Research Article

Flexible-Segmentation-Jumping Strategy to Reduce User-Perceived Latency for Video on Demand

Jun Li, Fengbin Li, and Xiaofeng Jiang

CAS, Key Laboratory of Network Communication System and Control, University of Science and Technology of China (USTC), Hefei, Anhui 230027, China

Correspondence should be addressed to Fengbin Li, lfb0132@mail.ustc.edu.cn

Received 11 August 2010; Revised 7 November 2010; Accepted 9 February 2011

Academic Editor: Miin-Shen Yang

Copyright © 2011 Jun Li et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Traditionally, people usually watch a video from the beginning and continuously to the end; this is changed by the concept and application of Video-On-Demand (VOD). Users do not want to wait for a long time when they seek some specific content in a video; they want to instantly watch any part in a video according to their needs. To resolve this challenge, in this paper, we propose a Flexible-Segmentation-Jumping Strategy (FSJS). This scheme considers users' randomly access behaviors, especially concerns the initial delay before watch point selection. By considering these behaviors and flexibly selecting jumping point, our scheme can significantly reduce user waiting time, in most cases can reduce the waiting time to zero. Our simulation implements the proposed FSJS scheme to the uniform segmentation and exponential segmentation algorithms to show how FSJS improves a user's perceived latency and reduces the extra average serving time. The simulation results show that FSJS can have a significant improvement in user-perceived latency.

1. Introduction

Nowadays, Video-On-Demand (VOD) services have been successfully applied in many applications; their unique merits have been accepted by most people and companies. Compared to text-based content systems, a VOD system usually serves large-size streaming media, which means it naturally requires larger disk space. In a VOD system, the initial delay has become one of the most important aspects for its quality of service (QoS). Using multiple servers to store media files cannot solve the large initial delay issue when there are a number of clients watching VOD at the same time; in addition, this method cannot effectively reduce user-perceived latency.

Figure 1 shows the general architecture of a VOD system. VOD servers are connected with each other, not isolated anymore when compared with the traditional VOD form; they all connect to a shared disk array, which is used to store all media objects. Such architecture has at least two advantages. First, the streaming media objects are stored not in individual servers but in a disk array, and, hence, the disk space can be shared by all servers to store more media files.

Second, since servers are connected with each other, we can apply some scheduling strategies to improve the performance of the system. For example, we can use Load-Balance strategy to schedule and manage the server resources or some other useful strategies to decrease a user's start-up latency (initial delay). In this paper, we focus on user's waiting time and how to reduce a user's perceived latency.

Traditionally, segment-based cache [1–4] is a good scheme to deal with how to hit as many segments as possible in a predefined caching space. It includes the uniform-segmentation strategy, exponential-segmentation strategy [1], and adaptive and lazy segmentation strategy [2–4]; each of them has its own advantages when used in a VOD system. Uniform-segmentation is easily to be used, and it considers the recently used segment of a file. Exponential-segmentation scheme concerns both the recently-used and the size of a segment of a file. Adaptive and lazy segmentation considers many factors, among which the most prominent factor that is the average access duration. All these algorithms effectively improve the Byte-Hit-Ratio (BHR) and reduce the network bandwidth.

FIGURE 1: Architecture of the VOD system.

However, because the algorithms introduced above all aim to boost BHR, they have little contribution in improving the response time of requested media object. Their focus is to reduce the server load, not the user-perceived latency (start-up latency). If we reduce the user-perceived latency, which directly corresponds to users' perception when they use a VOD system, user satisfaction will be highly increased. Therefore, it is urgent for us to design a new caching scheme focusing on reducing user startup latency. Another problem is that the algorithms introduced above all assume that users always play media files from the beginning and continuously towards the end, that is, users do not jump forward or backward or conduct other actions used in Video-Cassette-Recorder (VCR) operations. For VOD systems, this assumption clearly does not hold in practice and users will conduct random access operation. Hence, our designed caching scheme should take random access into account.

In this paper, we propose a Flexible-Segmentation-Jumping-Strategy (FSJS). This strategy pays attention to both the initial delay and user's random access behavior. Through dividing the BaseSegment and flexibly selecting the jumping point, FSJS fully considers the factor of user operation behaviors in VOD systems, significantly reducing user's average waiting time, and, in most cases, eliminating the waiting time completely.

In order to study the performance of FSJS, in our simulation study, we incorporate FSJS with the following two basic strategies that are currently widely used in VOD systems: uniform segmentation scheme and exponential segmentation scheme. Then, we demonstrate how to use our proposed scheme to flexibly select jumping point to obtain the best performance. The simulation outcomes fully verified that the FSJS has a good performance in improving the user-perceived latency and user's satisfaction.

The rest of paper is organized as follows. Related works is presented in Section 2, Caching scheme design is presented in Section 3, and Section 4 presents Simulation and performance evaluation. We will make the conclusion in Section 5.

2. Related Works

In order to reduce user average waiting time and improve quality of service, researchers have proposed many novel strategies, such as partial cache, proxy updating algorithms, prefetching technique, and media delivery technique. Partial cache [1–4], which have been briefly introduced above, can

minimize data traffic between servers and clients. But its focus is on improving BHR, and, hence, it has little benefit in improving user-perceived latency.

Proxies are widely used in web browsing and video streaming services to decrease client waiting time. Reference [5, 6] have proposed proxy cache update algorithms to dynamically manage a proxy's cache in order to improve quality of service in streaming media and startup latency. Reference [7] describes a video staging technique which is useful in maintaining a constant bit-rate stream between a proxy and a server.

References [8–10] studied prefetching of multimedia objects. Reference [8] proposed a proactive prefetching method utilizing partially fetched data to improve the utilization of network bandwidth. In [9], the prefetching was applied to preload a certain amount of data in order to take advantage of the caching power. For layered-encoded objects, [10] studied prefetching of layered video not in cache by maintaining a prefetching window of a cached stream.

Media delivery techniques include batching [11–13] stream merging [14], utilize multicast [15, 16] and patching [17]. Tokekar et al. [11] presented a model to analyze proposed batching policy in view of different user reneging behaviors and came up with the optimum value of batching interval to maximize the average number of users served and at the same time minimize reneging probability. Reference [12] considered a finite-buffer batch arrival and batch service queue with single and multiple vacations. Reference [13] analyzed a discrete-time two-phase queuing system, which receives batch service in the first queue and individual service in the second queue. It analyzed the queue length and the effect of batch size on the waiting time. However, it is not easy to decide the batching interval and size. Reference [14] discussed three main stream merging techniques: Patching, Transition Patching, and ERMT. It also presented two alternative implementations of MCF (MCF-T and MCF-P), and showed that MCF achieves significant performance benefits in terms of both the number of requests that can be served concurrently and the average waiting time for service. Reference [15] addressed a relevant optimization problem, and proposed a hybrid transmission scheme to tackle channel allocation problem. This scheme determines the most suitable delivery technique for each video and the appropriate number of channels to be allocated to the video using a dynamic programming approach. [16] evaluated and discussed the recent progress in developing multicast VOD systems, and studied QoS, fairness of multicast VOD server, and custom behavior. But it is hard to develop standard protocols for multicast VOD for practical applications. In terms of patching, [17] proposed a PeriodPatch, which only uses 38% streams of FIFO or 50% of Patching schedule to provide the same TVOD service.

3. Caching Scheme Design

3.1. Basic Design Idea. Since media files used in a VOD system are very large, it is impractical to cache all streaming media in a limited caching space. Thus, it is a good idea to

use partial caching scheme instead. First, we divide the media file into W segments. Each segment is called a BaseSegment and has a size of M . In order to reduce the initial delay and guarantee to have a relatively high BHR, when a user first requests a new media file, our strategy will cache the first BaseSegment unconditionally if the caching space has enough size to cache this segment.

Second, except the first BaseSegment, we divide each of the remaining BaseSegments into two parts: Prefix and Suffix. Since the caching space is limited, our caching scheme will only cache the Prefix of each segment as shown in Figure 2.

With the same caching capacity, this strategy allows us to cache more segments, both the total number of segments and the number of same media file's segments. The number of same media file's segments affects the user-perceived latency. However, since VOD services allow users to jump forward or backward randomly, these actions will still cause users to wait for some time. A user with little patience can still give up seeing such a movie. To deal with this challenge, we modify our scheme in the following way: if users' play in Prefix part which has been cached, the user can see the content without waiting, else in the Suffix part that not in cache, we do not immediately tell the server to send the frames to users; alternatively, we flexibly select the jumping point, like the Figure 3.

If a user plays in S_1 , he or she can watch the movie immediately; if the user's request point is in S_2 , our scheme will simply let the request jump to S_3 or S_4 (which one point to jump is determined by the length of S_2-S_3), and at the same time request the remote server to download the unavailable Suffix part. In a round trip time delay, when the user still watches in the Prefix part, the Suffix part has been continuously forwarded to the client, which leads to an uninterrupted playout at the user side. If we limit the size in S_3-S_2 or S_4-S_2 , this minor jumping will not disturb the user too much, and the user will no observe the difference. However, the jumping strategy brings us the benefit of almost no waiting time for a user when the user jumps forward or backward to see a specific content.

Compared with the traditional cache, like the uniform segmentation, exponential segmentation, or adaptive and lazy segmentation scheme, and so forth, FSJS's most striking feathers lie in the segmentation strategy and jumping strategy. One is that in traditional cache, the basic unit is BaseSegment, but in FSJS, which divides the BaseSegment to Prefix and Suffix, made the basic unit smaller, that eventually benefits for us to do better caching strategy. Another is that traditional caches in initial delay are very high; however, FSJS using the jumping strategy can let the users obtain almost zero waiting time in most cases. With the increase of server load (by increasing the number of VOD users), the FSJS's advantage becomes more obvious, as more and more users will be pulled to the same segment in cache, which definitely will increase the BHR and reduce the server load.

If a user's jumping point is in S_5 , the system will only have the previous cached Prefix part without the subsequent cached Prefix part. In this situation, the request point will be pulled to S_6 . However, if the length of S_5-S_6 is large enough

FIGURE 2: Segmentation strategy.

FIGURE 3: Jumping strategy.

to affect the user's normal watching, how to deal with such situation and size should be discussed.

FSJS can reduce a user's waiting time; however, in some cases, it may cause the remote server to introduce an extra serving time. In Figure 3, if the user's request point is in S_2 , and assuming that in our scheme it jumps to S_3 , the user can immediately see the video but he or she needs an extra time to download video streaming in the size of C from the remote server when reaching the first request point S_2 , which definitely will increase the server load and influence the server performance.

3.2. Analyzing the Model. Considering the basic structure of VOD system as shown in Figure 1, it consists of N heterogeneous servers and a disk array. Let the set $S = \{s_1, s_2, \dots, s_N\}$ denote servers and each server has the capacity of c_i ($1 \leq i \leq N$) to be used as buffer to cache some popular media segments. The disk array's storage is D , and it has enough space to cache the video media. All streaming media are stored in disk array, not in the servers. Each server s_i caches L different media files L , and each media file has size j ($1 \leq j \leq L$). For each media object, its caching status is denoted as $\delta \in \{1, 0\}$, in which 1 means the media object is in cache, 0 indicates the object is no in cache. Thus, in order to maximum the BHR and space utilization, the caching scheme should be made as

$$\begin{aligned} \max \sum_j \text{size}_j \times \delta &\leq c_i, \quad (1 \leq i \leq N, 1 \leq j \leq L), \\ \sum_i \sum_j \text{size}_j \times \delta &\leq D, \quad (1 \leq i \leq N, 1 \leq j \leq L). \end{aligned} \quad (1)$$

When a client requests a video media, the time of the request from the client to server is denoted as t_{cs} , the retrieval time of the media file in server is denoted as t_r , and the media's transportation from the server to client is denoted as t_{sc} . Assuming the network delay between server and client is t_d , such as network congestion, that can affect a user's waiting time. In such situation, each client's waiting time when playing a video can be defined as follows:

$$\text{user's waiting time} = t_{cs} + t_r + t_{sc} + t_d. \quad (2)$$

It is obvious that request from client to server needs far smaller time compared to the retrieval and the data transmission time from the server to client. Thus, t_{cs} in our calculation can be neglected. Simply, in our simulation, assume the network delay $t_d = 0$. Our scheme's objective can be formulated as

minimize user's waiting time $= t_r + t_{sc}$,

$$\begin{aligned} \text{subject to : } & \max_j \sum_j \text{size}_j \times \delta \leq c_i \\ & (1 \leq i \leq N, 1 \leq j \leq L), \quad (3) \\ & \sum_i \sum_j \text{size}_j \times \delta \leq D \\ & (1 \leq i \leq N, 1 \leq j \leq L). \end{aligned}$$

In Figure 2, suppose the size of the BaseSegment is M and the Prefix size is Lp ; thus, the percentage of Prefix, denoted as PerPrefix, is equal to $Lp/M * 100\%$, ($Lp \leq M$). The value of PerPrefix, which affects the BHR or other performances, will be studied through our simulation experiments.

Suppose we define $S2-S3 = L23$, $S4-S2 = L42$, and $S5-S6 = L56$. If a jump point has both previous Prefix and subsequence Prefix part, like $S2$, $L23$ is less than or equal to half of the BaseSegment or $L42$ more than half of BaseSegment, in our strategy, we will let the request point $S2$ jump to $S3$, and vice versa jump to $S4$. If the point only has the previous Prefix, like the $S5$, when $L56$ less than or equal to BaseSegment, we think it will not disturb the user too much, then jump to $S6$. But when $L56$ larger than BaseSegment, the users may notice the difference, so it should not jump, but stay in $S5$. The jumping point $S2$, $S5$ can be defined as follows:

$$\begin{aligned} S2 &= \begin{cases} S3, & L23 \leq \frac{M}{2} \quad \text{or} \quad L42 > \frac{M}{2}, \\ S4, & L23 > \frac{M}{2} \quad \text{or} \quad L42 \leq \frac{M}{2}, \end{cases} \quad (4) \\ S5 &= \begin{cases} S6, & L56 \leq M, \\ S5, & L56 > M. \end{cases} \end{aligned}$$

From above, if user's play in $S1$, $S2$, $S5$ ($L56 \leq M$), through select appropriate jumping point, $t_r + t_{sc} \approx 0$, users can obtain the best satisfaction. Only in $S5$ ($L56 > M$), users need to wait before the frames were sent from the servers to clients.

In the basic design as shown in Figure 3, the extra download size C is an important factor that needs to be studied. In this paper, we use Extra Average Serving Time (EAST) to describe the extra waiting time for downloading of C . EAST only happens when the system selects the previous point of the current request point, and selecting the next point or stand still will not exert EAST. Thus, in each media file i , $EAST_i$ can be simply calculated as

$$EAST_i = \begin{cases} 0, & S1 \quad \text{or} \quad S2 \rightarrow S4 \quad \text{or} \quad S5, \\ \frac{C}{R_i}, & S2 \rightarrow S3 \quad \text{or} \quad S5 \rightarrow S6, \end{cases} \quad (5)$$

where R_i represents the average transmitting rate between the servers and clients. $S2 \rightarrow S4$ means pulling the request point $S2$ to $S4$.

3.3. Simulation Strategy. Because FSJS is just a segment-divide strategy, it has no utility function. New media segment has to be added and the popularity of the segment also changes over time, which means, the caching segment should be updated periodically and we need to evict the smallest value segment. In order to simply show the merits of FSJS and also find out which caching scheme has the smallest EAST, we only use FSJS to update uniform segmentation and exponential segmentation algorithm.

3.3.1. FSJS Updating Uniform Segmentation. We cache the first BaseSegment when a media file is first requested and increase only one Prefix part for the next been requested. Then, we use LRU (Least Recently Used) replacement to decide which segment should be evicted from the cache. There are two eviction strategies assuming that we have found the media file which has the smallest utility value. (1) if the smallest utility value's media file has some Prefix in cache, our eviction only drop the last Prefix part; otherwise, we drop the BaseSegment. (2) we drop all segments of this selected media in cache, including all Prefix parts and the BaseSegment. The first strategy calls UniSeg (FSJS) and second one UniSeg (FSJS)-D.

3.3.2. FSJS Updating Exponential Segmentation. Similarly, we can form eviction strategies called ExpSeg (FSJS) and ExpSeg (FSJS)-D for using FSJS to update the exponential segmentation. Exponential segmentation scheme still uses its own utility function [2], $\varphi = 1/((T_c - T_i) \times i)$ (in which T_c is the current time, T_i is the last time to access the media segment, i is the number of the segment) and the difference is that the Prefix will increase double each time when the media file been requested. Right now, the form of FSJS updating the basic caching scheme has been discussed, but we have not known the ratio of the Prefix to BaseSegment.

4. Simulation and Performance Evaluation

4.1. Finding the Best Percentage of Prefix to BaseSegment. Our simulation uses 10 days' VOD log captured from our university's campus network to a VOD server. The log recorded all users' VCR operations in detail. The server ip is 202.38.64.88, and the number of media files is 7424. Considering each VCR operational jump as an independent play, the total play count is 279,765, the total media size is 2, 572, 949, 855, 454B, and the average flow size is 169, 175, 372B, the relationship of file popularity can be drawn in Figure 4.

In Figure 4, it shows the ratio of current size to total media size and the ratio of current flow size to total flow size, we can see from the figure that almost the first 1300 media files account the 20% of the total media size, but hold the 78% of the total flow size, thus the popularity of individual requests taken from a "80/20" laws.

FIGURE 4: The relationship of file popularity.

As the percentage of Prefix to BaseSegment significantly affects BHR and server's additional serving time, we need to find the best ratio of Prefix/BaseSegment. In our simulation, we assume BaseSegment = 2 MB, Caching Size = 30 GB, $R_i = 1$ Mbps. Our objective is that in the same caching capacity cache as many media file segments as possible, thus PerPrefix should be no bigger than 50%. At the same time, since a small ratio means low BHR, we define PerPrefix should be larger than 10%. The relationship about the PerPrefix, BHR, and EAST is shown in Figure 5 when using FSJS to update the uniform segmentation.

In Figure 5, the value of BHR increases when we increase the value of PerPrefix. This is what we want, since a higher BHR means a lower server load. But at the same time, the EAST also increases, which is not what we want, as it will bring an extra serving time. Next, in the same way like the UniSeg(FSJS)-D, Figure 6 shows the simulation results when using FSJS to update the exponential segmentation.

Figure 6 shows that ExpSeg(FSJS)-D has a similar performance like the UniSeg(FSJS)-D. When the value of PerPrefix increases, both BHR and EAST rise, but at a different rate at each PerPrefix. In order to find the best operation setting, which means in such PerPrefix, the BHR is not low, the EAST also not high, but their combination have the best satisfaction when used in VOD system. Thus, we have to compromise in BHR and EAST to find the best percentage of Prefix to BaseSegment. Using the following equation as a standard criterion:

$$\text{Satisfaction} = k \times \frac{\text{BHR}^p}{\text{EAST}^q} \times 100\%, \quad (6)$$

where Satisfaction is the percentage of $k \times \text{BHR}^p$ to EAST^q , k is a constant, as a control factor which can minor adjust system in order to achieve the expected result, p and q larger than zero are decided by different situation of VOD system. If the designer is more concerned with the server load, he or she can choose p larger than q ; if the designer cares more

(a)

(b)

FIGURE 5: UniSeg(FSJS)-D.

about the user waiting time, p should be chosen less than q . In our simulation, we think both BHR and EAST are important; thus, we choose $k = 1$ and $p = q$. Figure 7 shows the simulation results of BHR/EAST.

With the percentage of Prefix increasing, UniSeg(FSJS)-D and ExpSeg(FSJS)-D have different BHR/EAST, and the higher value of BHR/EAST, the more satisfaction a user will get. Thus, in Figure 7, it is easier to find the best point of UniSeg(FSJS)-D is PerPrefix = 15%, and ExpSeg(FSJS)-D's best point is PerPrefix = 35%.

4.2. Comparison in Extra Average Serving Time. Now, we have found the best percentage of Prefix to BaseSegment, next we use PerPrefix = 15% in UniSeg(FSJS) and UniSeg(FSJS)-D to compare with uniform segmentation and PerPrefix = 35% in ExpSeg(FSJS) and ExpSeg(FSJS)-D to compare with exponential segmentation in EAST, respectively. The simulation results are shown in Figures 8 and 9.

Figure 8 shows how using FSJS to update uniform segmentation affects the EAST. With the increase of the caching size, EAST decreases. But the extra serving time

(a)

(b)

FIGURE 6: ExpSeg(FSJS)-D.

FIGURE 7: Finding the best point.

FIGURE 8: Prefix/BaseSegment = 15%.

FIGURE 9: Prefix/BaseSegment = 35%.

almost stops decreasing in 820 ms when using the uniform segmentation strategy and the caching size becomes larger than 20 GB. Using FSJS, it continues to decrease in EAST with the expanding of the caching size. The reason is that when applying FSJS, with the same caching space, the system can cache more Prefix segments and reduces the EAST through selecting forward or backward jumping point in a more effective way. At the same time, when the caching size is less than or equal to 20 GB, UniSeg(FSJS) has a superior performance than the UniSeg(FSJS)-D, in other conditions, UniSeg(FSJS)-D exhibits better performance. Due to using UniSeg(FSJS) and UniSeg(FSJS)-D, their results in EAST are slight difference. However, when the caching space is full, UniSeg(FSJS) only evicts the smallest utility functional

FIGURE 10: Two best strategies' comparison.

Prefix part, which needs more iteration when it has enough space to cache the next new BaseSegment. In UniSeg(FSJS)-D, it drops all the smallest utility functional segments, only needs one eviction that can release enough space, therefore, UniSeg(FSJS)-D saves more time in replacing the segments.

Figure 9 shows the performance in terms of EAST when using FSJS to update exponential segmentation. ExpSeg(FSJS) is better than ExpSeg(FSJS)-D, at least saving 180 ms in average in EAST. The worst performance is exponential segmentation, which needs nearly more than 220 ms in average in EAST to the ExpSeg(FSJS)-D and 400 ms to the ExpSeg(FSJS). ExpSeg(FSJS)-D is superior to the ExpSeg(FSJS) in the replacing moment, ExpSeg(FSJS)-D needs less time to replace segments compared to the ExpSeg(FSJS). However, in exponential segmentation scheme, it evicts twice segments of the previous each time. In this way, it can quickly release enough space to cache new segments. So the difference between the ExpSeg(FSJS)-D and ExpSeg(FSJS) is less than the difference between UniSeg(FSJS)-D and UniSeg(FSJS) in terms of evicting time. Thus, in our simulation, ExpSeg(FSJS) is better than ExpSeg(FSJS)-D.

UniSeg(FSJS)-D (PerPrefix = 15%) is the best caching scheme when using FSJS to update uniform segmentation, and ExpSeg(FSJS) (PerPrefix = 35%) is the best caching scheme when applying the FSJS to exponential segmentation. As shown in Figure 10, it is easier to see that ExpSeg(FSJS) is superior to the UniSeg(FSJS)-D in EAST, especially the caching size is small, it almost saves 200 ms in EAST when caching size is 5 GB. With the caching size increasing to 30 GB, the UniSeg(FSJS)-D almost stops in 180 ms, but the ExpSeg(FSJS) can still decreasing, which nearly decrease to 120 ms in EAST when the caching size reach to 50 GB. So far, we have found that the best caching scheme using FSJS is ExpSeg(FSJS), and the PerPrefix is 35%. In such condition, we can achieve the smallest Extra Average Serving Time.

5. Conclusion

In this paper, we analyze the current exiting problems using traditional caching strategies, and propose a Flexible-Segmentation-Jumping-Strategy (FSJS). We use FSJS to update uniform segmentation and exponential segmentation. Based on the initial delay, the server load, the factor of user randomly access behavior, we developed four different strategies called UniSeg(FSJS), UniSeg(FSJS)-D and ExpSeg(FSJS), ExpSeg(FSJS)-D. We also found the best point of Prefix/BaseSegment. We compared the four proposed strategies, and found that FSJS not only can cut down a user's waiting time to zero in most cases, but also can reduce a server's extra serving time. Among the four strategies, ExpSeg(FSJS) can achieve the best performance. We plan to use ExpSeg(FSJS) in our future real VOD project. Our simulation results showed that FSJS can significantly reduce user-perceived latency.

Acknowledgment

This paper is sponsored by the National High Technology Research and Development Program of China (no. 2008AA01A317), the AnHui Colleges & Universities Provincial Science Research Project (no. KJ2008A106), and the Fundamental Research Funds for the Central Universities (no. WK2100100006).

References

- [1] K.-L. Wu, P. S. Yu, and J. L. Wolf, "Segment based proxy caching of multimedia streams," in *Proceedings of the 10th International Conference World Wide Web (WWW '01)*, pp. 36–44, ACM, Hong Kong, 2001.
- [2] S. Chen, B. O. Shen, S. Wee, and X. Zhang, "Adaptive and lazy segmentation based proxy caching for streaming media delivery," in *Proceedings of the 13th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '03)*, pp. 22–31, Monterey, Calif, USA, June 2003.
- [3] S. Chen, B. O. 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.
- [4] S. Chen, B. Shen, S. Wee, and X. Zhang, "Investigating performance insights of segment-based proxy caching of streaming media strategies," in *Proceedings of ACM/SPIE Conference on Multimedia Computing and Networking*, vol. 5305, pp. 148–165, San Jose, Calif, USA, January 2004.
- [5] M. Hofmann, T. S. N. Eugene, K. Guo, S. Paul, and H. Zhang, "Caching techniques for streaming multimedia over the internet," Tech. Rep. BL011345-990409-04TM, Bell Labs, Holmdel, NJ, USA, 1999.
- [6] E. Bommaiah, K. Guo, M. Hofmann, and S. Paul, "Design and implementation of a caching system for streaming media over the internet," in *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS '00)*, Washington, DC, USA, June 2000.
- [7] Y. Wang, Z. L. I. Zhang, D. H. C. Du, and D. Su, "A Network-conscious approach to end-to-end video delivery over wide area networks using proxy servers," in *Proceedings of the*

- 17th Annual IEEE Conference on Computer Communications (INFOCOM '98)*, pp. 660–667, April 1998.
- [8] J. Jung, D. Lee, and K. Chon, "Proactive web caching with cumulative prefetching for large multimedia data," in *Proceedings of the 9th International World Wide Web Conference (WWW '00)*, Amsterdam, The Netherlands, May 2000.
 - [9] J. I. Khan and Q. Tao, "Partial prefetch for faster surfing in composite hypermedia," in *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, San Francisco, Calif, USA, March 2001.
 - [10] R. Rejaie, H. Yu, M. Handley, and D. Estrin, "Multimedia proxy caching mechanism for quality adaptive streaming applications in the internet," in *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '00)*, pp. 980–989, Tel-Aviv, Israel, March 2000.
 - [11] V. Tokekar, A. K. Ramani, and S. Tokekar, "Analysis of batching policy in view of user reneging in VOD system," in *Proceedings of the International Conference of IEEE India Council (INDICON '05)*, pp. 399–403, Chennai, India, December 2005.
 - [12] K. Sikdar and U. C. Gupta, "On the batch arrival batch service queue with finite buffer under server's vacation," *Computers & Mathematics with Applications*, vol. 56, no. 11, pp. 2861–2873, 2008.
 - [13] S. Saha and A. S. Alfa, "Selecting batch size in discrete-time two-phase queuing system," *Mathematical and Computer Modelling*, vol. 47, no. 11-12, pp. 1246–1253, 2008.
 - [14] N. J. Sarhan and B. Qudah, "Efficient cost-based scheduling for scalable media streaming," in *Multimedia Computing and Networking*, vol. 6504 of *Proceedings of SPIE*, San Jose, Calif, USA, January 2007, C5040.
 - [15] S. A. Azad and M. Manzur, "An efficient transmission scheme for minimizing user waiting time in video-on-demand systems," *IEEE Communications Letters*, vol. 11, no. 3, pp. 285–287, 2007.
 - [16] H. Ma and K. G. Shin, "Multicast video-on-demand services," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 1, pp. 31–43, 2002.
 - [17] Z. Xiang, Y. Z. Zhong, and S. Q. Yang, "PeriodPatch: an efficient stream schedule for video-on-demand," in *Internet Multimedia Management Systems*, vol. 4210 of *Proceedings of SPIE*, pp. 341–347, Boston, Mass, USA, 2000.