistical analysis [11] to model the QoE. Other works develop an analytical model for QoE over a measurable quality metric, such as freezing time [12] or bitrate [13]. They then use subjective experiments to validate their assumptions.

With the recent advances in QoE modeling, existing works apply user QoE in the control and management of video streaming system. In [14], they study the QoE of the quality transitions and propose a QoE-aware rate-adaptation system for DASH streaming. In [15], the logarithm law is used to model the user QoE over bit-rate. It proposes an optimized caching algorithm to maximize users' QoE in wireless network. A varying QoE served from different servers in CDN is studied in [16]. It designs a client-side QoE based server selection algorithm for each client. This method assumes that a special "iBox" device can be deployed in the client's residential network and the device is pre-configured with addresses of servers in CDN. These assumptions limit its applicability in production systems as the installations of "iBox" in large-scale would be difficult.

The server selection is a topic extensively studied for the purpose of improving user QoE in video streaming system. Early studies of server selection utilize DNS to select a server located close to users [17]. In CDN based VoD system, DNS based server selection is used as a proximity aware server selection scheme [18]. To balance load among servers, various redirection schemes are combined with DNS based server selection at a finer level [19]. Such schemes could be effective to improve user QoE in CDN environment. They select a server for users according to both the network proximity and the server load. Measurement studies in YouTube also reveal that network latency and server load are major factors in their server selection scheme [3]. However, there still remain many other factors impacting user QoE and these factors can neither be considered completely nor be modeled accurately. Other works about Quality of Service (QoS) aware routing focus on QoS aware server selection in a pre-configured overlay network [20], however, their solutions can hardly be adapted to the Cloud environment where dynamic resource provisioning can change the topology of overlay networks.

III. System Overview

Our system consists of *cache agents* and *client agents*. They together form the following functions: 1) individual user QoE monitoring, 2) server performance inference, 3) the server selection and 4) failover control. *Client agent* is a management process running in the video player to monitor the client's QoE in real time. *Cache agent* runs in each video server, collects clients' QoE and infers server performance. Client agents work with cache agents to perform QoE based controls for the adaptive server selection and the failover response. Each client agent communicates with a cache agent located closest in network latency. In production VoD systems, we can use the cloud DNS to select the closest cache agent for each client agent.

We explain how our system operates step by step in Figure 1. We have a VoD client A requesting a video. A's client agent sends a query message with the requested video to its closest cache agent S_1 (Step ①). Upon receiving the query from A, S_1 looks up its cache table and finds several candidate servers with the requested video[2]. Each cache agent maintains a table of server QoE scores. The server QoE score represents how well a server performs. It is inferred by the QoE of clients served by the server. S_1 then looks up server QoE scores for candidate servers, selects S_2 that has the best score and responds to A's query (Step ②). Client A downloads the initial segment of the video from S_2 (Step ③). After receiving the initial segment, the video player on A starts playing the video. The client agent gets the QoS metrics, such as the freezing time and the bit-rate of the segment, to compute its QoE with server S_2 (Step ④). If A freezes frequently waiting for the segment, A gets a bad QoE for the initial segment. Client agent A then reports the bad QoE to its cache agent S_1 (Step ⑤). S_1 receives A's QoE and infers S_2 has unacceptable performance. S_1 then updates the server QoE score for S_2 in its QoE score table (Step ⑥). Cache agent S_1 looks up its latest QoE score table and selects S_1 with the best QoE score for client A (Step ⑦). Client A then switches to S_1 and downloads the next segment from S_1 (Step ⑧). Our system iteratively runs from ④ to ⑧ for all segments of the video in the streaming session. The segment is sufficiently sized to avoid unstable behavior.

Cache agents perform distributed control on the server side. The client agent only undertakes the monitoring task and sends QoE to the cache agent that infers system performance. When there are multiple clients reporting their QoE for different servers periodically, the cache agent learns multiple servers' performance in near real time. As there is one cache agent per video server, the QoE updates can be distributed to many cache agents to avoid traffic overload.

IV. QoE-based Adaptive Control System

A. Chunk QoE Model

For DASH streaming[3], existing works obtain QoE from various QoS metrics. These metrics including the streaming bit-rate [15], the freezing time and the join time [11], etc. Our system requires a real-time QoE model that can estimate user

[2]Distributed content discovery algorithms [21], like DHT, Chord, flooding algorithm, etc., can dynamically discover servers caching a video. In our work, we ran a MCCD algorithm in [22] on our cache agents to obtain the addresses of multiple servers caching the requested videos.

[3]Commercial DASH players include Microsoft Smooth Streaming and Adobe OSMF. Details of their bit-rate adaptation logic can refer to [23].

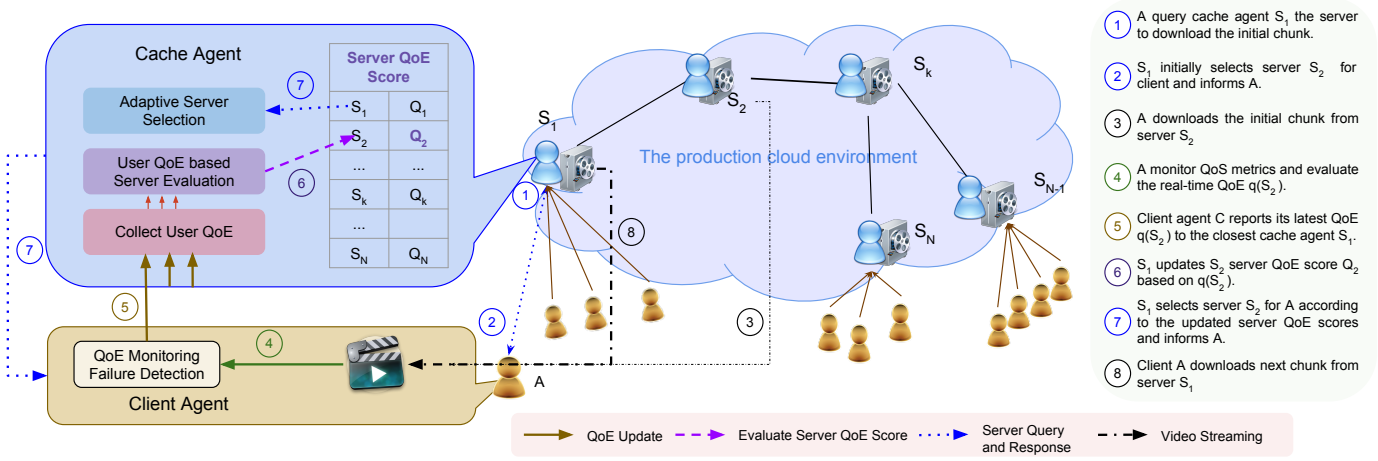


Fig. 1: System Overview

experience, so we ignore the join time because it does not change once the streaming starts. DASH encodes a video in multiple bit-rates and split each bit-rate version into a series of fixed length segments, called Chunks. During streaming, the DASH player detects the network throughput in real time and adaptively selects the bitrate for every Chunk. The finest granularity we can compute QoE is for a Chunk because the bitrate and the freezing time are measurable at Chunk-level. A psychology study validates that user perception follows a logarithm model [13] of bit-rate[4] as shown in Equation (1). A human vision study finds that user experience follows a logistic model of freezing time [12] as shown in Equation (2).

$$q_{\text{bit-rate}}(r) = a_1 \ln \frac{a_2 r}{r_{\max}} \tag{1}$$

r in Equation (1) is the bit-rate of a Chunk. $r_{\max}$ is the maximum bit-rate available in DASH. a_1 and a_2 are empirical parameters learned from subjective experiments.

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

τ in Equation (2) is the freezing time caused by downloading one Chunk. c_1 to c_3 are parameters learned by subjective studies. Both models follow Mean Opinion Score (MOS) standard [24] to value QoE on a scale of 1 to 5 that represents "bad", "poor", "fair", "good" and "excellent" levels respectively. We combine these two models to measure user QoE per Chunk as shown in Equation (3).

$$q(\tau, r) = \delta \cdot q_{\text{freezing}}(\tau) + (1-\delta) \cdot q_{\text{bit-rate}}(r) \tag{3}$$

Ideally in DASH streaming, the bit-rate is lowered as much as possible to avoid freezing. There would be no freezing until the bit-rate drops to the lowest level. δ here denotes the relative user experience (compared with the best possible experience) in the lowest bit-rate level without freezing[5]. We use the combination of existing analytical models as the Chunk QoE. Other Chunk based QoE models are applicable to our system if needed.

[4]The logarithm QoE model of bit-rate has been adapted to DASH streaming in [15] as shown in equation (1).

[5]In our system, we set $\delta \cdot 5 = 1$, which corresponds to "bad" experience in MOS score.

B. QoE Score for Server Evaluation

Clients connecting to the same cache agent are likely to be close to each other and have similar QoE. The cache agent can use QoE collected from different clients to learn performances of reported servers. Clients connecting to the same cache agent are denoted to be in a *client group*. QoE from different clients are used to infer server performance. The cache agent effectively distributes QoE information to clients in the client group. Each client learns from other clients' QoE when selecting the appropriate server. Consistency and variability of QoE from the client group determines the effectiveness of sharing QoE from various clients.

We use the reinforcement learning to infer the server performance using collected QoE from clients. In reinforcement learning [25], the rewards in earlier actions are used to evaluate and to determine the next action. The goal is to maximize the total reward gain of all actions. Collected QoE of a particular server in a cache agent is a time-series data. The client QoE represents the reward. The action is the server selection and the total reward gain is the total QoE for all clients in the client group. We define *Server QoE Score (SQS)* to infer server performance based on client QoEs. The SQS is non-stationary due to the changing server load, the varying network condition, and the dynamic background interference. We use the *exponential weighted moving average* to compute the *Server QoE Score* as shown in Equation (4).

$$Q^t(s) = (1-\alpha) \cdot Q^{t-1}(s) + \alpha \cdot q^t(s) \tag{4}$$

$Q^t(s)$ is the server s's SQS at time t. $q^t(s)$ is QoE received from clients at time t. α is the weight of the latest QoE reward of selecting the server. Each cache agent maintains $Q(s)$ for reported servers over time. The SQS is initialized as 5 ("Excellent") for the server with the cache agent and as 4 ("Good") for all other servers.

C. QoE based Adaptive Server Selection

Our QoE based adaptive server selection algorithm runs in client and cache agents as follows. The client monitors its QoE via the average of latest N chunk QoE. It then reports its QoE on its streaming server to the closest cache agent periodically (every N chunks). When the cache agent gets the client's QoE, it updates the SQS of the streaming server as (4)

shows, greedily chooses a server according to (5) and sends the server address to the client agent. The client agent then downloads following N chunks from the new server.

$$s^*(t) = \arg\max_{s \in S} Q^t(s) \quad (5)$$

We apply the greedy method for simplicity. As videos are distributed among different servers, the greedy method will seldom direct many clients to a single server. Experimental results validate our initial assumption of the system behavior.

D. QoE based Failover Control

Failures can happen in various components of the VoD system. Some failures are hard to detect and identify. Administrators can accidentally delete videos. Video server can hang due to software errors. A VM terminates due to PM failures. From the user's perspective, all these failures end up with a Chunk request timeout or a HTTP request error. We designate these errors perceived by the user as *unacceptable* QoE ($q(s) = 0$). When the cache agent receives $q(s) = 0$, it sets α to 1 thus SQS becomes 0. The cache agent then selects another server. The client then resends the Chunk request to the newly selected server to prevent the streaming session crash. When the cache agent itself fails, then we rely on DNS to recover a new cache agent as done in existing systems.

V. System Performance

A. Experimental Setup

We implement and deploy the cache agent and the client agent to evaluate our system. We deploy client agents in 284 PlanetLab nodes to emulate VoD clients. "f1-micro" instances are provisioned in Google Cloud to serve as video servers. The number of servers in each region is provisioned according to the number of users in that region as shown in Table I. The locations of clients and servers are shown in Figure 2. Google discloses all their datacenter locations as red squares in Figure 2. They do not reveal which datacenter is used for the cloud service. [26] infers that multiple zones of one region in Google Cloud are datacenters located at green crosses in Figure 2. To emulate a large number of videos, we rename the same video clip (ten minute length video) as 1000 distinct videos. We assume the popularity of these videos follow Zipf distribution [27]. We use popularity based caching method to cache videos. More popular videos are cached in more servers. Each video is cached in at least 3 servers as shown in Table I.

We implement the following server selection schemes to compare our system.

- *HOP*: Each request is redirected to the server with the minimum hop number among servers with the requested video.
- *LOAD*: Each request is redirected to the server with the minimum load among servers with the requested video.
- *RTT*: Each request is redirected to the server with the the minimum RTT among servers with the requested video.
- *RANDOM*: Each request is redirected to a randomly selected server from servers with the requested video.
- *QoE*: Our system.

All existing systems, the client selects server only once at the beginning of a video streaming session. These systems represent CDN [19]. In *RTT* and *LOAD*, each server periodically probes all other servers every 5 minutes. Smaller probing period would incur large amount of probing traffic as the number of servers increases. After redirection, the client remains with the selected server for the whole video session.

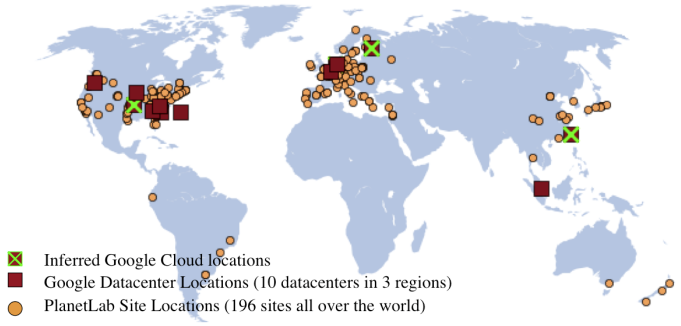


Fig. 2: The locations of clients and servers in our system

B. System Performance under VM Interferences

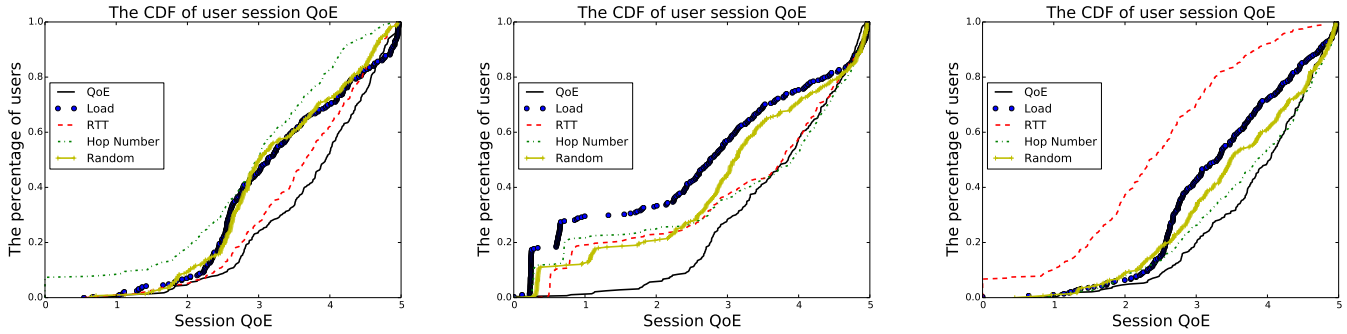
We set up three scenarios to show the system performance under various levels of interference. We first evaluate our system in a real production cloud environment, the Google Cloud without any additional stress on resources. Performance of VMs in Google Cloud varies depending on the time and physical machines Google allocates for our VMs. Our system would experience VM interference due to the nature of Google Cloud environment. The second scenario evaluates the system under severe interferences. As we do not control physical machines in Google Cloud, we emulate the severe interference by throttling the outbound bandwidth to 4Mbps from 1Gbps in two randomly selected servers. The third scenario evaluates our system under highly dynamic interferences. We emulate it by periodically throttling the outbound bandwidth to 4Mbps from 1Gbps every 1 minute in two randomly selected servers. We deploy 284 PlanetLab client nodes that request streaming videos at the same time. Each client randomly requests a 10-minute video according to its popularity.

First scenario, Google Cloud: Figure 3 (a) shows the cumulative distribution of all users' session QoE. The session QoE is the average QoE of all Chunks in a single video streaming session. The results show that our *QoE* method gets the best session QoE for most users. We have over 76% users with above QoE value 3 (3 in MOS corresponds to the user satisfaction level "fair"). The *RTT* has 73% and the *HOP* has only 49%. We have over 47% of users with above QoE level "good" (QoE value 4). The *RTT* has 38% and the *HOP* has only 17%. Our system has 9% and 30% more users with session QoE above "good" level than the *RTT* and the *HOP* respectively. It shows that, given the same amount of resources, "good" level QoE can be obtained for more users in our system. Our system has the 90th percentile QoE as 2.5708. The 90th percentile QoE are 2.2393, 2.4187, 1.4445, 2.0423 for *LOAD*, *RTT*, *HOP*, and *RANDOM* respectively. Our system performs 6.29% better than the RTT. The poor performance of *LOAD* shows that the QoE degradation is not caused by the server overload but other factors .The *HOP* method is designed to select the closest server for users in terms of the network distance. However, it has the worst performance in Figure 3 (a). We tested the same experiment in Google Cloud multiple times at different hours. The performance of the *HOP* varies a lot. We suspect that it is due to dynamic interference in Google Cloud or varying network conditions. The results show that our system always selects servers that serve better QoE without having to identify causes of performance degradation.

TABLE I: Resource Provisioning & Content Caching in Experimental VoD System

Regions	asia-east1			europe-west1					us-central1				
Zones	a	b	c	b		c		d	a	b		c	d
# of clients[a]	42			123					132				
Server	S_1		S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}
# of servers	2			5					5				
# of servers	1	0	1	2		2		1	1	2		1	1
# of videos	307		346	326	332	331	330	331	344	324	313	333	337

[a]The client belongs to a region if its average RTT to all data centers in the region is smaller than other regions.



(a) Scenario 1: Google Cloud (b) Scenario 2: Severe Interference (c) Scenario 3: Dynamic Interference

Fig. 3: The cumulative distribution of session QoEs for all users

Second scenario, Severe Interference: Figure 3 (b) shows that the *QoE* method has significant advantages over other methods for those clients who are strongly impacted by the interference. We observe that around 10% to 20% clients using methods other than the *QoE* get session QoE below 1. These affected clients selected servers with severe interference and they suffered more freezing events due to the background traffic interference. As QoE is neither monitored nor reported from these clients, the system has no knowledge about their performance. In contrast, the *QoE* method learns server performance from user QoE and adaptively selects well-performing servers.

Third scenario, Dynamic Interference: Figure 3 (c) shows the *QoE* method has an absolute advantage over the *RTT* in providing users with better QoE. The *RTT* performs the worst. The *RTT* method probes servers every 5 minutes but the interference appears every other minute, so it misses the interferences. Periodic probing fails to capture the dynamic changes of background interference.

We also test our system under various types of interference in CPU, I/O, and memory. We emulate these interferences by stressing corresponding resources on two randomly selected servers. Our system outperforms other methods similarly. With regard to different types of interference, there is a slight difference on how much the interference impacts user QoE. I/O and bandwidth interference seem to have higher QoE impact than CPU and memory.

Extensive experimental results show that our system can manage QoE better. There are more users obtaining QoE above a pre-defined level and better QoE guarantees.

C. System Performance under Failures

We study how our system reacts to two types of failures. The first one emulates an unresponsive video server on a working VM due to various software errors and bursty user demand. This failure leads a server to hang or crash. The second one emulates an unresponsive VM. This is caused by PM to hang or crash due to various failures in other coexisting VMs. We initiate 284 clients requesting streaming videos at the same time for 1 hour. Each client randomly chooses one method from *HOP*, *LOAD*, *RTT*, *RANDOM* and *QoE*.

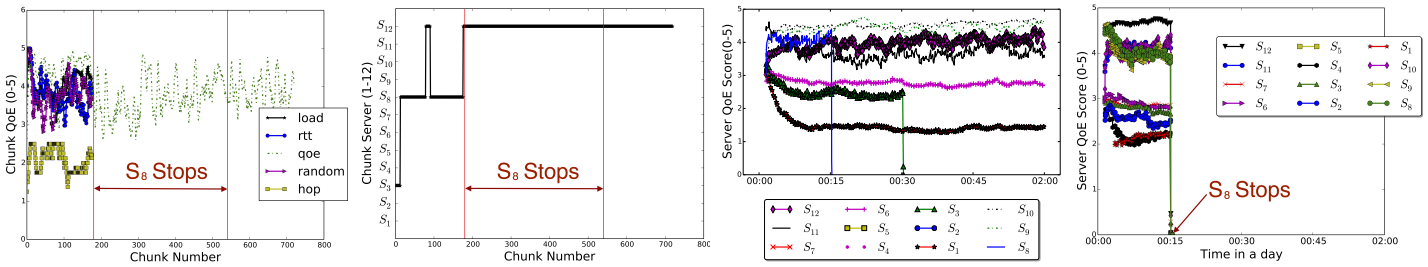
Clients using methods other than the *QoE* remain with selected servers for the entire streaming session. They would crash in both failure scenarios in the middle of streaming session. Clients with QoE monitor QoE in real time and report their real-time QoE to their respective cache agents. Our system can operate despite server/VM failures. The cache agent always selects a new server for clients when the client reports an unacceptable QoE.

Figure 4 shows how our system reacts to the unresponsive server failure on a working VM (first type of failure). We emulate this failure by stoping the Apache server on S_8 for 30 minutes in the middle of the streaming. Figure 4 (a) shows several clients' QoE curvesover time. Clients with *HOP*, *LOAD*, *RTT*, and *RANDOM* all crashed after the server was stopped. We find the available buffer of the client with the *QoE* dropped from 22.35 seconds to 11.55 seconds right after the Apache server stopped. While playing out the video from the buffer, the client with the *QoE* method overcame the failure by switching to S_{12} as shown in Figure 4 (b). The cache agent of the client is S_{10}, so we show the server QoE scores evaluated on S_{10} in Figure 4 (c). It shows that at time 0:15, the QoE score of S_8 dropped to 0 indicating the server failure. Before the S_8 failure, S_8's SQS was higher than S_{12}'s SQS. Thus S_8 was selected. After the S_8 failure, S_{12}'s SQS became the highest thus selected for the client and overcame the failure. Figure 4 (d) shows the S_8's SQS on all cache agents. All cache agents discovered S_8 failure right after 0:15.

For the second type of failure (the unresponsive VM failure), the observed SQS is similar to the first scenario. The *RTT* and *LOAD* discovered the VM failure within 5 minutes, but this did not prevent crashes of on-going streaming sessions.

D. Overhead Analysis

There are $N = 284$ clients connecting to $M = 12$ cache agents to report their Chunk QoE periodically. On average,



(a) QoE of Clients impacted by S_8

(b) Server selected for all Chunks in the client using QoE method

(c) SQS evaluated on cache agent S_{10}.S_{10} is the cache agent for the client using *QoE* method shown in (a)(b).

(d) S_8's SQS evaluated on all cache agents over time.

Fig. 4: System performance under the failure of unresponsive server

each cache agent receives $\frac{2N}{M}$ messages in every period[6]. Assume that we fix the value $\frac{N}{M} = c$ as a constant meaning that the resource is provisioned according to the user demand. The average number of QoE update messages sent to each cache agent is constant determined by c, the average number of clients served by one server. *LOAD* and *RTT* probe every 5 minutes. Their total traffic is $\frac{M^2}{5}$ per minute. It increases quadratically with the number of servers. *LOAD* and *RTT* cannot adapt to a large-scale system with thousands of servers.

VI. Conclusion

We propose a QoE based adaptive control system for VoD in the Cloud. Extensive experimental results on our system show that our system can manage users' QoE better than existing measurement based server selection systems. Our system has more users obtaining QoE above a predefined level, provides better QoE guarantees and operates despite server/VM failures. These results validate our belief that QoE gives the best perception of system performances, well over widely used measurements, such as RTT and server load. Our future work can proceed in using QoE in other aspects of system control, such as content caching and resource provisioning.

References

[1] Y. Wu, C. Wu, B. Li, X. Qiu, and F. C. Lau, "CloudMedia: When Cloud on Demand meets Video on Demand," in *ICDCS*. IEEE, 2011, pp. 268–277.

[2] T. Hobfeld, R. Schatz, M. Varela, and C. Timmerer, "Challenges of QoE management for cloud applications," *Communications Magazine*, vol. 50, no. 4, pp. 28–36, 2012.

[3] 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.

[4] A.-J. Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante, "Drafting behind akamai (travelocity-based detouring)," in *SIGCOMM*, vol. 36, no. 4. ACM, 2006, pp. 435–446.

[5] S. K. Barker and P. Shenoy, "Empirical evaluation of latency-sensitive application performance in the cloud," in *MMSys*. ACM, 2010, pp. 35–46.

[6] S. Pertet and P. Narasimhan, "Causes of failure in web applications (cmu-pdl-05-109)," *Parallel Data Laboratory*, p. 48, 2005.

[6]In our system, the video is split into 5-second Chunks and clients report the average QoE of every 6 Chunks to reduce the amount of QoE reporting traffic. Considering a large-scale VoD system with 1 million clients and 1000 video servers, there would be $2 \times 10\text{B} \times 1\ \text{million}/(1000 \times 6 \times 5\ \text{seconds}) = 40\text{KB/minute}$ QoE traffic sent to one server if we count 10B for one message (including an IP address and a float type QoE value). We believe the amount of QoE traffic has minimal impact on both servers and networks compared to the volume of traffic generated by video streaming.

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

[8] (2015, March) Google cloud platform. [Online]. Available: https://cloud.google.com/

[9] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *SIGCOMM*, vol. 33, no. 3, pp. 3–12, 2003.

[10] I. SG12, "Definition of Quality of Experience," *TD 109rev2 (PLEN/12), Geneva, Switzerland*, pp. 16–25, 2007.

[11] M.-N. Garcia, F. De Simone, S. Tavakoli, N. Staelens, S. Egger, K. Brunnstrom, and A. Raake, "Quality of experience and HTTP adaptive streaming: A review of subjective studies," in *QoMEX*, Sept 2014, pp. 141–146.

[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. Reichl, B. Tuffin, and R. Schatz, "Logarithmic laws in service quality perception: where microeconomics meets psychophysics and quality of experience," *Telecommunication Systems*, vol. 52, no. 2, pp. 587–600, 2013.

[14] R. K. Mok, X. Luo, E. W. Chan, and R. K. Chang, "QDASH: a QoE-aware DASH system," in *MMSys*. ACM, 2012, pp. 11–22.

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

[16] H. A. Tran, S. Hoceini, A. Mellouk, J. Perez, and S. Zeadally, "QoE-based server selection for Content Distribution Networks," *IEEE Transactions on Computers*, no. 11, pp. 2803–2815, 2014.

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

[18] 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.

[19] S. Bakiras, "Approximate server selection algorithms in content distribution networks," in *ICC*, vol. 3. IEEE, 2005, pp. 1490–1494.

[20] Z. Li and P. Mohapatra, "QRON: QoS-aware routing in overlay networks," *Selected Areas in Communications, IEEE Journal on*, vol. 22, no. 1, pp. 29–40, 2004.

[21] J. Gao and P. Steenkiste, "Design and evaluation of a distributed scalable content discovery system," *JSAC*, vol. 22, no. 1, pp. 54–66, 2004.

[22] C. Wang, H. Kim, and R. Morla, "QoE driven server selection for VoD in the Cloud," in *IEEE Proceedings of 8th International Conference on Cloud Computing (CLOUD)*, June 2015, pp. 917–924.

[23] 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.

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

[25] A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 1998.

[26] D. Mytthon. (2014, March) 5 things you probably don't know about Google Cloud. [Online]. Available: https://gigaom.com/2014/03/02/5-things-you-probably-dont-know-about-google-cloud/

[27] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "Analyzing the video popularity characteristics of large-scale user generated content systems," *TON*, vol. 17, no. 5, pp. 1357–1370, 2009.