This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE Globecom 2011 proceedings.

Dynamic Adaptive Streaming over HTTP from Multiple Content Distribution Servers

Wei Pu*, Zixuan Zou†, and Chang Wen Chen*
*The State University of New York at Buffalo, NY, U.S.A. Email: {weipu, chencw}@buffalo.edu
†Huawei Technologies, Co. LTD., Shenzhen 518129, P. R. China. Email: zixuan.zou@huawei.com

***Abstract*—Dynamic adaptive streaming over HTTP (DASH[1]) is attractive. It reuses popular web servers rather than relying on expensive streaming server. It takes the benefits of router and firewall optimizations for HTTP traffic. It promises to automatically adapt to bandwidth dynamics to provide smooth playback experience with best achievable visual quality. In this paper, we study a new architecture for DASH, i.e. DASH from multiple content distribution servers with scalable coded video. We take the advantage of content distribution network (CDN) architecture to offer even better DASH experience. In our scheme, users request one video program simultaneously from more than one CDN servers over HTTP. To optimize resource allocation among different servers, we propose a collaborative multi-scale scheduling algorithm (CMSS) which can effectively: 1) mitigate web server load and automatically balance loads among servers; 2) adapt to bandwidth dynamics of each server; 3) optimize aggregated streaming quality. We implemented a prototype of our algorithm and deployed it on PlanetLab open platform. Measurement results verify the effectiveness of CMSS.**

I. Introduction

Recently, DASH has been raising a lot of attentions, including Microsoft smooth streaming, Apple HTTP live streaming, Adobe HTTP dynamic streaming, etc. These popular DASH solutions adopt stream switching, in which the server stores multiple versions of the same video program with different bit rate. The server switches among these versions according to the rate adaptation algorithm.

Stream switching does not take the benefit of advanced scalable video coding technology, i.e. H.264/SVC international standard. It takes large storage space and can only provide small number of candidate bit rate streams (usually no more than 5, [1]). With H.264/SVC's median granularity scalability (MGS), server can provide progressive streaming bit rates, which makes itself able to better adapt to bandwidth variations. Taking the advantage of scalable video coding, it has been shown in [2] that simple deadline-driven rate adaptation algorithm achieves very good performance.

In this research, we identify a new problem of DASH which, to the best of our knowledge, has not been dressed previously. Specifically, we study how to use replica servers to improve DASH quality, taking the benefits of scalable video coding. In single source parallel TCP streaming [3][4], video source locates in the single server. It is relatively easy to perform server side rate control. However, in multi-server DASH, video sources are stored in different geographically distributed CDN servers. Server side rate adaptation would require close server collaboration to decide which video segment to transmit at each time from each server, which is not preferred in practice due to large communication cost. Therefore, user driven rate adaptation and segment selection are preferred, in which users are in charge of negotiating with CDN servers to adapt video bit rate to available aggregate bandwidth.

Although exploiting path diversity to improve streaming quality has been studied for years in P2P streaming literature, however, multi-server DASH is significantly different from the well studied P2P streaming model. First, different from a regular peer in P2P streaming, CDN server is assumed to be simultaneously serving hundreds to thousands of users. Reducing server's load should be explicitly taken as one of the primal goals of DASH rate control algorithm design. For example, assume that a HD video is encoded with bit rate 10Mbps. If each HTTP request asks one 20kB data chunk in HTTP request, then for 1000 users, the server have to process up to 62,500 HTTP requests per second. Second, CDN servers are relatively stable and they do not require incentive mechanisms to deliver video to users. Third, CDN servers usually have full replica of the video program. Finally, we would like to point out that even in P2P streaming, video bit rate adaptation is sill lacking a good solution.

We study multi-server DASH with scalable coded

Supported by a Gift Funding from Huawei Technologies.

[1]In this paper, we use DASH as a general acronym, which does not represent 3GPP-DASH/MPEG-DASH standards.

978-1-4244-9268-8/11/$26.00 ©2011 IEEE

video source. The challenge of this problem is that mitigating server load (reducing number of HTTP requests) and optimizing bandwidth utilization usually conflict with each other. Coordinating servers to balance these two conflicting aspects is a big challenge. As we know, there are mismatches between estimated TCP bandwidth and practical TCP bandwidth. So user should frequently update its bandwidth estimation. When the number of HTTP requests is reduced, the user loses the precision of selecting optimal video rate according to estimated TCP throughput. Therefore reduces bandwidth utilization. However, over excessive HTTP requests would exhaust server resources. To address these challenges, we propose and implement collaborative multi-scale scheduler (CMSS). By introducing a new multi-scale rate prediction algorithm and a new multi-scale resource allocation algorithm, the scheduler can accurately track bandwidth variation and accurately allocate resources with logarithmic number of HTTP requests. CMSS is able to effectively release server load, especially when video bit rate is very high (e.g., HD video streaming).

II. Related Work

Wang et al [3] studied the benefit of exploiting multipath diversity for TCP streaming. Tullimas et al [4] studied streaming using parallel TCP connections along single path. Their aim is to use multiple TCP to mitigate TCP variations. They assume single source without rate adaptation mechanism.

Cicco et al [1] measured Akamai HD DASH over Adobe Flash. Based on their measurement results, they proposed their improved rate control algorithm using classic control theory [5]. Kuschnig et al [6] proposed an HTTP request-response based DASH system. They proposed to use fix size chunk as basic scheduling unit and to dynamically adjust inter HTTP request time intervals to guarantee fairness. However, their solution does not scale well from server's perspecitve due to the huge number of HTTP requests. Stockhammer [7] introduced the initial version of 3GPP standard for DASH and compared it with existing industrial DASH systems. Akhshabi et al [8] experimentally compared several widely used DASH system, including Microsoft's smooth streaming, Netflix player, and Adobe's dynamic streaming. Kuschnig et al [2] tested several rate adaptation algorithm for H.264/SVC steaming. Schierl et al [9] studied priority-based media delivery using H.264/SVC. Sánchez et al [10] studied web cache for DASH with H.264/SVC coding. Padmanabhan et al [11] proposed a different approach using multiple description coding (MDC), which is a non-standard video coding technology. Fortino et al [12] proposed Comodin, a CDN based media streaming system using RTP.

III. Collaborative Multi-scale Scheduling

In this paper, we assume that H.264/SVC bit stream is stored segment by segment in the servers. Each segment contains T seconds of video program. As SVC stream contains intrinsic rate adoption functionality (CGS, MGS), we can select relatively longer segment length $T \in [10s, 30s]$. A video stream is presented as segment sequence $(S_0, S_1, \cdots)$, where $S_i (i = 0, 1, \cdots)$ denotes one T-second segment. The size (in byte) of S_i is V_i. H.264/SVC nework abstraction layer units (NALUs) within one segment are rearranged in rate-distortion optimized order. I.e. important NALUs (i.e. with lower $priority\_id$) locate in front of less important NALUs in the segment. How to map NALUs to $priority\_id$ is non-standard and out of the scope of this paper. Readers please refer to [13]. Here, we assume that each segment has already been reordered.

Let N denote the number of available servers, with rate $r_i(t), 0 \le i < N$. Note that the rate variations are determined by TCP, rather than the CDN servers. Assume that $m(I)$ denotes the Lebesgue measure of real interval I. Let

$$r_i(t_1, t_2) = \frac{\int_{t_1}^{t_2^-} r_i(\tau)\, \mathrm{d}\tau}{m(\{t : t \in [t_1, t_2),\ i \text{ transmits at } t\})} \quad (1)$$

denotes the measured average TCP rate in the time interval $[t_1, t_2)$. Let $t_S(i)$ be the starting of the transmission time of segment i. To match the playback rate with the streaming rate, except for several initial segments, which might be optimized to reduce initial playback rate, we assume that all the other segments should be transmitted within T seconds. If the highest available layer is successfully delivered before using out its T-second quota, the servers will be suspended and resumed at the beginning of the next segment's scheduled transmission time.

First, we give the optimal scheduling policy. As this policy requires exactly prediction of TCP rate in the future ($r_j(t), t > t_S(i)$), it is not a causal system. I. e. it is physically infeasible. The algorithm is summarized in Algorithm 1. The return value $\mathbf{V} = (V_i^{(0)}, V_i^{(1)}, \cdots, V_i^{(N-1)})$ is a vector indicating how to allocate S_i. At time $t_S(i)$, user sends HTTP request with byte range $[0, V_i^{(0)})$ of S_i from server 0, $[V_i^{(0)}, V_i^{(0)}+V_i^{(1)})$ of S_i from server 1, and so forth. As within each segment, the optimal scheduling policy requires the user to make only one request to each server, which is independent with the length of the segment, we name it $O(1)$ scheduler in this paper for simplicity.

Algorithm 1 Ideal Multi-Server DASH

procedure SCHEDULE($\{t_S(i), r_j(t)\,(0 \le j < N, t \ge t_S(i)), T, V_i\}$)
  $V_i^R \leftarrow \sum_{j=0}^{N-1} \int_{t_S(i)}^{t_S(i)+T} r_j(\tau)\,\mathrm{d}\tau$
  if $V_i^R \ge V_i$ **then**
    $\left\{\sum_{j=0}^{N-1} \int_{t_S(i)}^{t_S(i)+T(i)} r_j(\tau)\,\mathrm{d}\tau = V_i\right\} \Rightarrow T(i)$
  else ▷ Bit stream truncation
    $T(i) \leftarrow T$
  end if
  $V_i^{(j)} \leftarrow \int_{t_S(i)}^{t_S(i)+T(i)} r_j(\tau)\,\mathrm{d}\tau$
  $\mathbf{V} \leftarrow (V_i^{(0)}, V_i^{(1)}, \cdots, V_i^{(N-1)})$
  return $t_S(i+1) \leftarrow t_S(i) + T$, $\mathbf{V}$
end procedure

Now we discuss the problem of $O(1)$ scheduler, which motivates our CMSS algorithm. First, as we cannot perfectly predict TCP flow rates $r_j(t), t > t_S(i), 0 \le j < N$, we can only use predicted rate r_j^{PRED} as the average rate for future transmission. I.e.

$$r_j(t) = r_j^{PRED},\ t > t_S(i) \tag{2}$$

The problem is that generally speaking, scalable coded video data within one segment (has already been reordered) are used by the decoder sequentially. This means the received data is useful if and only if all its precedent bytes are successfully received. Note that video decoder has certain degree of error recovery capability. In this research, we do not model this issue. If $\exists 0 \le j < N-1$, r_j^{PRED} is over estimated and $T(i) = T$, then:

$$V_i^{(j)} = r_j^{PRED} \cdot T > \int_{t_S(i)}^{t_S(i)+T} r_j(\tau)\,\mathrm{d}\tau = \overline{V_i^{(j)}} \tag{3}$$

where $\overline{V_i^{(j)}}$ is practically received volume of data from server j. Therefore, the bytes in

$$[\sum_{k=0}^{j-1} V_i^{(k)} + \overline{V_i^{(j)}}, \sum_{k=0}^{j} V_i^{(k)}) \tag{4}$$

are not received. As a results, any byte received in $[\sum_{k=0}^{j} V_i^{(k)}, V_i)$ is useless. This results in a waste of bandwidth.

Statistically speaking, prediction accuracy deteriorates with time. I.e. prediction of near future on average has higher fidelity than far future. Therefore, a candidate solution is to use small chunk size and frequently update scheduling decisions. Using small chunks as basic scheduling unit also has the potential of reducing the mismatch gap (4) as ARQ mechanism can be applied. However, this approach results to extremely large amount of HTTP requests. Number of HTTP request increases linearly with respect to segment size.

To address these shortcomings, we propose our CMSS algorithm as Algorithm 2. At the beginning of segment S_i 's transmission time, input $[v_1, v_2) = [0, V_i)$, $t_p^{(j)} = -1$. Δ is a constant. The basic ideas of CMSS algorithm are:

1) Multi-scale separation of video segments;
2) Multi-scale rate prediction.

Algorithm 2 Collaborative Multi-Scale Scheduling

procedure SCHEDULE($\{t_S(i), t_p^{(j)}, [v_1, v_2)\}$)
  if $t_p^{(j)} < 0$ **then**
    $r_j^{PRED} \leftarrow r_j(t_S(i-1), t_S(i))$
  else
    $r_j^{PRED} \leftarrow r_j(t_p^{(j)}, now())$
  end if
  $t_p^{(j)} \leftarrow now()$
  $T_l = \frac{t_S(i)+T-now()}{2}$
  $V_i^{(k)} \leftarrow \min(\max(\Delta, T_l \cdot r_j^{PRED}), v_2 - v_1)$
  $S_i^{(k)} \leftarrow [v_1, v_1 + V_i^{(k)})$
  $v_1 \leftarrow v_1 + V_i^{(k)}$
  HTTP request $S_i^{(k)}$ from server k
  return $[v_1, v_2)$, $t_p^{(j)}$
end procedure

Rather than separating a segment S_i into small fixed size chunks, CMSS separates it into a series of various size chunks. The separation process is illustrated in Fig. 1. At the beginning of a segment's

transmission quota, rather than allocating server j's volume once for all as in the $O(1)$ algorithm, CMSS allocates half of it, according to the predicted rate. In Fig. 1 (assume $j = 1$), this volume is denoted by $r_1^{PRED}(0) \cdot \frac{T}{2}$. When this volume is delivered successfully at time T_1, the scheduler allocates another half of the currently available volume, i.e. $r_1^{PRED}(T_1) \cdot \frac{T-T_1}{2}$. Note that T_1 is often not equal to $\frac{T}{2}$ due to rate prediction error. When this volume is finished at T_2, another volume $r_1^{PRED}(T_1+T_2) \cdot \frac{T-T_1-T_2}{2}$ is allocated and so forth. Such *multi-scale segment separation* algorithm is expressed at line 8-10 in Algorithm 2).

The primary principle of *multi-scale segment separation* is to avoid gap (4) by reserving enough buffer time. As long as the predicted rate is no lower than half of the practical rate, everything is going to be fine. Rate prediction is performed by *multi-scale rate prediction*. The predicted TCP rate is determined recursively by the average transmission rate of its predecessor chunk. In ideal case when TCP throughput is stable, the previous chunk requires twice of current chunk's transmission time. Therefore, *multi-scale segment separation* can be viewed as a 2-tap low pass filter with the same tap value, i.e. $(\frac{1}{2}, \frac{1}{2})$. The advantage of this approach is that at the beginning, chunk size is large. Accordingly, its transmission time is large. Therefore, rate prediction might not be able very accurate. However, at this period, segment i is far from its deadline $t_S(i + 1)$. Therefore, relative large rate prediction error is acceptable. With transmission proceeds, chunk size becomes small, rate prediction becomes precise as statistically speaking, near future is easier to predict. This process in fact also gives higher priority to the chunks transmitted near the beginning of a segment's transmission period. As we have explained, for scalable coded video, these chunks are more important than the later chunks. *Multi-scale rate prediction* algorithm is expressed at line 3 and 5 in Algorithm 2). Using CMSS, HTTP request has logarithmic complexity, compared with linear complexity of fixed size chunk solution.

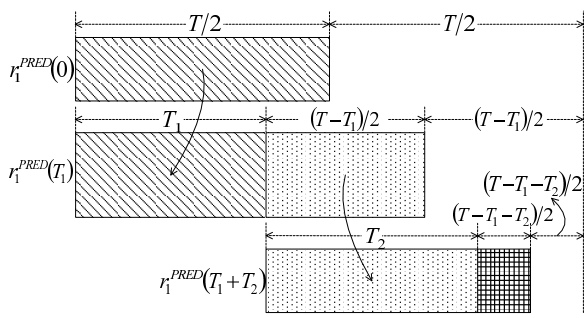

Fig. 1. Illustration of CMSS.

IV. Testbed Measurement

We develop a prototype system of CMSS using Python Twisted library. We deploy it in PlanetLab open platform. Two PlanetLab nodes are selected as CDN servers. They are: *planetlab1.eecs.wsu.edu* and *ricepl-1.cs.rice.edu*. We connect these two servers using *ubmm-deaktop9.cse.buffalo.edu*. Video source is encoded with maximum bit rate of 1.5625MBps, $T = 32$s.

A. Reference Solutions

We compare CMSS with $O(1)$ algorithm. However, as we have explained previously, $O(1)$ algorithm has very bad bandwidth utilization efficiency. To mitigate this problem and get a fair comparison result, $r_j^{PRED}, 0 \le j < N$ is calculated by the average rate of server j multiplied by a coefficient 0.9.

B. Effective Throughput

In this subsection, we study the effective throughput of different schemes. The result is presented in Fig. 2. From the result, we can see that CMSS works better than $O(1)$ algorithm. This is because CMSS can effectively reduce 'gap' in the received segments. In fact, when comparing the total received data, $O(1)$ algorithm is slightly larger than CMSS because the service time of HTTP request for size x content can be expressed as $T(x) = a + bx$ [14], where a, b are platform dependent parameters. Making more HTTP request increases overhead time a. However, as $O(1)$ algorithm cannot allocate bandwidth in an optimized way, quite a lot of received are not usable (see Fig. 4). As a result, when comparing the really usable throughput, CMSS beats $O(1)$ algorithm.

C. Server Load

Fig. 3 represents average HTTP request number, normalized by received data volume. It verifies that CMSS makes comparable HTTP requests with $O(1)$ algorithm. In most of the cases, CMSS only requires 3-5 times of $O(1)$ algorithm's requests.

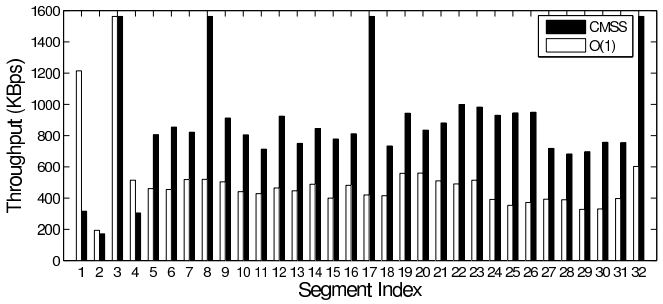

Fig. 2. Effective throughput of different schemes.

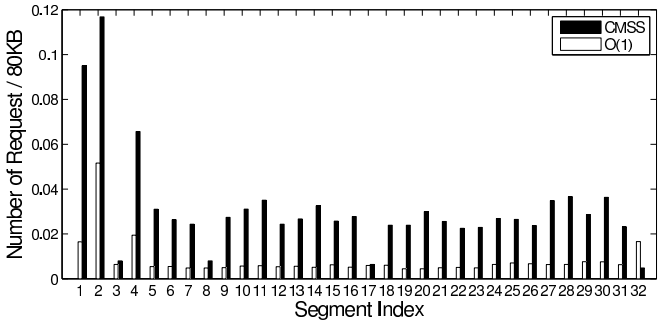

Fig. 3. Number of HTTP request per 80kB.

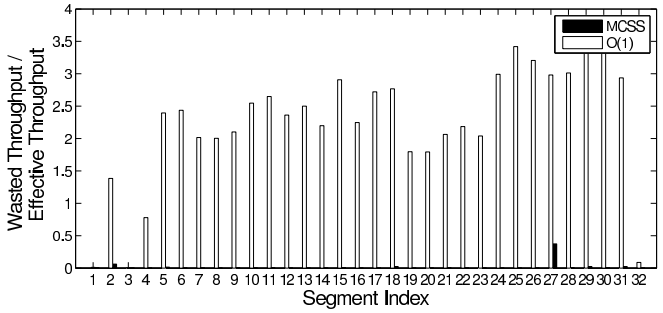

Fig. 4. Normalized Wasted Bandwidth.

D. Bandwidth Utilization Efficiency

Fig. 4 presents the normalized wasted bandwidth. The vertical axis is the useless data received by the user, normalized by effective throughput. $O(1)$ algorithm wastes lots of bandwidth transmitting useless data because its bandwidth estimation algorithm cannot be accurate and its segment separation algorithm is too aggressive.

V. Conclusion

In this paper, we studied multi-server dynamic adaptive streaming over HTTP. Different from previous research, which pay most of their attentions to network transmission or video signal processing related issues, we focus on both network transmission, video signal processing, and server load mitigation. We proposed a novel collaborative multi-scale scheduling algorithm, which requires logarithmic number of HTTP requests for each video segment. Compared with the linear complexity fixed chunk size scheme, CMSS significantly reduced server load without sacrificing band width utilization efficiency. Compared with the constant complexity prediction based algorithm, CMSS results in better bandwidth utilization with comparable server load.

References

[1] L. D. Cicco and S. Mascolo, "An experimental investigation of the Akamai adaptive video streaming," in *Proc. USAB WIMA'10*, Klagenfurt, Austria, Nov. 3–4, 2010, pp. 447–464.

[2] R. Kuschnig, I. Kofler, and H. Hellwagner, "An evaluation of TCP-based rate-control algorithms for adaptive internet streaming of H.264/SVC," in *Proc. ACM MMSys'10*, Scottsdale, AZ, USA, Feb. 22–23, 2010, pp. 157–168.

[3] B. Wang, W. Wei, and Z. Guo, "Multipath live streaming via TCP: Scheme, performance and benefits," *ACM Trans. Multimedia Comput., Commun. Appl.*, vol. 5, no. 3, pp. 25:1–25:23, Aug. 2009.

[4] S. Tullimas, T. Nguyen, and R. Edgecomb, "Multimedia streaming using multiple TCP connections," *ACM Trans. Multimedia Comput., Commun. Appl.*, vol. 4, no. 2, pp. 12:1–12:20, May 2008.

[5] L. D. Cicco, S. Mascolo, and V. Palmisano, "Feedback control for adaptive live video streaming," in *Proc. ACM MMSys'11*, San Jose, CA, USA, Feb. 23–25, 2011, pp. 145–156.

[6] R. Kuschnig, I. Kofler, and H. Hellwagner, "Evaluation of HTTP-based request-response streams for internet video streaming," in *Proc. ACM MMSys'11*, San Jose, CA, USA, Feb. 23–25, 2011, pp. 245–256.

[7] T. Stockhammer, "Dynamic adaptive streaming over HTTP: Standards and design principles," in *Proc. ACM MMSys'11*, San Jose, CA, USA, Feb. 23–25, 2011, pp. 133–144.

[8] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP," in *Proc. ACM MMSys'11*, San Jose, CA, USA, Feb. 23–25, 2011, pp. 157–168.

[9] T. Schierl, Y. S. de la Fuente, R. Globisch, C. Hellge, and T. Wiegand, "Priority-based media delivery using SVC with RTP and HTTP streaming," *J. Multimed. Tools Appl.*, Sep. 2010.

[10] Y. Sánchez, T. Schierl, C. Hellge, T. Wiegand, D. Hong, D. D. Vleeschauwer, W. V. Leekwijck, and Y. L. Louédec, "iDASH: Improved dynamic adaptive streaming over http using scalable video coding," in *Proc. ACM MMSys'11*, San Jose, CA, USA, Feb. 23–25, 2011, pp. 257–264.

[11] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributed streaming media content using cooperative networking," in *Proc. ACM NOSSDAV'02*, Miami Beach, FL, USA, May 12–14, 2002, pp. 177–186.

[12] G. Fortino, W. Russo, C. Mastroianni, C. E. Palau, and M. Esteve, "CDN-supported collaborative media streaming control," *IEEE Multimedia*, vol. 14, no. 2, pp. 60–71, Apr. 2007.

[13] I. Amonou, N. Cammas, S. Kervadec, and S. Pateux, "Optimized rate-distortion extraction with quality layers in the scalable extension of H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 9, pp. 1186–1193, Sep. 2007.

[14] T. Abdelzaher and N. Bhatti, "Web server QoS management by adaptive content delivery," in *Proc. IEEE IWQoS'99*, San Francisco, CA, USA, May 31–Jun. 4, 1999, pp. 216–225.