Hindawi Publishing Corporation
International Journal of Digital Multimedia Broadcasting
Volume 2012, Article ID 192314, 8 pages
doi:10.1155/2012/192314

Research Article

Adjustable Two-Tier Cache for IPTV Based on Segmented Streaming

Kai-Chun Liang and Hsiang-Fu Yu

Department of Computer Science, National Taipei University of Education, Taipei 106, Taiwan

Correspondence should be addressed to Hsiang-Fu Yu, yu@tea.ntue.edu.tw

Received 31 January 2012; Accepted 10 March 2012

Academic Editor: Pin-Han Ho

Internet protocol TV (IPTV) is a promising Internet killer application, which integrates video, voice, and data onto a single IP network, and offers viewers an innovative set of choices and control over their TV content. To provide high-quality IPTV services, an effective strategy is based on caching. This work proposes a segment-based two-tier caching approach, which divides each video into multiple segments to be cached. This approach also partitions the cache space into two layers, where the first layer mainly caches to-be-played segments and the second layer saves possibly played segments. As the segment access becomes frequent, the proposed approach enlarges the first layer and reduces the second layer, and vice versa. Because requested segments may not be accessed frequently, this work further designs an admission control mechanism to determine whether an incoming segment should be cached or not. The cache architecture takes forward/stop playback into account and may replace the unused segments under the interrupted playback. Finally, we conduct comprehensive simulation experiments to evaluate the performance of the proposed approach. The results show that our approach can yield higher hit ratio than previous work under various environmental parameters.

1. Introduction

Internet protocol TV (IPTV) is a promising Internet killer application, which integrates video, voice, and data onto a single IP network, and offers viewers an innovative set of choices and control over their TV content. Many major telecommunication companies, such as AT&T, Verizon, and Bell, have announced their IPTV solutions by replacing the copper lines in their networks with fiber optic cables to create sufficient bandwidths for delivering many TV contents. The Bell Entertainment Service in Bell Canada, for example, uses a single VDSL line with a consistent download speed of 20 Mbps and an upload rate of 8 Mbps to provide a converged Internet and television service. The trend is similar in other areas, such as Europe and Asia. Major cities in Japan, for example, already provide high-speed networks which allow customers to obtain video over IP. In Taiwan, the largest telecommunication company, Chunghwa Telecom, offers the multimedia on-demand (MOD) services, which allow clients to watch traditional TV contents over IPTV infrastructures.

A conventional solution to provide IPTV services is through a content distribution network (CDN), in which the service provider installs multiple video servers at different locations to transmit video contents to local customers. In general, multimedia streaming objects are far bigger than web objects. Additionally, real-time transmission is necessary for continuous video playback. A video server thus yields much larger disk load and bandwidth consumption than a web server. Once viewer arrival rate increases significantly, the video server is easily overloaded and reduces service quality. To alleviate the limits, most CDNs are based on cache servers. Figure 1 depicts a popular architecture, which is composed of a video server, cache servers, and clients. Usually, the video server is in WAN, and the cache servers are deployed near clients. An incoming video request is first forwarded to the cache server, instead of the video server. Once receiving the request, the cache server checks whether

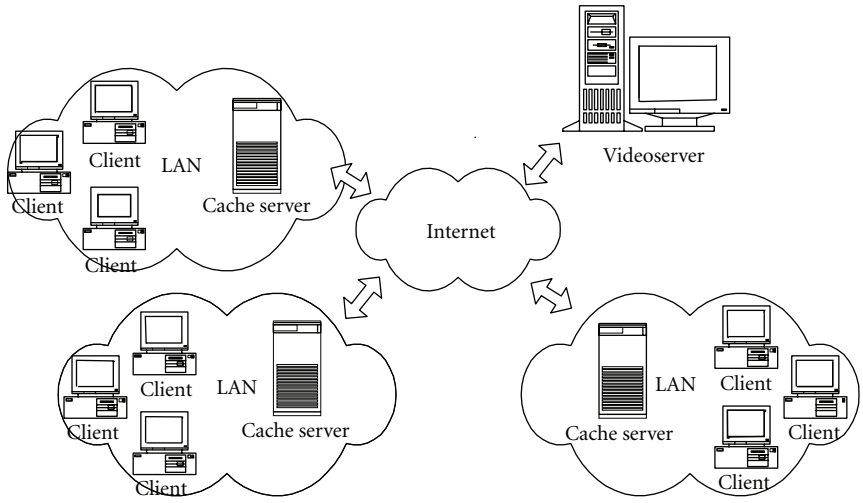

Figure 1: Basic streaming cache architecture.

the video data are available in the cache storage. If so, the cache server transmits the data to the client directly. Otherwise, the cache server connects to the video server for the video data, which are then forwarded to the client.

The studies in [1–3] investigated how to allow time-shifted IPTV services, which could enable the end user to watch a broadcasted TV program with a time shift. Wauters et al. [1, 2] proposed a network-based time-shifted television (tsTV) solution using cooperative proxy caches. The study in [3] proposed a hybrid strategy combining genetic algorithms to determine the optimal cache locations for supporting time-shifted IPTV services. Simsarian and Duelk [4] analyzed the bandwidth requirements in metropolitan area networks (MANs) for providing IPTV services and developed a model of the IPTV network to determine the optimum location of the cached video content. The study in [5] introduced the concept of content cacheability and proposed a cache-partition algorithm using the cacheability to serve the maximum amount of video requests subject to constraints on cache memory and throughput. Sofman and Krogfoss [6] indicated that a portion of the video content could be stored in caches closer to clients to reduce the IPTV traffic and further presented an analytical model of hierarchical cache optimization depending on traffic, topology, and cost parameters. A heuristic model [7] was proposed for hierarchical cache optimization in an IPTV network. Chen et al. [8] presented an IPTV system based on a peer-to-peer hierarchical cache architecture. The work [9] devised a caching algorithm that tracked the popularity of objects to make intelligent caching decisions in IPTV services.

A web cache server generally considers a web page an atomic object, and thus caches a complete page. However, caching a complete streaming object is not suitable to a streaming cache server. If a streaming cache server always caches a complete video object, the number of cached videos will be very small because a streaming object is much larger than a web page. When incoming requests increase, cached videos are easily swapped out because the cached objects are not enough, leading to poor cache performance. In addition, caching an entire video also results in long playback latency because the data transmission time is too large to be ignored. Suppose that a client can download a 100-minute MPEG2 video encoded by 6 Mbps at a bandwidth of 10 Mbps. The video size is $100 * 60 * 6$ bits, and the data transmission time equals 3600 seconds. Clearly, playback after downloading is unrealistic. To alleviate these problems, many studies [10–19] partition a streaming object into multiple smaller segments, which are cached partially. A prefix caching [10] stores the initial frames of popular videos. Upon receiving a video request, the cache server transmits the initial frames to the client and simultaneously requests the remaining frames from the video server. Wu et al. [11, 12] investigate how to partition videos to achieve higher hit ratio. Three video-segmentation approaches—fixed, pyramid, and skyscraper—are proposed. Their simulation results indicate that the pyramid segmentation is the best segmentation approach. Compared with whole video caching, segmentation-based caching is more effective in increased byte-hit ratio. Lazy segmentation approach [13] delays the video partition until a video is accessed. The study in [14] introduces the proxy jitter, which results in playback jitter at the client side due to proxy delay in fetching the uncached segments. The proposed hyperproxy [14] can generate minimum proxy jitter with a low delayed startup ratio and a small decrease of byte-hit ratio. SProxy [15] implements a segment-based streaming cache system on Squid [16]. The study in [17] devises a segment-based cache mechanism to support VCR functions on the client side. extending popularity-aware partial caching algorithm (PAPA) [18], dynamic segment-based caching algorithm (DECA) [19] determines the segment size according to segment popularity.

This paper proposes a two-layer segment-based cache for streaming objects, as shown in Figure 2. The cache server divides the cache storage into two layers—L1 and L2—in

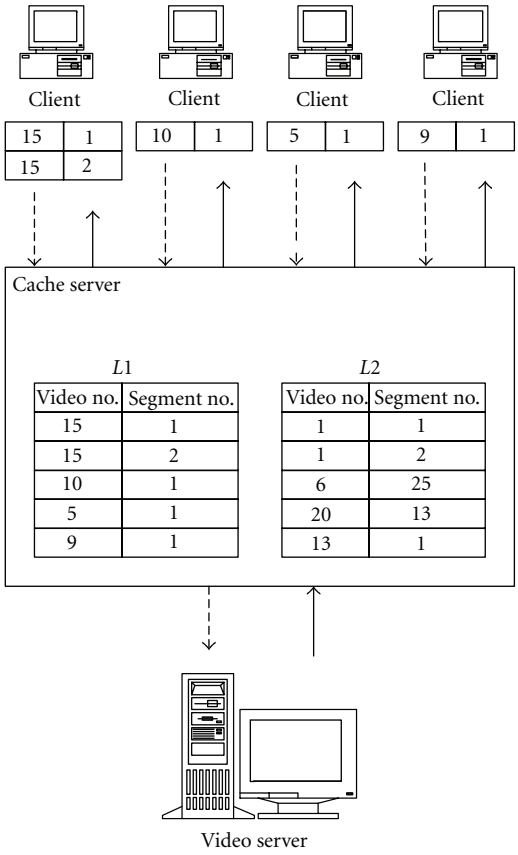

Figure 2: The proposed streaming cache.

which cache L1 mainly caches to-be-played segments and cache L2 saves possibly played segments. As the segment access becomes frequent, the proposed approach enlarges cache L1 and reduces cache L2, and vice versa. Once the space of cache L1 is not enough, cache L1 uses LRU to choose a victim, which is then moved to cache L2. If cache L2 also has not enough space, cache L2 first swaps out a selected segment according to LRU-K [20] and saves the segment coming from cache L1. Because requested segments may not be accessed frequently, this work further designs an admission control mechanism to determine whether an incoming segment should be cached or not. The cache architecture takes forward/stop playback into account and may replace the unused segments under the interrupted playback. Table 1 briefly compares the proposed caching architecture with previous approaches. This work conducts a comprehensive simulation to evaluate the proposed cache under various cache size, video popularity, request arrival rate, and playback interrupt rate. In comparison with the segment-based LRU, the video-based LRU, and Wu's approach [12], our approach mostly yields higher hit ratio.

The rest of this paper is organized as follows. Section 2 presents the proposed cache. The simulation results and performance comparisons are shown in Section 3. Brief conclusions are drawn in Section 4.

$\text{Ratio}_{\text{viewing}} = C_{\text{viewing}}/C_{\text{L1+L2}}$
If($\text{Ratio}_{\text{viewing}} \geq \text{L1\_Cache\_Ratio}_{\max}$){
  $C_{\text{L1\_new\_size}} = C_{\text{L1+L2}} \cdot \text{L1\_Cache\_Ratio}_{\max}$
  $C_{\text{L2\_new\_size}} = C_{\text{L1+L2}} - C_{\text{L1\_current\_size}}$}
Else{
  $C_{\text{L1\_new\_size}} = \text{Ratio}_{\text{viewing}} \cdot C_{\text{L1+L2}}$
  $C_{\text{L2\_new\_size}} = C_{\text{L1+L2}} - C_{\text{L1\_new\_size}}$}

Algorithm 1: The algorithm to determine the sizes of caches L1 and L2.

2. The Proposed Cache

The work devises a segment-based two-layer caching approach to increase byte-hit ratio. Suppose that the bandwidth between a video server and a cache server is unlimited, and the bandwidth between the cache server and a client is larger than playback rate. Each video is partitioned into multiple fixed-length segments, as indicated in Figure 2. The proposed approach periodically adjusts the sizes of caches L1 and L2 according to the size of segments accessed, as shown in Figure 3. With the decreasing of the number of the segments currently played, the proposed approach reduces the size of cache L1 and enlarges the size of cache L2, and vice versa. In order to avoid frequent size adjustment, the adjustment is executed periodically, rather than when a segment request arrives. Table 2 lists the parameters to determine the sizes of caches L1 and L2. Algorithm 1 shows how to determine the sizes of caches L1 and L2. The size of cache L1 has an upper bound to avoid the size of cache L2 being zero when many video requests arrive. Upon determining the cache sizes, the cache server adjusts the cache space by moving recently or seldom used segments, as indicated in Algorithm 2.

Besides LRU, the replacement of cache L1 is also based on the observation that a video is played continuously. If a video is played currently, its segments not played yet are very possibly accessed later. Accordingly, cache L1 avoids swapping out these segments. It is well known that the popularity of video data varies with time. The condition that video segments in cache L1 are no longer played may reflect that the segments become less popular. We thus move the segments to cache L2 when cache L1 is full. Figure 4 depicts the complete operation of the proposed approach once cache hits. When a requested segment hits the cache, the segment can be either in cache L1 or in cache L2. If the segment hits cache L1, cache L1 reorders the segment according to LRU and transmits the segment to the requested client. Otherwise, the segment hits cache L2 and is moved to cache L1. If cache L1 has enough space, cache L2 directly moves the segment to cache L1; else, cache L1 swaps cache L2 a played segment for the hit segment.

When a segment is neither in cache L1 nor in cache L2, the segment is missed. Figure 5 shows how to process a missed segment. If cache L1 has enough space, the segment is saved in cache L1 according to LRU. Otherwise, if cache L2 has free space, cache L1 swaps cache L2 a played segment

```
If(Ratio_viewing > C_L1_current_size/C_L1+L2){ // if cache L1 is not enough
   While(C_L1_current_size < C_L1_new_size){
      Select recently-used segments in cache L2 to move to the bottom of cache L1.
      Update current cache sizes of caches L1 and L2. }}
Elsif(Ratio_viewing < C_L1_current_size/C_L1+L2){
   While (C_L1_current_size > C_L1_new_size){
      Select seldom-used segments in cache L1 to move to the top of cache L2.
      Update current cache sizes of caches L1 and L2.}}
```

Algorithm 2: The algorithm to adjust the sizes of caches L1 and L2.

Table 1: Comparison among related streaming caches.

Approach	Wu's approach	Hyperproxy	SProxy	PAPA	DECA	Segment-based two-layer caching
Segment partition	Pyramid	Lazy	Fixed	Segment prefix	Dynamic segment prefix	Fixed
Cache replacement	Cost function + LRU	Cost function	NA	Cost function	Cost function	LRU + LRU-K
Number of cache layers	2	1	1	1	2	2
Admission control	YES	YES	NA	NA	NA	YES
Precache	NA	YES	YES	NA	NA	NA

Table 2: Terms used by the algorithm to determine the cache size.

Term	Definition
C_{L1+L2}	Entire cache size
$C_{L1\_current\_size}$	Current size of cache L1
$C_{L1\_new\_size}$	New size of cache L1
$C_{L2\_current\_size}$	Current size of cache L2
$C_{L2\_new\_size}$	New size of cache L2
$C_{viewing}$	Size of playing segments
$L1\_Cache\_Ratio_{max}$	Maximum ratio of size of cache L1 to entire cache size
$Ratio_{viewing}$	Ratio of playing segments to entire cache size

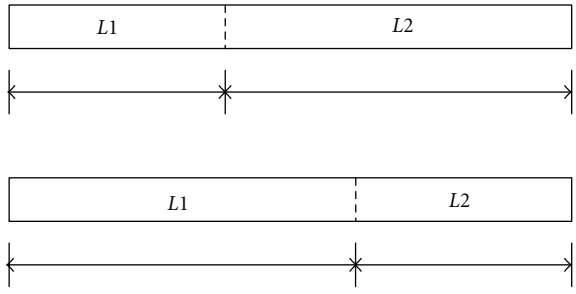

Figure 3: Size adjustment on caches L1 and L2.

for the missed segment. If cache L2 is also full, cache L2 performs an admission control to determine whether to cache the segment. If not, the missed segment is not cached and simply transmitted to the client. If so, cache L2 drops a victim segment according to LRU-K. Cache L1 then moves a segment chosen by LRU to cache L2 and saves the missed segment.

The admission control is based on LRU-K. When both of the caches L1 and L2 are full, the admission control compares the previous Kth access time of a missed segment with that of the victim segment chosen by cache L2. If the access time of the missed segment is later than that of the victim segment cache L2 drops the victim segment and saves the missed segment.

3. Performance Analysis and Comparison

We implemented an event-driven simulator by Perl to evaluate the performance of the proposed cache approach. The simulation settings are listed in Table 3. Suppose that the number of videos equals 2000. Assume that the video size is uniformly distributed between 10 segments and 110 segments, where the length of each segment equals 1 minute. The cache size is expressed in terms of ratio of total videos, and the default value is 0.2. The interarrival time is assumed to follow a Poisson distribution. For each request, it is generated by a Poisson process, which is exponentially distributed with a mean of $1/\lambda$, where λ is the request arrival rate. The default value is 6 requests per minute. The requested videos are drawing from a total of M distinct videos. The popularity of each video follows a Zipf-like distribution Zipf(x, M) [21]. A Zipf-like distribution contains two parameters, x and M, the former corresponding to the degree of skew. The distribution of each video i equals $p_i = c/i^{1-x}$, where $i \in \{1, \ldots, M\}$ and $c = 1/\sum_{j=1}^{M}(1/j^{1-x})$. Setting $x = 0$ corresponds to a pure Zipf distribution, which is highly skewed. On the other hand, setting $x = 1$ corresponds to a uniform distribution without skew. The default value for x is 0.2 and that for M is 2,000. The popularity of each video changes

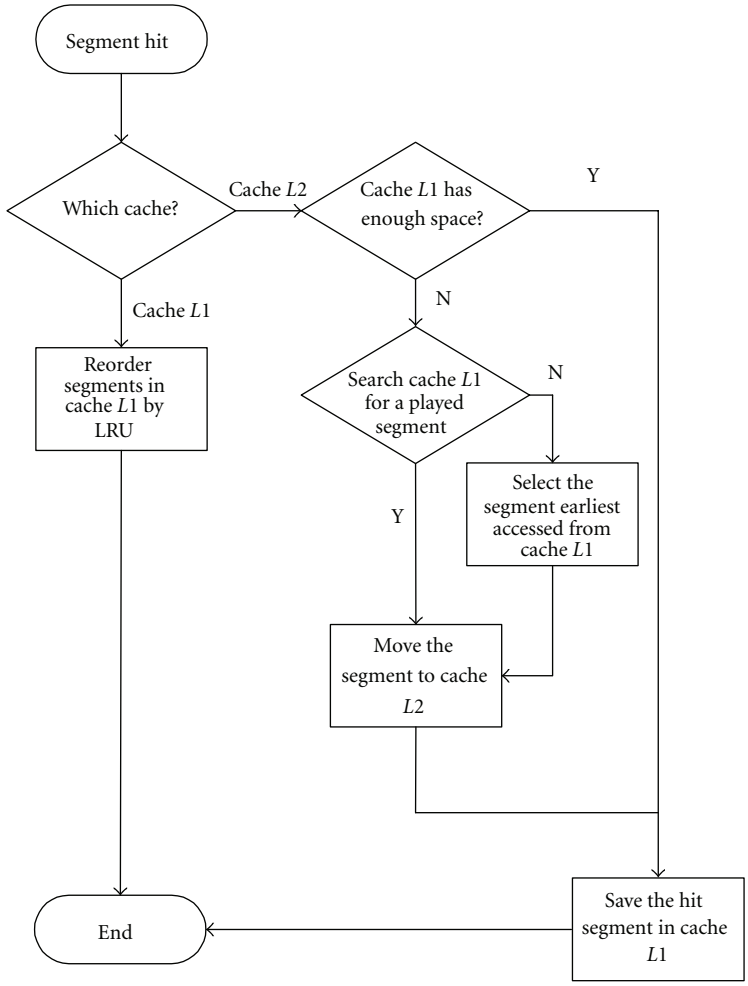

Figure 4: The algorithm to process the segment hit.

Table 3: Simulation parameters and default values.

Number of videos	2000
Video length	10–110 minutes
Request arrival rate	6 requests per minute
Simulation duration	43200 minutes
Cache size	0.2 (in the percentage of the entire video size)
Video popularity	Zipf-like distribution for video titles, Zipf(0.2, 2000)
Probability of forward/stop playback	0.2
Popularity shift distance	10 videos every 21600 minutes

over time to simulate the scenario that there may be different user groups accessing the videos at different time and their interest may be different. Similar to Wu's study [11], the popularity distribution changes every 21600 minutes, and the shift distance equals 10 videos. The default probability of forward/stop playback is 0.2.

This work compares the proposed approach with the video-based LRU, the segment-based LRU, and Wu's approach [12]. The video-based LRU caches a complete video and selects a replaced video according to LRU. For the segment-based LRU, a video is partitioned into multiple equal-size segments. Wu's approach divides a video into unequal segments under pyramid segmentation. The simulator is installed on FreeBSD 8.0 running on HP ProLiant DL380G6 and HP ProLiant DL320G6.

Figure 6 shows the impact of the cache size on the byte-hit ratio. For a wide range of the cache size, the proposed approach has higher byte-hit ratio than other approaches. The advantage in byte-hit ratio of our approach is more significant for a smaller cache size. For instance, the hit ratio of our approach is 11% higher than that of Wu's approach at the cache size of 0.1, while 26% better than those of the video-based LRU and the segment-based LRU. With the growth of the cache size, all the schemes can cache most videos, and thus their performance is similar.

We next examine the impact of the skew in video popularity on the byte-hit ratio, as indicated in Figure 7. The proposed approach has the higher byte-hit ratio under skewed video popularity. When the video popularity becomes normal distribution, our scheme performs less effectively. For example, the hit ratio of the proposed scheme is 7% better

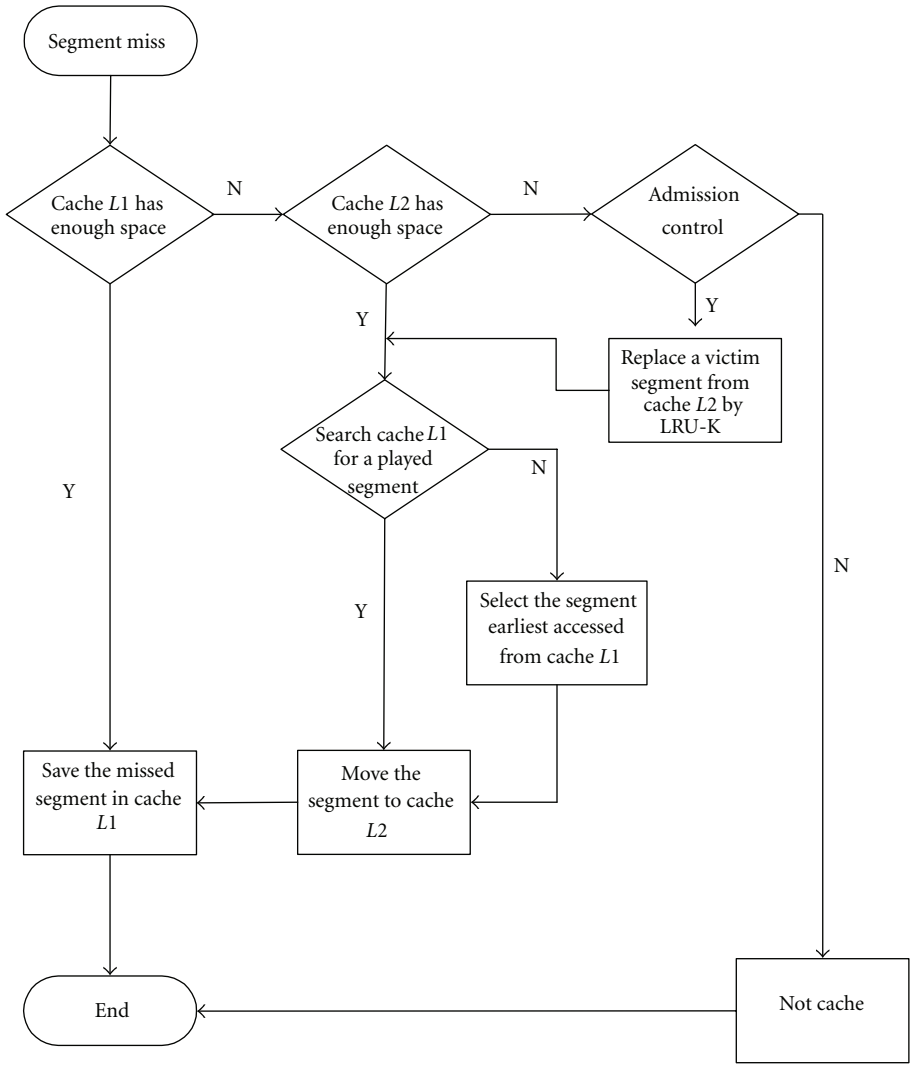

Figure 5: The algorithm to process the segment missed.

than that of Wu's approach at the skew factor of 0.2. The scheme also outperforms the video-based LRU and the segment-based LRU. However, when the skew factor is larger than 0.6, Wu's approach performs better than the proposed.

Figure 8 shows the impact of the request arrival rate on the byte-hit ratio. For a wide range of the arrival rate, the proposed approach outperforms other schemes. In comparison with Wu's approach, our approach yields 3–11% higher-hit ratio. The hit ratio of the approach is also 11–13% larger than those of the video-based LRU and the segment-based LRU. The results reflect that the proposed cache performs steadily under various request arrival rates.

Figure 9 depicts the impact of the rate of the forward/stop playback on the byte-hit ratio. The rate indicates the probability that a user performs forward/stop playback during watching a video. The rate of 0.1 represents that one of ten videos happens forward/stop playback. The figure shows that the proposed cache yields 8-9% larger hit ratio than Wu's

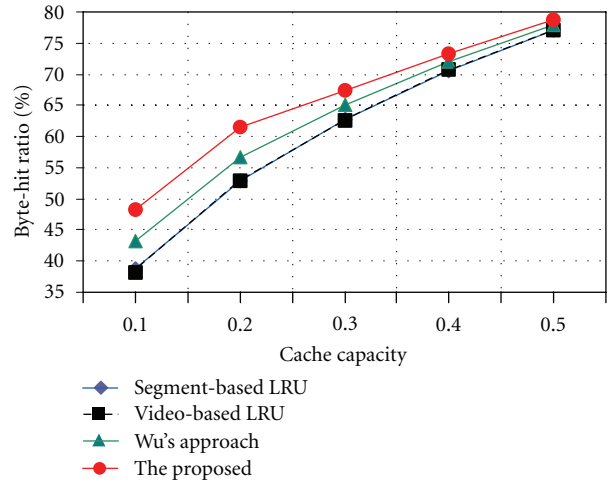

Figure 6: Influence of cache size on byte-hit ratio.

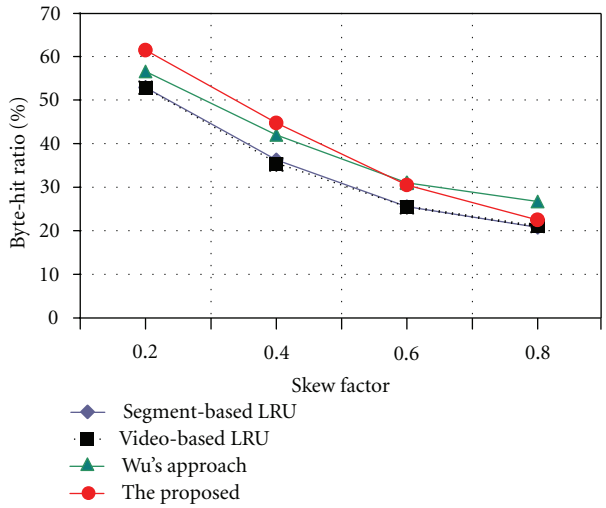

Figure 7: Impact of skew in video popularity on byte-hit ratio.

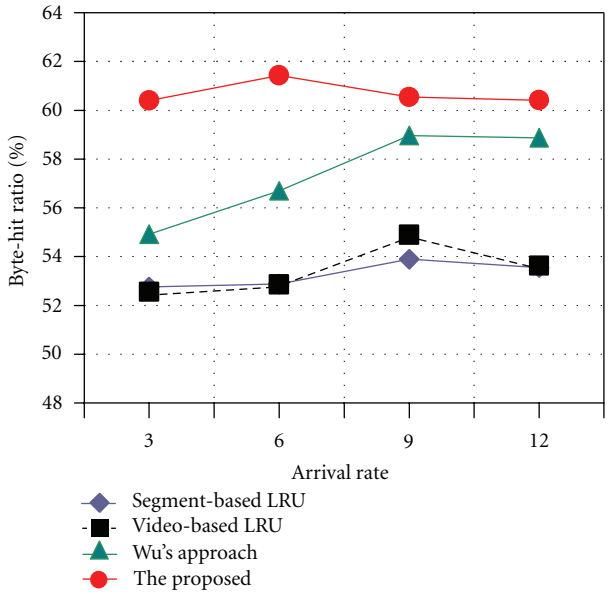

Figure 8: Influence of request arrival rate on byte-hit ratio.

approach. In comparison with the video-based LRU and the segment-based LRU, the proposed approach also achieves 13–17% better.

4. Conclusions

Internet protocol TV (IPTV) is a promising Internet killer application, which integrates video, voice, and data onto a single IP network, and offers viewers an innovative set of choices and control over their TV content. To provide high-quality IPTV services, this work proposes a two-layer segment-based cache, which divides the cache storage into caches L1 and L2, and dynamically adjusts their sizes according to video popularity. As the segment access becomes frequent, the proposed approach enlarges cache L1 and reduces cache L2, and vice versa. Once the space of cache

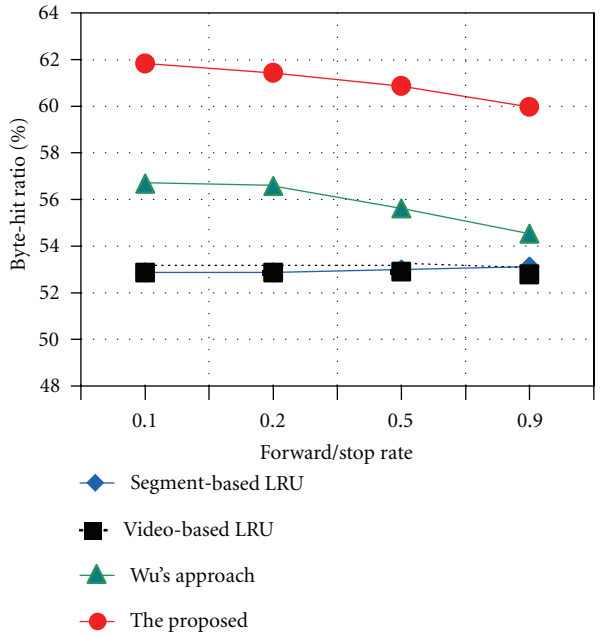

Figure 9: Impact of forward/stop playback on byte-hit ratio.

L1 is not enough, cache L1 uses LRU to choose a victim, which is then moved to cache L2. If cache L2 also has not enough space, cache L2 first swaps out a selected segment according to LRU-K and saves the segment coming from cache L1. To enlarge the hit ratio, this study also presents an admission control to determine which accessed segments should be cached. The proposed cache further considers the situations that clients may suddenly perform forward/stop playback. This work conducts a comprehensive simulation to evaluate the proposed cache under different cache size, video popularity, request arrival rate, and playback interrupt rate. The simulation results indicate that our approach mostly outperforms the segment-based LRU, the video-based LRU, and Wu's approach under various settings.

Acknowledgment

The work was financially supported by National Science Council, Taiwan under a research Grant no. NSC 100-2221-E-152-005.

References

[1] T. Wauters, W. Van De Meerssche, F. De Turck et al., "Co-operative proxy caching algorithms for time-shifted IPTV services," in *Proceedings of the 32nd Euromicro Conference on Software Engineering and Advanced Applications (SEAA '06)*, pp. 379–386, September 2006.

[2] T. Wauters, W. Van De Meerssche, F. De Turck et al., "Management of time-shifted IPTV services through transparent proxy deployment," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM '06)*, pp. 1–5, December 2006.

[3] Z. Juchao, L. Jun, and W. Gang, "Study of cache placement for time-shifted TV cluster using genetic algorithm," in

Proceedings of the Genetic and Evolutionary Computation Conference (GEC '09), pp. 781–785, June 2009.

[4] J. E. Simsarian and M. Duelk, "IPTV bandwidth demands in Metropolitan Area Networks," in *Proceedings of the 15th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN '07)*, pp. 31–36, June 2007.

[5] L. B. Sofman, B. Krogfoss, and A. Agrawal, "Optimal cache partitioning in IPTV network," in *Proceedings of the 11th Communications and Networking Simulation Symposium (CNS '08)*, pp. 79–84, April 2008.

[6] L. B. Sofman and B. Krogfoss, "Analytical model for hierarchical cache optimization in IPTV network," *IEEE Transactions on Broadcasting*, vol. 55, no. 1, pp. 62–70, 2009.

[7] B. Krogfoss, L. B. Sofman, and A. Agrawal, "Hierarchical cache optimization in IPTV networks," in *Proceedings of the IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB '09)*, pp. 1–10, May 2009.

[8] L. Chen, M. Meo, and A. Scicchitano, "Caching video contents in IPTV systems with hierarchical architecture," in *Proceedings of the IEEE International Conference on Communications (ICC '09)*, pp. 1–6, June 2009.

[9] D. De Vleeschauwer and K. Laevens, "Performance of caching algorithms for IPTV on-demand services," *IEEE Transactions on Broadcasting*, vol. 55, no. 2, pp. 491–501, 2009.

[10] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societie (INFOCOM '99)*, pp. 1310–1319, New York, NY, USA, March 1999.

[11] K. L. Wu, P. S. Yu, and J. L. Wolf, "Segment-based proxy caching of multimedia streams," in *Proceedings of the International Conference on World Wide Web (WWW '01)*, Hongkong, May 2001.

[12] K. L. Wu, P. S. Yu, and J. L. Wolf, "Segmentation of multimedia streams for proxy caching," *IEEE Transactions on Multimedia*, vol. 6, no. 5, pp. 770–780, 2004.

[13] S. Chen, H. Wang, X. Zhang, B. Shen, and S. Wee, "Segment-based proxy caching for Internet streaming media delivery," *IEEE Multimedia*, vol. 12, no. 3, pp. 59–67, 2005.

[14] S. Chen, B. Shen, S. Wee, and X. Zhang, "Segment-based streaming media proxy: modeling and optimization," *IEEE Transactions on Multimedia*, vol. 8, no. 2, pp. 243–256, 2006.

[15] S. Chen, B. Shen, S. Wee, and X. Zhang, "SProxy: a caching infrastructure to support internet streaming," *IEEE Transactions on Multimedia*, vol. 9, no. 5, pp. 1062–1072, 2007.

[16] http://www.squid-cache.org/.

[17] J. Z. Wang and P. S. Yu, "Fragmental proxy caching for streaming multimedia objects," *IEEE Transactions on Multimedia*, vol. 9, no. 1, pp. 147–156, 2007.

[18] L. Shen, W. Tu, and E. Steinbach, "A flexible starting point based partial caching algorithm for video on demand," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME '07)*, pp. 76–79, Beijing, China, July 2007.

[19] W. Tu, E. Steinbach, M. Muhammad, and X. Li, "Proxy caching for video-on-demand using flexible starting point selection," *IEEE Transactions on Multimedia*, vol. 11, no. 4, pp. 716–729, 2009.

[20] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "LRU-K page replacement algorithm for database disk buffering," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 297–306, May 1993.

[21] G. K. Zipf, *Human Behavior and the Principles of Least Effort*, Addison-Wesley, Cambridge, Mass, USA, 1949.