International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:7, No:10, 2013

Extension of the Client-Centric Approach under Small Buffer Space

Hsiang-Fu Yu, Yu-Chan Hsu, Chun Fang, Hao-Yun Yang

***Abstract*—**Periodic broadcast is a cost-effective solution for large-scale distribution of popular videos because this approach guarantees constant worst service latency, regardless of the number of video requests. An essential periodic broadcast method is the client-centric approach (CCA), which allows clients to use smaller receiving bandwidth to download broadcast data. An enhanced version, namely CCA++, was proposed to yield a shorter waiting time. This work further improves CCA++ in reducing client buffer requirements. The new scheme decreases the buffer requirements by as much as 52% when compared to CCA++. This study also provides an analytical evaluation to demonstrate the performance advantage, as compared with particular schemes.

***Keywords*—**Periodic broadcast, client-centric approach, buffer space, multimedia communications.

I. Introduction

WITH the fast boost of video-on-demand (VOD) services, VOD systems easily run out of bandwidth because the growth in bandwidth can never keep up with the increasing of the number of clients. One approach to alleviate this problem is to broadcast only popular videos. This approach is feasible because relatively few popular videos form most client requests [1], [2]. An efficient broadcasting method is periodic broadcast, which divides a video into multiple segments that are simultaneously and periodically transmitted to clients across individual data channels. The method can improve system throughput by allowing numerous clients to share the channels. Because the clients usually wait for the occurrence of the first segment before playing the video, the scheme provides only near-VOD services.

Suppose that a server allocates a broadcasting bandwidth of k playback video rates. The fast broadcasting (FB) scheme [3] divides a video of length L into $2^k - 1$ segments and yields a maximal waiting time $L/(2^k - 1)$. Assume that $k = 7$ and $L = 3600$ s. The maximal waiting time for an FB client equals 28 s. To achieve a near-minimal waiting time, the recursive frequency-splitting (RFS) scheme [4] broadcasts a segment at a frequency that can guarantee continuous video playback.

With the fast growth of wireless networks, mobile video services become more and more popular. The issue on how to broadcast videos under rather restricted client resources, such as client buffering space and receiving bandwidth, is increasingly important. The reverse fast broadcasting (RFB) scheme [5] improves FB to reduce buffer requirements by as much as 50%. The hybrid broadcasting scheme (HyB) [6] combines RFS and FB to reduce client buffer space, as well as service latency. Motivated by HyB, the study in [7] integrates the fixed-delay pagoda broadcasting scheme [8] and RFB to yield small client waiting time and buffer demand. A generalized reverse sequence-based model [9] was proposed to explain why broadcasting segments in reverse order could improve buffer requirements. The skyscraper broadcasting (SkB) scheme [10] allows a client to download video data using only a bandwidth of two channels. The client-centric approach (CCA) [11] also permits a client to download video data via a small bandwidth, and CCA+ [12] improves CCA to yield a shorter waiting time than that of SkB. CCA++ [12] further refines CCA+ by leveraging client bandwidth for more efficient video broadcast, and reduces the broadcast latency by as much as 39% when compared to CCA+ and 78% when compared to CCA.

Extending CCA++, this work proposes an enhanced version, called Reverse CCA++ (RCCA++). Similar to CCA++, the new scheme considers both broadcasting bandwidth and receiving bandwidth. However, the proposed scheme reduces client buffer requirements remarkably, when compared to CCA++. This work finds that RCCA++ requires 52% smaller buffer requirements than CCA++. This study also conducts an analytical evaluation to demonstrate the performance advantage, as compared with particular schemes.

The remainder of this study is organized as follows. Section II briefly reviews CCA++. Section III presents RCCA++. The performance of RCCA++ is evaluated in Section IV. Section V draws brief conclusions.

II. Review of CCA++

Table I defines terms frequently used in this paper. The CCA++ scheme divides a video into N segments, where $N = \sum_{i=1}^{k} n_i$ and

$$n_i = \begin{cases} 2^{i-1}, & \text{if } 1 \le i \le u, \\ n_{i-1}, & \text{if } i \bmod u = 1 \text{ and } i > u, \\ \sum_{j=i-u-1}^{i-1} n_j, & \text{otherwise} \end{cases} . \quad (1)$$

Hsiang-Fu Yu, Yu-Chan Hsu, Chun Fang, and Hao-Yun Yang are with the Department of Computer Science, National Taipei University of Education, Taiwan (e-mail: yu@tea.ntue.edu.tw).

TABLE I
LIST OF TERMS USED IN THIS PAPER AND THEIR RESPECTIVE DEFINITIONS

Term	Definition
L	Video length
k	Number of server broadcasting channels for each video
C_i	i^{th} broadcasting channel, $i = 1,...,k$
S_i	i^{th} video segment, $i = 1,...,N$
G_i	i^{th} segment group, $i = 1,...,k$
n_i	Number of the segments of group G_i
b	Video playback rate assumed to equal the data transmission rate of each channel
u	Client receiving bandwidth, expressed as u receiving channels
δ	Client maximum waiting time
T_i	Time a client starts playing the segments of group G_i, and $T_i = 1 + \sum_{j=1}^{i-1} n_j$
time unit	Basic unit on the time axis, whose length equals the length of S_1

The segments, denoted by S_1, S_2, . . . , and S_N, are then classified into k groups in sequence. A group G_i includes n_i segments. Without loss of generality, we set the length of S_1 (that is, n_1) to be 1, and the video duration L is a value relative to n_1. On the server side, the CCA++ scheme broadcasts the segments of group G_i on channel C_i periodically and simultaneously. At receiving ends, clients use u loaders to receive video segments systematically by group. Each loader downloads at most one segment from a channel in every time unit. Initially, the loaders receive the segments of groups G_1 to G_u from channels C_1 to C_u. When a loader finishes receiving the segments of the x^{th} group, the loader determines to download the segments of which group by checking the value of $x \bmod u$. If $x \bmod u = 0$, the loader downloads the segments of the $x+1^{\text{th}}$ group. Otherwise, the loader accesses the segments of the $x+u+1^{\text{th}}$ group. The loaders repeat the process until all of the segments are received.

III. PROPOSED EXTENSION OF CCA++

Because the number of segments buffered before played by a client determines the client's buffer size, the key to minimizing the size is to ensure the number as small as possible, under the client continuous playback. In CCA++, the maximum buffer size comes from the buffering of the segments of the last group.

The proposed RCCA++ scheme adopts the same strategy of CCA++ to ensure that a client can continue to download the next group after accessing the current group. Our research thus focuses on how to make the number of the buffered-before-played segments of the last group as small as possible. In CCA++, a loader is required to continuously receive the segments of group G_i during time $T_{i-1}+1$ to T_i, and the loader buffers $n_i - 1$ segments at time T_i [12].

A. Server Side

Motivated by the observation, this work devises the reverse CCA++ scheme, which broadcasts video segments according to the following steps.

1. Equally divide a video into N segments, where $N = \sum_{i=1}^{k} n_i$ and n_i is based on (1). The segments are then classified into k groups, where group G_i includes n_i segments.
2. Periodically broadcast the segments of group G_i on channel C_i according to the following conditions.

For $u = 2$:

a. If $k \bmod u = 0$:
- Transmit the segments of group G_i on channel C_i in sequence, where $1 \le i \le k-3$.
- Reversely transmit the segments of group G_{k-2} on channel C_{k-2}, the segments of group G_{k-1} on channel C_{k-1}, and the segments of group G_k on channel C_k, respectively.

b. If $k \bmod u = 1$:
- Transmit the segments of group G_i on channel C_i in sequence, where $1 \le i \le k-4$.
- Reversely transmit the segments of group G_i on channel, where $k-3 \le i \le k$.

For $u > 2$:

a. If $k \bmod u = 1$ or $k \bmod u = 2$:
- Transmit the segments of group G_i on channel C_i in sequence, where $1 \le i \le k-3$.
- Reversely transmit the segments of group G_{k-2} on channel C_{k-2}, the segments of group G_{k-1} on channel C_{k-1}, and the segments of group G_k on channel C_k, respectively.

b. Otherwise:
- Transmit the segments of group G_i on channel C_i in sequence, where $1 \le i \le k-2$.
- Reversely transmit the segments of group G_{k-1} on channel C_{k-1} and the segments of group G_k on channel C_k, respectively.

B. Client Side

A client is assumed to have enough buffers to store downloaded video segments. We also suppose that a client uses u loaders to receive video segments, where each loader accesses at most one segment from one channel in every time

unit. Initially, the loaders receive the segments of groups G_1 to G_u from channels C_1 to C_u, respectively. When a loader finishes receiving the segments of the x^{th} group, the loader determines to download the segments of which group by checking the value of $x \bmod u$. If $x \bmod u = 0$, the loader downloads the segments of the $x+1^{\text{th}}$ group. Otherwise, the loader accesses the segments of the $x+u+1^{\text{th}}$ group. The loaders repeat the process until all of the segments are received.

Furthermore, each loader must check the segment-broadcasting order on a channel before it starts segment downloading. If the segments are transmitted in sequence, the loader immediately and continuously downloads the segments until all the segments of the same group are received. When the loader finishes receiving the segments of group G_i, the loader determines to download the segments of which group by checking the value of $i \bmod u$. If $i \bmod u = 0$, the loader receives the segments of group G_{i+1}. Otherwise, the loader accesses the segments of group G_{i+u+1}. The downloading process is the same as that of CCA++.

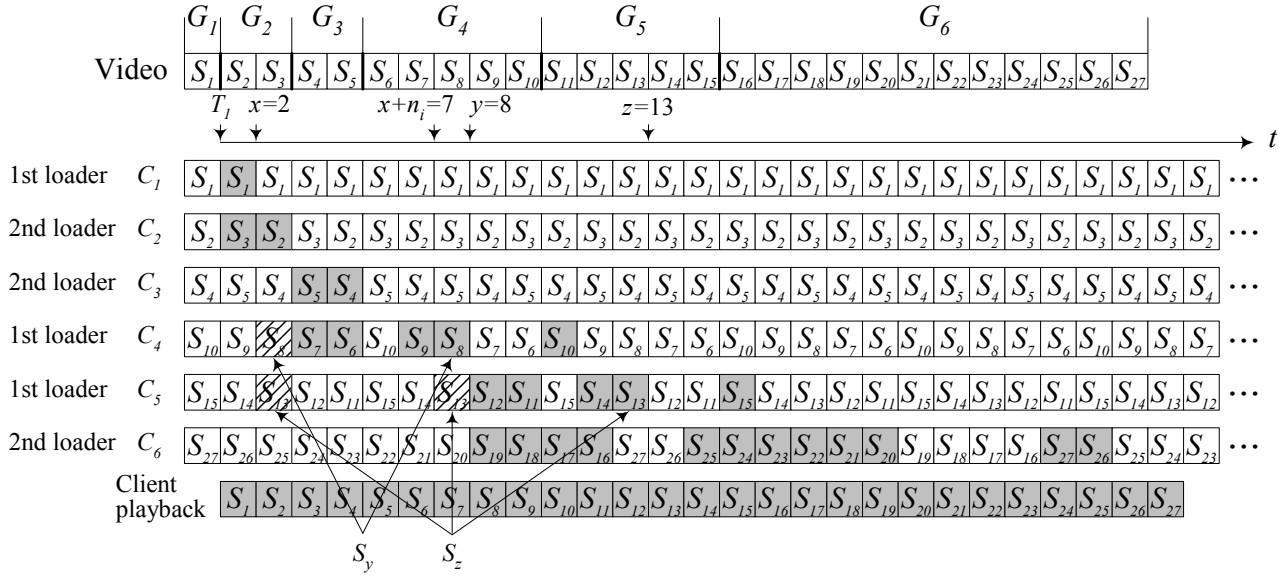

Fig. 1 Illustration of the segment downloading for the proposed extension, where $u = 2$

If the segment-broadcasting order on channel C_i is reverse, the loader performs a different downloading process as the following.

For $i \bmod u > 1$: Suppose that the loader sees a segment S_y on channel C_i at time x, and the segment will be played at time y. If $x \le y$, the segment has not been played, and the loader must decide whether to download it or not. Because segment S_y is broadcast once every n_i time units, the loader next sees it at time $x + n_i$. If $y < x + n_i$, the next appearance of segment S_y is after its playback. Thus, the time unit x is the last chance to download the segment before its playback. Combining $x \le y$ and $y < x + n_i$, we obtain

$$x \le y < x + n_i \tag{2}$$

If the inequality is true, the loader must download segment S_y at time unit x to ensure continuous playback. Otherwise, the loader does not receive the segment. When all the segments of group G_i have been received, the loader stops the downloading.

For $i \bmod u = 0$: The loader also performs the segment downloading on channel C_i according to (2). However, the loader additionally receives segments from channel C_{i+1}. Suppose that the loader sees a segment S_z on channel C_{i+1} at time x, and next sees it at time $x + n_{i+1}$ because it is broadcast once every n_{i+1} time units. The loader determines whether to download the segment by checking

$$x \le z < x + n_{i+1} \tag{3}$$

If the inequality is true, the loader downloads segment S_z; otherwise, the loader waits for its next appearance. When all the segments of group G_{i+1} have been received, the loader stops the downloading.

Fig. 1 demonstrates how to download segments, where $k=6$ and $u=2$. Because $u=2$, two loaders, saying the first loader and the second loader, are used to receive the segments of groups G_1 and G_2 at time T_1. The loaders continuously download the segments since they are broadcast in sequence. Once the segment downloading on channel C_1 is complete, the first loader next accesses the segments of group G_4 on channel C_4 because $i \bmod u \neq 0$, where $i=1$ and $u=2$. The segment downloading on channel C_4 is according to (2) because $4 \bmod 2 = 0$. For example, when the loader first sees segment S_8 with diagonal lines on channel C_4 at the second time unit, the inequality of (2) does not hold (*i.e.,* $y=8>x+n_i=7$), and thus the loader does not receive this segment. When the loader next sees segment S_8 colored gray on channel C_4 at the seventh time unit, (2) becomes true because $7=x<y=8<x+n_i=12$, where $x=7$. The loader thus downloads this segment. Because $i \bmod u = 0$, where $i=4$ and $u=2$, the first loader also receives the segments from channel C_5 according to (3). For instance, when the loader first sees segment S_{13} with diagonal lines on channel C_5 at the second time unit, the inequality of (3) does not hold (*i.e.,* $z=13>x+n_j=7$), and thus the loader does not receive this segment. When the loader next sees segment S_{13} at the seventh time unit, the inequality of (3) does not hold still, and the loader skips the segment again. The segment S_{13} is finally downloaded at the 12^{th} time unit because (3) becomes true, $12=x\le z=13<x+n_j=17$. When all the segments of group G_5 have been received, the first loader stops the downloading.

Furthermore, once the second loader finishes the segment downloading on channel C_2 at the second time unit, it continuously receives the segments from channel C_3 because $2 \bmod u = 0$ and the segments are broadcast in sequence. When the downloading on channel C_3 is complete, the loader next accesses the segments of group G_6 because $3 \bmod 2 \neq 0$ and $3+2+1=6$. The segment downloading on channel C_6 is according to (2). Once all the segments of group G_6 have been received, the second loader stops the downloading.

IV. Performance Analysis and Comparison

This section evaluates the performance of the proposed scheme. We compare RCCA++ with CCA, CCA+, and CCA++. When a client barely misses segment S_1 of a requested video, the maximal waiting time δ equals its access time from the first channel. Thus, $\delta = L/N = L/\sum_{j=1}^{k} n_j$. The video length L is assumed to be 120 minutes, and the server bandwidth is varied from 1 to 15 channels. The performance results of CCA, CCA+, CCA++, and RCCA++ are plotted in Fig. 2, where $u=2$. The figure shows that as the server bandwidth increases, the waiting time under all four schemes reduces sharply. Because CCA++ and RCCA++ are based on the same video partition scheme, they have the same waiting time. Furthermore, CCA++ and RCCA++ yield the shortest waiting time. For example, when the server bandwidth equals 15 channels, CCA++ and RCCA++ reduce the broadcast latency to less than 5.2 seconds. By contrast, CCA and CCA+ incur more than 14.1 and 6.8 seconds, respectively.

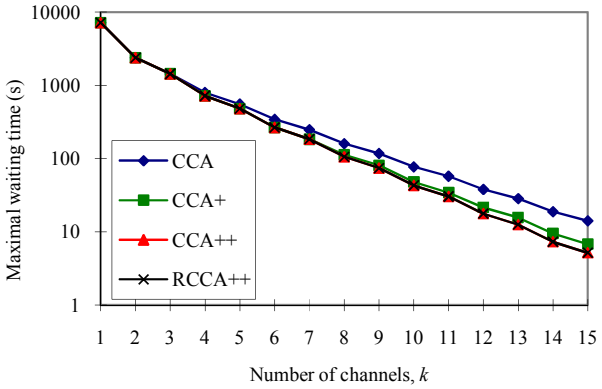

Fig. 2 Maximum waiting time incurred on new clients at different numbers of channels ($L=120$ minutes)

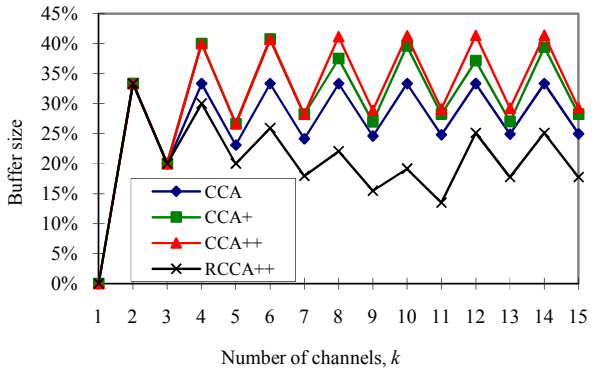

Fig. 3 Maximum buffer requirements in percentage of video size at different numbers of channels

This work used the Perl language to implement a simulator to yield the buffer requirements for CCA, CCA+, CCA++, and RCCA++ under various broadcasting bandwidths. The results are plotted in Fig. 3, where $u=2$. For $k>3$, RCCA++ requires the smallest buffer space than other schemes. For example, when the server bandwidth equals 11 channels, RCCA++ requires a client to buffer 14% of video size. By

contrast, CCA, CCA+, and CCA++ need 25%, 28%, and 29%, respectively. RCCA++ thus reduces the buffer requirements by 44% for CCA, 50% for CCA+, and 52% for CCA++.

V. Conclusions

This study presents an improved version of CCA++ for reducing client buffer space. The proposed scheme, RCCA++, requires far smaller buffer space than CCA++. Based on the same amount of server bandwidth and client receiving bandwidth, RCCA++ can yield 52% smaller buffer size than CCA++ can. In evaluating the performance, this study compares RCCA++ with CCA, CCA+, and CCA++. The results clearly show the advantages of using RCCA++ under various parameter settings.

Acknowledgment

This work was supported in part by the National Science Council of Taiwan under Contract NSC 101-2221-E-152-004.

References

[1] M. Vilas, X. G. Paneda, R. Garcia, D. Melendi, and V. G. Garcia, "User behavior analysis of a video-on-demand service with a wide variety of subjects and lengths," in *Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*, pp. 330-337, August 2005.

[2] H. Yu, D. Zheng, B. Y. Zhao, and W Zheng, "Understanding user behavior in large-scale video-on-demand systems," in *Proceedings of EuroSys 2006*, pp. 333-344, October 2006.

[3] L.-S. Juhn and L.-M. Tseng, "Fast data broadcasting and receiving scheme for popular video services," *IEEE Transactions on Broadcasting*, vol. 44, no. 1, pp. 100–105, March 1998.

[4] Yu-Chee Tseng, Ming-Hour Yang, and Chi-He Chang, "A recursive frequency-splitting scheme for broadcasting hot videos in VOD service," *IEEE Transactions on Communications*, vol. 50, no. 8, pp. 1348–1355, August 2002.

[5] H.-F. Yu, H.-C. Yang, and L.-M. Tseng, "Reverse Fast Broadcasting (RFB) for Video-on-Demand Applications," *IEEE Transactions on Broadcasting*, vol. 53, no. 1, pp. 103-111, March 2007.

[6] H.-F. Yu, "Hybrid Broadcasting with Small Buffer Demand and Waiting Time for Video-on-Demand Applications," *IEEE Transactions on Broadcasting*, vol. 54, no. 2, pp. 304-311, June 2008.

[7] Y.-N. Chen and L.-M. Tseng, "An Efficient Periodic Broadcasting with Small Latency and Buffer Demand for Near Video on Demand," *International Journal of Digital Multimedia Broadcasting*, vol. 2012, Article ID 717538, 7 pages, 2012.

[8] J. F. Paris, "A fixed-delay broadcasting protocol for video-on-demand," in *Proceedings of the 10th International Conference on Computer Communications and Networks (ICCCN '01)*, pp. 418-423, October 2001.

[9] H.-F. Yu, P.-H. Ho and H.-C. Yang, "Generalized Sequence-based and Reverse Sequence-based Models for Broadcasting Hot Videos," *IEEE Transactions on Multimedia*, vol. 11, no. 1, pp. 152-165, January 2009.

[10] K.A. Hua and S. Sheu, "Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems," *ACM SIGCOMM*, September 1997.

[11] Y. Cai, A. Hua and S. Sheu, "Leverage client bandwidth to improve service latency of distributed multimedia applications," *Journal of Applied Systems Studies*, 2(3), 2001.

[12] A. Natarajan, Y. Cai, J. Wong, "An enhanced client-centric approach for efficient video broadcast," *Multimedia Tools and Application*, vol. 43, no. 2, pp. 179-193, 2009.

[13] H.-F. Yu, "Improvement of the Client-Centric Approach for Broadcasting Popular Videos," *Multimedia Tools and Applications*, vol. 67, no. 3, pp. 629-639, December 2013.