World Wide Web: Internet and Web Information Systems, 7, 297–314, 2004

Designing Efficient and Robust Caching Algorithms for Streaming-on-Demand Services on the Internet

K. Y. LEUNG, ERIC W. M. WONG and K. H. YEUNG
{kyleung,ewong}@ee.cityu.edu.hk, eeayeung@cityu.edu.hk
Department of Electronic Engineering, City University of Hong Kong, Tat Chee Avenue, Hong Kong SAR, China

Abstract

Content Delivery Networks (CDN) have been used on the Internet to cache media content so as to reduce the load on the original media server, network congestion, and latency. Due to the large size of media content compared to normal web objects, current caching algorithms used in the Internet are no longer suitable. This paper presents a high-performance prefetch system that accommodates user time-varying behavior. A hybrid caching technique, which combines prefetch and replacement algorithms, is also introduced. The robustness of the cache system against imperfect user request information is evaluated using three request noise models. Two prefetch performance indices are also presented to help content administrators in deciding when to update the user request profile for caching algorithms.

Keywords: content delivery networks, streaming-on-demand services, caching algorithms, time-varying behavior, prefetching

1. Introduction

With the rapid development of the broadband Internet, streaming-on-demand services are becoming more prevalent. Similarly to Web content, the caching of streaming content between servers and clients can improve the efficiency of distributing multimedia objects to clients. The benefits of caching are likely to be especially great for streaming media owing to the large file sizes involved.

In the design of content management for video cache servers, two observations from traditional Video-on-Demand (VOD) systems are very relevant. The first observation is that different age groups of users generally access different types of video in different time periods. We refer to this user behavior as time varying behavior (TVB). Therefore, the caching strategy must also be time varying. The second observation is that video requests usually focus on a limited number of popular videos. Therefore, caching small amounts of the most popular videos is enough to serve most of the requests. This makes the design of the prefetch algorithm to cache the whole video for cache systems with limited resources become practical. As a result, a TVB-aware prefetch algorithm can be designed to prefetch videos to optimize the usage of cache space.

Although TVB-aware prefetch algorithms can greatly increase the performance of distributed server systems, user request patterns are of course sometimes unpredictable and the prefetch algorithm may therefore mistakenly cache unpopular videos. For example, programs that were popular in the past may become unpopular in the future. A caching algorithm is said to be robust if it can maintain good performance automatically in the face of changing and uncertain user behavior.

The objective of this paper is the design of efficient and robust caching algorithms for streaming-on-demand services on the Internet in TVB environments with or without perfect user behavior information. One prefetch algorithm and two cache replacement algorithms are presented. The robustness of the proposed algorithms is evaluated using three noise models that capture unpredictable/unknown user behavior under various situations. In addition, two measurement indices, prefetch quality and prefetch quantity, are presented and their potential use for describing and predicting the performance of prefetch algorithms on a cache server is discussed.

The paper is organized as follows. In Section 2, a TVB-aware prefetch algorithm is proposed for a distributed video server system. In Section 3, two cache replacement algorithms are proposed to further enhance the system performance. In Section 4, three noise models are introduced to evaluate the robustness of the proposed cache algorithms. In Section 5, two measurement indices, prefetch quality and prefetch quantity, are presented. Finally, the conclusions are summarized in Section 6.

2. TVB-aware prefetch algorithm

2.1. Time varying behavior

The time-varying behavior and program preferences of users can be characterized by demographic segment (e.g., according to whether the users are children, college students, adults, men, women, unemployed or elderly). Typically, users belonging to the same demographic segment have similar viewing schedules and similar viewing preferences. Carroll and Davis [4] discuss the dependence of television viewing behavior and audience demographic segment on time of day. Table 1 summarizes the core TV audience for the five dayparts described in [4]. It can be seen that different dayparts are associated with specific demographic groups.

It might be expected that Internet video users have similar user behavior with TV users. In order to verify this conjecture, the behavior of typical Web users was studied. Figures 1 and 2 show the results obtained by analyzing a 14-working-days proxy log file obtained from CA*netII [2]. From the proxy log, we selected the four most frequently requested web sites and analyzed their access patterns. These Web sites are grouped into two classes:

Table 1. Core audience for five dayparts.

TV dayparts	Time	Core audiences
Early morning	7 a.m.–9 a.m.	Adults (awake early), children
Daytime	9 a.m.–4 p.m.	Women, elderly, unemployed
Early fringe	4 p.m.–8 p.m.	Children, men, working women
Prime time	8 p.m.–11 p.m.	Adults
Late fringe	11 p.m.–1 a.m.	College students, adults (awake later)

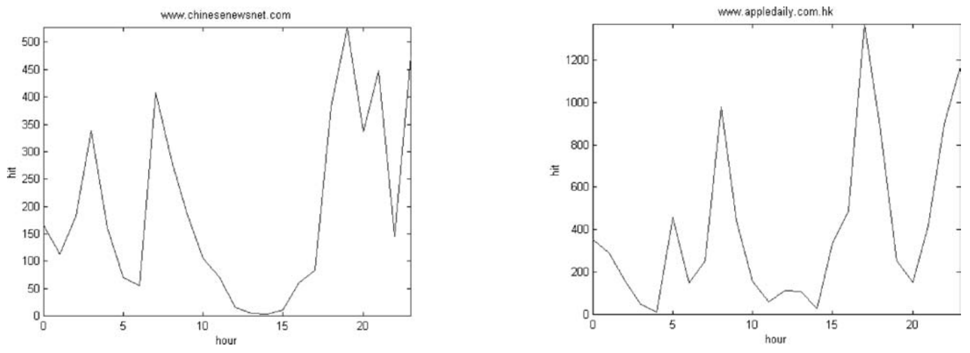

Figure 1. Access patterns for news type Web sites.

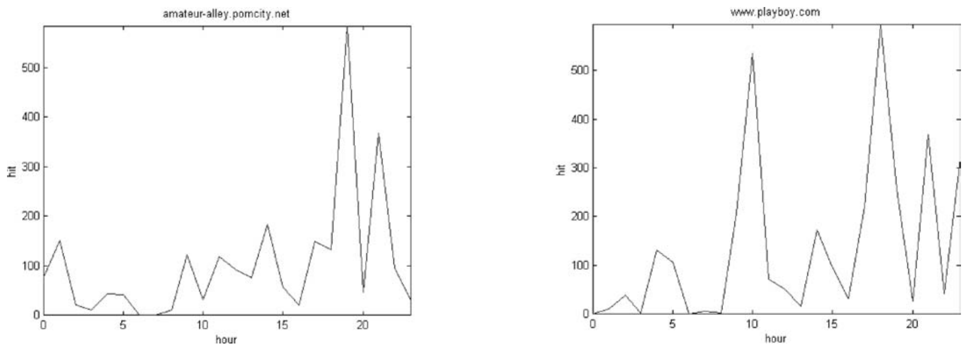

Figure 2. Access patterns for adult type Web sites.

news and adult content. From the figures, it can be seen that both classes of web site have quite similar access patterns/dayparts. This suggests that dayparts also exist for Internet users.

2.2. A TVB-aware prefetch algorithm

Conventional caching algorithms in Content Delivery Networks (CDNs) manage the cache content by prefetching the most popular objects on a daily, weekly or monthly basis, which depends on the frequency with which the original contents are updated and the preferences of the administrator. These algorithms work well for small Web objects because surrogates have sufficient resources to simply cache all of the most popular Web objects. They also work well when the cache content remains unchanged for several days or even weeks because the popularity of the content changes only slowly. However, previous studies [1,6,12,15,16] have highlighted the significant penalty incurred with streaming media when a system wrongly prefetches a large but unpopular video object since such a wrong decision wastes valuable bandwidth and cache space. Clearly, the potential exists for designing more efficient prefetch algorithms by taking better account of the time-varying nature of video popularity.

We assume that video content is categorized into a number of classes and consider a TVB-aware prefetch algorithm that divides the day into a number of dayparts. In each daypart, the most popular videos from the class associated with that daypart class are prefetched first. The dayparts are derived by dividing the day into small time intervals (typically 30 minute intervals). These intervals are aggregated into contiguous dayparts based on the popularity distribution of the identified video classes (for each short time interval the popularity of each video class may be determined from the server access histories).

2.3. Simulation results

We compare simulation results for a cache system using a conventional prefetch algorithm and one using a TVB-aware prefetch algorithm. For the conventional prefetch algorithm, videos are prefetched before the simulation starts and cached until simulation ends. The TVB-aware prefetch algorithm prefetches videos before the start of each daypart according to the access profile in Figure 3. This access profile is also used to determine the user requests generated in the simulation. The access frequencies of the videos in each class are assumed to follow the Zipf distribution with parameter 0.271 [9].

For the TVB-aware prefetch algorithm, each daypart is assigned a parameter called the *prefetch time*. This is the time allocated to prefetching videos before the start of a daypart. Simulation results indicate that small changes in this parameter have little effect on performance as long as it is not too small. During the prefetch time, videos with the higher popularity in the upcoming daypart are prefetched first. Once a decision has been made to prefetch a video, a currently cached video with lower popularity in the upcoming daypart is discarded to release the cache space needed by the new video. This cache replacement operation continues until all of the desired videos are prefetched or the start of the next daypart is reached. For example, assume the prefetch time is 30 minutes. When a time section in the prefetch profile starts at 6:30 a.m. and assume all requests belong to news class. The prefetch operation will start at 6:00 a.m. and it shall prefetch the most popular news videos.

The cache size and stream size capacity of the cache server used in our simulations are based on the Cisco content engine CE-7320-ICDN [7]. In the simulations, we assume that all videos are 90 minutes duration and in the MPEG1 format. Each 90-minute 1.5 Mbps MPEG1 video is about 1 GB in size. With this setup the CE-7320-ICDN can hold 360 videos and provide 250 concurrent video streams for retrieval of MPEG1 videos. Table 2 summarizes the system parameters we used in our simulation.

The simulation results are presented in Figure 4. It can be seen that with a cache size of 50 videos the TVB-aware algorithm achieves a 35% improvement over the conventional algorithm while a 40% improvement is achieved when the cache size is increased to 500 videos. It can also be seen that the cache size required for both systems to achieve the same performance is very different. For example, the TVB-aware prefetch algorithm

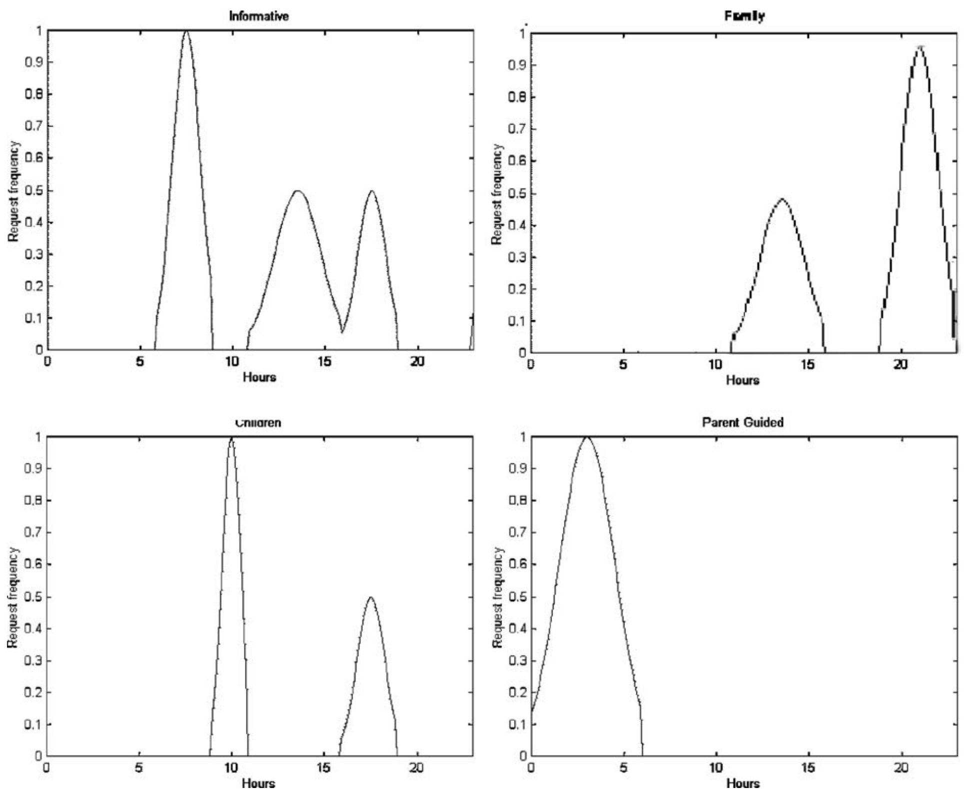

Figure 3. Access profile for the four video classes used in simulation.

Table 2. Simulation parameters and descriptions.

Parameters	Descriptions	Default values
Video length	All videos are assumed to have same length	90 minutes
Number of types	Video types exists in the original server	4
Sample size	Number of videos exists for each video genera	1000
Total sample size	Total number of videos can be requested (number of types $\times$ sample size)	4000
Stream size	Number of video streams supported by video cache system	250
Arrival rate	Maximum arrival rate of video requests per hour	150
Prefetch size	Total number of videos to be prefetch in a time session	360
Prefetch time	Time (minutes) for TVB-aware prefetch algorithm to prefetch videos before the corresponding time section reaches	30

requires 200 prefetch videos to maintain a 52% hit ratio while the conventional prefetch algorithm requires 500 prefetch videos to achieve this performance. In other words, the conventional algorithm requires more than twice the storage space of the TVB-aware algorithm to achieve the same hit ratio. Of course, these simulation results assume that

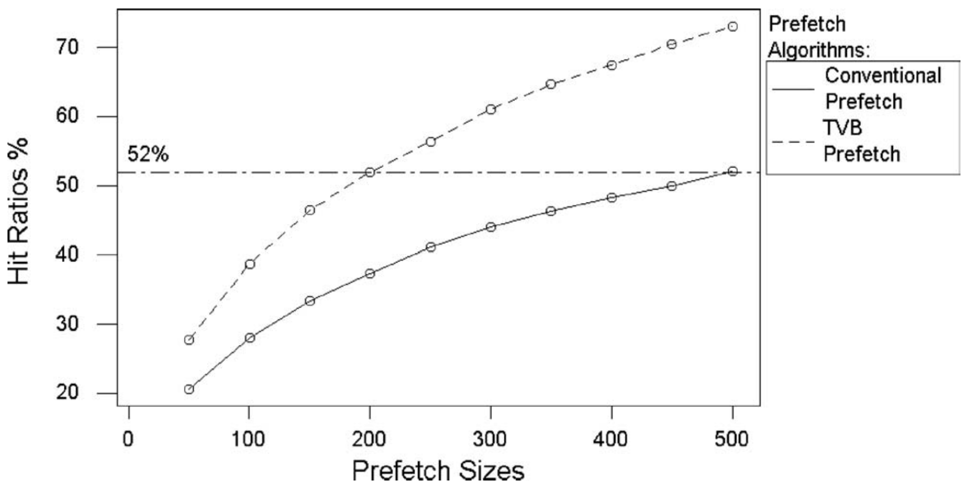

Figure 4. Performance analysis for prefetch algorithms with various prefetch sizes.

the TVB-aware prefetch algorithm has perfect knowledge of user behavior and the performance achieved by the TVB-aware algorithm can be expected to degrade as the user behavior becomes less predictable.

3. Cache replacement algorithms

3.1. Introduction

Prefetching the most frequently requested videos before users actually request them can dramatically increase the performance of cache systems. It is especially useful for distributing videos on the Internet as it can save quite a substantial amount of bandwidth while allowing users to experience high quality videos and providing faster start-up play time. However, a pure (TVB-aware) prefetch algorithm has two drawbacks. First, it is sensitive to the accuracy of the user behavior information. User request patterns are sometimes unpredictable or change rapidly, with videos that may have been popular in the past becoming unpopular in the future. The fundamental assumption of prefetching is that videos, which have been popular in the past, will remain popular in the future. If this assumption is invalid, an incorrect prefetch decision may be made. Second, a TVB-aware prefetch algorithm may require a significant amount of I/O and network bandwidth in order to provision the cache with the desired videos prior to the start of each daypart.

To address the first problem, we can use a replacement algorithm that learns new/unknown user behavior. To address the second problem, we seek to only cache specific programs at the time we really need them. This method acts like a delayed fetch algorithm. However, to be effective these approaches require a good cache replacement algorithm. In this section we therefore consider two TVB-aware cache replacement algorithms.

3.2. Design of cache replacement algorithms

To optimize the performance of a TVB-aware prefetch algorithm, prefetch and cache replacement can be used together to complement each other. We refer to this method as a combined technique. The prefetch algorithm identifies the general trends of past arrival patterns and prefetches videos before users request them. The replacement algorithm can then fine tune the cache content on the surrogate in a real-time fashion by storing current popular videos and removing unpopular videos. By combining these two techniques we can create a high performance and robust caching algorithm for CDN systems.

Previous studies have investigated the addition of a prefetch model to existing caching models in the context of file systems and the conventional Web [3,10,11,13,14,17]. However, the relationship between the prefetching and caching models is not fully discussed. These studies focus on how to implement prefetch and seldom consider the interaction between prefetching and replacement. Literature relating to the combined prefetch and replacement techniques in the context of modern CDN systems is notably lacking. To implement such a combined technique, a TVB-aware prefetch algorithm is used as in Section 2. Using historical data, popular videos are prefetched before the start of each daypart. However, some cache space is now reserved for a replacement algorithm to cache videos on demand. In this way we can cache videos whose popularity has increased relative to the historical record. Note that the replacement algorithm will not replace the videos cached by the prefetch algorithm and, of course, it will not seek to duplicate the prefetched videos. When a user makes a request, the cache is inspected to determine if it can be used to serve the user directly. If the requested video is not present in the cache, a copy of the video is fetched from the original server and sent to user. The replacement algorithm determines whether this video copy is also added to the cache server.

Previously studied cache replacement algorithms include the least recently used (LRU) and least frequently used (LFU) algorithms. However, these algorithms do not take the TVB-aware prefetch action into account. Here, we study two replacement algorithms which have TVB characteristics and work with a prefetch algorithm. We refer to them as the Absolute Popularity (AbsPop) and Virtual Partition (VirtPart) algorithms, respectively.

3.2.1. LRU algorithm In the conventional LRU algorithm, one of the most common replacement algorithms, cache content is managed by storing newly requested videos and removing those cached videos which have not been least recently accessed. This is achieved by placing each newly requested video at the top of an LRU list. When there is insufficient cache space, the cached video, which is at the bottom of the LRU list, is removed first. In this way, LRU gradually learns user behavior. This conventional LRU algorithm is used in our simulation as a baseline reference.

3.2.2. Absolute popularity algorithm The Absolute Popularity replacement algorithm (AbsPop) is based on each individual video's absolute popularity. The relative popularity of a video is the ratio between the number of accesses to the video and the total number of accesses to all videos in the same class and can be calculated from the access log. The absolute popularity of a video is the product of the popularity weighting w_{it} of its class i

in daypart t and the relative popularity P_{ix} of video x in class i. The first factor w_{it} is calculated as *access frequency of all videos in class i* divided by *access frequency of all videos*. The second factor P_{ix} is determined by analyzing a proxy log file. Note that the sum of P_{ix} over all videos in the same class i is one. When replacement is needed (i.e., a newly requested video has higher popularity than the one in the cache with the least popularity), the cached video with the least absolute popularity value is replaced by the newly requested video. We assume that the daypart affects only the popularity distribution of the class. For videos in the same class, we assume that their relative popularities do not change frequently. Therefore, the relative popularity of videos does not change with the daypart.

A disadvantage of this algorithm is that it requires quite a lot of knowledge on the popularity of videos and hence has a relatively high computational complexity. Moreover, this algorithm relies on information from past user behavior. While it can be expected to perform well if such information is accurate, if the popularity of videos has not been reviewed for a long time, such information will not up-to-date and the performance of the algorithm may degrade.

3.2.3. Virtual partition algorithm In the Virtual Partitioning replacement algorithm (VirtPart) the cache is divided into n virtual partitions corresponding to n video classes. The size of each partition is selected to be proportional to the popularity weighting w_{it} of the corresponding class i in daypart t. It is assumed that these weightings accurately reflect the ratio of the numbers of popular videos among the different video classes. In each partition, an LRU list is used to monitor the least recently used video for the corresponding class. The virtual partition for each video class is found by comparing the ratio of the *preferred cache size* to *actual cache size* for each virtual partition. Note that the *preferred cache size* is the caching target for a particular video class which can be calculated by the product of popularity weighting w_{it} and the total cache size. The *actual cache size* is the cache size currently used by that video class. The ratio indicates how well the virtual partition algorithm works. For good performance it should be as close to one as possible. A high ratio (say greater than one) means that the system currently caches the video content for that class at a rate which is less than the expected/target value. Therefore, when performing cache replacement we should cache the new video content in the partition having the highest ratio. The virtual partition with the lowest ratio is selected when it is necessary to remove an old cached video. A video is chosen from the LRU list of this virtual partition and is removed, i.e., the least recently used videos are replaced first within the selected class.

The VirtPar algorithm makes use of predictable TVB characteristics such as video class popularity in different dayparts while ignoring less predictable TVB characteristics such as the relative popularity of individual videos (in a particular video class). The philosophy behind this is that in a particular time section during a day the popularity of video class is much more stable and reliable than the popularity of individual videos.

A disadvantage of the VirtPar algorithm is that when accurate request statistics are available it may not perform as well as the AbsPop algorithm. However, VirtPart requires only the information on the popularity of video classes, and not the popularity of each video.

Therefore, the VirtPart algorithm is easier to implement, less computationally intensive than AbsPop and may be more robust when only partial knowledge of user behavior is available (see Figure 7).

4. Imperfect user behavior

4.1. Overview

Most recent VOD models rely on results reported in two studies: Chervenak [5] and Dan, Sitaram and Shahabuddin [9]. These studies examined statistics in magazines for video rentals and reports from video store owners. Both studies concluded that the popularity distribution of video titles could be fitted to a Zipf distribution. However, user behavior may change after some time and so this "ideal" request model may not be appropriate for use in evaluating the performance of cache servers. In order to test the robustness of a caching algorithm, we propose a number of non-ideal models. These are (i) an individual request noise model, (ii) a group request noise model, and (iii) an unidentified request noise model. The individual request noise model describes the popularity change of individual videos. The group request noise model describes the popularity changes of video classes. The unidentified request noise model describes the request arrivals of videos which have been newly added to the original server.

4.2. Individual request noise

The popularity of videos often changes with time. For example, viewers tend to request each video once only. They rarely play the same video again and again. Once most viewers have played a popular video, this video is likely to become less popular. As a result, the prefetch profile will change. We refer to this phenomenon as individual request noise. One solution is to update the prefetch profile frequently to reflect such changes. However, a frequent prefetch process will increase the load of the cache server and waste network bandwidth between the original server and the cache server. An alternative solution is to employ a cache replacement algorithm.

The individual request noise model described in this paper is controlled by a noise level parameter. The noise level defines the degree of deviation from the historical request pattern. To reduce complexity and maintain the properties of the request model, we make the following assumptions. First, the popularity of a video is assumed not to undergo large changes. That is, it is assumed that the popularity of a video clip does not increase or decrease rapidly (once a video is popular, it remains popular for several days or weeks). Second, we assume that the individual request noise only changes the popularity ranking of videos. The resulting access frequencies of all requested videos are still assumed to follow the Zipf distribution. The change in popularity ranking is calculated with the help of Zipf distribution. Namely, we let $f'(x) = f(x) \cdot (1 + \mathit{noise} \cdot p)$ where $f(x)$ is the original popularity (or Zipf distribution), $f'(x)$ is the new popularity distribution, *noise* is the noise level (which is a positive number) and p is a random variable between -0.5

and 0.5. This calculation is used to change the popularity ranking of videos only and video access frequencies continue to follow the Zipf distribution.

4.3. Group request noise

Group request noise is associated with changes in the popularity of video classes across dayparts. The duration and request frequency for each daypart can change on a daily basis. It is impossible to determine in advance accurate dayparts for all video classes every day. Moreover, the prefetch profile used by TVB-aware algorithms may have been simplified to reduce computational burden. We model this type of noise by altering the video class selection when generating requests. First, we introduce a probability (the group request noise level) for choosing a video class randomly from the set of all known classes, rather than simply selecting the class from the access profile as in the noise-free case. Within this class, a video is selected based on the Zipf distribution. Note that a group request noise level of 0 means that the class selection exactly follows the patterns found in the historical access profile, while a group noise level of 1 corresponds to a purely random class selection.

4.4. Unidentified request noise

In CDN systems, the choice of new videos to be cached can be explicitly specified by the web master of the content provider. However, the web master may not publish the new contents to the surrogate fast enough. Moreover, the popularity of the new contents may not have been estimated accurately by the content provider. Therefore, the caching algorithm in the surrogate generally has inaccurate knowledge as to the popularity of new videos. If a caching algorithm relies heavily on past request statistics, new videos may also lead to degradation in overall cache performance. We refer to these phenomena as unidentified request noise.

We model unidentified request noise as follows. Video requests are generated from two video pools. The first pool is referred to as the known video pool and contains videos with known popularity. The second pool is called unknown video pool and contains new videos for which the request statistics are unknown to the cache server. The number of unknown videos is specified as a percentage of the total number of known videos. We assume that the popularities of the unknown videos follow the Zipf distribution. In the noise-free case, a video request is always selected from the known video pool. With unidentified request noise, when a video request arrives, there is a probability, called the *unidentified noise level*, that the video is selected from the unknown video pool instead.

4.5. Simulation results

Figures 5–7 show simulation results illustrating, respectively, the impact of individual request noise, group request noise and unidentified request noise on a cache server system.

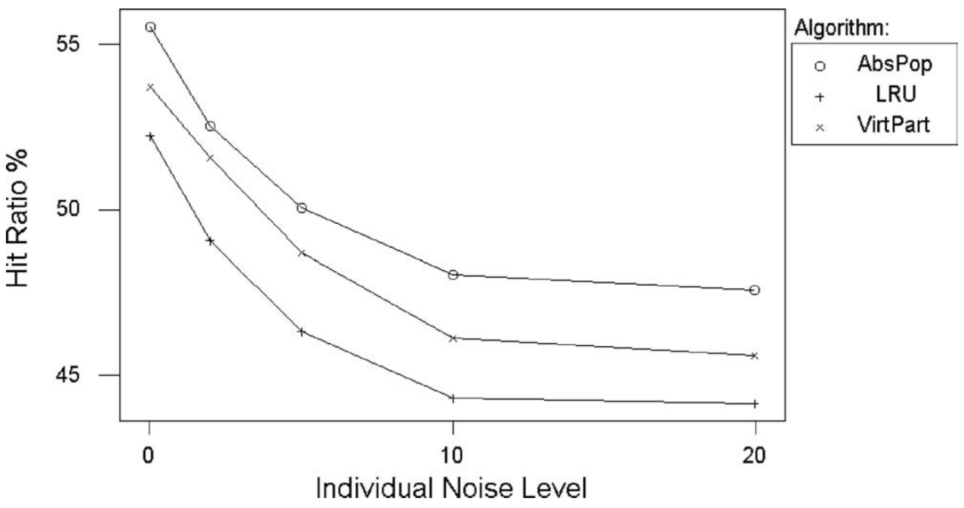

Figure 5. Performance of cache replacement algorithms with individual request noise.

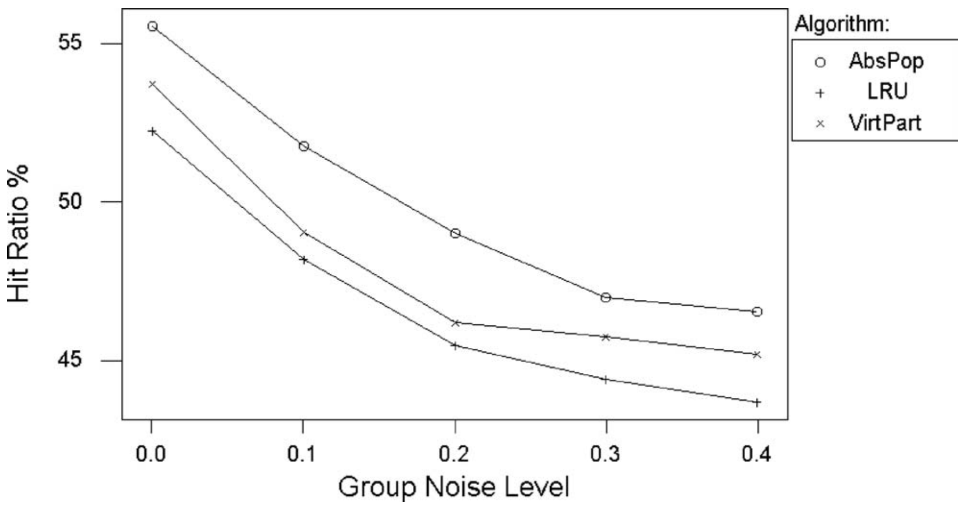

Figure 6. Performance of cache replacement algorithms with group request noise.

Of the three replacement algorithms studied, it can be seen that the AbsPop algorithm exhibits the best performance in general, followed by the VirtPart algorithm and then the LRU algorithm. However, when the level of unidentified noise is high, AbsPop can be seen to perform worse than the VirtPart and LRU algorithms because in that situation the video access characteristics are only weakly related to historical request statistics. Moreover, it can be seen that the performance of both TVB-aware replacement algorithms follows similar trends against noise level for all three request noise models.

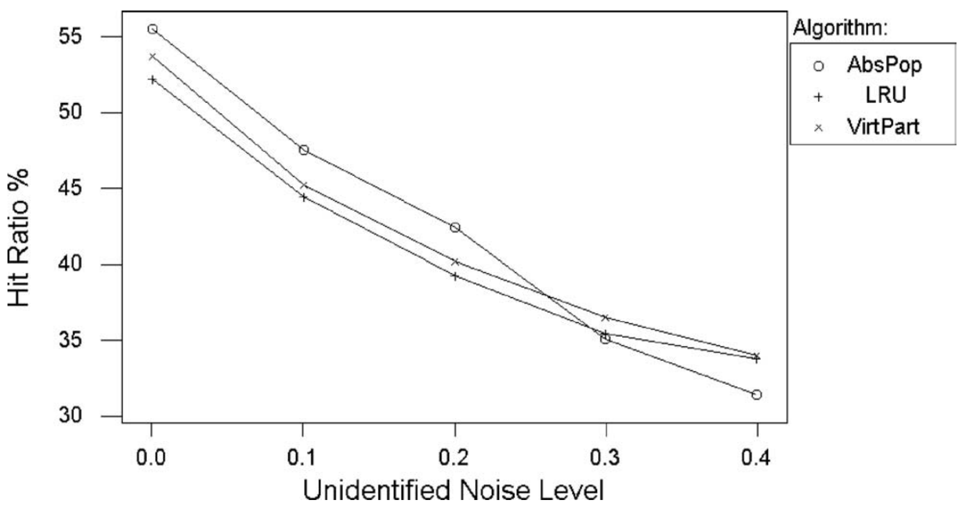

Figure 7. Performance of cache replacement algorithms with unidentified request noise.

It is interesting to note that the two replacement algorithms with learning mechanism (VirtPart and LRU) do not in general perform better than the one without learning mechanism (AbsPop). Note that in our simulations the most popular video will be on average requested 7 times per hour. For the 10th most popular video, it is requested once per hour only. If the duration of each daypart is only last few hours, it may be difficult for replacement algorithms to learn the user request profile.

4.6. Combined cache replacement and prefetch vs. pure prefetch algorithms

Figures 8–10 compare simulation results obtained using a combined TVB-aware prefetch and cache replacement technique and a pure (TVB-aware) prefetch algorithm for a variety of prefetch sizes and for user request patterns with individual request noise, group request noise and unidentified request noise, respectively. Only the replacement algorithm AbsPop is used here as the results in the previous section indicate that it in general has the best performance among the three replacement algorithms studied.

From the figures, it can be seen that the performance (hit ratio) of the pure prefetch algorithm generally improves as the prefetch size increases, regardless of noise level or noise model, as is to be expected. Note, however, that when prefetch size is very high a slight performance drop occurs because the overhead of prefetching increases with prefetch size.

For the same prefetch size and noise level, the AbsPop (combined) algorithm consistently achieves a higher hit ratio than the pure replacement algorithm, except for very large prefetch sizes. Moreover, it can be seen that the AbsPop algorithm is less sensitive to unidentified request noise than the pure prefetch algorithm, as expected. Unlike the pure prefetch algorithm (where performance generally improves with prefetch size),

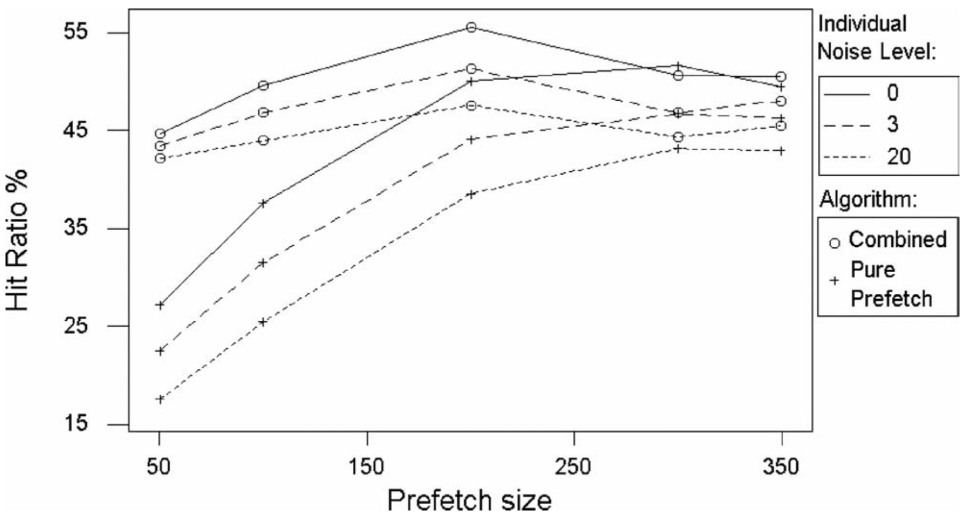

Figure 8. Simulation results with individual request noise.

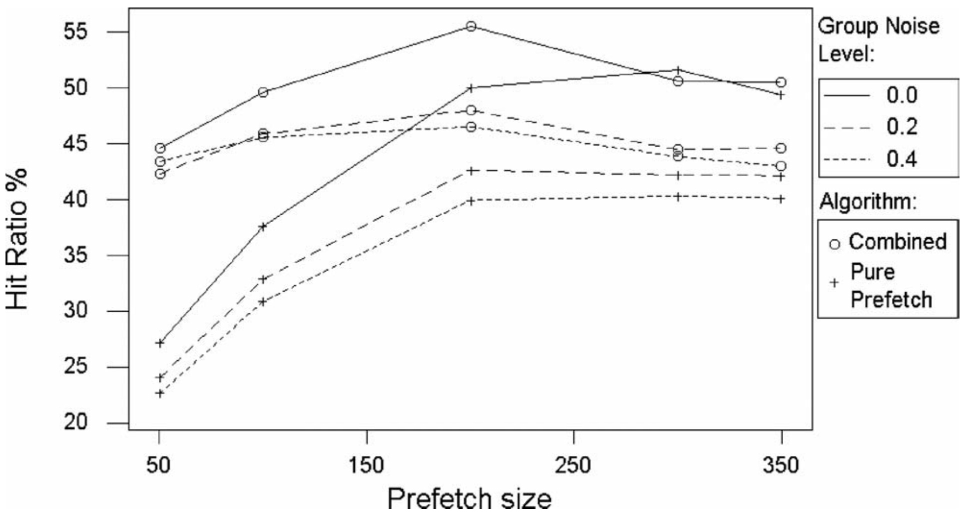

Figure 9. Simulation results with group request noise.

the AbsPop algorithm has an optimum prefetch size. Namely, regardless of noise level or type, the AbsPop algorithm achieves its highest hit ratio when the prefetch size is around 200 videos, i.e., when about half of the streaming capacity is used for content prefetching and the other half for content replacement.

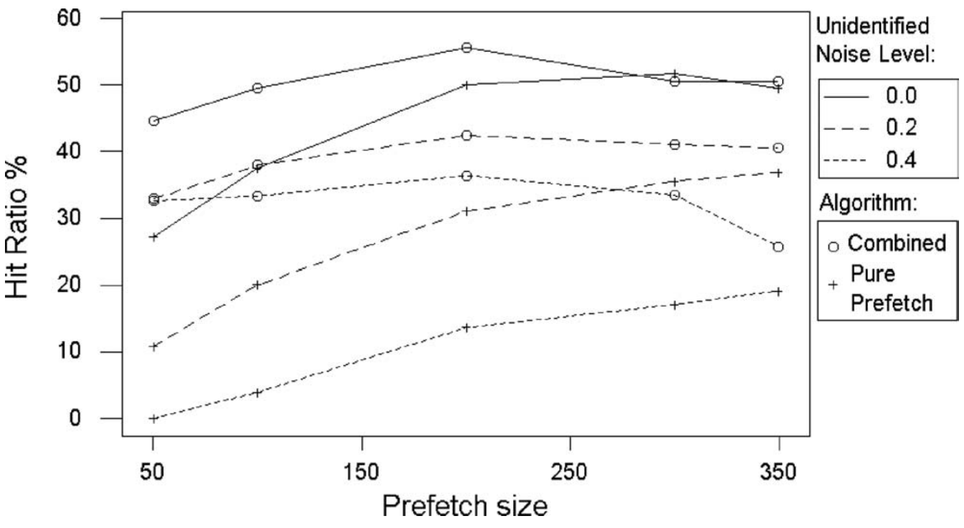

Figure 10. Simulation results with unidentified request noise.

5. Measurement indices for prefetching

To capture the performance of prefetch algorithms under different types of request noise two measurement indices, “prefetch quality” and “prefetch quantity,” are introduced.

5.1. *Prefetch quality*

Prefetch quality is defined as the summation of popularities of all the videos prefetched in the cache system. Note that the sum over all video popularities is one. In Figure 11, the curve represents the popularity of videos. This popularity is assumed to follow the Zipf distribution. In this figure, the request pattern is ideal because all of the high popularity videos are included in the prefetch region determined by the prefetch size. The prefetched/cached videos which contribute the calculation of prefetch quality is included in the shade area A where $A + B = 1$. Note that prefetch quality measures the popularity of the videos that have been prefetched. A higher prefetch quality implies that the prefetch algorithm has a higher hit ratio. Using the prefetch quality, an administrator can therefore identify the impact of inaccurate user behavior information (request noise) on the performance of the caching system. Using the prefetch quality index, the performance of a surrogate can be roughly estimated under both ideal and non-ideal request patterns.

5.2. *Prefetch quantity*

The term “prefetch quantity” is defined as the percentage of videos which are correctly prefetched in the cache server system (after applying individual request noise). It indicates

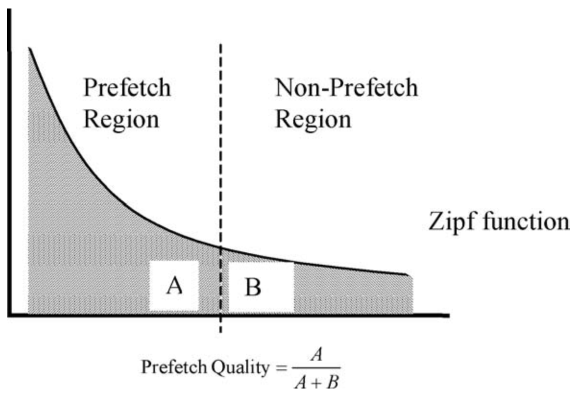

Figure 11. Illustration of prefetch quality.

the impact of noise on performance. The prefetch quantity can tell us how many videos are correctly prefetched, although it cannot estimate the hit ratio of the cache server system. This information can help content administrators to predict the trend of the hit ratio of the prefetch/cache sever system. When the popularity of a video starts to decrease, its popularity tends to continue to decreases in the future. A reduction in the prefetch quantity indicates that some videos have become less popular and thus there is a high probability that the system performance will drop in the near future.

5.3. *Uses of the measurement indices*

Using knowledge of video popularity, the prefetch quality and prefetch quantity can be easily calculated. These indices can be used as performance prediction indicators for a prefetch system. Figures 12 and 13 show examples of the use of prefetch quality and prefetch quantity, respectively. In Figure 12, it can be seen that the shape of the prefetch quality curve is a good match to the shape of the hit ratio curve (and it can be seen from Figure 13 that prefetch quantity is a less good match). Note, however, that the prefetch quality is not equal to the hit ratio. Prefetch quality predicts the performance in the ideal case where the most popular videos are always prefetched in each daypart. In reality, of course, this will not happen for two main reasons:

1. Not all popular videos are prefetched before the start of their corresponding daypart.
2. Prefetched videos are replaced before the end of their daypart to allow prefetching of popular videos for the next daypart.

Although prefetch quality captures the hit ratio trends in cache performance against noise, it does not provide quantitative information as to the number of videos which are correctly/incorrectly prefetched (which is needed to allow content administrators to decide when cache server content should be updated). Here, we suggest how the prefetch quality index may be combined with the prefetch quantity index to address this issue. In general, there are four cases to consider.

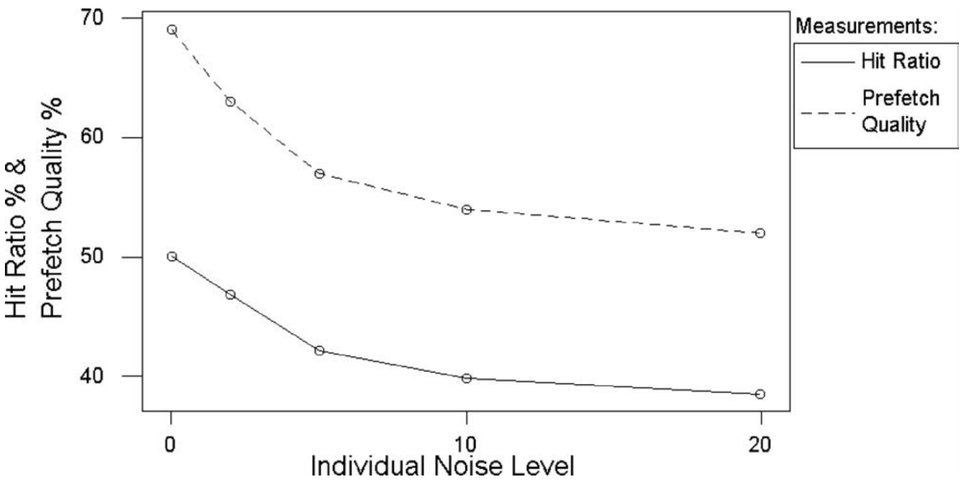

Figure 12. Variation in prefetch quality and hit ratio with individual request noise.

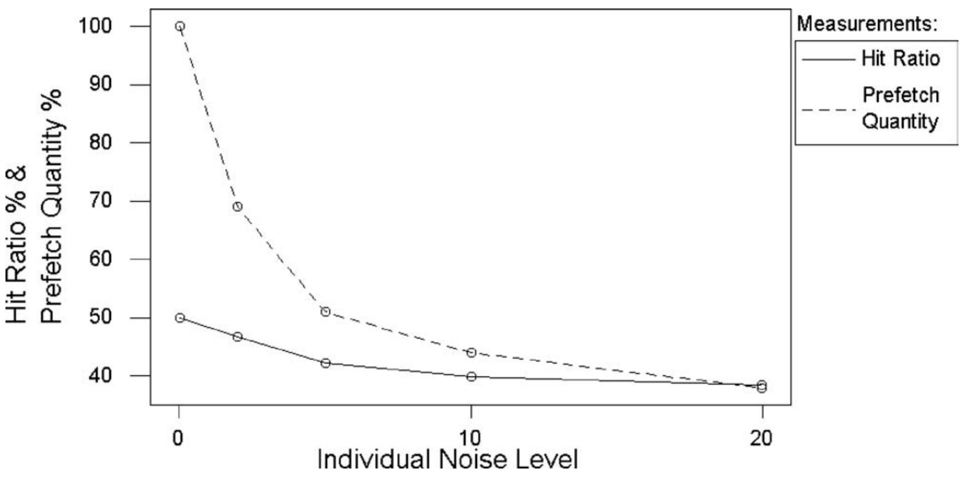

Figure 13. Variation in prefetch quantity and hit ratio with individual request noise.

1. *High prefetch quality and high prefetch quantity.* Most of the popular videos are prefetched and the cache has a good hit ratio. The cache server is performing well and can be expected to continue to have good performance in the near future.
2. *High prefetch quality and low prefetch quantity.* The prefetch algorithm is achieving a good hit ratio. However, a low value of prefetch quantity indicates that the prefetch algorithm is identifying only a small number of popular videos. Therefore, it is recommended that the content provider should update the prefetch profile to get the latest user behavior.

3. *Low prefetch quality and high prefetch quantity.* The prefetch algorithm is correctly identifying the most popular videos. However, the low prefetch quality indicates that the hit ratio is poor. This is likely to occur when there is not enough cache space to hold all of the popular videos. Therefore, increasing the cache size is recommended if the current performance is not acceptable.
4. *Low prefetch quality and low prefetch quantity.* The prefetch profile urgently requires to be updated. In some cases, if the prefetch profile cannot be updated immediately, the use of a combined prefetch and cache replacement technique with a larger cache size might improve the system performance.

6. Conclusions

This paper studies caching algorithms for a media server system that take account of the time-varying nature of user behavior (TVB). In particular, a TVB-aware prefetch algorithm and two cache replacement algorithms are proposed. Simulation results show that the TVB-aware prefetch algorithm performs significantly better than a conventional prefetch algorithm. In addition, it is shown that by combining the TVB-aware prefetch algorithm with a suitable cache replacement algorithm further performance improvements can be achieved. In order to test robustness to imperfect user request information, we consider three request noise models for generating unpredictable/unknown user requests. Simulation results show that under imperfect user request information the combined prefetch and AbsPop replacement algorithm achieves better performance and is more robust against noise than the pure TVB-aware prefetch algorithm. Finally, two prefetch performance indices are presented to assist content administrator in deciding when to update the user request profile for caching algorithms and hence to maintain high efficiency in their media server systems.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable comments, which substantially improved the quality of the paper. The work described in this paper was substantially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (project number: 7001224).

References

[1] S. Acharya and B. Smith, “MiddleMan: A video caching proxy server,” in *Proc. 10th International Workshop Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, June 2000.

[2] CA*netII (Canada’s coast to coast broadband research network), sanitized log files, parent cache, available at `http://ardnoc41.canet2.net/cache/squid/rawlogs/`

[3] P. Cao, E. W. Felten, A. R. Karlin, and K. Li, “A study of integrated prefetching and caching strategies,” in *Proc. of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1995.

[4] R. L. Carroll and D. M. Davis, *Electronic Media Programming: Strategies and Decision Making*, McGraw-Hill, 1993, pp. 62, 289–291.

[5] A. L. Chervenak, "Tertiary storage: An evaluation of new applications," Ph.D. Thesis, University of California at Berkeley, Computer Science Division Technical Report UDB/CSD 94/847, December 1994.

[6] M. Chesire, A. Wolman, G. M. Voelker, and H. M. Levy, "Measurement and analysis of a streaming media workload," in *Proc. Usenix Sympos. Internet Technologies & Systems (USITS)*, March 2001.

[7] Cisco Systems, Inc., "Cisco content engines for the Internet content delivery network," `http://www.cisco.com/warp/public/cc/so/neso/cxne/cceng_ds.pdf`

[8] Cisco Systems, Inc., "Cisco content networking technology," `http://www.cisco.com/warp/public/44/solutions/network/content_networking.shtml`

[9] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *Proc. 2nd Annual ACM Multimedia Conference and Exposition*, San Francisco, CA, 1994.

[10] D. Duchamp, "Prefetching hyperlinks," in *Proc. of the 2nd USENIX Symposium on Internet Technologies and Systems (USITS'99)*, Boulder, CO, 1999.

[11] J. Griffioen and R. Appleton, "Reducing file system latency using a predictive approach," in *Proc. of USENIX Summer Conf.*, 1994, pp. 197–207.

[12] J. Kangasharju, F. Hartanto, M. Reisslein, and K. W. Ross, "Distributing layered encoded video through caches," *IEEE Transactions on Computers* 51(6), 2002, 622–636.

[13] E. P. Markatos and C. E. Chironaki, "A Top 10 approach for prefetching the Web," in *Proc. of INET'98: Internet Global Summit*, 1998.

[14] T. Palpanas and A. Mendelzon, "Web prefetching using partial match prediction," in *Web Caching Workshop*, San Diego, CA, 1999.

[15] M. Reisslein, F. Hartanto, and K. W. Ross, "Interactive video streaming with proxy servers," *Information Sciences* 140(1–2), Special Issue on Interactive Virtual Environment and Distance Education, 2001, 3–31.

[16] R. Rejaie, H. Yu, M. Handley, and D. Estrin, "Multimedia proxy caching mechanism for quality adaptive streaming applications in the Internet," in *Proc. IEEE INFOCOM 2000*, 2, March 2000, pp. 980–989.

[17] Q. Yang and H. H. Zhang, "Integrating Web prefetching and caching using prediction models," *World Wide Web* 4(4), 2001, 299–321.