DESIGN, ANALYSIS, AND EXPERIMENTAL VERIFICATION OF CONTINUOUS MEDIA RETRIEVAL AND CACHING STRATEGIES FOR NETWORK-BASED MULTIMEDIA SERVICES

DONG, LIGANG

(*M.Eng.& B.Eng., Zhejiang University, PRC*)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2002

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Assistant Professor Dr.Bharadwaj Veeravali. His very friendly guidance, constant encouragement, insightful ideas, and rigorous research style accompany the entire progress during which I study for the PhD degree. I am also very grateful to my co-supervisor, Associate Professor Dr. Chi Chung Ko, for his valuable suggestions and enlightening instructions on how to do researches and make impressive presentations during the weekly seminar.

I would like to thank very much the National University of Singapore (NUS) for granting me the research scholarship in past three years.

I am also very grateful to the support from the project - High Speed Information Retrieval, Processing, Management and Communications on Very Large Scale Distributed Networks (funded by SingAREN and NSTB Broadband 21 Programme).

My special thanks to my parents, sister, and brother-in-law for their continuous encouragement and supports. I could not have come so far in my long study life without them.

My sincere thanks to my wife, Dan, who put up with a three-year-long separation without a grudge. Her selfless love provided an important support for my study.

Finally, my thanks also go to all of my friends in Open Source Software Lab, Computer Communication Network Lab, and Digital System Application Lab. The friendship with

them made my study and life in NUS fruitful and enjoyable.

Contents

Summary

Network-based multimedia services are attractive for both users and service providers. Network-based multimedia applications have widely appeared in the recent years. Multimedia is either continuous media (CM) (e.g., video and audio) or non-CM (e.g., text and image). Owing to their large sizes, large playback rates, and the continuous-playback constraint, CM data pose more challenges than non-CM data on the design of services. In this thesis, we carry out design, analysis, and experimental verification of *retrieval* and *caching* strategies for CM data to improve the quality of service.

Requested CM documents are retrieved from the server for the playback at the client using either of two modes - streaming or downloading. In the streaming mode, the users enjoy a shorter start-up delay and need less storage spaces than the downloading mode. The multiple-server retrieval strategy can reduce the load on a single server and achieves a better performance by partitioning a retrieval task among several servers. In this thesis, we focus on the multiple-server retrieval strategy in the case of the streaming mode. Our multiple-server retrieval is realized by using the Multiple-Channel Retrieval (MCR) algorithm, which can be used in either a single-server or a multiple-server retrieval. The MCR algorithm not only meets the requirement of a continuous playback, but also outperforms the single-channel and single-server retrieval in important performance metrics, e.g., start-up delay, block ratio, and retrieval duration. The MCR algorithm includes strategies

of channel partition, static scheduling, and dynamic scheduling. The channel partition strategy allocates available bandwidths to form retrieval channels. The static scheduling strategy, which is applied before the playback begins, determines when and what data are retrieved from synchronous or asynchronous channels. The dynamic scheduling strategy, which is carried out during the retrieval process, handles variable-size channels caused by variable network traffics. Besides, the server can dynamically change the channel size to improve the acceptance ratio of coming requests.

The experiment of the multiple-channel and multiple-server retrieval has been carried out. We retrieve video data from local and remote video servers by using HTTP and TCP. This experiment gives more insights on designing the retrieval strategies. The experiment complements the simulation and shows the advantage of the multiple-channel and multiple-server retrieval. The experiment also implies the applicability of proposed retrieval strategies.

Caching can reduce the load on the original server and improve the quality of services for clients. The interval-level caching strategy is a class of most popular caching strategies for CM documents. The interval-level caching strategy caches only a part of a CM document, thus, less cache spaces are required. Nevertheless, there exist several drawbacks in past interval-level caching strategies. Firstly, past interval-level caching strategies consider only the constant-size interval. In fact, the bandwidth of a stream is neither fixed nor changeless, therefore, an interval, which is formed by stream(s), will not be constant-size. Therefore, the resource allocation should be *dynamic*. Secondly, past interval-level caching strategies ignore the existence of *switching* operations, which happens when a stream finds no readable data in the cache or there are not sufficient bandwidths. The switching operation will affect the continuous playback, hence we propose some strategies to avoid switching operations. These strategies direct the replacement operation among intervals. Finally, in past interval-

level caching strategies, the bandwidth is *reserved* before the usage. Instead, we allocate the bandwidth *just-in-time* to efficiently utilize the bandwidth resource.

In all, our research contribution is to improve performances in retrieval and caching issues. They have very important effects on the quality of network-based multimedia services.

List of Tables

List of Figures

Chapter 1

Introduction

MultiMedia Information Technology (MMIT) provides an attractive means of communication in the modern day era. Network-based multimedia services [44] have been proven efficient, cost-effective, and adaptable. The popularity of such network-based services is increasingly becoming attractive. A significant advantage comes from providing a complete flexibility in the presentation control to the users. Thus, users may *interact* with a multimedia presentation just as they would do with a Video Cassette Recorder (VCR) presentation. From service providers' perspective, network-based services are attractive in terms of attracting a large group of clients (maximizing the number of clients) while promising a guaranteed Quality of Service (QoS) at a cheaper price.

As exemplified in the multimedia literatures [112], media are categorized into two types - *continuous media* (CM) (or called *streaming media*), e.g., video and audio, and *non-continuous media*, e.g., text and graphics. A significant challenge is posed in handling CM as opposed to non-CM. This is primarily due to their large sizes, large storage requirements, large communication bandwidth consumption, etc. Furthermore, the temporal and spatial properties inside CM are also important constraints that significantly affect QoS.

The respective details are discussed exhaustively in the available multimedia literature and can be found in [23, 151].

Starting from the mid-1980s, the manifold development in CPU processing power, storage device capacities, and network bandwidths, has made it feasible to support network-based multimedia services. One of the most popular applications of such network-based multimedia services is Video-on-Demand (VoD) [144, 124, 103, 92], the research of which started gaining attention in the late 1980s. Besides VoD, other typical services include digital library [184], distance education [105], video conferencing [4], interactive TV [142], interactive games [25], home shopping [161], and so on. Network-based multimedia applications greatly challenge the computing, storage, and networking technologies. Most recently, content distribution/delivery network [29] is considered to provide efficient distribution and delivery of multimedia contents (mainly for CM data). In this thesis, we particularly focus on two different important issues - *retrieval* and *caching* of CM data.

1.1 Related Work

In this section, we shall discuss on the literatures that are closely related to the focus of this thesis. In addition, we shall expose most of the relevant techniques that are currently in place on the retrieval and caching of CM data. In this section, we will indistinguishably use "video" and "CM document". Besides, some researches for non-CM data are also applicable for CM documents. When we introduce them, we use "object" or "document" to refer to both non-CM and CM documents.

1.1.1 Admission control

Admission control policies are used to control the admission of user requests so as to guarantee the combined QoS requirement of all admitted requests. The capacity or throughput of a multimedia system is usually represented with the maximum number of streams, which the system can support while satisfying a certain level of QoS. Admission control algorithms for CM data can be categorized into two types: *deterministic* and *statistical* (or called strict and predictive [99]). Deterministic admission control algorithms make worst-case estimates of the bit rate and disk access times, and are used when users cannot tolerate any losses. Statistical admission control algorithms use estimated probability distributions of the bit rate and disk access times to guarantee that deadlines will be met with a certain probability. Such algorithms achieve much higher utilization than deterministic algorithms, and are used when clients can tolerate *infrequent* losses [164, 165]. For video data, if users can allow some degree of data loss, then more requests can be admissible [116]. Mundur [110] proposed and analyzed threshold-based admission control, which groups new requests into priority classes. Priorities are based on popularity of videos.

1.1.2 Load balancing

Load balancing among disks or servers help to improve the total throughput of a system. Load imbalance can be reduced by combining the storage device into striping groups and interleaving the data among the devices. Wolf *et al.* [175] considered two load-balancing schemes. The static component determines good assignments of videos to groups of striped disks. The dynamic component carries out load balancing through a real-time disk retrieval scheduling. In detail, in processing a block request for a replicated object, the server will dynamically put the retrieval operation to the most lightly loaded disk to carry the load balancing [114]. In [135], data are randomly allocated and partially replicated on

disks to achieve load balancing. Replication allows some of the load of the disks with smaller Bandwidth to Space Ratio (BSR) to be redirected to the disks with higher BSR. In [63], five different load allocation polices were designed and analyzed. Dynamic Policy of Segment Replication (DPSR) divides multimedia files into fixed-size segments and replicates segments (which have the highest payoff) from highly loaded disks to lightly loaded disks. For copying from the highly loaded disks, the DPSR policy does not require an additional stream but uses a stream that is already playing, which is called the copyback stream. The Caching for Load Balancing (CLB) policy [146] attempts to balance the load among the various storage devices by caching only streams from heavily loaded disks (whose load are greater than the average load) or overloaded devices, hence it minimizes the rejection rate.

1.1.3 Placement strategies in storage devices

Good data placement policies result in a high operational efficiency of the server to achieve a high utilization of the storage space as well as the bandwidth of the storage devices.

Firstly, we introduce the data placement in a single disk. Gemmell *et al.* [55] categorized the placement strategies as being interleaved, noninterleaved, contiguous, or scattered. Of them, the contiguous placement strategy is the most important. For eliminating intrafile seeks, data in a file should be stored in contiguous blocks. An application may require the presentation of multiple kinds of media (e.g., video and multiple sound tracks). One approach is that the required multimedia data may be stored as a single file with multiplexing of the various media streams, e.g., MPEG [52]. An alternative approach is to store the individual streams as individual files and to transmit these files to the client. In this case, the blocks of the files should be stored close together to facilitate a continuous retrieval [125]. The organ-pipe placement policy [156] emphasized that most frequently referenced blocks are placed in the location with the maximum block access rate (e.g., the periphery

of a multizone disk) to improve the throughput of the disk. Secondly, we can consider the data placement from the viewpoint of disk arrays. In [15], two kinds of striping methods, RAID-3 and RAID-5, were compared in terms of the cost and the revenue-earning potential. The result showed that for large-scale video servers, coarse-grained striping in a RAID-5 style is more cost effective. In [168], the studied problem was how to replicate, stripe, and place the CM documents over a minimum number of magnetic disk arrays, such that a given access profile can be supported. The authors demonstrated that it is an NP-hard problem, and presented some heuristic algorithms to find the near-optimal solutions. In [86], the authors considered how to move (add or remove) disks to support dynamic required retrieval bandwidths in striped disks without reorganizing the striping, as the cost of reorganization is much higher than the movement cost of disks.

1.1.4 Request scheduling

Request scheduling considers how to maximize the capacity of an entire system through the cooperation among requests. Originally, we allocate respective resources to every request once the request arrives. It is called unicasting. In this case, all of the VCR functions can be supported. However, this scheme is cost-expensive. The following strategies are proposed to make the server support more requests.

Merging. Merging of multiple adjacent streams can reduce the bandwidth consumption. One method of merging streams for the same video is to slow down the playback of leading streams and/or speed up the playback of lagging streams. This method is called piggybacking in [58]. Another merging strategy is to delay the streams through displaying some filler materials such as previews [78]. In [78], the authors used merging to deal with the failure or overloading situation. The authors in [16, 2, 83] considered optimal/heuristic stream merging algorithm for minimizing the I/O consumption. In [17], the implementation details

of merging were provided.

Patching [6, 141, 74, 61]. In patching, a client will receive data from multiple streams. The beginning part of a video (so-called "patch") is from unicasting, and the other part is from earlier opened stream(s). The client plays the patch part while it buffers data for the late playback. In [24], the authors derived an optimal patching window, after which it is more bandwidth efficient to start a complete stream rather than send the patch.

Bridging [133, 69, 70]. In bridging, two successive streams are bridged by buffering/caching a segment of data between them, so that the server only needs to provide the bandwidth of supporting one stream. Buffered/cached data can be stored in any node in the path from the server to the client, e.g., server, proxy, and client. For instance, in [133], after every retrieval begins, retrieved data will be retained in the server memory for a certain period of time, which is termed as the *viewer enrollment window*, so that the requests that arrive during the viewer enrollment window can get the data from the memory. Essentially, bridging is interval caching [36] or generalized interval caching [39] (caching strategies will be introduced later). If two bridging streams can be merged after a period of time, so that the utilization of the buffer is saved.

Non-periodic multicasting (or called batching) [37, 38]. In the batching-by-timeout (or called forced-wait) policy, the first queued request for each video is forced to wait for a certain time interval. In the batching-by-size policy, a stream is opened only when a specified number of requests for the same video are grouped together.

Periodic multicasting/broadcasting. With a periodic multicasting/broadcasting, the total required bandwidth is constant for a server, irrespective of arrival rates of requests. The basic idea is that each video is partitioned into several segments and broadcasted peri-

odically towards a goal of achieving a minimum start-up delay. Multicasting/broadcasting provides the most cost-effective solution for popular videos. The simplest broadcasting protocol is staggered broadcasting. However, the following broadcasting schemes provide better performances (i.e., less bandwidth requirement when the maximum start-up delay is limited in a fixed value) because the client receives data from multiple channels. For instance, pyramid broadcasting [166], permutation-based pyramid broadcasting [3], skyscraper broadcasting [60], fast broadcasting [67], harmonic broadcasting [66], fixed-delay broadcasting [117] etc. The authors of [117] claimed that if given the bandwidth with six times of the playback rate, then the waiting time will be less than 32 seconds for a two-hour video. In the broadcasting, this video will be divided into 2046 segments. More segments will further reduce the start-up delay. Besides, there is a kind of dynamic broadcasting protocols [183], which are like common broadcasting protocols. However, dynamic broadcasting protocols keep track of user requests, so that when there are less users, some segment transmissions are skipped. Broadcasting is cost-efficient, nevertheless, there are two main drawbacks. Firstly, VCR functions cannot be supported (except pause/resume). Secondly, the allocation of bandwidth must be strictly satisfied, and the video must have a constant bit rate, otherwise the continuous playback cannot be satisfied.

Combined scheme. When batching is combined to patching, a better performance (in terms of the server bandwidth consumption) can be achieved than single patching at the expense of higher latency [172]. In [30], a method of combining unicasting, patching, staggered broadcasting, and stream-bundling broadcasting was proposed to meet the various requirement of "hot", "warm", and "cold" videos. Lee [92] analyzed the combination of unicasting, patching, and staggered broadcasting. Poon *et al.* [121] considered the combination of unicasting, bridging, and staggered broadcasting to minimize the reneging probability.

1.1.5 Support of VCR functions

The possible VCR functions include play forward, play backward, pause/resume, fast forward, fast backward, slow forward, slow backward, jump forward, and jump backforward [81]. In unicasting, VCR functions are easily supported. Here we introduce how to support VCR function in the case of batching or multicasting/broadcasting. Pause/resume is the most common VCR operation. When a stream performs a pause operation, this stream will leave the batching retrieval. In this case, there are two choices for this stream. One method is the contingency channel policy [39], in which a small number of shared contingency channels (which cannot be used by new requests) are set aside for handling unpredictable demands due to VCR control operations. The emergency interactive channels [5] have similar functions. In the other method, which is used in the split and merge policy [98], required data are buffered in the proxy or the client, thus, the server need not transmit the data again. Fast forward and fast backward probably cause additional bandwidth requirement [41]. However, they also can be implemented using two approaches without increasing bandwidth consumption. One method is that special files for fast-forward are generated and stored beforehand. The other method is that selected frames or blocks are transmitted from the server to the client [179]. For supporting VCR functions, the resource (i.e., buffer space and disk bandwidth) requirement for satisfying a certain QoS is analyzed in [95]. The authors in [48, 81] studied how to support VCR functions in staggered broadcasting in the segment-level and the block-level, respectively.

1.1.6 CPU and I/O scheduling

The scheduling policy determines how to allocate the utilization of resources among competitive tasks/requests. In the multimedia application, the resources are mainly referred to the CPU computing power and I/O bandwidth. For the multimedia application, we usually

adopt the scheduling strategy for *real-time* applications [55, 13]. Real-time applications include hard real-time applications, which require deterministic guarantees for the response time of each task, and soft real-time applications, which require statistical guarantees for the response time of each tasks. For the scheduling of periodic real-time tasks, there are two scheduling priority policies [146]. In rate monotonic scheduling, the task with a shorter interrequest time has higher priority. The deadline scheduling policy sets the priority of each task to its deadline. Tasks with the same deadline are processed in an arbitrary order. Besides, there are scheduling strategies for *best-effort* applications [126, 182]. For instance, interactive applications require low response times, and throughput-intensive applications require high throughputs. The authors in [140, 174, 132] proposed the disk scheduling framework for meeting the *mixed* (i.e., including real-time and best-effort) service requirements of applications. They served a non-real-time request in a round, only if all the remaining real-time requests in this round will not miss their deadlines. The hybrid rate monotonic policy [124] classifies tasks into three types - isochronous, guaranteed-service, and background. Isochronous tasks are real-time periodic tasks, such as video streams. Guaranteed-service tasks are tasks that require guaranteed throughputs and bounded delays, e.g., polling service drivers. Background tasks are low-priority tasks with no guarantees of QoS. In [139], the authors focused the scheduling of a presentation, in which both the intraobject time dependency and the interobject time dependency were considered.

The traditional objectives of disk scheduling policies are to maximize the disk throughput and minimize the disk response time. In multimedia servers, an additional objective is to ensure that each stream is able to retrieve its blocks without missing deadlines. Thus, we can summarize that various disk scheduling algorithms consider some of three factors - (1) Maximum throughput, (2) Minimum response time, and (3) Real-time. Table 1.1 is the

taxonomy of existing single disk scheduling algorithms/policies. The introduction of FCFS,

Table 1.1: Taxonomy of disk scheduling algorithms/policies

Algorithms	Factor (1)	Factor (2)	Factor (3)
Shortest Seek Time First (SSTF), Smallest Positioning Time First (SPTF)	•		
FCFS, Shortest Total/Access Time First (STF/SATF), Aged Shortest Access Time First (ASATF)		•	
EDF, round-robin			•
SCAN/C-SCAN, LOOK/C-LOOK, V(R)	•	•	
Priority SCAN, Shortest Seek Earliest Deadline by Order/Value (SSEDO and SSEDV), Feasible Deadline SCAN (FD-SCAN), SCAN-EDF, earliest deadline SCAN, GSS	•	•	•

SSTF, SPTF, SCAN/C-SCAN, and LOOK/C-LOOK can be found in [178]. STF/SATF and ASATF can be found in [62]. SSEDO, SSEDO, and FD-SCAN can be found in [32]. In round-robin, each stream is served according to a fixed order in a round, however the order is randomly chosen. In disk scheduling, the Earliest Deadline First (EDF) policy [126] may have a high seek overhead because only the deadline is considered to determine the service order of I/O requests. In SCAN, the disk head repeatedly sweeps outward from the center of the disk to the periphery and back to the center. The policy has lower seek overheads, however the read-ahead buffer needed per stream is equal to that needed for two rounds in round-robin. C-SCAN, a variant of the SCAN policy, performs sweeps in only one direction

(inward or outward). The SCAN-EDF policy [126] reduces the seek overhead associated with EDF by grouping requests with deadlines in a small time interval T_{group}. V(R) is the parameterized generation of SSTF and SCAN by adding a penalty for every change of the direction. The Group Sweeping Scheduling (GSS) policy [182] obtains the tradeoff between round-robin and SCAN. GSS partitions V active streams into M_{gr} groups, services M_{gr} groups in round-robin order, and services the streams within each group using C-SCAN.

Pang *et al.* [114] proposed to give each disk an advance notification about the blocks that have to be fetched in the impending time periods, so that the disk can optimize its service schedule. Lau *et al.* [82] studied the retrieval scheduling from the magnetic tape for real-time applications.

The scheduling theory can be classified into the share scheduling and the non-share scheduling. The traditional scheduling theory [119, 22] is usually referred to the non-share scheduling, in which, a job exclusively occupies the resource of one machine/processor. In multimedia applications, the scheduling of either CPU or I/O belongs to share scheduling. In the scheduling of CPU computing power, hierarchical scheduling [59] is about how to partition the server between task groups. The partitioning of the server among various groups is independent of the load on the groups, so that it is not possible for a task in any group to maliciously or accidentally monopolize the server. A fair-share policy services the various groups in a round-robin way. The fair-share policies include Weighted Fair Queuing (WFQ) policy and Start-time Fair Queuing (SFQ) policy [59]. SFQ has greater fairness and behaves better under variable loads. In I/O scheduling or retrieval scheduling, we must determine how to partition the server bandwidth into several channels and allocate channels among the streams. In [72], the share scheduling was converted to the non-share scheduling by fixing the sizes of channels. The study mainly focused on minimizing the

number of tardy frames in the end-to-end delivery of CM data.

1.1.7 Multiple-server approach

As the single-server approach is expensive for large-scale applications [89], this motivates us to aggregate the capacity and bandwidth of multiple servers to provide cost-efficient scalable performances.

Parallel video servers (or called clustered servers or server arrays). Lee [87] gave a comprehensive study of architectural alternatives and approaches employed by existing (before 1998) parallel-server systems. For the parallel video server, there are two kind of service models - server-push [88] and client-pull [89]. Under the server-push model, the server schedules the periodic retrieval and transmission of video data, once a video session is started. Under the client-pull model, the client periodically sends requests to the servers to retrieve blocks of video data. Thus, for these two models, the data synchronization is carried out in servers and clients, respectively. In [88, 89], various performance metrics (such as service delay and client/server buffer requirement) have been analyzed. In [91], the buffer requirement in the client-pull mode was analyzed in detail.

Multiple-server retrieval scheduling. Bharadwaj *et al.* [163] introduced a novel retrieval method, in which a single long duration multimedia document is retrieved from a pool of servers as opposed to the idea of employing a single server in the network. This is different from the parallel-server approach, as different servers are unrelated from each other in [163]. The authors of [163] assumed that the clients cannot start the playback of the ith portion *until* the client downloads it entirely from the ith server. Under this assumption, the authors presented a schedule to minimize the access time. Ping *et al.* also considered employing multiple servers to retrieve a CM document [120]. The authors

designed an optimal retrieval scheduling scheme that postpones the buffer overflow at the client as much as possible. The study mainly focused on the design of buffer management strategies.

1.1.8 Reliability issues

To reduce the impact of device failures, it is possible to replicate multimedia objects (e.g., mirroring) [80] or store redundant (parity) information (together with striping) [90]. The parity approach requires less disk spaces while there is the overhead in re-computing unavailable data due to disk failures. Besides, Cohen *et al.* [34] proposed the SID scheme to allow a tradeoff between data replications and parity encoding. Lee *et al.* [93] studied the rebuild algorithms for rebuilding data stored in a failed disk into a spare disk.

1.1.9 Overview of cache management

Data can be stored in either an *origin* or a *cache*. An origin is an initial or original storage location of data, whereas a cache is a storage location in which data are uploaded for future accesses when documents are delivered from the origin to clients. *Caching* concerns the use of the cache to avoid delays and/or overheads in accessing the origin.

There are two typical caching - *memory caching* and *disk caching*. In memory caching, the high-speed main memory is used as the cache of the relatively slow-speed disk. In disk caching, the near-distance disk (e.g., in proxy) is used as the cache of the far-distance disk (e.g., in original server), or the disk is used as the cache of tertiary storage, e.g., CD, tape. The bandwidth of a memory cache is rarely a bottleneck, i.e., the bandwidth of main memory can be assumed to be virtually infinite. On the contrary, disk caching policies have to consider constraints imposed by disk bandwidths as well as disk spaces.

Caches are used because of the following two advantages. Firstly, caches are usually "nearer" to the client than origins or need less access times. Hence, the bandwidth consumption of networks or the server I/O and the access time are reduced significantly. Thus, caching increases the system capacity. Secondly, to a certain extent, clients can obtain the data even when the origin cannot accept the requests under failure or overloaded situations. Thus, we achieve a good persistence of data.

Buffering and caching are two similar techniques with a little differences. The storage locations of buffering and caching can be the same. However, in buffering, the data blocks, which are transmitted from a sender to a receiver, are temporarily stored until the receiver consumes them. The occupied buffer space is released when the data have been consumed by the receiver. In caching, the data blocks are stored for future accesses. Unlike buffering, the copy may be retained as long as there are storage spaces available to hold it.

Buffering may be used to connecting two continuous delivery process, e.g., communication buffer, I/O buffer. Furthermore, buffering can smooth burstiness in the instantaneous data consumption rate in various components in the delivery path, so as to avoid jitters in the presentation of CM data.

In the caching problem, we are concerned on issues on *who* caches a document, *when* to cache a document, *where* to cache a document, *what* documents to be cached, *how* to find the cached documents, *where* to place the caches in the network, and *consistency* of cached documents.

Consistency of cached documents. This issue is concerned with the update of any stale cached documents that may exist in the system, owing to the presence of multiple copies [104]. The consistency policies/protocols can be characterized by some parameters,

e.g., replica responsiveness, replica reaction, change distribution, write set, coherence group [118]. The authors of [134] presented an overview of various policies. As different web documents have different features, Pierre *et al.* [118] decided separately the consistency policy for each document to minimize the response time, the number of stale document, and the consumed bandwidth.

Push caching and pull caching. This is a problem on *who* decides to cache the documents. The push caching scheme is also called the *server-initiated strategy* [18, 106], where the origins decide on caching the documents. This scheme ensures a strong consistency, however it does not cope up with the rapid changes in request arrival patterns (e.g., a burst request arrival). In addition, in this case, the origins need an authority to *command* the caches which are often autonomous. On the other hand, the pull caching scheme is also called the *client-initiated strategy*, in which the documents are cached only when a client requests them. As opposed to the former strategy, the client-initiated strategy adapts to the request arrival pattern that is rapidly changing. However, in this case, the problem of cache consistency arises. Besides, through making the content of cached video be known by users, the users can adjust their requirement. Thus, the cached video can be fully utilized [148, 149].

On-demand caching and on-command caching. This is a problem on *when* to cache the documents. In the *on-demand caching* strategy [102], documents are cached when they are accessed by the client. In other words, when there are no requests for a document, we do not consider to cache that document in the cache. In contrast, in the *on-command caching* strategy [102], the cache is set up to automatically retrieve certain documents, or possibly replicating all the documents from an origin at regular intervals. The prefetching strategy is a kind of on-command caching strategy. The basic prefetching techniques are

always prefetch [160] and stride prediction (e.g., RPT [33]). In prefetching, an estimate of the future access probabilities (i.e., request rates) is computed and relevant documents are cached for future accesses [46, 68]. In [35], a bi-dimensional spatial locality in images was exploited for prefetching. In interactive and/or composite media documents, the access pattern is not sequential but may follow a set of likely access sequences over many small media objects. The access pattern can be modeled as navigation of a hypergraph, where each node represents playback of a small media object. Thus, the most likely set of follow-up nodes can be prefetched [146].

In the following Sections 1.1.10 and 1.1.11, we consider the problem on *what* documents are cached, in terms of a partial document or a full document, respectively.

1.1.10 Full caching

Before determining which document is cached, the "importance" of the documents will be measured by the following four factors.

(1) Temporal locality (or locality of reference). This aspect suggests that a recently accessed document is likely to be accessed again in the future. According to [65], long-term popularity and short-term temporal correlation among requests (accesses) are two sources of temporal locality. There is a detailed study about temporal locality in [147].

(2) Access frequency. This refers to the rate at which the requests arrive for a document.

(3) Document size. The size of a document is equal to the space requirement if that document is cached. Therefore the size of a document has an importance effect on deciding whether or not to cache a document.

(4) Miss penalty. This is the retrieval cost of a document from the origin upon a miss in the cache [65].

Table 1.2 is the taxonomy of existing cache replacement algorithms/policies that consider the above four factors. In addition to the above factors, the lifetime of a document and the type of a document are also important factors that be considered in the design of replacement algorithms. Perfect LFU and in-cache LFU are two variants of LFU. Perfect

Table 1.2: Taxonomy of cache replacement algorithms/policies

Algorithms	Factor (1)	Factor (2)	Factor (3)	Factor (4)
FIFO				
LRU (e.g., segmented LRU, EELRU [147])	•			
LFU (e.g., perfect LFU, in-cache LFU)		•		
LFF (Largest File First) [20]			•	
Latency				•
LRU-k [111], LRFU [84], LFU-DA [10]	•	•		
LRUMIN, var-page-LRU [143]	•		•	
GD (i.e., GreedyDual) [181]	•			•
LRV [129], GDS [27], size-adjusted LRU [3]	•		•	•
Hybrid [177]		•	•	•
GD-LFU [77],GDSF [9], GDSP [64], LNC-W3 [136], GD* [65], LUV [14]	•	•	•	•

LFU keeps track of all the past accesses to all the documents (using counters to register these accesses) even when a document is evicted from the cache, whereas, in-cache LFU removes the record of all the past accesses when a document is evicted from the cache [21]. Segmented LRU [8] is a variant of LRU, in which the documents that are referenced only

once will be evicted out quickly. Among the caching algorithms that consider identical set of factors in the design, the difference lies in the amount of algorithm overheads incurred and in the manner in which these factors is represented. In the literature, there are three methods that are used to handle these factors. The first fashion is to combine the above mentioned factors using a heuristic or analytical scheme with some weights assigned to each of these factors, e.g., QoS in [1], utility value in [64]. Alternatively, one may prioritize some of the above factors over others in key-based policies [173]. Finally, in regression-based combination scheme proposed in [49], a past access record is used to do the regression calculation to obtain an optimal set of weights for these factors. However, the computation involved in the regression calculation is very large.

1.1.11 Partial caching

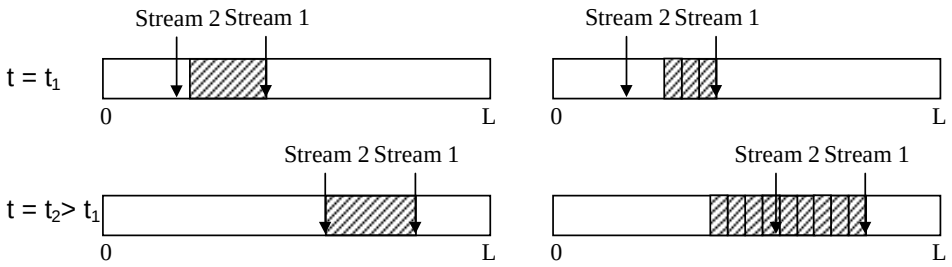
Because of large sizes of CM documents, it is not cost-efficient to store an entire document in the cache [38]. In this case, instead, we cache partial data of a document. The prefix-caching [138, 100, 115] caches the initial portion of the stream, and the prefix is transmitted from the cache to the client, so that the start-up delay is reduced. Verscheure and Frossard [50, 167] considered caching prefixes and patches (which are used in the patching method) to minimize the backbone bandwidth consumption. In addition, the cache space requirement was calculated. In layered caching, the multimedia document is (virtually) split into a number of layers. The lowest level contains the most important data. The layered encoded document is used to handle heterogeneous accesses [71]. If the network bandwidth to the source is limited, then only the lowest layers are fetched and played [127].

Besides, there are two classes of caching strategies/algorithms for CM documents. One class is the block-level algorithm, e.g. BASIC [113]. In a block-level caching algorithm, a block of data is the basic caching entity and the cache space is allocated for a single

block. BASIC selects to replace a block that would not be accessed for the longest period of time. The other class is the interval-level algorithm, e.g., DISTANCE [113], Interval Caching (IC) [36], Resource-Based Caching (RBC) [157], and Generalized Interval Caching (GIC) [38]. In an interval-level algorithm, the basic caching entity is an *interval*, which is an amount of data between two adjacent streams. Once an interval is chosen to be cached, its former stream places the read blocks in the cache upon consumption, so that the latter stream always can read the date cached by the former stream. Here, a stream is referred to a session that CM data are retrieved from the server (either the origin or the cache). The cache space is allocated for a single interval. IC orders only current intervals, in which both the former stream and the latter stream exist. However there is a kind of "anticipated" interval, in which the latter stream has not arrived. The GIC policy orders all the intervals (current or anticipated) in terms of increasing interval sizes and allocates the cache space to as many intervals as possible. Thus, IC is suitable for streams accessing long documents, and GIC extends IC so that both long and short CM documents can be managed. DISTANCE is similar to IC. A distance in DISTANCE is an interval in IC. RBC is a disk-caching algorithm while DISTANCE, IC, and GIC are memory-caching algorithms. In the RBC policy, each cacheable entity (entire object or fragments of an object) has been associated with resource requirements consisting of bandwidths and spaces.

Figure 1.1 compares the basic principles of the interval-level caching and the block-level caching algorithms. In comparison with the interval-level caching algorithm (e.g., GIC and RBC), the block-level caching algorithm (e.g., BASIC) has two drawbacks.

- In the block-level caching algorithm, the blocks are frequently ordered, then the replacement operations are carried out. In the interval-level caching algorithm, the intervals are ordered when intervals are generated or changed. Therefore the block-



(a) Interval-level caching (b) block-level caching

In the above diagrams, the long rectangle represents a CM document. L is the size of this document. In a document, the hashed part is cached data and the blank part is uncached data. In case of the block-level caching, the hashed part consists of cached blocks.
In the case of the interval-level caching, an interval of data between two adjacent streams is cached. The former stream (e.g., stream 1 in (a)) writes the data into the cache and the latter stream (e.g., stream 2 in (a)) reads cached data. Unless there is a following stream, which forms another interval together with stream 2, stream 2 will swap out read data from the cache.
In the case of the block-level caching, the stream chooses to cache important blocks for following stream(s). Following streams will not automatically swap out read data from the cache.

Figure 1.1: Comparison between the interval-level caching and the block-level caching

level caching algorithm has much higher operational overheads than the interval-level caching algorithm.

- The block-level caching algorithm caches unrelated sets of blocks and not continuous blocks of data, hence, it is difficult to guarantee a continuous playback at the client end.

Thus, designing block-level caching strategies is unrealistic to handle CM streams.

1.1.12 Distributed caches

Architecture of caches. There is probably single cache or multiple caches in a system. If several caches share their cached documents among all the clients (i.e., not limited to local clients), then the hit ratio will be improved greatly [43]. It is referred to as the *cooperative caching* strategy [128, 123]. Wolman *et al.* [176] evaluated quantitatively the potential of the performance improvement for the cooperative proxy caching using a trace-based method as well as an analytical approach. Their research showed the cooperative web proxy caching is an effective architecture for small individual caches that comprise user populations in the tens of thousands. In cooperative caching, two kinds of architectures were proposed in the literature [131] - mesh (or distributed) architecture [122] and hierarchical architectures [31]. Tewari *et al.* [158] provided a performance comparison between these two architectures and derived some design guidelines for a large-scale distributed cache environment. In the hierarchical architecture, a client needs to pass *several* hops for accessing the data in a *distant* proxy, while in a distributed architecture, a client is inclined to access neighboring and/or directly-connected proxy. Therefore, the latter has a shorter response time.

Location of cached documents. This is a problem on *how* to find cached documents. The research issue is to efficiently discover (i.e., routing of requests), select, and deliver

the desired document(s) from neighboring or remote caches in a cooperative-cache environment, e.g., [11, 107]. In a distributed architecture, the order of looking for a document follows from the local proxy to a neighboring proxy, and then to the original server. On the other hand, in a hierarchical architecture, the order of looking for a document is from the local proxy to parent proxies until an original server is reached. Basically, two topics are studied. One topic is how to route the request to the appropriate cache to retrieve the object. The ICP [170, 171] protocol allows proxy caches to broadcast requests for data not in the cache and retrieve data. However, this broadcasting of the query is often low efficient as we do not exactly know whether or not the required object is stored in a special cache. Thus, the second topic appears. It is how to store the information about objects and caches. For example, the directory of objects, load state of caches. This information is used for the reference of choosing the cache. A CRISP [53] cache consists of a group of cooperating caching servers sharing a central directory of cached objects. One alternative on [53] is to fully replicate the directory on every proxy and asynchronously propagate local changes in each cache to the rest to maintain all directories weakly consistent [54]. Another alternative on [53] is to store only the subdirectory of objects, which are shared by more than one cache, in the central directory [54]. A Cache Digest [26] or Summary Cache [47] is a summary of the contents of caches. It contains, in a compact (i.e. compressed) format, an indication of whether or not particular URLs are in the cache. Again, we return to the first topic. Now it is how to route the request to the *most appropriate* cache. In [123], the required object is retrieved directly from the Internet instead of the remote cache, when it is faster in the former way than in the latter way. Now these researches have been extended to the content distribution/delivery network [29].

Placement/replacement algorithms in distributed caches. This is a problem on

where (i.e., which cache) to cache documents. Sinnwell *et al.* [145] used cooperative caching to minimize the mean response time in Networks Of Workstations (NOWs). In [42], cooperative caching algorithm for web objects was proposed and multiple performance metrics were discussed. In [180], the authors considered cooperative caching for wireless multimedia streaming. The authors in [145, 180, 42] studied the caching strategies used in the distributed architecture. In comparison, the caching strategies used in a hierarchical architecture can be found in [19, 155]. In [19], an object is cached at the nodes that are a fixed number of hops apart on the path from the client to the server. In [155], a dynamic programming method was used to choose the caches, in which web objects are placed. The CARP [162, 28] protocol divides a set of URLs among a set of loosely coupled proxy caches. A hashing function is used to determine the proxy cache that should be requested for any particular URL. The usage of CARP is tightly related to affinity routing [39], in which requests are automatically routed to caches dedicated to caching a special set of objects. By enhancing the content locality, the access time is improved.

Cache location. This is a problem on *where* to place the caches in the network. In [97, 79], the placement of caches in the network is studied to maximize the service capability.

1.2 Motivation

Firstly, we explain why we concern the CM retrieval problem. In most of the existing literature, most of studies are limited to the retrieval using a *single* bandwidth channel from a *single* server. In an environment with various traffics in the network and various loads on different servers, if a long CM document is divided to several segments, each of which is retrieved from different bandwidth channels in different servers, then the load on the networks and servers will can balanced very well. Good load balancing makes the servers

in distributed networks accept more requests. This idea of employing a pool of servers to retrieve CM documents makes more sense especially for very long duration videos, as the amount of data to be transported is very large. In fact, many techniques in partial caching already contain this idea. For instance, in prefix-caching [138, 100, 115], the prefix and the remainder of a video come from different servers. Thus, we need consider how to coordinate the retrieval from different servers. This problem is resolved by scheduling. Bharadwaj *et al.* [163] employed multiple servers to retrieve a multimedia document. The authors assumed that the clients cannot start the playback of the ith portion *until* the client downloads it entirely from the ith server. In fact, the clients can adopt the *playback-while-receive* (i.e., streaming) mode to further reduce the access time, i.e., the clients begin the playback, once they receive the initiate portion (which is buffered for smoothing the variable bit rate of data) of a CM document. Ping *et al.* [120] also considered employing multiple servers to retrieve CM documents. Their study aims to minimize the consumption of the buffer space at the client. However, in fact, a client usually only need to support the buffering of one video, hence, the buffer space of the client is never insufficient. In comparison, the access time and the block ratio are more important performance metrics for the client. The above analysis motivates us to revisit the retrieval problem employing the multiple-server retrieval of CM documents in the case of play-while-receive strategy.

In the rest of this section, we present the motivation of studying the CM caching. First of all, we analyze the drawbacks in past interval-level caching algorithms.

Firstly, for any of the interval-level caching algorithms, there is a possibility of contacting the origin when the required data are not available in the cache, referred to as *switching* (also called hiccup in [85]) in this thesis. In detail, if a stream that is reading from the cache finds that there are no required data in the cache before its retrieval finishes, this

stream has to retrieve the corresponding data from the origin. We refer to this simply as *switching back* to the origin. In order to avoid any playback discontinuity owing to the shortage of resources (refer to the I/O bandwidth and the network bandwidth), the required amount of resources should be reserved in advance. However, in fact, this reservation is impossible, as the estimation of reserved resources is very difficult. Furthermore, the switching operation incurs additional operation overheads, and hence, it is undesirable. Thus, an efficient caching algorithm must also attempt to *minimize the number of switches to origin*, apart from improving the *byte hit ratio*, which is the most important performance metric in caching. Lee *et al.* [85] also noticed the disadvantage of switching operations, and proposed the following strategy to eliminate the switching operation: an interval is replaced only when the second stream of that interval can be serviced from a disk. However the studies in [85] were only based on IC (i.e., interval caching) used in the memory caching. In fact, the shortage of bandwidths in disk caching also may result in the switching operation. Moreover, in disk caching, the implementation of the method in [85] is difficult, as the cache server is far from the origin server, which is independent from the cache server.

Secondly, previous interval-level algorithms assumed that two streams that form one interval have the same retrieval bandwidth. However this assumption is not always true because of the following reasons.

- Two streams can come from different servers and arrive at different clients with different qualities of services (e.g., low-bandwidth or high-bandwidth).
- The retrieval bandwidth of a stream probably varies with time.
  - CM data are transmitted through the underlying interconnection network. Thus, time-varying network traffics can result in a time-varying retrieval bandwidth

consumption.

- At the client end, a stream may experience pause/resume, slow forward, fast forward, and other VCR operations. These VCR operations are equivalent to a variable bit rate retrieval situation for a stream. Although the authors of [39] introduced a method to deal with the pause/resume in GIC, by and large, handling a variable bit rate scenario, which is owing to several other VCR operations for CM data caching systems, still remains as an open and challenging problem to tackle.
- Servers probably adjust the retrieval bandwidth of streams for some purposes. If the server can dynamically adjust the retrieval bandwidth, then the throughput of an entire system can be improved. This issue will be demonstrated in Chapter 3.

In an interval, if the two streams have different retrieval bandwidths, then the cache space requirement of this interval and the cache bandwidth requirement of these two streams will be variable. Past interval-level caching algorithms allocate the resource (including the bandwidth and the space) only once for every interval and every stream, and therefore past interval-level caching algorithms clearly cannot handle the case of the variable retrieval bandwidth.

Finally, in the case of the disk caching algorithm RBC, the allocation of the cache bandwidth for a stream is often less efficient. In detail, once an interval is formed, the bandwidth is allocated to the two streams that form that interval. However, the latter stream will not use the bandwidth to read data from the cache until there are available cached data, which are written by the former stream. As the cache bandwidth is crucial, if we can allocate the bandwidth just-in-time instead of by a reservation method, then the caching performance

can be improved. Figure 1.2 shows this idea.

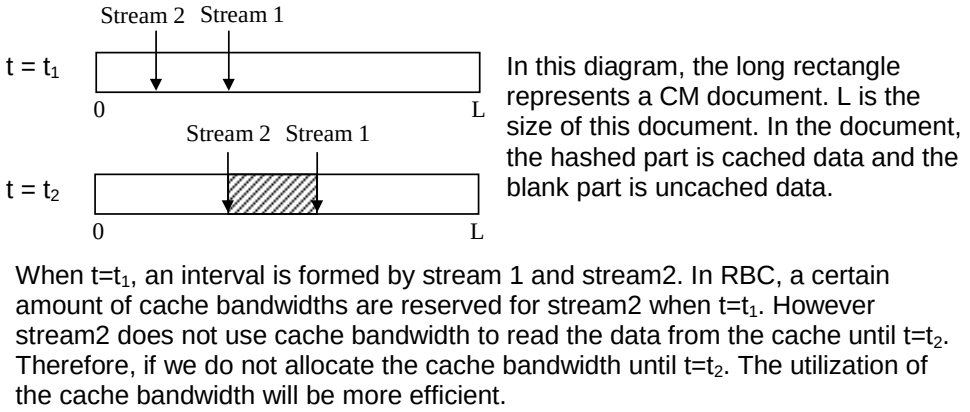


Figure 1.2: Bandwidth requirement for a stream in retrieving a document

1.3 Issues to be Studied and Main Contributions

The above-mentioned issues in Section 1.2 considerably motivate us to propose novel strategies for the retrieval and caching of CM data.

As for the retrieval issue, we realize the multiple-server retrieval by adopting the multiple-channel scheduling. Firstly, we present the partition strategy of the retrieval channel, i.e., allocate the bandwidth from one or multiple servers to form retrieval channels. Secondly. we design a multiple-channel scheduling strategy used in the streaming mode. The retrieval scheduling includes synchronous-channel retrieval scheduling, asynchronous-channel retrieval scheduling, static-size channel scheduling, and variable-size channel scheduling. The static-size channel scheduling is carried out before the playback starts, while the variable-size channel scheduling is carried out dynamically during the playback. The dynamic scheduling prevents some slow channels from missing their deadlines. Also, the dynamic scheduling can reduce the retrieval duration and improve the block ratio of re-

quests. Thirdly, combining the above two parts, a complete *Multiple-Channel Retrieval* (MCR) algorithm is presented. This algorithm can be used in either a single-server retrieval or a multiple-server retrieval. Finally, we compare the multiple-server retrieval and the single-server retrieval by means of rigorous simulation studies. We also compare the multiple-channel retrieval and the single-channel retrieval. Additionally, we implement the multiple-channel and multiple-server retrievals in the experiment. This experiment not only shows the advantage of the multiple-channel and multiple-server retrieval strategies, but also imply the applicability of proposed strategies.

In tackling the caching issue, firstly, we propose some important rules to eliminate switching of streams. Secondly, we design strategies to handle the case of the variable retrieval bandwidth. Thirdly, we design a bandwidth management strategy of caching CM data to improve the efficiency of utilizing bandwidth. Fourthly, combining the above strategies, we propose a novel caching algorithm referred to as the *Variable Bit Rate Caching* (VBRC) algorithm. In this algorithm, the switching operation is avoided to the maximum extent. As far as we know, VBRC is the first caching algorithm that can be used to cache variable bit rate streams. Furthermore, VBRC will be shown to outperform RBC in the case of disk caching. Finally, we carry out plenty of rigorous simulation experiments to show the performances of all the strategies designed in this thesis. This simulation study is shown to testify all our analytical findings.

1.4 Organization of the Thesis

The rest of the thesis is organized as follows. In Chapter 2, we describe the main elements in network-based multimedia system and analyze the significance of our research. Also, we model the retrieval sub-system and the caching sub-system. Besides, we introduce

the terminology and simulation model used in this thesis. In Chapter 3, we design and analyze the strategies for the multiple-server and multiple-channel retrieval. We focus on the channel partition and the retrieval scheduling. Combining all the strategies in this chapter, we present the Multiple-Channel Retrieval (MCR) algorithm, which can be used in either single-server or multiple-server retrieval. In Chapter 4, we design and analyze the strategies for improving past interval-level caching algorithms. We focus on how to reduce the switching operation, how to handle variable bit rate, and how to efficiently utilize the cache bandwidth. By combining all the strategies, we present the Variable Bit Rate Caching (VBRC) algorithm. In Chapter 5, we implement the multiple-channel and multiple-server retrieval, and testify the advantage and applicability of proposed strategies. In Chapter 6, we summarize our research work and discuss on possible extensions.

Chapter 2

System Modeling and Problem Setting

2.1 Network-Based Multimedia System

Network-based multimedia services are provided by network-based multimedia systems or distributed multimedia systems. Li *et al.* [96] defined a distributed multimedia system as a integrated communication, computing, and information system that enables the processing, management, delivery, and presentation of synchronical multimedia information with quality of service guarantees. Furht *et al.* [51] proposed a general architecture of the network-based multimedia system to provide interactive services for thousands of users.

A typical network-based multimedia system consists of three parts: servers, networks, and clients. Servers store CM data (In this thesis, we only consider CM data) and deliver data to clients. When servers provide multimedia services to clients, the requirement of QoS can be described in the following issues.

- *Continuous playback.* In this case, QoS means CM data should be delivered to the client on or before the deadline, which is determined by the playback time of data.

- *Short start-up delay* (short access time). The start-up delay is the waiting time of the clients from the time instant when the client sends a request to the server for requiring a CM document to the time instant when the CM document is played back at the client. The user will not wait for a long time before the playback begins at the client.

- *Low block probability/ratio* (low rejection probability/ratio). A request for a CM document is blocked (i.e., rejected) in the following two cases - (a) servers do not store the requested document, and (b) the bandwidth resource in the server(s) is not sufficient. Under these two cases, the requested CM document cannot be delivered and consequently played back at the client *in a bounded delay.* Clearly, the client does not wish to meet too frequent refusals. Enhancing the throughput of the server can reduce the block ratio. A similar performance metric is the reneging probability, which is the probability that users become impatient and cancel their requests as users experience a long wait time. The reneging probability cannot be calculated unless we determine the reneging behavior of users [121].

- *Short retrieval duration.* The user usually wish that all the requested data are retrieved as soon as possible since a long retrieval process is easily interrupted by the network congestion or other factors.

- *VCR function support.* It is also an important QoS of allowing users to freely control the presentation of CM documents.

Besides, jitter free is also concerned in QoS. Jitter can be easily resolved by buffering, so we do not consider it in this thesis. Besides, scalability and reliability are two important

Table 2.1: Typical storage capacities and bandwidths

Resource	Storage capacity (GB)	Bandwidth (MB/s)
Disk (e.g., Seagate Cheetah 36ES[137])	18.4	63.2
CD (e.g, Philips PCA36XCD)	600MB	5.3
DVD (e.g., Philips PCA532DK)	8.5	6.5
Tape (e.g., Quantum DLT-7000)	35	5
Main Memory (e.g., Micron Crucial	128MB	2.1GB/s
PC2100 DDR-SDRAM memory [108])		
SCSI (e.g., Ultra2 Wide/Ultra 160)		80/160
PCI (e.g, 32-bit/64-bit)		133/266
NIC (e.g., 3COM Gigabit Server NIC)		125
LAN network (e.g., fast/Gigabit Ethernet)		12.5/125

objectives in designing a large-scale multimedia system. However they are not tightly related with our research, as these two objectives are usually considered in constructing a entire multimedia system.

Good or acceptable QoS is not an easy task since available resources are *limited*. The concerned resources are mainly referred to the storage space and the bandwidth capacity in servers. The bandwidth capacity is determined by the minimum of the I/O bandwidth and the network bandwidth. The I/O bandwidth is generally determined by the storage device (e.g., disk, tape, CD, DVD) drives, the SCSI interface, the PCI bus, and the NIC in the computers. Table 2.1 shows the typical values of storage spaces and bandwidths. Obviously, hard disks have good storage capacities and bandwidths, therefore hard disks are

used as main storage devices in distributed multimedia systems. Tertiary storage devices are usually used as the backup of data. Comparing with main memories, disks usually have much larger spaces and smaller I/O bandwidths. A single disk has less bandwidths than the SCSI interface, the PCI bus, and the NIC, therefore, disk arrays are often used instead of a single disk. In this thesis, we do not consider CPU speed as a key resource, as the technological advancement in the design of CPUs has improved at a more rapid pace than memory systems (refer to the figure in [153]). Moreover, the retrieval of data in servers is not a CPU-intense task. Besides, as pointed out in a recent article, the technology of improving the capacity of network bandwidths is more faster than that of CPU and data storage [152]. Thus, comparing with the network bandwidth, the I/O bandwidth easily becomes the bottleneck.

As mentioned in [65], the growth rate of the information content and the traffic on the Internet is much higher than the rate at which memory or disk sizes are likely to grow. For example, the Internet traffic after 1997 doubles every 6 months [130] while the maximum storage space of a single disk doubles only every year, in the past 5 years. In particular, the number of multimedia documents with larger sizes keeps increasing on the Internet. Experts predict that by the year 2005, more than 50% of the information available over the Internet will be large multimedia documents [57]. Also, the percentage of requests to such large-size multimedia documents increases more than a linear fashion with respect to time [101]. Note that the large-size multimedia documents are mainly CM documents. Besides, the increase of the I/O bandwidth is even much slow than the improvement on the storage space because of various limits in IT technologies. Therefore, while both the bandwidth and the storage space are important for achieving an excellent performance, the bandwidth easily becomes the more important bottleneck.

In all, what we observe is that the rate at which client demands on networks for CM documents overtakes the speed at which the storage space and bandwidth technology is progressing. Therefore, *how to efficiently utilize the resources while retrieving and caching to obtain a satisfied QoS* becomes a core concern of this thesis. Also, it is worth mentioning at this juncture that our research is applicable to *various* network-based multimedia systems.

2.2 Retrieval Model

CM data can be retrieved from the server (through a network) using one of two methods - downloading (or called as *play-after-download*) and streaming (or called as *play-while-receive*). In the case of downloading, only after an entire data segment (even an entire document) is received by the client, the playback can begin. This method results in a long waiting time at the client. In the case of streaming, the audio and video content is being delivered to clients while the playback can begin after a short start-up latency (only for buffering to smooth the variable bit rate).

In streaming a CM document, the server can use unicasting (i.e., on-demand), batching, or broadcasting. In this thesis, we consider the on-demand mode of the retrieval. In on-demand, the client can control when the stream is started or stopped, e.g., VoD. When a client sends a request to a server, one or multiple servers delivers the requested document to the client if the request can be accepted from the perspective of resources. The retrieval process excludes the transmission detail in networks.

2.3 Caching Model

In this thesis, we consider the memory caching, where *the main memory is used as the cache of the disk in the same computer*, and the disk caching, where *the disk in a cache server is used as the cache of the disk in an original server*. For a simple description, we adopt two terms - *cache* and *origin* in the caching relation. Hence, in the memory caching, the main memory is the cache and the disk is the origin. In the disk caching, the disk in cache server is the cache and the disk in the original server is the origin.

The connection between the origin and the cache, or between the cache and the client is via networks. In this thesis, for both disk caching and memory caching schemes, we only consider a single cache case while the origin and the client can be multiple, as attempted by other researchers [113, 36, 157, 38]. In other words, we do not consider the choice of caches.

How the client retrieves the requested document from the cache and/or the origin can be described as follows. At the beginning, we search the required document in cache. If there is a miss, we contact the origin for the document.

2.4 Terminology

Some important terms cited in this thesis are listed in Table 2.2. Some common quantities including the symbol and definition in this thesis are presented below in Table 2.3.

Table 2.2: Important terms

Term	Description
Interval	An interval is formed by a pair of consecutive streams (a former stream and a latter stream) to the same document [39]. However, in an interval at a certain time instant, there probably does not exist the former stream (i.e., the former stream has finished its retrieval) and/or the latter stream (i.e., the latter stream has not arrived yet).
Channel	A certain amount of bandwidths that are allocated for delivering CM data.
Cached Interval	An interval has been allocated some cache spaces and bandwidths according to the size of the interval and the retrieval bandwidth of streams, respectively.

2.5 Simulation Model

2.5.1 Performance metrics

In the retrieval problem, we study on how to minimize the access time, the block ratio, and the retrieval duration while satisfying the continuity constraint of the playback.

In the caching problem, the performance metrics are different. The miss ratio [113] is defined as the ratio of the total number of blocks accessed from the origin to the total number of accessed blocks. With different retrieval bandwidths, blocks have different sizes. Therefore, the miss ratio is an inaccurate performance metric to measure the reduction of the network traffic. The hit ratio [157, 38] is defined as the ratio of the total number of documents accessed from the cache and the total number of accessed documents. However, in interval-level caching strategies, only part of an entire document is fetched from the cache.

Table 2.3: Important quantities

Symbol	definition
r	Playback rate of a CM document
L	Size of an entire CM document
H	Length of an entire CM document. $H = L/r$
bw	Retrieval bandwidth (or called bit rate) of a stream
g	Size of an interval. $g = min(g_f(t) - g_l(t), L)$, where $g_f(t)$ is the retrieved size of the former stream and $g_l(t)$ is the retrieved size of the latter stream.
h	Length of an interval. $h = g/r$
R_s	Cache space requirement of an interval. $R_s = min(g, L - G)$, where G is total size of the allocated space for other intervals on the same document.
R_b	Cache bandwidth requirement of a stream. R_b is identical with bw of this stream.

Thus, a measure on the number of "hits" cannot reflect the performance, as different hits may access different amounts of cached data. Therefore, it is not accurate to use a simple hit ratio as the performance metric. Finally, it may be noted that if an entire document is cached, then the time of accessing the cached document is shorter than the time of accessing the same document from the origin. However, when we cache a CM document by interval-level caching strategies, the beginning blocks of a document is often not cached, and hence, we still need to retrieve the initial blocks from the origin. It means that we often cannot obtain the benefit on the access time from the interval-level caching strategy. Therefore, the average access time cannot be an appropriate performance metric.

The design of caching strategies fundamentally aims to reduce the overload on the network,

through which the requested documents are transferred, and the origin. In other words, a *saving* on the bandwidth is treated as the main *benefit* of the caching. Therefore, we evaluate the caching strategies in terms of their abilities to improve the *byte hit ratio*. The byte hit ratio is the ratio of the total number of bytes accessed from the cache to the total number of bytes accessed. Thus, in all our simulation experiments, we analyze the behaviour of the system by measuring the byte hit ratio with respect to several parameters such as, cache size, disk bandwidth, and request arrival rate.

2.5.2 Workload characteristics

In our simulation, requests arrive according to the Poisson distribution with λ, which is the most frequently adopted access pattern.

There are two kinds of probability distributions used to calculate access probabilities (or access rates) of a document i, $i = 1, ..., M$ (M is the number of documents). These are,

$$\text{Zipf distribution [94]:} \qquad p_i = \frac{e^{-\theta i}}{T}, \quad \text{where}, T = \sum_{i=1}^{M} e^{-\theta i} \tag{2.1}$$

and

$$\text{Zipf-like distribution [3]:} \qquad p_i = \frac{i^{-\theta}}{T}, \quad \text{where}, T = \sum_{i=1}^{M} i^{-\theta} \tag{2.2}$$

where, θ is the skew factor of the access probability distribution. In this thesis, we adopt $70-20$ access skew (i.e., about 70% of the accesses are restricted to 20% of the documents) as in [157]. From (2.1) or (2.2), we can calculate θ values corresponding to the $70-20$ access skew when M varies (see Table 2.4). In our simulation experiments, M CM documents are divided into two groups [157]. The size of a group, which is called the *large group*, is uniformly distributed between $125MB$ and $1000MB$, and the size of the other group, which is called the *small group*, is uniformly distributed between $5MB$ and $50MB$. The large

Table 2.4: Skew factor value in the 70-20 access skew case

M	100	200	300	400	500
θ in (2.1)	0.06	0.03	0.02	0.015	0.012
θ in (2.2)	1.012	0.952	0.927	0.911	0.901

group refers to long videos (e.g., movies), whereas the small group refers to short videos (e.g., advertisement clips and MTV). The default size (i.e., the number of documents) of the large group is $P \times M$, where P is the percent of the large group in M documents.

Chapter 3

Multiple-Server/Multiple-Channel Retrieval Strategies

3.1 Why Multiple-Server/Multiple-Channel Retrieval?

In this thesis, a "channel" is defined as a certain amount of bandwidths that are allocated for retrieving CM data. In a single-channel retrieval, only a single channel, which is either a constant bandwidth (i.e., constant-size channel) or a variable bandwidth (i.e., variable-size channel), is used to retrieve a CM document. In the multiple-channel retrieval, multiple independent channels are adopted in the retrieval of a CM document. Thus, the retrieval of a single CM document is carried out with several channels, say, channel 1, channel 2,..., channel N. N may be different for the retrievals of different CM documents, however N must be at least 2 for the multiple-channel retrieval ($N = 1$ for the single-channel retrieval).

We design the multiple-channel retrieval to realize the multiple-server retrieval. In comparison with the single-server retrieval, the multiple-server retrieval has the following advantages.

- *Resource utilization.* In the single-server retrieval, suppose if a server has not enough idle time to support the retrieval of a long-length video, or if it has an inadequate retrieval bandwidth capacity that leads to a unbearable start-up delay, these server resources will be discarded in the case of single-server retrieval. However, in the multiple-server retrieval, such low-bandwidth servers can also participate by carefully considering them during our scheduling process.

- *Reliability.* If multiple servers join the retrieval of a document, then the failure of a single server will not have a fatal affect on the retrieval.

- *Load balancing.* In the single-server retrieval, the server that ensures the minimum access time for that retrieval is not necessarily the server with the lightest load. In the multiple-server retrieval, since the access time is only related to the first retrieved portion and the first retrieval channel, we minimize the access time in channel 1, and also consider to balance the load in other channels in other servers.

Besides the benefit from the multiple-server retrieval, the multiple-channel retrieval has its benefit. In particular, variable network traffics cause the size of the actual retrieval bandwidth to fluctuate with time, therefore most streams will have to re-buffer at some time instants. Because of inferior QoS of streaming CM, some users would like to play the document after downloading it. As a cost, users have to experience a long downloading time. In the case of the multiple-channel retrieval, the possibility of the buffer operation during the playback is greatly reduced.

When multiple channels are used to retrieve a CM document, we need suitable scheduling strategies to achieve good performances.

3.2 Two Kinds of Retrieval Scheduling Strategies

We can adopt two fashions (play-after-download and play-while-receive) to retrieve CM documents. For both of cases, once the playback of the CM document begins in the client, the continuity of the playback should be satisfied until the retrieval is completed. Figure 3.1 shows the timing diagram of the multiple-channel retrieval scheduling when the retrieval in different channels has the same starting time. These channels are *synchronous* channels.

3.2.1 Scheduling strategy in the case of play-after-download

Figure 3.1(a) shows the timing diagram of the multiple-channel retrieval scheduling in the case of play-after-download (refer to [163]). From this figure, we can obtain

$$a_{1,1} = t_0 bw_1, \tag{3.1}$$

and

$$a_{1,j+1} = a_{1,1} \prod_{k=1}^{j} \rho_k, \quad j = 1, ..., n-1, \tag{3.2}$$

where,

$$\rho_k = bw_{k+1}(\frac{1}{bw_k} + \frac{1}{r}).$$

Also, we can obtain

$$a_{i+1,1} = \frac{\sum_{k=1}^{n} a_{i,k} bw_1}{r}, \quad i = 1, ..., m-1, \tag{3.3}$$

and

$$a_{i+1,j+1} = a_{i+1,j}\rho_j - \frac{a_{i,j} bw_{j+1}}{r}, \quad i = 1, ..., m-1, \ j = 1, ..., n-1. \tag{3.4}$$

Using (3.2), (3.3), and (3.4), we can derive $a_{i,j}$ as follows.

$$a_{i,j} = a_{1,1} f_{i,j}(bw_1, ..., bw_n, r), \quad i = 1, ..., m, \ j = 1, ..., n, \tag{3.5}$$

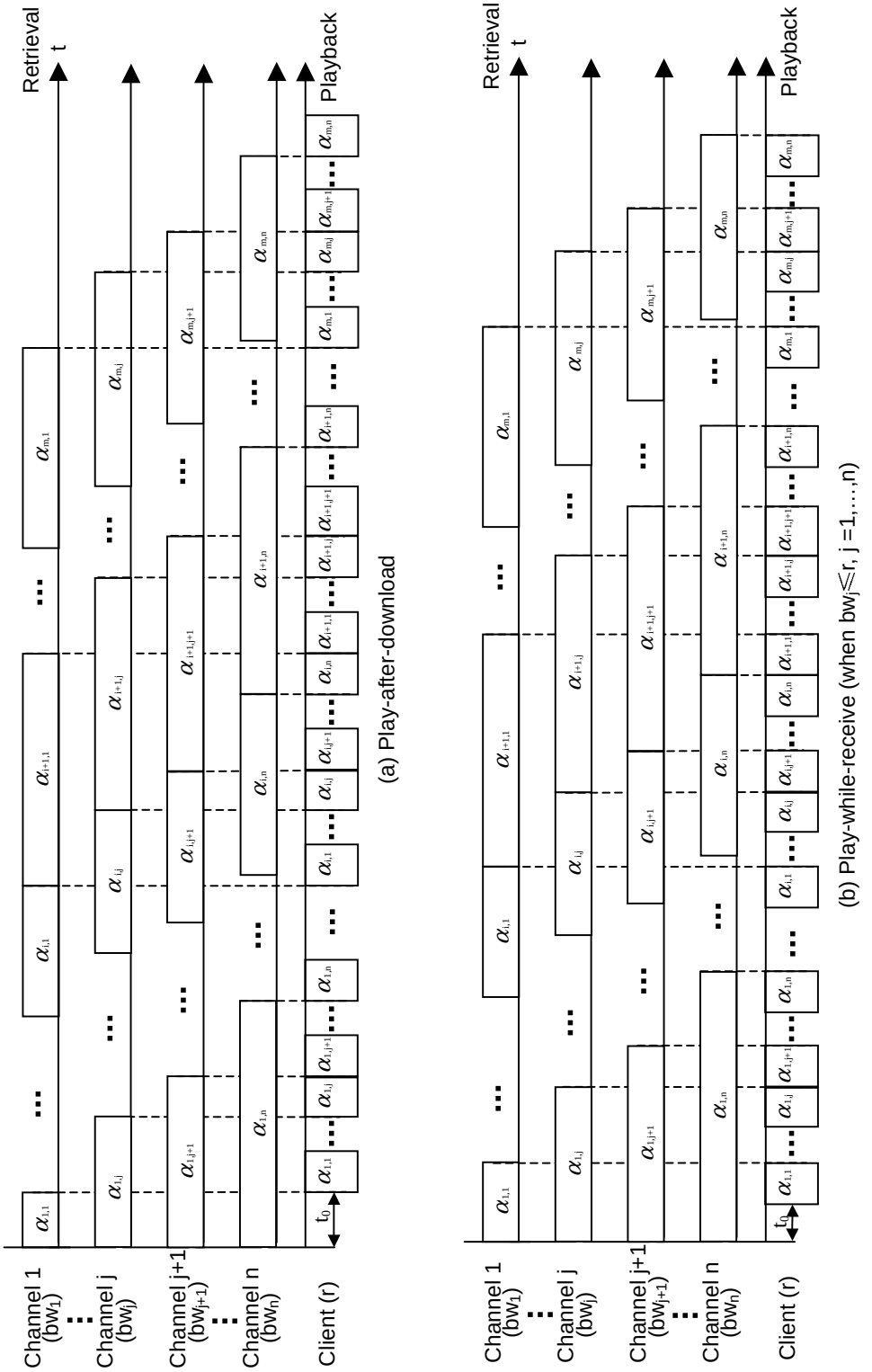


Figure 3.1: Timing diagrams of the multiple-channel retrieval scheduling

where, $f_{i,j}(bw_1, ..., bw_n, r)$ is the function of variables $bw_1, ..., bw_n$, and r.

In addition, we can obtain

$$\sum_{i=1}^{m}\sum_{j=1}^{n} a_{i,j} = L.$$

So,

$$a_{1,1}\sum_{i=1}^{m}\sum_{j=1}^{n} f_{i,j} = L \tag{3.6}$$

Thus, $a_{1,1}$ can be obtained by solving (3.6). Consequently, t_0 and $a_{i,j}$ (for $i = 1, ..., m,\ j = 1, ..., n$) can be obtained by solving (3.1) and (3.5), respectively.

3.2.2 Scheduling strategy in the case of play-while-receive

In the case of play-while-receive, when $bw_j \leq r$ (for $j = 1, ..., n$), the timing diagram of the multiple-channel retrieval scheduling is shown in Figure 3.1(b). From this figure, we can obtain

$$a_{1,1} = \frac{t_0 r bw_1}{(r - bw_1)}, \tag{3.7}$$

and

$$a_{1,j+1} = a_{1,1}\prod_{k=1}^{j}\beta_k, \quad j = 1, ..., n-1, \tag{3.8}$$

where,

$$\beta_k = \frac{r bw_{k+1}}{bw_k(r - bw_{k+1})}.$$

Also, we can obtain

$$a_{i+1,1} = \frac{\sum_{k=2}^{n} a_{i,k} bw_1}{r - bw_1}, \quad i = 1, ..., m-1, \tag{3.9}$$

and

$$a_{i+1,j+1} = a_{i+1,j}\beta_j - \frac{a_{i,j+1}\beta_j bw_j}{r}, \quad i = 1, ..., m-1,\ j = 1, ..., n-1. \tag{3.10}$$

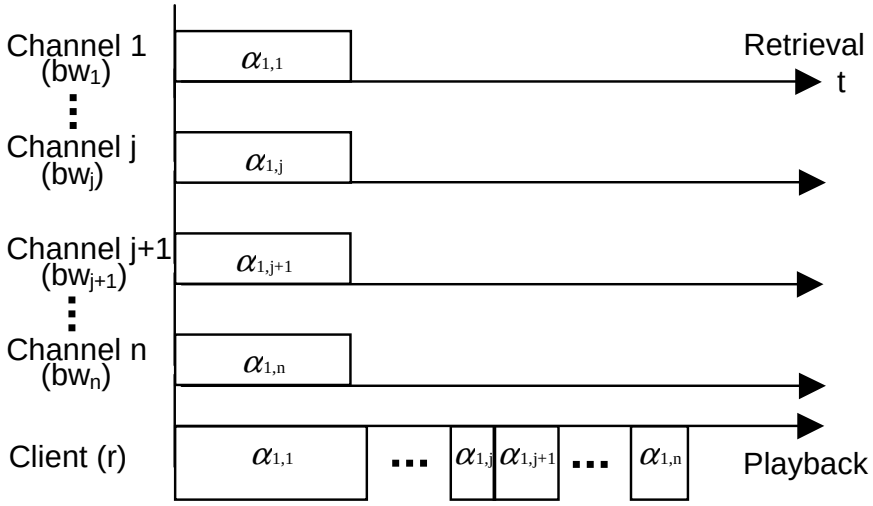


Figure 3.2: Timing diagram of the multiple-channel retrieval scheduling in the case of play-while-receive (when $bw_1 \geq r$)

Using a similar method used in the case of play-after-download, we can obtain t_0 and $a_{i,j}$ (for $i = 1, ..., m,\ j = 1, ..., n$).

Figure 3.2 shows the retrieval scheduling of one case when there is at least one channel whose size is not less than r. We choose this channel as the first retrieval channel for minimizing the access time. Thus, the access time is already zero even using single-installment (i.e., $m = 1$). Moreover, the multiple-installment retrieval induces more operation overheads than the single-installment retrieval. Therefore, in this case, we only use the single-installment retrieval. Thus, we can obtain

$$a_{1,j} = \frac{Lbw_j}{\sum_{k=1}^{n} bw_k}, \quad j = 1, ..., n, \tag{3.11}$$

which ensures every channel completes the retrieval task at the same time instant.

3.2.3 Comparison between two scheduling strategies

Lemma 3.1 *In both play-after-download and play-while-receive strategies,* $a_{1,1}$ *is non-increasing when* m *increases.*

Proof. The proofs for play-after-download and play-while-receive strategies are similar, so we only consider the play-after-download strategy.

Firstly, (3.5) shows that $f_{i,j}(bw_1, ..., bw_n, r) \geq 0$ for $i = 1, ..., m$ and $j = 1, ..., n$. Secondly, (3.4) shows $a_{i+1,j+1}$ depends on $a_{i+1,j}$ and $a_{i,j}$, and $a_{i+1,j+1}$ is independent with m. Hence, $\sum_{i=1}^{m} \sum_{j=1}^{n} f_{i,j}$ is non-decreasing when m increases. Therefore, according to (3.6), $a_{1,1}$ is non-increasing when m increases. ■

Lemma 3.1 means that $\{a_{1,1}(m)\}$ is a monotone sequence with respect to m. Every monotone sequence has a limit [7]. However, the expression of $a_{1,1}(m)$ is difficult to be formulated, hence the close form of the limit of $\{a_{1,1}(m)\}$ and the access time t_0 are difficult to be obtained.

If n channels comes from the same server, then we can treat this kind of multiple-channel retrieval as a single-channel retrieval. In this case the access time t_0 can be calculated as follows (refer to Figure 3.3).

$$t_0 = \begin{cases} L(\frac{1}{\sum_{k=1}^{n} bw_k} - \frac{1}{r}) & \text{for } r \geq \sum_{k=1}^{n} bw_k \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

(3.12) can be treated as the lower bound of the access time in cases of both play-while-receive and play-after-download strategies. Thus, we have the following two suppositions.

- **Supposition 3.1** *For both play-while-receive and play-after-download strategies, if* $r \geq \sum_{k=1}^{n} bw_k$, *then,* $\lim_{m \to \infty} t_0 = \frac{L}{\sum_{k=1}^{n} bw_k} - \frac{L}{r}$.

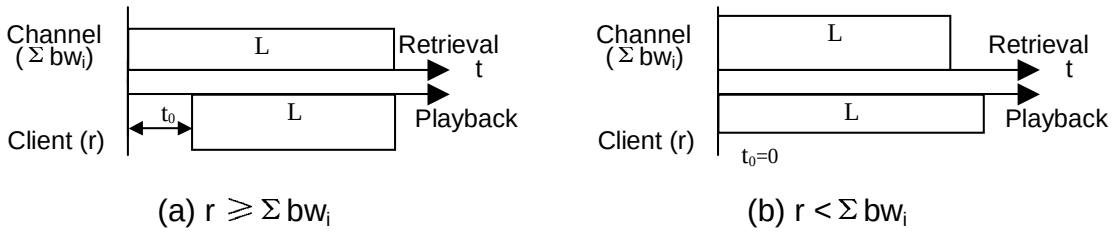


Figure 3.3: Timing diagrams of the single-channel retrieval scheduling in the case of play-while-receive

- **Supposition 3.2** *For both play-while-receive and play-after-download strategies, when* $r < \sum_{k=1}^{n} bw_k$, *then,* $\lim_{m\to\infty} t_0 = 0$.

We carried out a numerical experiment to compare the access time of play-after-download and play-while-receive strategies. Simultaneously, we want to use this experiment to testify the above two suppositions. In this experiment, the CM document size is $2GB$, the playback rate is $0.5MB/s$, every channel has the same size of $0.2MB/s$, the number of servers and channels varied from 1 to 10. The experiment result shown in Figure 3.4 - 3.7 testifies our analysis in Lemma 3.1, Supposition 3.1, and Supposition 3.2. All the figures show that there are less access times by using the play-while-receive strategy than the play-after-download strategy when an identical number of installments are adopted. When the number of installments or channels is very large, the performance of play-after-download is close to that of play-while-receive.

In practice, the retrieval duration of every portion, i.e., $\frac{a_{i,j}}{bw_j}$, cannot be chosen to be smaller than operation overheads, otherwise it is not efficient. We assume the minimum duration of a portion is δt.

Theorem 3.1 *There are less access times by using the play-while-receive strategy than the play-after-download strategy when minimum duration of* $a_{1,1}$ *is considered.*

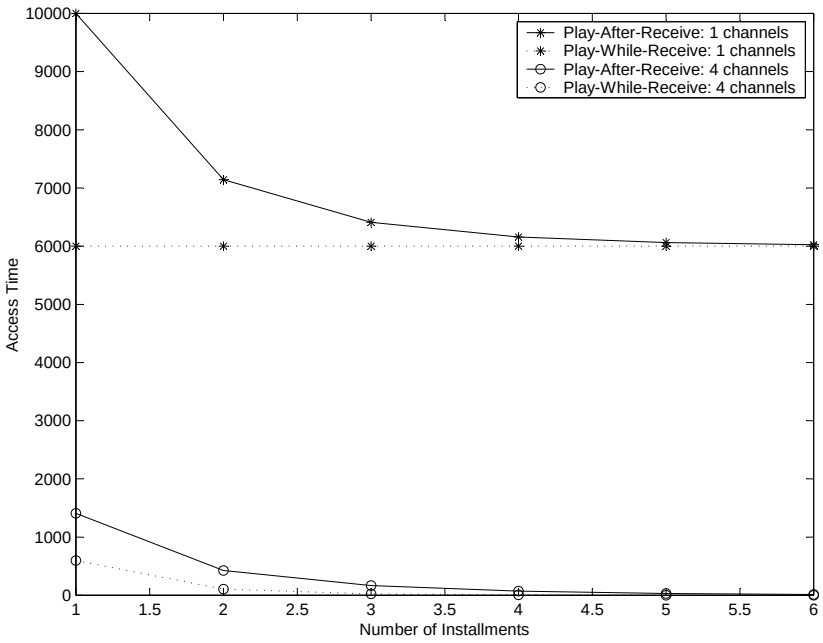


Figure 3.4: Comparison between play-after-download and play-while-receive (1)

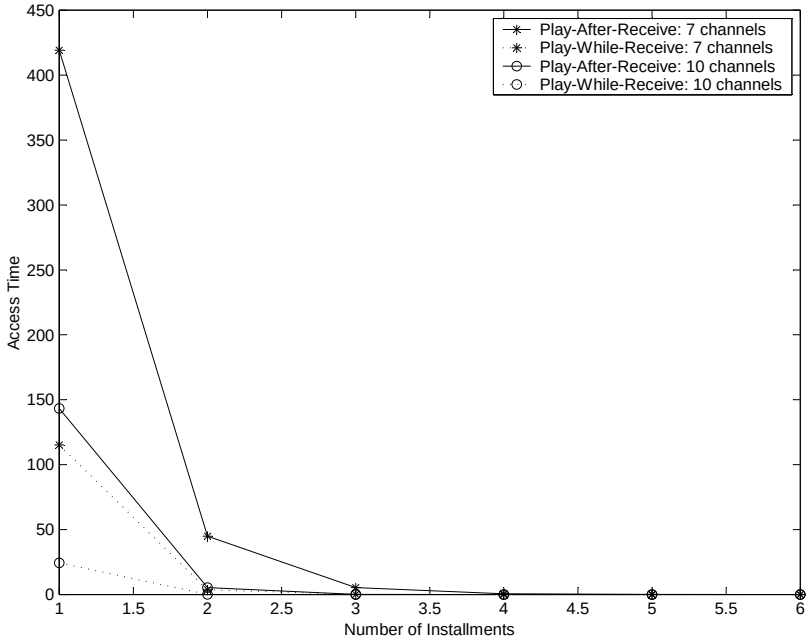


Figure 3.5: Comparison between play-after-download and play-while-receive (2)

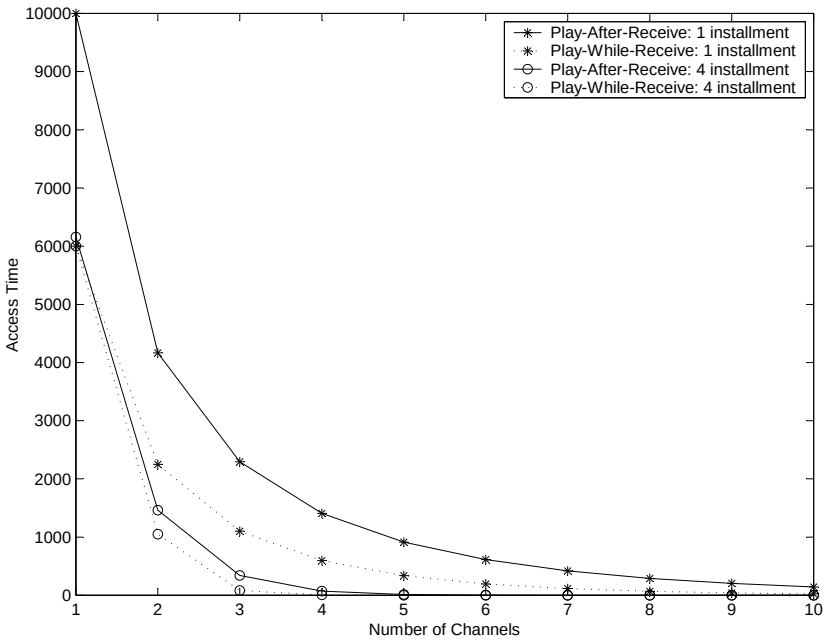


Figure 3.6: Comparison between play-after-download and play-while-receive (3)

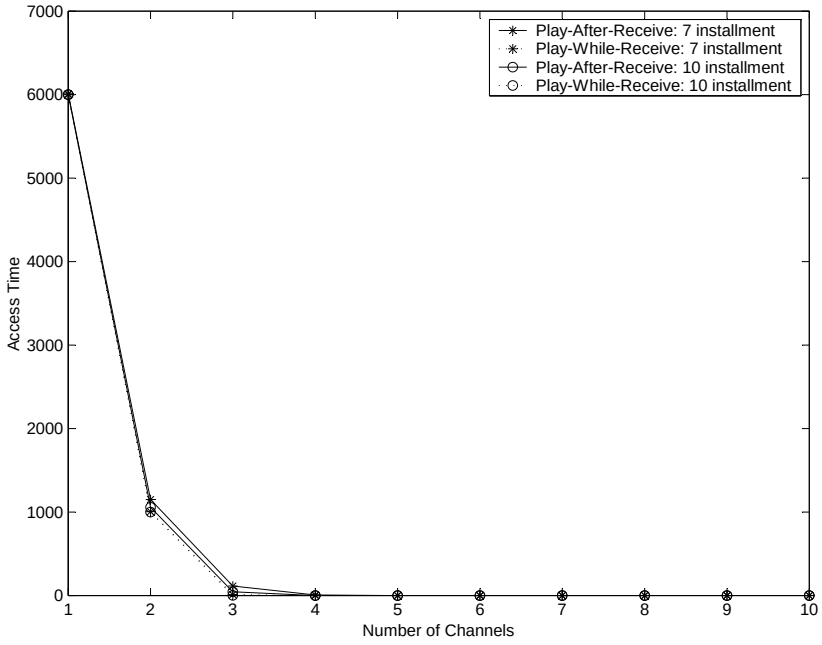


Figure 3.7: Comparison between play-after-download and play-while-receive (4)

Proof. Using the play-after-download strategy, the access time is calculated from (3.1), i.e.,

$$t_0 = \frac{a_{1,1}}{bw_1}.$$

When $\frac{a_{1,1}}{bw_1} = \delta t$,

$$t_0 = \delta t. \tag{3.13}$$

Using the play-while-receive strategy, the access time is calculated from (3.7), i.e.,

$$t_0 = \begin{cases} \frac{a_{1,1}(r-bw_1)}{rbw_1} & \text{for } r \geq bw_1 \\ 0 & \text{otherwise.} \end{cases}$$

When $\frac{a_{1,1}}{bw_1} = \delta t$,

$$t_0 = \begin{cases} \frac{\delta t(r-bw_1)}{r} < \delta t, & \text{for } r \geq bw_1 \\ 0 & \text{otherwise.} \end{cases} \tag{3.14}$$

By comparing (3.13) and (3.14), we prove this theorem. ■

With the above numerical experiments and the theorem, we have shown: when the multiple-channel retrieval is used, the play-while-receive strategy achieves less access times than the play-after-download strategy. Therefore in the remainder of this chapter, we only consider the play-while-receive strategy.

3.3 Asynchronous-Channel Retrieval Scheduling

In Section 3.2, the synchronous-channel case, in which all the channels have identical starting times, has been studied. In this section, we shall further study the retrieval scheduling when all the channels have not identical starting times. In other words, the channels are *asynchronous*. In this case, we only consider the single installment.

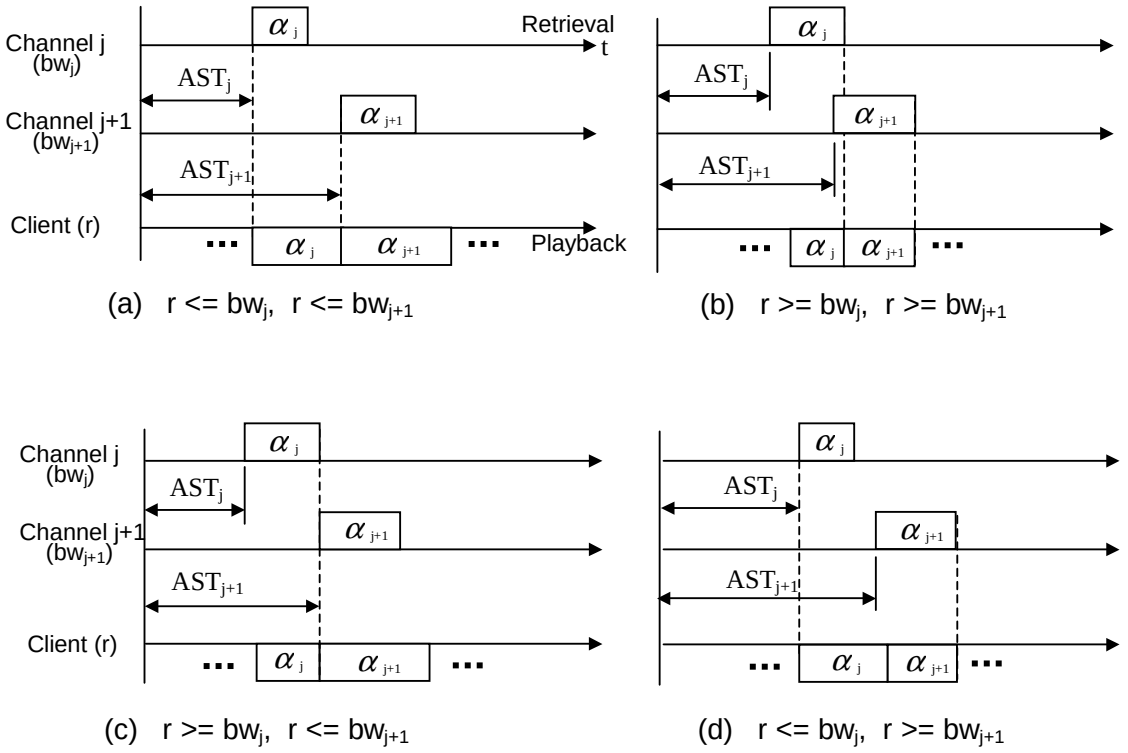


Figure 3.8: Timing diagrams of the asynchronous-channel retrieval scheduling (single installment)

We now derive recursive relationships among the playback rate of a CM document (r), the retrieval bandwidth (bw_j and bw_{j+1}), and the starting times of channels (AST_j and AST_{j+1}).

1. $r \leq bw_j$ and $r \leq bw_{j+1}$ (Figure 3.8(a))

This is the case when the playback rate is less than the retrieval bandwidth.

$$AST_{j+1} - AST_j = \frac{a_j}{r}, \quad j = 1, ..., n-1 \tag{3.15}$$

2. $r \geq bw_j$ and $r \geq bw_{j+1}$ (Figure 3.8(b))

This is the case when the playback rate is greater than the retrieval bandwidth.

$$AST_{j+1} - AST_j + \frac{a_{j+1}}{bw_{j+1}} = \frac{a_j}{bw_j} + \frac{a_{j+1}}{r}, \quad j = 1, ..., n-1 \tag{3.16}$$

3. $r \geq bw_j$ and $r \leq bw_{j+1}$ (Figure 3.8(c))

This is the case when the jth channel has a smaller retrieval bandwidth than the playback rate and the $(j+1)$th channel has a greater retrieval bandwidth than the playback rate.

$$AST_{j+1} - AST_j = \frac{a_j}{bw_j}, \quad j = 1, ..., n-1 \tag{3.17}$$

4. $r \leq bw_j$ and $r \geq bw_{j+1}$ (Figure 3.8(d))

This is the case when the jth channel has a greater retrieval bandwidth than the playback rate and the $(j+1)$th channel has a less retrieval bandwidth than the playback rate.

$$AST_{j+1} - AST_j + \frac{a_{j+1}}{bw_{j+1}} = \frac{(a_j + a_{j+1})}{r}, \quad j = 1, ..., n-1 \tag{3.18}$$

We observe that the continuity relationships during presentation are inherently captured in the above set of equations, involving bw_j, r, and AST_j, for all j values, respectively.

In the asynchronous-channel retrieval scheduling, the access time t_0 can be calculated as follows (refer to Figure 3.9).

$$t_0 = AST_1 + \begin{cases} a_1(\frac{1}{bw_1} - \frac{1}{r}) & \text{for } r \geq bw_1 \\ 0 & \text{otherwise} \end{cases} \tag{3.19}$$

3.4 Channel Partition Strategies

In this section, we propose channel partition strategies, which are about how to allocate bandwidths to form channels. Once the channels are partitioned, a CM document can retrieved from these channels, i.e., channel 1, channel 2, ..., and channel n.

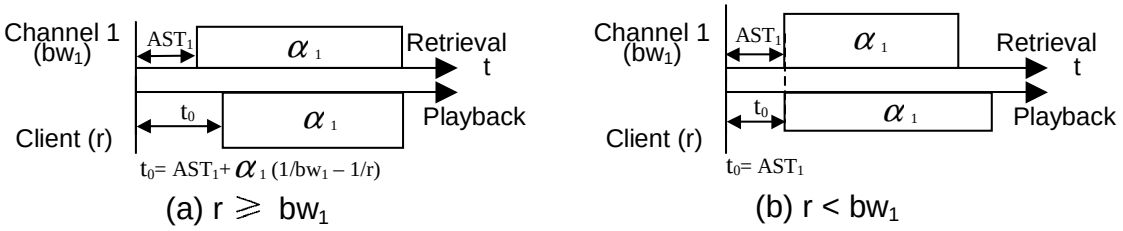


Figure 3.9: Access time in the asynchronous-channel retrieval scheduling (single installment)

(a). Strategy for forming Channel 1

With channel 1, we transmit the first portion of a CM document and hence, channel 1 determines the access time of the CM document. According to Figure 3.9 and (3.19), a small value of AST_1 and a large size of channel 1 are beneficial for minimizing the access time. For obtaining a large-size channel 1, we adopt an *aggregate retrieval bandwidth* from a pool of servers. In other words, a group independent channels, which come from different servers and have identical starting times, form channel 1 by using the scheduling strategy shown in Figure 3.1 (b).

(b). Strategy for forming other channels

According to (3.19), a small value of a_1 helps to reduce the access time. Also, from equations (3.15)-(3.18), a small value of AST_2 aids to decrease the value of a_1. Therefore, we should choose as small AST_2 as possible for channel 2. Besides, our another objective is to minimize the block ratio. To achieve this goal, we try to balance the total load among the servers. The basic idea is to choose other channels (i.e., channel 2, ..., n) in servers with the lightest load. The total number of channels in retrieving the CM document is not strictly limited. However, two channels are at least needed. Channel 1 is used to realize the goal of minimizing the access time, and channel 2 aids in balancing the server load.

3.5 Variable-Size Channel Retrieval Scheduling Strategies

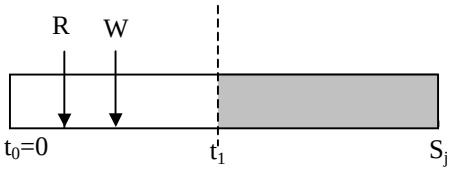
Sections 3.2 and 3.3 present the *constant-size* channel retrieval scheduling, which is carried out before the playback begins and hence called *static scheduling*. In this section, we consider the *variable-size* channel retrieval scheduling, which is carried out during the retrieval process and hence called *dynamic scheduling*. A variable-size channel is needed in the following two cases.

- If a certain retrieval session is stopped by its users, or the latter part of the required document can be retrieved from the cache server, then the previously allocated bandwidth can be reclaimed. These extra bandwidths are not expected at the time when we carry out the static scheduling.
- Because of the reasons from servers or networks, a channel is probably "freezed". Thus, the related stream may miss its deadline, i.e., the playback overtakes the retrieval. Thus, the continuous playback at the client cannot be satisfied.

In the section, we study the strategies of variable-size channel retrieval scheduling.

3.5.1 Retrieval strategy for ensuring the continuous playback

If every channel has a non-varying size, then no retrievals will miss their deadline, as is ensured by the static scheduling . When the retrieval bandwidth of a stream probably varies with time, we need the dynamic scheduling strategy to ensure that the retrieval of every channel will not miss their deadlines. The idea is to let idle channels (extra bandwidths) help channels that will probably miss their deadlines. In detail (refer to Figure 3.10), when the channel i completes its retrieval task, it will help another channel to retrieve data. Firstly, we find all the channels that will probably miss the deadline. Of these channels, we further find the channel with the earliest deadline, e.g., channel j. Secondly, we assume



Suppose that according to the retrieval bandwidth and the playback rate, the playback stream R will overtake the retrieval stream W at t_1. For preventing the starvation of the playback stream, we let other channels to share the load with W, i.e., other idle channels retrieve the portion colored by grey.

Figure 3.10: Retrieval strategy for ensuring a continuous playback

that the size of not retrieved data in channel j is s_j. We also assume that t_1 is the instant when the playback stream will overtake the retrieval stream in channel j. We let channel i help the channel j to retrieve the final portion with the data amount of $s_j - bw_j(t_1 - t_0)$, where t_0 is the current instant and bw_j is the current retrieval bandwidth of the channel j.

3.5.2 Retrieval strategy for improving the block ratio

If there are rich available bandwidths, then we usually form a large-size (i.e., more than the playback rate) channel to retrieve data, so that the bandwidth is fully utilized and the retrieval duration is shortened. However the after effect is that late arriving requests cannot be supported with sufficient bandwidths. Thus, the block ratio may become worse. An intuitive solution is when the new requests arrive, the size of previous allocated channels is reduced. If so, a new problem may appear: the change of a channel size means the change of the previous schedule, and it will probably affect the continuous playback of documents that are being retrieved. So, we propose a novel strategy. With this strategy, the bandwidth resource can be dynamically utilized, also, the continuous playback in previous scheduling is still be ensured. Our strategy is described as follows.

- The maximum size of a channel is equal to the playback rate.

- Extra available bandwidths can be used to help any partitioned channel to retrieve data, however extra available bandwidths will not be considered in the retrieval scheduling.

Thus, we can flexibly allocate the extra available bandwidths to partitioned channels. Extra available bandwidths will speed the retrieval in partitioned channels. This strategy can be used by servers to improve the block ratio.

3.5.3 Retrieval strategy for shortening the retrieval duration

Assume there are n channels used for retrieving a requested document. When channel i (with a size of bw_i) has completed its retrieval task. Firstly, we find out the channel with the longest retrieval task, e.g., channel j (with a size of bw_j). We assume that the size of not retrieved data in channel j is s_j. Secondly, we let channel j share the retrieval task with channel i, i.e., channel i retrieves the amount of $\frac{s_j bw_i}{bw_i+bw_j}$, and channel j retrieves the amount of $\frac{s_j bw_j}{bw_i+bw_j}$. Thus, channel i and channel j are expected to stop retrieving at the same instant. This strategy is similar to the strategy used to ensure the continuous playback. As the continuous playback is more important than reducing the retrieval duration, we will not carry out this strategy unless the retrieval of every channel will not miss the deadline.

3.6 Multiple-Channel Retrieval Algorithm

The problem we are attacking can be precisely described as follows at this juncture. There are a group of CM documents requested by respective users. The problem is to determine the exact (optimal) sizes of the portions to be retrieved from the respective servers in such a fashion that the access time, the block ratio, and the retrieval duration are minimized while ensuring the continuous playback of CM documents. To achieve this goal, we propose

the following algorithm.

Multiple-Channel Retrieval (MCR) Algorithm

Step A. Sort the CM documents in the order of importance.

Step B. Form n channels that can be used to retrieve every CM document. Every channel has a AST_j (i.e., starting time of the retrieval).

Step C. Derive the schedule constraints, which include,

I . The temporal constraints imposed by the availability of channels (namely the values of AST_j), i.e., (3.15)-(3.18);

II . The constraint of the CM document size. $\sum_{j=1}^{n} a_j = L$

Step D. Calculate a_j, $j = 1, ..., n$ using the derived equations in Step C. On the whole, we have $n - 1$ constraints of type I and one constraint of type II. Thus, we have n equations involving n unknowns (a_j values). Hence, we can obtain optimal sizes of the retrieved portions of the CM document.

Step E. Once we obtain a_j values, we can calculate the access time of the CM documents by using (3.19). If the access time is less than allowed maximum access time, the CM document will be retrieved according to the schedule, otherwise the request is blocked.

Step F. In the process of retrieving CM documents, we adopts the variable-size channel strategies shown in Section 3.5.

Now we shall demonstrate the scheduling process through a numerical example presented below. Example 3.1 clarifies all the calculations in detail and is presented for the ease of understanding.

Example 3.1**.** Assume that there a request for a video and we have 3 channels. Their

Table 3.1: Known parameters (before calculation) in Example 3.1

Parameter	Value (MB/s)	Parameter	Value ($min.$)
bw_1	0.40	AST_1	0
bw_2	0.35	AST_2	10
bw_3	0.45	AST_3	40
r	0.42	H	110

parameters are listed in Table 3.1. Then, we can obtain 3 constraint equations for the video. They are the follows.

(1). Since $r > bw_1$, $r > bw_2$, we choose (3.16). Thus, we have,

$$AST_2 - AST_1 + \frac{a_2}{bw_2} = \frac{a_1}{bw_1} + \frac{a_2}{r} \tag{3.20}$$

(2). Since $r > bw_2$, $r < bw_3$, we choose (3.17). Thus, we have,

$$AST_3 - AST_2 = \frac{a_2}{bw_2} \tag{3.21}$$

(3). The size constraint of this video is given by,

$$a_1 + a_2 + a_3 = H \times r \tag{3.22}$$

Using the above set of equations, we obtain the optimal sizes (see Table 3.2) of the portions of the video to be retrieved. Using (3.19), we can calculate the access time: $t_0 = 0.6min.$. Thus, the entire timing diagram is as shown in Figure 3.11.

If we use a single channel 1 to retrieve the video as channel 1 has a minimum AST. The result are: $t_0 = 5.5min.$ The comparison of results shows that the multiple-channel retrieval strategy, in general, is better than the single-channel retrieval. The subsequent simulation experiments will further demonstrate this fact.

Table 3.2: Optimal sizes of the portions in Example 3.1

Parameter	a_1	a_2	a_3
Value (MB)	309	684	1779

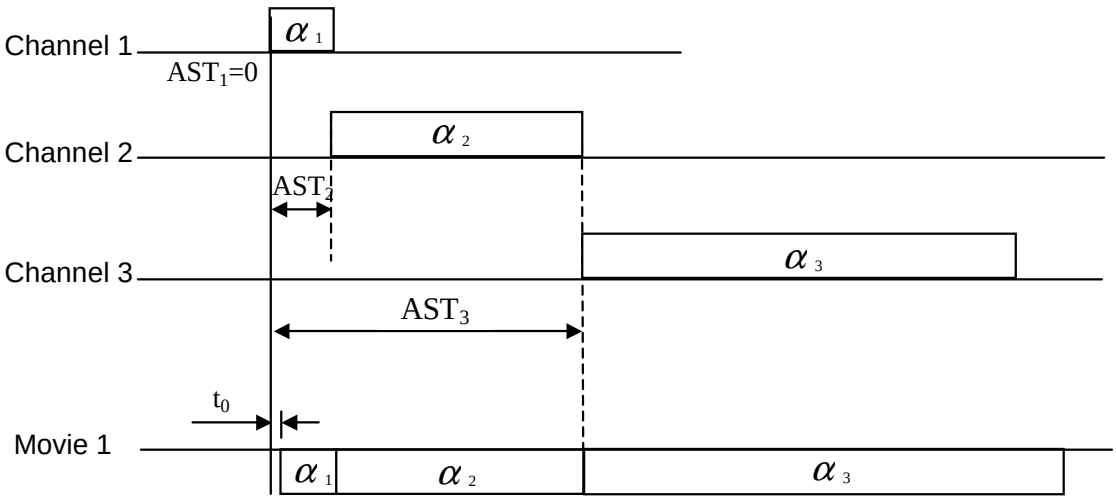


Figure 3.11: Timing diagram for Example 3.1

3.7 Performance Evaluation

In this section, we compare the performance of the single-server retrieval and the multiple-server retrieval.

3.7.1 Simulation test-bed

Clients retrieve CM documents from N_s servers. The performance metrics in our simulation are the average access time and the block ratio under the following system dependent parameters - server bandwidth, arrival rate of request, percentage of large documents, and allowed maximum access time. The system parameters in this simulation are summarized in Table 3.3. At the beginning of the simulation process, the servers have a light load. When more requests come, the servers have a heavy load. What we want to obtain is actually the performance in a variable-load process. As far as we know, in a real-life situation, servers also experience either a light load or a heavy load at different times. Therefore, our simulation acts in accord with the real scenario. Besides, if servers are always under a heavy load, the multiple-server retrieval cannot show its benefit, since *every* server is "busy".

We set $bw_{max} = 2MB/s$ according to our experiences. The transmit bandwidth of servers and the receive bandwidth of client can be larger than $2MB/s$ (refer to the capacities of computer components introduced in Chapter 2). However, the transmission of data from servers to clients is via networks. Through longer network paths, more bottlenecks (in backbone networks, WANs, or LANs) may be met. Hence it is known that the transmission throughput often drops with greater distances.

Table 3.3: System parameters in comparing the single-server retrieval and the multiple-server retrieval

System Parameters	Symbol	Parameter Values
Number of Servers	N_s	10
Number of Different Documents	M	100
Server Bandwidth Size	BW	$10 - 50MB/s$
Skew Factor of Access Prob. Distribution	θ	0.06
Number of Requests	N	500
Arrival Rate of Requests	λ	$0.5 - 5.0s^{-1}$
Playback Rate	r	Uniform distribution in $0.25 - 0.75MB/s$
Percentage of Large Documents	P	$0 - 100\%$
Allowed Maximum Access Time	AT_{max}	$0.2 - 2min$
Allowed Maximum Retrieval Bandwidth for Retrieving a Document in a Server	bw_{max}	$2MB/s$

3.7.2 Simulation result

In our simulation, three retrieval policies are compared, i.e.,

I. Single-server, single-channel, and constant-size channel

II. Single-server, single-channel, and variable-size channel

III. Multiple-server, multiple-channel, and variable-size channel

Figure 3.12 and Figure 3.13 show the access time and the block ratio of Policies I, II, and III, respectively, when the server bandwidth varies. With the increase in the server band-

width, all of three policies have a lower block ratio, as larger bandwidths can accommodate more streams.

Figure 3.14 and Figure 3.15 show the access time and the block ratio of Policies I, II, and III, respectively, when the fraction of requests for larger documents varies. With the increase in the fraction of requests for larger documents, all of three policies have worse block ratios, as to retrieve larger documents need more bandwidth resources.

Figure 3.16 and Figure 3.17 show the access time and the block ratio of Policies I, II, and III, respectively, when the arrival rate of requests varies. With the increase in the arrival rate, all of three policies have worse block ratios, as a larger arrive ratio means more requests comes in the same time while the bandwidth resources are held constant. Therefore, more requests are refused.

Figure 3.18 and Figure 3.19 show the access time and the block ratio of Policies I, II, and III, respectively, when the allowed maximum access time varies. With the increase in the allowed maximum access time, all of three policies keep the unchanged block ratio. In other words, to enhance moderately the allowed maximum access time will not improve the block ratio. Besides, it is obvious that the access time increase when allowed maximum access time increases.

Figures 3.12, 3.14, and 3.16 show that the changes of the server bandwidth, the fraction of requests for the larger documents, and the arrival rate of requests have no obvious correlations with the access time. The reason is we only consider the average access time of *accepted* requests. In addition, Figure 3.12, 3.14, 3.16, and 3.18 show that average access time of accepted requests are similar for Policies I, II, and III.

In terms of the block ratio, Policy III is better than Policies I and II, and Policy II is better

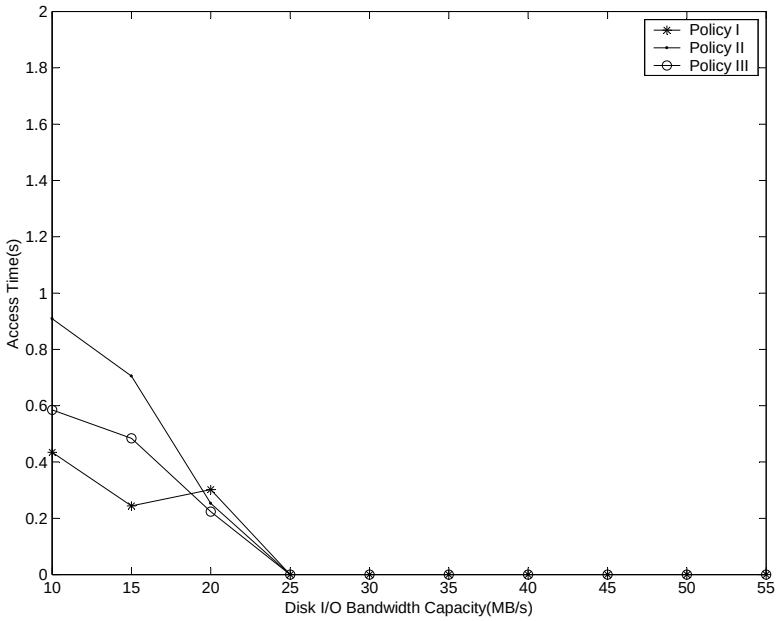


Figure 3.12: Access time of the single-server retrieval and the multiple-server retrieval ($P = 100\%, AT_{max} = 1min.$, and $\lambda = 2s^{-1}$)

than Policy I. It means that the variable-size channel retrieval strategy outperforms the constant-size channel retrieval strategy, and the multiple-server retrieval strategy shows better performances than the single-server retrieval. Through efficient resource utilization and load balancing, the multiple-server retrieval helps obtain smaller access time, consequently, more requests are accepted, i.e., the block ratio is improved.

3.8 Concluding Remarks

Although, the multiple-channel retrieval has been adopted in downloading the document from networks. However their retrieval is simple: every channel retrieves the data portion with an equal size. Khan *et al.* [72] also consider the multiple-channel retrieval of CM data. However our research is different from theirs in two issues. One is we schedule the retrieval of an entire CM document while Khan *et al.* scheduled the retrieval of uniform

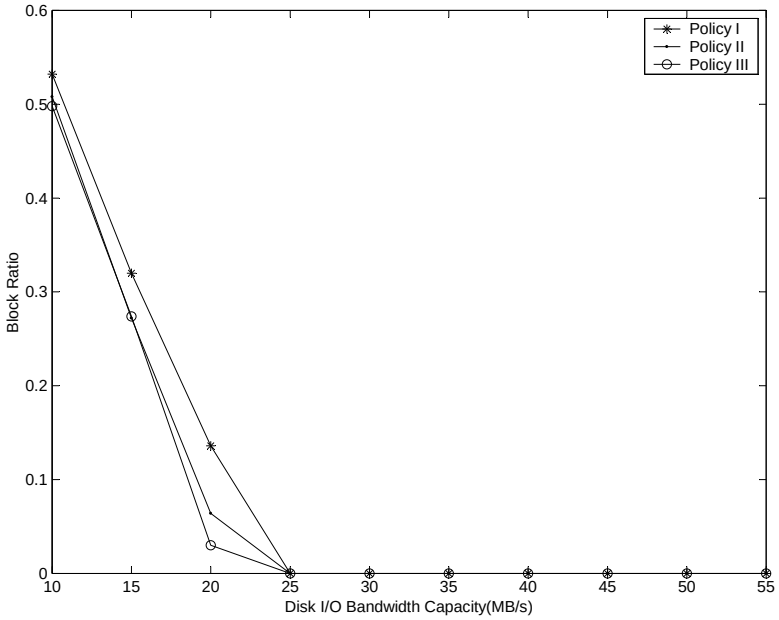


Figure 3.13: Block ratio of the single-server retrieval and the multiple-server retrieval ($P = 100\%, AT_{max} = 1min.$, and $\lambda = 2s^{-1}$)

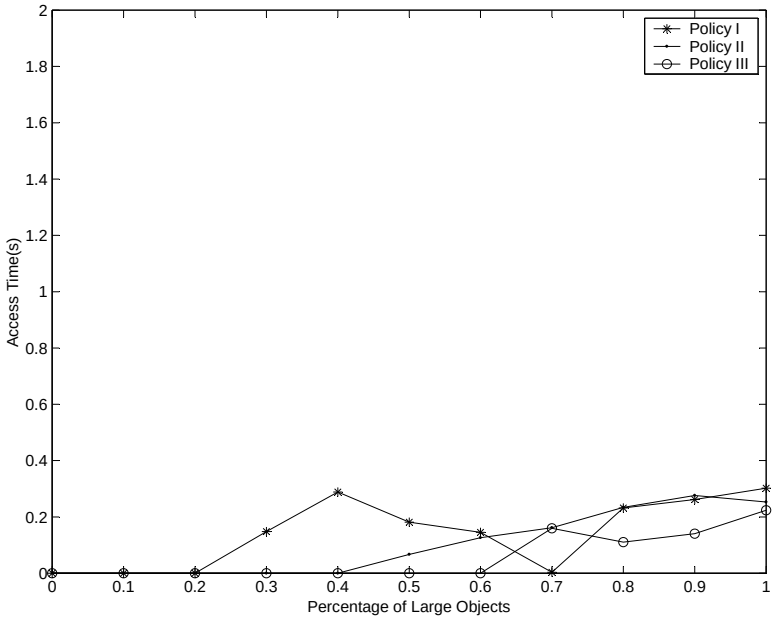


Figure 3.14: Access time of the single-server retrieval and the multiple-server retrieval ($BW = 20MB/s, AT_{max} = 1min.$, and $\lambda = 2s^{-1}$)

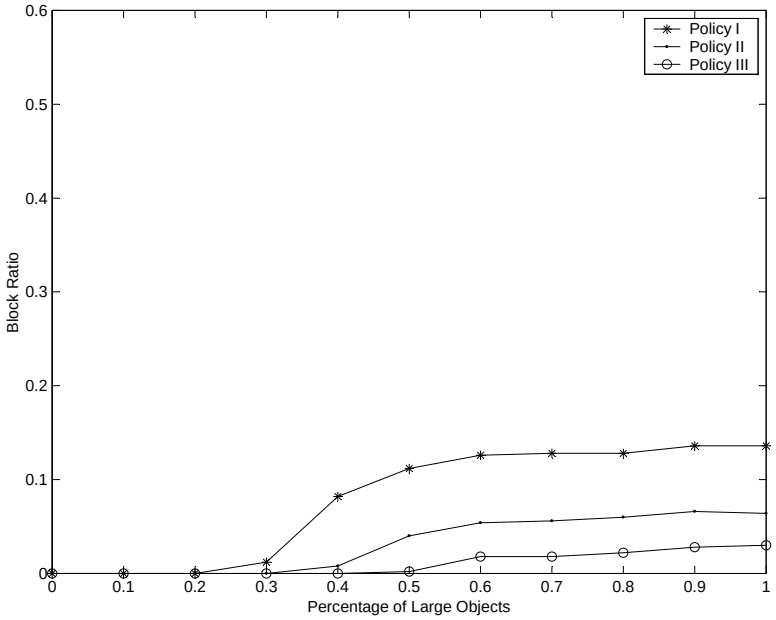


Figure 3.15: Block ratio of the single-server retrieval and the multiple-server retrieval ($BW = 20MB/s, AT_{max} = 1min$, and $\lambda = 2s^{-1}$)

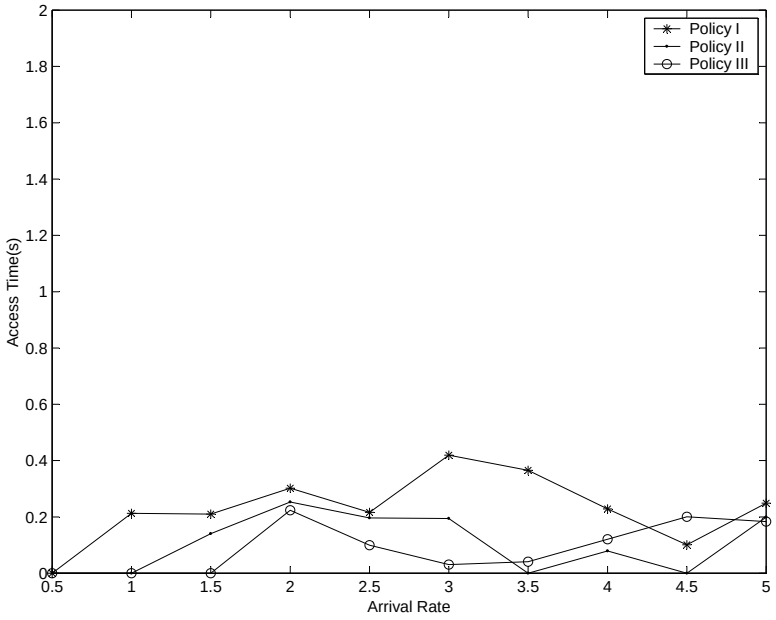


Figure 3.16: Access time of the single-server retrieval and the multiple-server retrieval ($BW = 20MB/s, P = 100\%$, and $AT_{max} = 1min$)

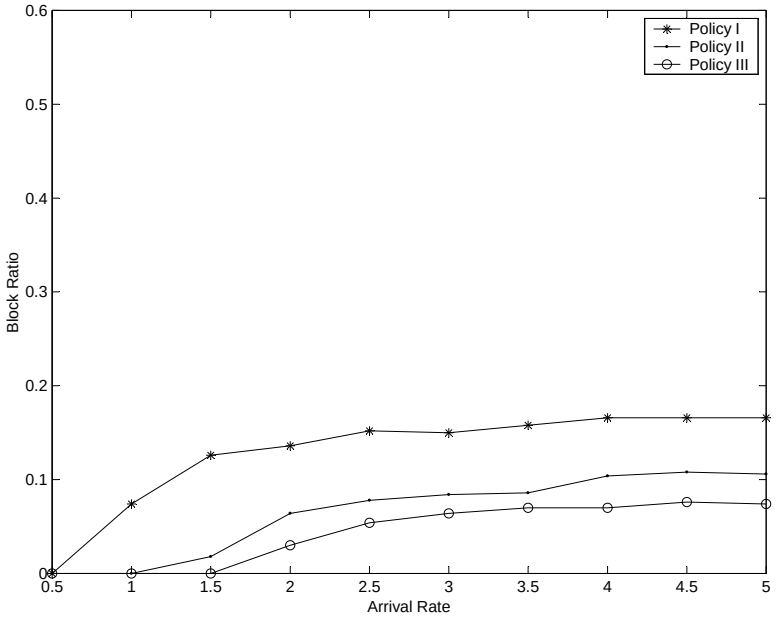


Figure 3.17: Block ratio of the single-server retrieval and the multiple-server retrieval ($BW = 20MB/s, P = 100\%$, and $AT_{max} = 1min.$)

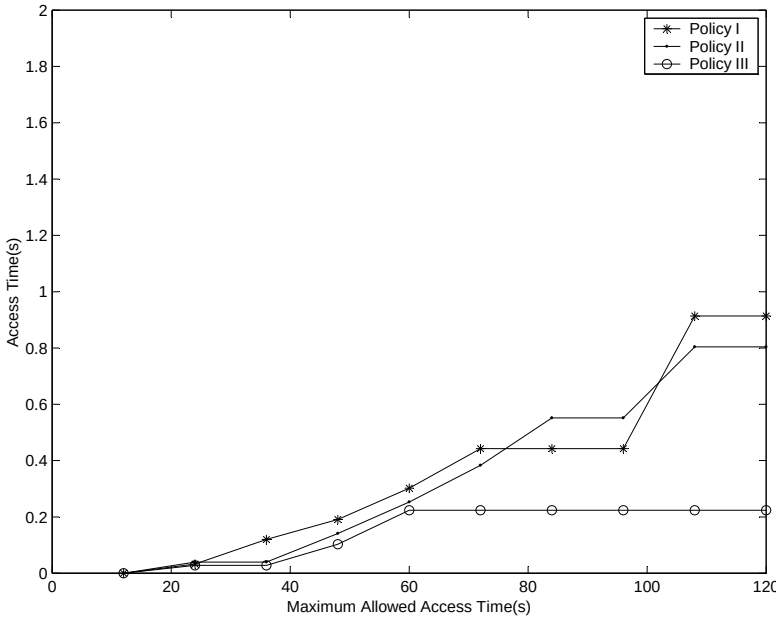


Figure 3.18: Access time of the single-server retrieval and the multiple-server retrieval ($BW = 20MB/s, P = 100\%$, and $\lambda = 2s^{-1}$)

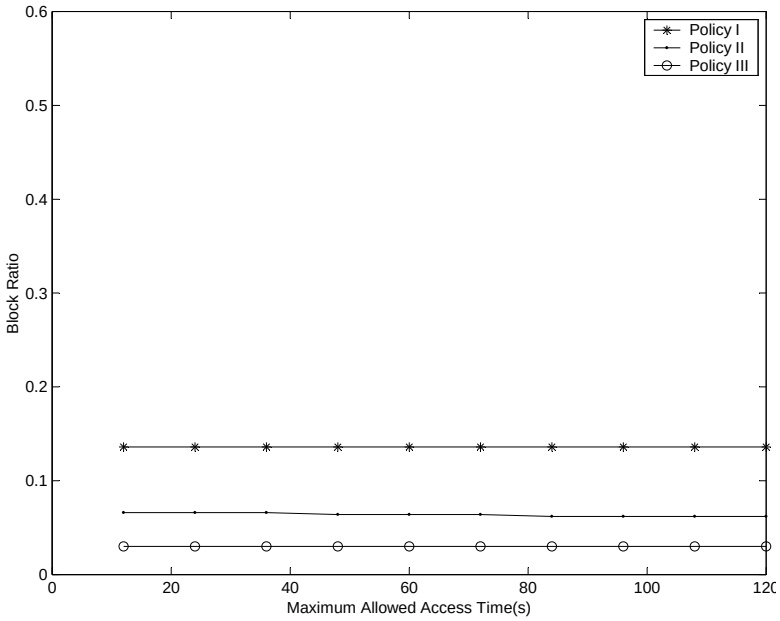


Figure 3.19: Block ratio of the single-server retrieval and the multiple-server retrieval ($BW = 20MB/s, P = 100\%$, and $\lambda = 2s^{-1}$)

units referred as an FMDs (frames of multimedia) [72]. This means that our scheduling is high-level while Khan's scheduling is low-level (disk level). The other one is Khan *et al.* did not concern the channel partition strategy and the variable-size channel. In [72], the channels have been given before the scheduling.

We realize the multiple-server retrieval using the multiple-channel retrieval. In this chapter, we have provided a complete solution for the multiple-server/multiple-channel retrieval problem. A three-step approach has been introduced to deal with a request for the retrieval and the playback an CM document. Firstly, partition the channel from the available bandwidth. Secondly, schedule the retrieval using allocated channels and determine the exact size of the individual disjoint portions that will be retrieved from each of the channels. Finally, apply variable-size channel retrieval strategies to complete the retrieval task. These contents have included in the Multiple-Channel Retrieval (MCR) algorithm. In designing

the retrieval strategies, we synthetically satisfy/improve all of main performance metrics, i.e., continuous playback, block ratio, access time, and retrieval duration, according to their priorities.

Chapter 4

Variable Bit Rate Caching Strategies

In this chapter, firstly, we present three basic strategies. They are,

(1). Caching strategy for the variable retrieval bandwidth

(2). Caching strategy under the non-switch constraint

(3). Allocation strategy of the cache bandwidth

Secondly, we present our caching algorithm that contains the above three strategies.

4.1 Caching Strategy for the Variable Retrieval Bandwidth

In Section 1.2 (about the motivation), we explained why the retrieval bandwidth is often a variable. In this section, we shall present strategies to handle the variable retrieval bandwidth.

In the case of constant retrieval bandwidths, once we allocate the cache bandwidth for a

stream, or allocate the cache space for an interval, we do not need to check the availability of the bandwidth or the space until two streams in the interval finishes their retrieval. When both of streams in the interval finish their retrieval, the allocated bandwidth or space is reclaimed. However, in the case of variable retrieval bandwidth, the cache bandwidth requirement of a stream and the cache space requirement of an interval are time-varying. Therefore we need to check the availability of the bandwidth and the space in the cache every service cycle D. D is tunable, e.g., $D = 1s$. On the one hand, a shorter service cycle results in more operational overheads. On the other hand, a larger service cycle probably cannot keep up with the change of the retrieval bandwidth.

For every service cycle, the availability of the bandwidth and the space in the cache is checked. If the total requirement for the bandwidth ($\sum R_b$) is greater than the cache bandwidth capacity BW, some bandwidths will be reclaimed from some streams using a *Cache Bandwidth Reclaiming* (CBR) strategy, shown in Figure 4.1.

The byte hit ratio is directly affected by the amount of data read from the cache. In comparison, after data are written into the cache, these cached data are probably swapped out of the cache *before* data are read because of insufficient cache spaces. Therefore, a stream that is reading data from the cache is more *important* than a writing stream. As a result, we do not reclaim the bandwidth from a stream that is reading from the cache unless there is no writing stream.

Similarly, if the total requirement for the space ($\sum R_s$) is greater than the cache space capacity SP, some spaces will be reclaimed from some intervals (refer to the *Cache Space Reclaiming* (CSR) strategy in Figure 4.2).

In the case of variable retrieval bandwidth, a following stream probably overtakes a pre-

The stream that is writing data into the cache must be the former stream of an interval. Insert all the writing streams into a queue named Ψ_w in a descending order of the interval size g of their respective intervals.

The stream that is reading data from the cache must be the latter stream of an interval. Insert all the streams that are reading data from the cache into a queue named Ψ_r in a descending order of the interval size g of their respective intervals.

Repeatedly carry out steps 1 and step 2 until ($\sum R_b \leq BW$).

Step 1. When Ψ_w is not empty, pop a stream that is at the head location of Ψ_w and reclaim the allocated cache bandwidth from this writing stream. Simultaneously, the relative interval is swapped out of the cache, i.e., the allocated space for the relative interval is reclaimed.

Step 2. When Ψ_w is empty and Ψ_r is not empty, pop a stream that is at the head location of Ψ_r and reclaim the allocated cache bandwidth from this reading stream.

Figure 4.1: CBR (cache bandwidth reclaiming) strategy

Insert all the cached intervals into a queue named Ψ_i in a descending order of the interval size g.

Repeatedly carry out the following step until ($\sum R_s \leq SP$).

When Ψ_i is not empty, pop an interval that is at the head location of Ψ_i and reclaim the allocated cache space from this interval.

Figure 4.2: CSR (cache space reclaiming) strategy

When $t = t_1$, let us suppose a following stream S_2 exceeds a preceding stream S_1. In other words, when $t > t_1$, the size of the retrieved data by S_2 is greater than S_1. At $t = t_1$, the following steps 1, 2, and 3 are carried out.

Step 1. If, when $t < t_1$, S_1 and S_2 form an interval H_0, then when $t \geq t_1$, S_1 becomes the latter stream of H_0 and S_2 becomes the former stream of H_0.

Step 2. If, when $t < t_1$, S_1 is a latter stream of interval H_1, then when $t \geq t_1$, S_2 becomes the latter stream of H_1.

Step 3. If, when $t < t_1$, S_2 is a former stream of interval H_2, then when $t \geq t_1$, S_1 becomes the former stream of H_2.

Figure 4.3: ERS (exchange strategy for repositioning the streams) strategy

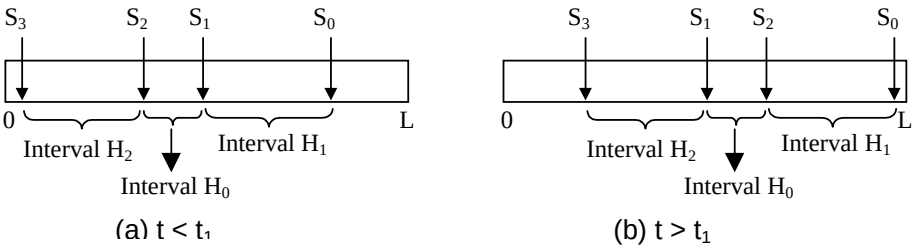
ceding stream which reads/writes the same document. For example, a preceding stream may be paused by a VCR operation. In this case, to handle the repositioning of streams within an interval, we propose a strategy referred to as *Exchange strategy for Repositioning the Streams* (ERS) as shown in Figure 4.3. Figure 4.4 also explains this strategy.

In short, the CBR (cache bandwidth reclaiming) strategy, the CSR (cache space reclaiming) strategy, and the ERS (exchange strategy for repositioning the streams) strategy are specially designed for the case of the variable retrieval bandwidth. In other words, they are not needed in the case of constant retrieval bandwidth. With these strategies, the resource allocation in the case of variable retrieval bandwidth can be handled successfully.

4.2 Caching Strategy under the Non-Switch Constraint

In Section 1.2 (about the motivation), we explained the concept of switching. In detail, a switching operation is due to two reasons.

- Shortage of cached data. A stream that is reading data from the cache cannot find



In this diagram, the long rectangle represents a CM document. L is the size of this document.
When $t<t_1$, S_3 and S_2 form interval H_2, S_2 and S_1 form interval H_0, S_1 and S_0 form H_1. When $t >= t_1$, S_3 and S_1 form interval H_2, S_2 and S_0 form H_1. S_1 becomes the latter stream of interval H_0 and S_2 becomes the former stream of interval H_0.

Figure 4.4: How a stream overtakes another stream

available cached data to read in the cache.

- Shortage of cache bandwidths. The allocated bandwidth for a stream that is reading data from the cache is reclaimed.

Because of the shortage of cached data or cache bandwidths, the stream has to switch back to the origin and begin to read data from the origin.

4.2.1 Influence of the switching operation on the performance

The switching operation affects the performance in two ways. One is the bandwidth reservation. The other is the overhead of switching operations.

Firstly, since streams that are reading data from the cache probably switch back to the origin, a solution of satisfying the playback continuity of these streams is to reserve enough retrieval bandwidths in the origin. However the estimation is very difficult because of the following reasons.

- In the case of disk caching, the cache and the access pattern in the cache are unknown by the origin. When the number of switching operations is very large, the origin

probably cannot accept all the switching streams even the origin reserves all of its bandwidth.

- In the case of memory caching, the disk bandwidth is much less than the memory bandwidth. The number of streams reading data from the memory are probably large, consequently, the number of switching streams is also unbearably high for the disk with a smaller bandwidth.

Secondly, it may be noted that the switching operation causes additional operational overheads. As a block-level caching algorithm, BASIC caches unrelated sets of blocks, and hence, BASIC cannot guarantee a continuous playback. Therefore, BASIC is not applicable for the CM caching. In fact, the actual reason why BASIC cannot guarantee a continuous playback is too many switching operations result in additional operational overheads that are prohibitively high.

4.2.2 Strategies for reducing the switching operation

Since a switching operation worsens the performance, some strategies should be proposed to reduce the probability of switching operations. In this section, a *non-switch constraint* is presented. If the non-switch constraint is satisfied in the caching algorithm, the number of switching operations will be reduced. In the case of constant retrieval bandwidth, the switching operation can be eliminated completely. In the case of variable retrieval bandwidth, the probability of a switching operation will be reduced to a maximum extent.

The following two rules constitute the **non-switch constraint**.

- *Allocated retrieval bandwidths for a stream that is reading from the cache will not be released until the current retrieval finishes. Thus, once a stream begins to read from the cache, the stream will never switch back to the origin because of the shortage of bandwidths.*

- *When a stream is reading the data from the cache, there is at least one preceding stream, which is writing the data into the cache, or all the data of the requested document have been cached. Thus, a stream can always read available data from the cache, and the stream will not switch back to the origin because of the shortage of cached data.*

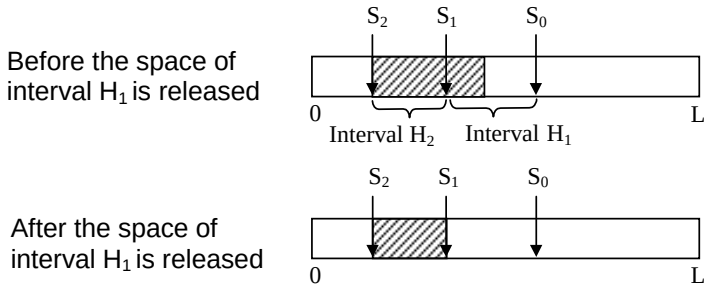
The above two rules inherently impose some limits in retrieving the data and the consumption of bandwidths associated with the cache. Through satisfying the non-switch constraint, the switching operation is eliminated or reduced, and therefore, the bandwidth of servers and networks is saved to a significant extent. Consequently, the acceptance ratio of requests in the entire system will be improved.

If the non-switch constraint is imposed in the CBR (cache bandwidth reclaiming) strategy (Figure 4.1) and the CSR (cache space reclaiming) strategy (Figure 4.2), then, the following should be observed in reclaiming strategies.

- Consider two cases. One is the space of the interval, in which the latter stream is *not* reading data from the cache, is reclaimed. The other is the space of the interval, in which the latter stream is reading data from the cache, is reclaimed. The second case must cause a switching operation, hence we should handle the first case before the second case.

- Consider two consecutive cached intervals, H_1 and H_2. S_1 is the latter stream of H_1 and the former stream of H_2. If the space of H_1 is reclaimed, then we try to make S_1 be a writing stream if the available bandwidth is sufficient. The caused benefit is that the stream S_2, which is the latter stream of H_2, will not carry out a switching operation in the future. Figure 4.5 provides a detailed explanation.

- The bandwidth of a writing stream is reclaimed before the bandwidth of a reading



In this diagram, the long rectangle represents a CM document. L is the size of this document. In the document, the hashed part is cached data and the blank part is uncached data. S_0 and S_1 form interval H_1, S_1 and S_2 form interval H_2. When interval H_1 is released space, S_1 is changed to a writing stream, so that S_2 can avoid the switch operation in the future.

Figure 4.5: Change from a reading stream to a writing stream

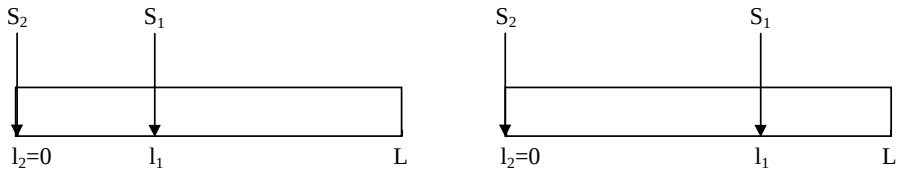
stream that is reading data from the cache, so that a switching operation is avoided to the maximum extent for the stream that is reading data from the cache.

All the above strategies aim to reduce the possibility of switching operations.

4.3 Allocation Strategy of the Cache Bandwidth

Before presenting a better allocation strategy of the bandwidth, the bandwidth requirement in the case of disk caching is analyzed. Figure 4.6 and Figure 4.7 show the bandwidth requirement for an interval and two consecutive intervals, respectively. For a simply description, the analyzed examples adopt constant retrieval bandwidths.

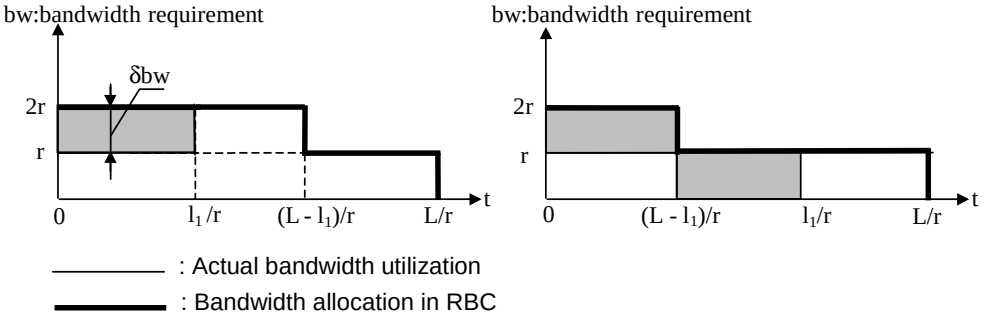
In RBC, when an interval is formed, the bandwidth is allocated for the former stream and the latter stream. However, Figure 4.6 and Figure 4.7 show the allocated bandwidth for a stream is not actually utilized until this stream start to read cached data. If we allocate the cache bandwidth when it is actually needed by the stream, then the bandwidth resource can be considerably saved. Our caching algorithm adopts this kind of so-called *just-in-time*



(a) $l_1 - l_0 \leq L/2$ (t=0)

(b) $l_1 - l_0 > L/2$ (t=0)

The diagrams (a) and (b) show two different cases. In (a) and (b), the long rectangle represents a CM document. L is the size of this document. S is the stream. S_1 and S_2 form an interval.

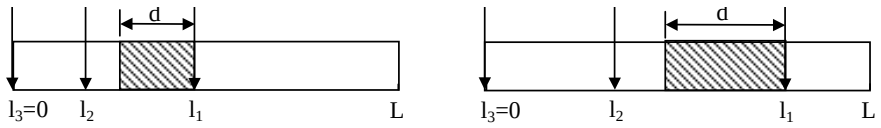


(c) $l_1 - l_0 \leq L/2$

(d) $l_1 - l_0 > L/2$

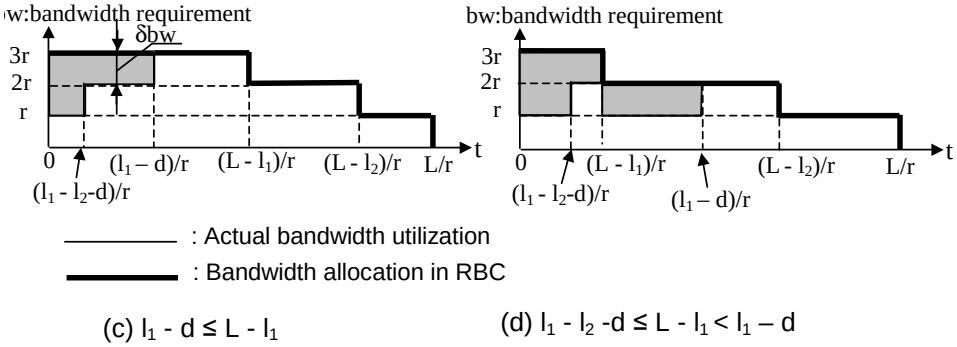
The diagrams (c) and (d) compare the bandwidth allocation in RBC and actual bandwidth utilization, The diagram (c) shows the case of (a), and the diagram (d) shows the case of (b). For a simple description, (c) and (d) show the case of constant retrieval bandwidth. Also the retrieval bandwidth is r, which is playback rate of the document.

Figure 4.6: Bandwidth requirement for an interval



(a) $l_1 - d \le L - l_1$ (t=0) (b) $l_1 - l_2 - d \le L - l_1 < l_1 - d$ (t=0)

The diagrams (a) and (b) show two cases. Because of limited spaces, we ignore the third case: $L - l_1 \le l_1 - l_2 - d$. S is the stream. S_1 and S_2, S_2 and S_3 form two consecutive intervals.



(c) $l_1 - d \le L - l_1$ (d) $l_1 - l_2 - d \le L - l_1 < l_1 - d$

The diagrams (c) and (d) compare the bandwidth allocation in RBC and actual bandwidth utilization. The diagram (c) shows the case of (a), and the diagram (d) shows the case of (b). For simplicity, (c) and (d) show the case of constant retrieval bandwidth. Also the retrieval bandwidth is r, which is playback rate of the document.

Figure 4.7: Bandwidth requirement for two consecutive intervals

method instead of the reservation method employed in RBC. The saving of bandwidths in our method, when compared to the bandwidth reservation method by RBC, is defined as,

$$G = \int_0^{L/r} \delta bw dt \tag{4.1}$$

where, δbw is the difference between the bandwidth usage in RBC and actual bandwidth requirement. Our calculation shows that the saving of bandwidth in both of Figure 4.6 (c) and (d) is l_1. The saving of bandwidth in both of Figure 4.7 (c) and (d) is $(2l_1 - l_2 - 2d)$.

In our strategy, the bandwidth of the disk cache is allocated for a stream in the following two cases. One case is when a stream begins to write the data into the cache, and the

Assume that the stream S_i needs the bandwidth to read data from cache. Insert all the writing streams whose following stream is not reading data from the cache (so that the non-switch constraint can be satisfied) into a queue named Ψ_w in a descending order of the interval size g of their respective intervals.

Repeatedly carry out the following step until ($R_b \leq A_b$) or (Ψ_w is empty).

When Ψ_w is not empty, pop a stream that is at the head location of Ψ_w and release the allocated cache bandwidth from the writing stream. Simultaneously, the respective interval is swapped out of the cache, i.e., the allocated space for the relative interval is released, otherwise the switching operation will happen in the future (refer to an example in Figure 4.9).

However, if there are not enough bandwidths for S_i, then S_i continues to read data from the origin.

Figure 4.8: BA (bandwidth allocation) strategy in the disk caching

other case is when a stream begins to read the data from the cache. For both of these two cases, if there are available bandwidths, then the stream is allocated with the bandwidth. Furthermore, in the latter case, if the available bandwidth A_b is less than the bandwidth requirement of a stream R_b, then we capture the bandwidth from the writing streams by using the *Bandwidth Allocation* (BA) strategy shown in Figure 4.8. With the BA (bandwidth allocation) strategy in Figure 4.8, we try to reclaim the bandwidth from some special streams that are writing data into the cache, since reading data from the cache is more important than writing data into the cache.

4.4 Variable Bit Rate Caching Algorithm

Using the three strategies introduced in Sections 4.1 - 4.3, we design a new caching algorithm - Variable Bit Rate Caching (VBRC) algorithm, which can be used in the disk caching or the memory caching. Without any loss of the generality, we describe VBRC in the context

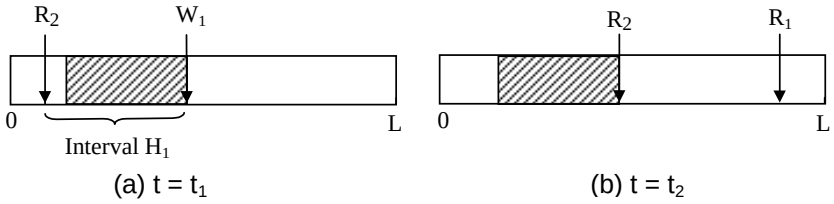


Figure 4.9: An example illustrating the BA (bandwidth allocation) strategy

of the disk caching. When we ignore the bandwidth constraint, VBRC is directly applicable for the memory caching.

4.4.1 Outline of the VBRC algorithm

We introduce the entire process of the VBRC algorithm in Figures 4.10 and 4.11.

4.4.2 Remarks

In the original GIC proposed in [39], the non-switch constraint was not considered. If the non-switch constraint is imposed on the original GIC, then the resultant caching algorithm is referred as GIC+. Clearly, the memory caching version of VBRC is identical with GIC+ when both of them are adopted in the case of constant retrieval bandwidth.

In the proposed algorithm VBRC, an interval size g is adopted to measure the *importance* of intervals for the allocation of the bandwidth and the space. In fact, if the interval length (h) is adopted instead of the interval size, then the performance of VBRC will not be considerably affected.

In comparison with RBC, VBRC have two significant advantages except the consideration

This algorithm includes three phases, which are carried out once in every service cycle D. D is chosen in such a fashion that it is at least much longer than the time overhead of carrying out three phases.

Phase 1. For every unfinished stream, we choose one of the following three cases appropriately.

Case 1. For a stream that is reading from the origin, if the block to be read is available in the cache, then the stream is switched from the origin to the cache. Meanwhile, the available bandwidth A_b decreases by an amount R_b, which is the retrieval bandwidth of the stream.

Case 2. For a stream that is reading from the cache, if the stream is not the former stream of an interval, the read block by the stream is swapped out of the cache.

Case 3. For a writing stream, the written data are cached.

If the stream will exceed the preceding stream during D, then the ERS strategy for handling the exceeding stream is carried out as described in Figure 4.3.

Phase 2. Check the available bandwidth A_b and the available space A_s

1. If $A_b < 0$, then the CBR (cache bandwidth reclaiming) strategy (refer to Figure 4.1) is carried out. 2. If $A_s < 0$, then the CSR (cache space reclaiming) strategy (refer to Figure 4.2) is carried out.

Phase 3. For every arriving request m for a document i, a stream S_m is formed. Assume that the retrieval bandwidth of S_m is R_b. We choose one of the following three cases appropriately.

Case 1. If the first block to be read has been cached, a new interval is formed (refer to Figure 4.12 for an example).

- If the available bandwidth A_b is not enough (e.g., $R_b > A_b$), the BA (bandwidth allocation) strategy (see Figure 4.8) is carried out.
- If the available bandwidth A_b is enough, then the stream begins to read data from the cache.

Figure 4.10: VBRC algorithm

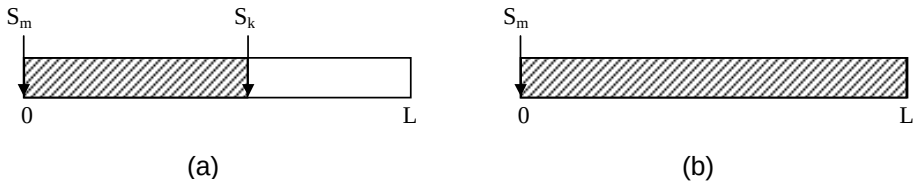
Case 2. If the first block has not been cached and there are not any preceding streams for the same document i, then the stream starts to read data from the origin.

- A new interval H_n is formed by stream S_m and another expected following stream. Simultaneously, S_m becomes a writing stream.
- Check the space and bandwidth requirements (R_s and R_b) of H_n. If A_s is not enough (i.e., $R_s > A_s$), then the *Space Allocation* (SA) strategy (see Figure 4.13) is carried out.
- If available bandwidths and spaces are not enough, then H_n is deleted and S_m will not write data into the cache.

Case 3. If the first block has not been cached and there is a preceding stream for the same document i, e.g., stream S_k, then the stream S_m starts to read data from the origin.

- A new interval H_n is formed by stream S_m and S_k. Simultaneously, if S_k is reading from the origin, then S_k becomes a writing stream.
- Check the space and bandwidth requirements of H_n. If A_s is not enough (i.e., $R_s > A_s$), then the SA (space allocation) strategy (see Figure 4.13) is carried out.
- If available bandwidths and spaces are not enough, then H_n is deleted and S_k will not write data into the cache.

Figure 4.11: VBRC algorithm (continue)



In this diagram, the long rectangle represents a CM document. L is the size of this document. The hashed part is cached data and the blank part is uncached data. S_k and S_m are streams. For both of cases (a) and (b), before S_m comes, there must exists an interval H_1. When S_m comes, S_m becomes the latter stream of H_1. Besides, a new interval, in which S_m is the former stream, is generated by S_m.

Figure 4.12: An example illustrating the form of a new interval

Assume that an interval H_n needs the space with the size of R_s. Insert all the cached intervals, in which the interval size is greater than the size of H_n and there are no streams which are reading data from the cache, into a queue named Ψ_i in a descending order of the interval size g.

Repeatedly carry out the following step until ($R_s \leq A_s$) or (Ψ_i is empty).

When Ψ_i is not empty, pop an interval that is at the head location of Ψ_i and release the allocated cache space from this interval. However, if there are not enough spaces for H_n, then H_n will be deleted.

Figure 4.13: SA (space allocation) strategy in the disk caching

of the non-switch constraint and the variable retrieval bandwidth.

- In VBRC, the allocation of the bandwidth adopts the just-in-time method instead of the reservation method in RBC.
- In VBRC, the allocation of the space is carried out among intervals instead of intervals and runs. A run [157] is formed by some consecutive intervals. By and large, an interval is smaller in sizes than a run, hence the allocation of the space among intervals is expected to be more efficient than the allocation of the space among runs.

Besides, both GIC and RBC do not analyze the bad effect of the switching operation and not try to reduce the number of the switching operation, also these two algorithms cannot directly used in the case of variable retrieval bandwidth.

4.5 Performance Evaluation

In this section, we conduct rigorous simulation experiments to testify all our strategies presented in Section 4.4. In the simulation, we consider an architecture: "the origins - the cache - the clients". In detail, we shall present the following studies.

- Effect of the non-switch constraint on the performance
- Comparing the performance of RBC and VBRC
- Performance of VBRC

The performance metric in our study is the byte hit ratio under several influencing factors such as, disk cache size, disk bandwidth, arrival rate, and percentage of requests for large documents, respectively.

4.5.1 Simulation test-bed

Table 4.1 shows the system parameters that are used in our simulation experiments. These parameters and their values used in our experiments below are typical to real-life situations, and the similar values are considered in the past literatures [157, 39]. In the simulation

Table 4.1: System parameters in the CM caching

System Parameters	Symbol	Parameter Values
Number of Different documents	M	100
Disk Cache Storage Size	B	$1G - 80GB$
Disk Cache Bandwidth Size	BW	$10 - 90MB/s$
Memory Cache Storage Size	B	$100 - 1000MB$
Skew Factor of Access Prob. Distribution	θ	0.06
Number of Requests	N	1500
Arrival Rate of Requests	λ	$0.1 - 1.0s^{-1}$
Service Cycle	D	$1s$
Playback Rate	r	Uniform distribution in $0.25 - 0.75MB/s$
Variable Scope of the Retrieval Bandwidth	V	$0 - 20\%$
Percentage of Large Documents	P	$0 - 100\%$

of caching strategies, the performance is usually measured in the steady stage [39, 157]. Similarly, we evaluate the performance of 1500 requests excluding the "warming up". In our simulation, the parameter V - the variable scope of the retrieval bandwidth is used to simulate the variation of the retrieval bandwidth. In the case of constant retrieval bandwidth, the retrieval bandwidth is equal to the playback rate (i.e., $V = 0$). In the case of variable retrieval bandwidth, in every service cycle, the retrieval bandwidth of

every existing stream is randomly chosen following a uniform distribution in the range $[(1-V)\times r, (1+V)\times r]$.

4.5.2 Effect on the performance due to the non-switch constraint

In this section, the non-switch constraint is imposed on GIC, which is the simplest caching algorithm, so that the influence on the performance due to the non-switch constraint can be observed. GIC is an algorithm introduced in [38] and GIC+ employs the non-switch constraint presented in Section 4.2. Thus, the latter is a variant of conventional GIC presented in [38]. Figure 4.14 shows the performance of GIC+ and GIC, when the

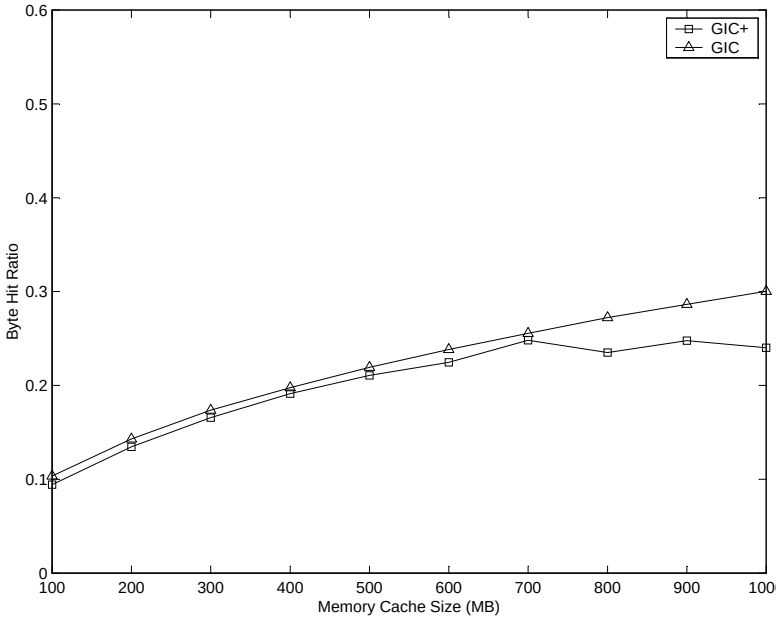


Figure 4.14: Performance comparison between GIC and GIC+ ($\lambda = 0.25s^{-1}$ and $P = 80\%$)

memory cache space varies. With the increase in size of the memory cache, both of the caching algorithms have better performances in the byte hit ratio, as a larger cache space can accommodate more intervals.

Figure 4.15 shows the performance of GIC+ and GIC, when the arrival rate varies. When

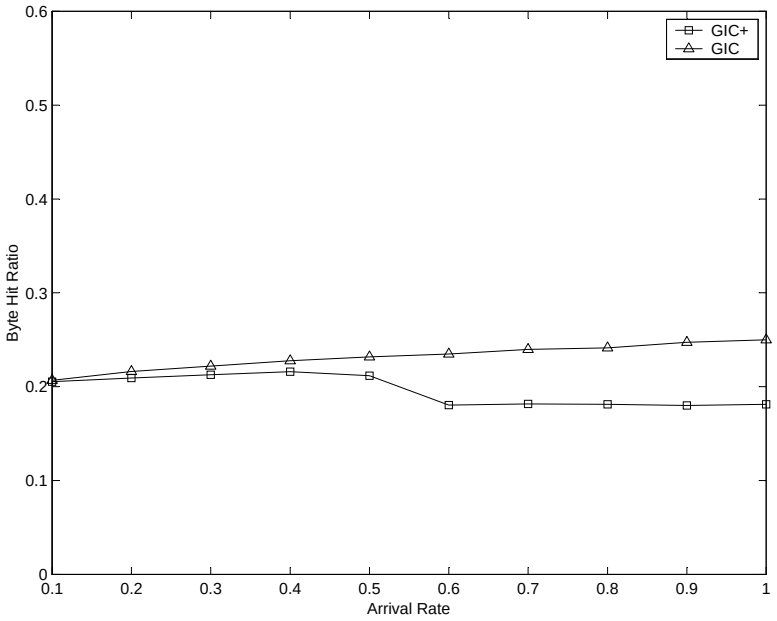


Figure 4.15: Performance comparison between GIC and GIC+ ($B = 500MB$ and $P = 80\%$)

the arrival rate increases, there are two factors that influence the performance.

- On the one hand, more small-size intervals can be formed, so the performance might be improved due to the fact that following requests can be satisfied with the already formed intervals.

- On the other hand, when arrival rate increases, the number of the requests in a unit time increases, however, due to a limited cache space more data cannot be cached, and hence, the performance might become worse.

With the combined influence of the above two factors, Figure 4.15 shows the arrival rates have little effects on the performance.

Figure 4.16 shows the performance of GIC+ and GIC, when the fraction of requests for large documents varies. When the request for the large documents increases, there are also two factors that influence the performance.

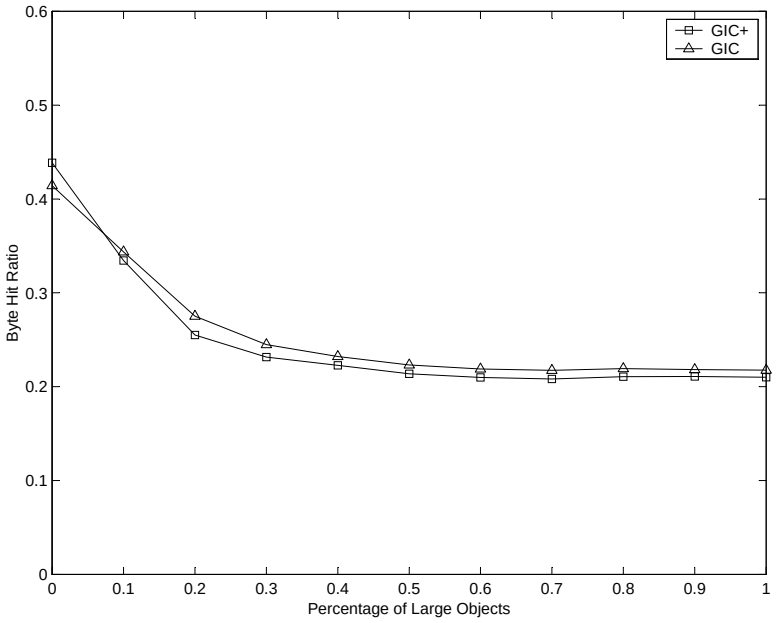


Figure 4.16: Performance comparison between GIC and GIC+ ($B = 500MB$ and $\lambda = 0.25s^{-1}$)

- On the one hand, more larger documents means that the total amount of data to access increases, while the amount of data that can be read from the cache has less changed. Hence, the performance might become worse.

- On the other hand, large-size documents make it easy to generate an interval to cache more data, thus, allowing more future accesses that request the same document to access cached data. Hence, the performance might be improved.

Figures 4.14 to 4.16, show the performance when the retrieval bandwidth is held constant and is equal to the playback rate. In this case, there are no switching operations in GIC+. In comparison, in GIC, the number of the switching operations is shown in Table 4.2.

Figure 4.14 - Figure 4.16 show GIC exhibits a better performance than GIC+ with respect to the byte hit ratio. However, note that here, the performance of *only the cache* is shown.

Table 4.2: Number of the switching operation in GIC (1500 requests)

$B(MB)$	100	200	300	400	500	600	700	800	900	1000	
	67	79	91	97	88	91	91	88	97	93	
λ	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	
	73	85	89	104	94	98	131	130	174	158	
P	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
	38	68	82	72	89	92	92	90	88	88	88

The performance improvement of GIC is achieved at the cost of a poor utilization of the bandwidth resource in the origin. If the performance of an entire system (including the origin and the cache) is considered, then GIC+ should exhibit a better performance than GIC, since the non-switch constraint eliminates all the unnecessary switching operations and the bandwidth resource available with the origin is saved to a significant extent (refer to the analysis in Section 4.2).

Table 4.2 shows the number of the switching operations increases as the memory cache size, the arrival rate, or the percentage of requests for large documents increases. However, it is difficult to estimate the number of the switching operations.

On the one hand, if we reserve the bandwidth of the origin for every stream that is reading data from the cache, then the total amount of bytes read from the origin is obviously reduced. In this case, the amount of bytes read from the cache is equivalent to a resource wastage in the origin. Assume that A_1 is the amount of bytes read from the origin and A_2 is the amount of data read from cache. Thus, the total amount of data to read is $A_1 + A_2$. The amount of resource wasted in the origin, by reserving a bandwidth, is equivalent to A_2. The ratio $A_2/(A_1 + A_2)$ gives the percentage of reserved resources that are not used

Table 4.3: Effect of the non-switch constraint (1500 requests)

$B(MB)$	100	200	300	400	500	600	700	800	900	1000	
A_4/A_2	0.06	0.03	0.05	0.04	0.04	0.04	0.03	0.50	0.34	0.26	
λ	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	
A_4/A_2	0.01	0.03	0.03	0.04	0.07	0.24	0.25	0.26	0.26	0.27	
P	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
A_4/A_2	0.01	0.02	0.03	0.05	0.02	0.02	0.02	0.02	0.04	0.04	0.04

at the origin. On the other hand, when we adopt the non-switch constraint to eliminate the switching operation, the byte hit ratio decreases (See Figure 4.14 - 4.16). We denote A_4 as the difference in the number of bytes that are read from the cache between GIC and GIC+. Thus, we compare these two cases by observing the ratio A_4/A_2 which is shown in Table 4.3. From the table we observe that, in most cases, the quantity A_2 is much more than A_4, and their difference shows the effect of the non-switch constraint.

4.5.3 Performance comparison between RBC and VBRC

In the case of memory caching, it should be clear at this stage that VBRC is GIC+ (i.e., GIC with the non-switch constraint). Hence we do not need to compare the performance of VBRC and GIC. Thus, in this section, we compare the performance of RBC and VBRC in the case of constant retrieval bandwidth (since RBC cannot be used in the case of variable retrieval bandwidth).

Figure 4.17 shows the performance of RBC and VBRC when the disk cache space varies. When $B \geq 5000MB$, the performance of RBC and VBRC have no noticeable improvement, since the disk space is already abundant. In this case, the cache bandwidth becomes a

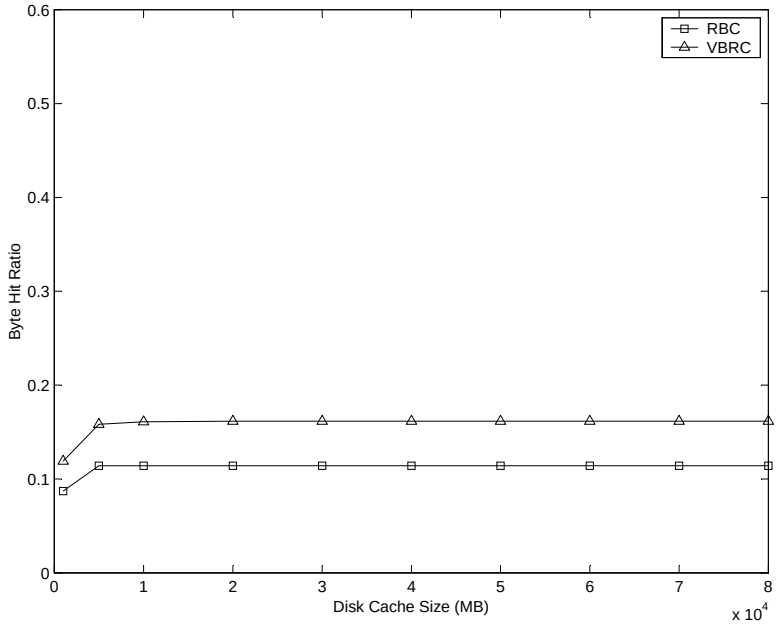


Figure 4.17: Performance comparison between RBC and VBRC ($BW = 20MB/s, \lambda = 0.25s^{-1}$, and $P = 80\%$)

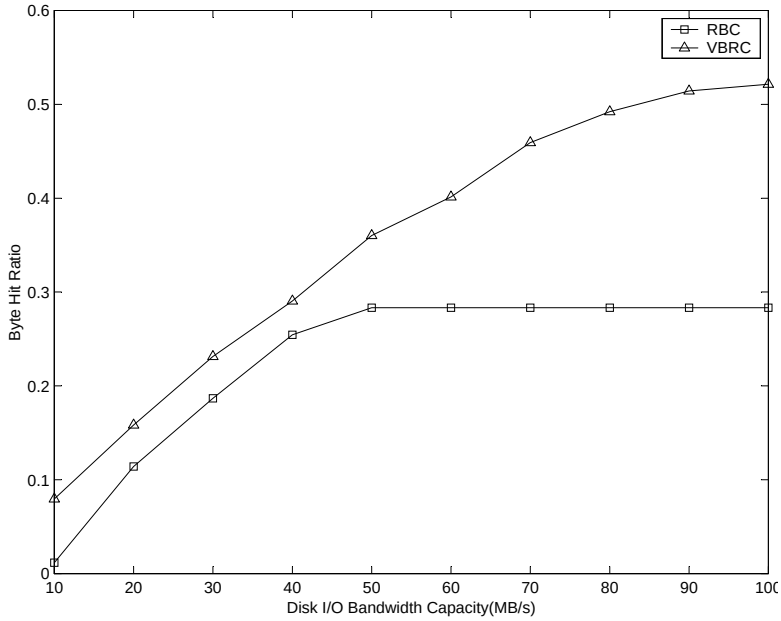


Figure 4.18: Performance comparison between RBC and VBRC ($B = 5000MB, \lambda = 0.25s^{-1}$, and $P = 80\%$)

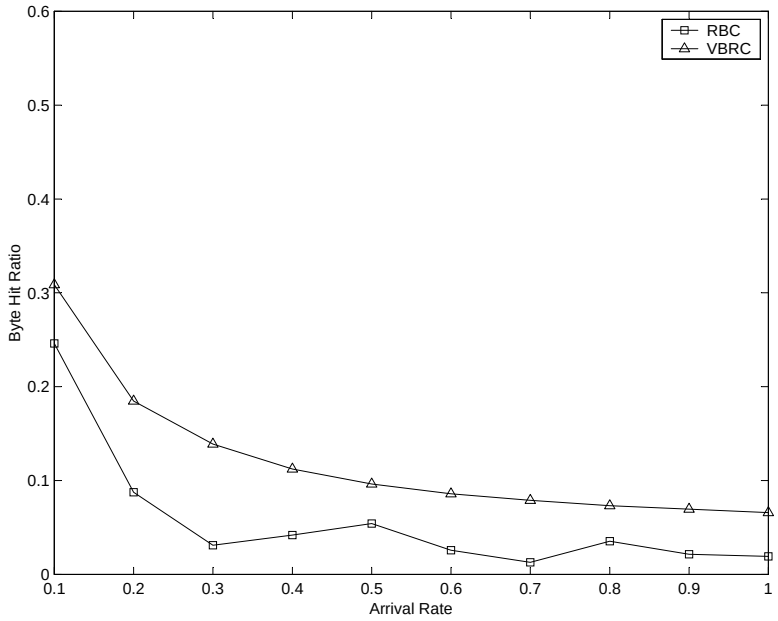


Figure 4.19: Performance comparison between RBC and VBRC ($B = 5000MB, BW = 20MB/s$, and $P = 80\%$)

bottleneck.

Figure 4.18 shows the performance of RBC and VBRC when the bandwidth of the disk cache varies. Certainly, more bandwidths can aid to read more cached data. Hence, a larger-size bandwidth means a better performance for both of RBC and VBRC. In particular, the byte hit ratio of RBC stops increasing (saturation) after the bandwidth reaches $50MB/s$, while the byte hit ratio of VBRC can increase till $100MB/s$. This is due to the fact that in the case of RBC, when BW is more than $50MB/s$, the disk space becomes a bottleneck in terms of utilization, and hence bears no influence on the increase in the disk bandwidth. This phenomena is also observed in [157].

Figure 4.19 shows the performance of RBC and VBRC when the arrival rate varies. When the arrival rate increases, with a limited storage space, more data cannot be cached,

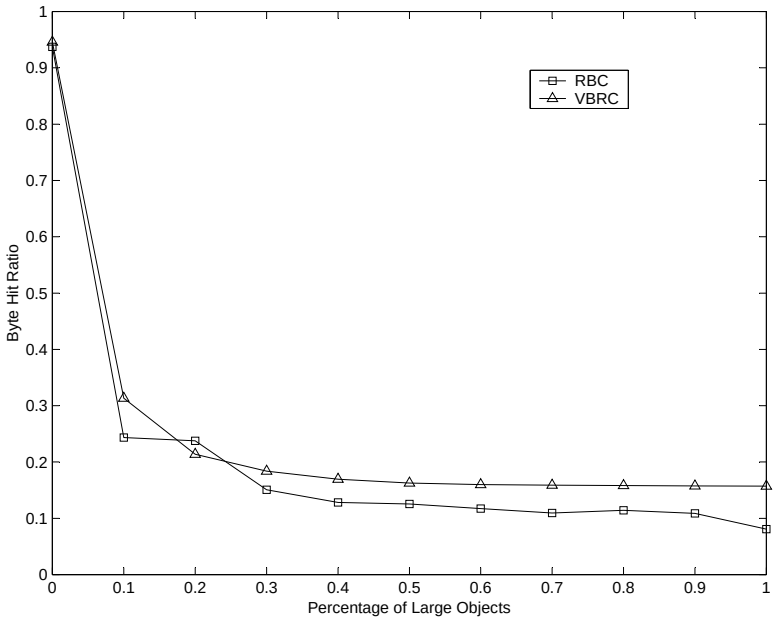


Figure 4.20: Performance comparison between RBC and VBRC ($B = 5000MB, BW = 20MB/s$, and $\lambda = 0.25s^{-1}$)

however, the number of the requests in a unit time increases, and hence, the byte hit ratio decreases, resulting in worse performances.

Figure 4.20 shows the performance of RBC and VBRC when the percentage of requests for large documents varies. More requests for large documents means that the total amount of data to access increases, while the amount of data that can be read from the cache is less changed. Hence, the performance might become worse.

In comparison with RBC, our VBRC algorithm adopts the "just-in-time" allocation method of the bandwidth, and the allocation of spaces is among intervals (not runs!). Therefore, from Figure 4.17 - Figure 4.20, we observe that VBRC has a much better performance than RBC. Besides, there are no switching operations in VBRC (as we use the non-switch constraint), while the switching operation in RBC is shown in Table 4.4.

Table 4.4: Number of the switching operation in RBC (1500 requests)

$B(GB)$	1	5	10	20	30	40	50	60	70	80	
	0	59	64	56	56	56	56	56	56	56	
$BW(MB/s)$	10	20	30	40	50	60	70	80	90	100	
	44	59	74	24	0	0	0	0	0	0	
λ	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	
	11	50	84	116	107	115	54	109	88	40	
P	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
	0	68	75	109	54	59	78	112	59	50	100

4.5.4 Performance of VBRC in the case of variable retrieval bandwidth

Figure 4.21- Figure 4.24 compares the performance of VBRC when $V = 0, 10\%, 20\%$ and when the disk cache space, the retrieval bandwidth of the disk cache, the arrival rate, or the percentage of requests for large documents varies, respectively. Tables 4.5 and 4.6 show the number of switching operations.

In the case of variable retrieval bandwidth ($V \neq 0$), the caching performance becomes a little worse in comparison with the case when $V = 0$. In general, the bandwidth is the bottleneck in the case of disk caching.

The switching operations are completely eliminated in the case of constant retrieval bandwidth in VBRC. However, in the case of variable retrieval bandwidth, the switching to server cannot be avoided, although VBRC employing the non-switch constraint thrives to minimize the number of switching operations.

It may be noted that GIC and RBC cannot be directly employed to handle a variable

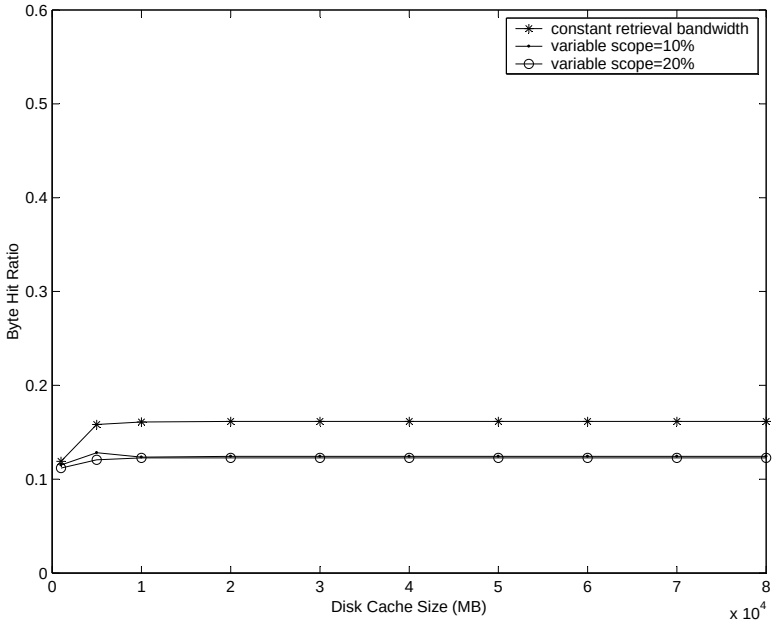


Figure 4.21: Performance of VBRC with the variable retrieval bandwidth ($BW = 20MB/s, \lambda = 0.25s^{-1}$, and $P = 80\%$)

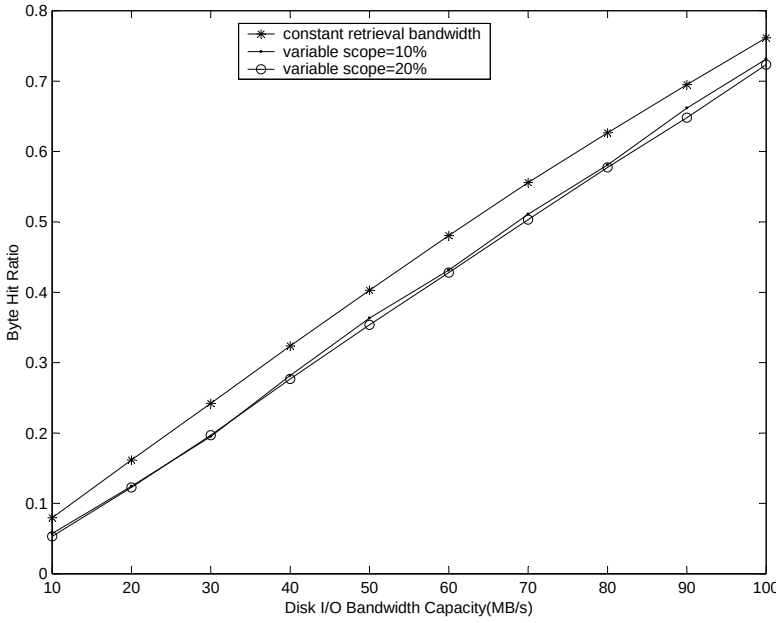


Figure 4.22: Performance of VBRC with the variable retrieval bandwidth ($B = 20GB, \lambda = 0.25s^{-1}$, and $P = 80\%$)

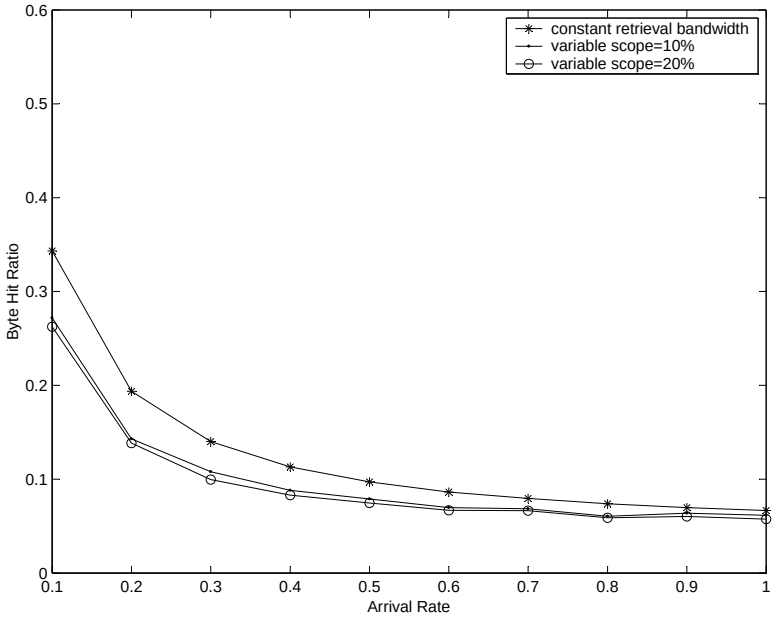


Figure 4.23: Performance of VBRC with the variable retrieval bandwidth ($B = 20GB, BW = 20MB/s$, and $P = 80\%$)

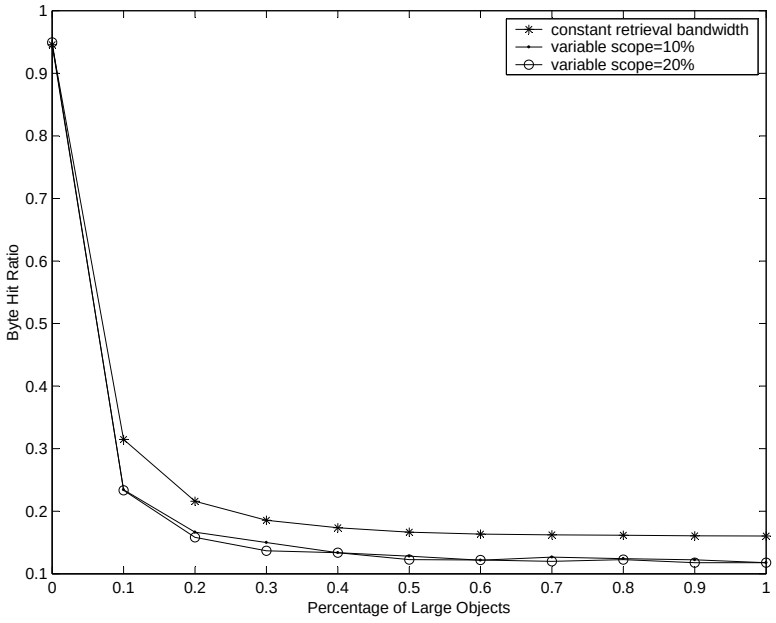


Figure 4.24: Performance of VBRC with the variable retrieval bandwidth ($B = 20GB, BW = 20MB/s$, and $\lambda = 0.25s^{-1}$)

Table 4.5: Number of the switching operation in VBRC ($V = 10\%$ and 1500 requests)

$B(GB)$	1	5	10	20	30	40	50	60	70	80	
	143	156	209	164	164	164	164	164	164	164	
$BW(MB/s)$	10	20	30	40	50	60	70	80	90	100	
	101	164	186	197	177	141	142	112	76	30	
λ	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	
	134	164	213	161	189	193	218	204	152	153	
P	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
	0	198	214	195	217	154	202	232	164	136	148

retrieval bandwidth situation. Thus, what we have conclusively shown from our simulation experiments is the adaptable nature of VBRC to the variable bandwidth scenario.

Finally, the overhead of VBRC can be observed in Table 4.7 (in a 733MHz PC). Although there are a number of operations in every service cycle (due to Phases 1, 2, and 3 in Figures 4.10 and 4.11), the total time overhead is insignificant, e.g., the average time overhead for processing every request is only about 10^{-2}s. Also, Table 4.7 shows that the overhead has no obvious relationship with the size of the cache space or the cache bandwidth. These observations imply that VBRC is cost-effective in the performance and applicable to the real-life situation.

4.6 Concluding Remarks

In this chapter, we have proposed a novel algorithm VBRC, which is used in caching CM data in the disk and the main-memory storage device. The design of VBRC stems from two reasons. The first reason is due to the fact that current interval-level caching algorithms

Table 4.6: Number of the switching operation in VBRC ($V = 20\%$ and 1500 requests)

$B(GB)$	1	5	10	20	30	40	50	60	70	80	
	191	176	230	230	230	230	230	230	230	230	
$BW(MB/s)$	10	20	30	40	50	60	70	80	90	100	
	189	230	186	193	207	164	180	140	96	68	
λ	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	
	153	215	247	249	208	232	230	223	246	205	
P	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
	0	220	252	221	223	228	204	215	230	229	180

Table 4.7: Time overhead of VBRC ($V = 20\%$ and 1500 requests)

$B(GB)$	1	5	10	20	30	40	50	60	70	80	
(min.)	0.17	0.18	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	
$BW(MB/s)$	10	20	30	40	50	60	70	80	90	100	
(min.)	0.18	0.27	0.23	0.25	0.27	0.27	0.23	0.25	0.22	0.23	
λ	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	
(min.)	0.17	0.17	0.15	0.15	0.13	0.13	0.12	0.13	0.10	0.12	
P	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
(min.)	0.02	0.1	0.17	0.17	0.15	0.17	0.15	0.17	0.15	0.15	0.15

cannot handle the case of the variable retrieval bandwidth (or variable bit rate). Note that we do not refer to VBR of CM data themselves, in which the change of bit rates is much smaller. We can easily use buffering or other resource reservation methods [73] to handle it. By contrast, we are considering the variable bit rate that is caused by the outside (e.g., networks, servers, or clients). While there are no previous researches devoted to similar topics, the variable retrieval bandwidth demand is commonly encountered in network-based multimedia services. Our VBRC algorithm checks the space and bandwidth requirements in every service cycle, and handles the exceeding streams in the case of variable retrieval bandwidth. The second reason is that there exist switching operations in all so far proposed caching algorithms in the literature, and these switching operations easily result in a disruption of the continuous playback. In fact, as mentioned in Section 4.2, a large amount of overheads due to the switching operation is one of the primary reasons why BASIC may not be suitable to handle streams in a practical caching system.

The design of VBRC essentially focuses on the management of resources (the cache space and bandwidth). In VBRC, the bandwidth resource is allocated in a *just-in-time* manner instead of carrying out any reservation. The allocation of the cache space is among intervals not runs, so that the utilization of the space can be more efficient. The allocation of the cache space and bandwidth considers the case of the variable retrieval bandwidth and the strategy of reducing the switching operation. In the simulation experiments, we studied the performances of VBRC in terms of the byte hit ratio under several influencing parameters such as, cache space, cache bandwidth, request arrival rate, and percentage of large documents. Our rigorous simulation experiments provide a corroborative evidence for the performance of VBRC and confirm our analysis. VBRC is conclusively testified to perform better than the popular disk caching algorithm RBC [157]. Also, the overhead and

the number of switching operations in VBRC are shown satisfactory.

In our research, all of examples belong to the single-channel retrieval. However the idea of proposed strategies can be easily used in the multiple-channel/multiple-server retrieval.

Chapter 5

Experiments on the CM Data Retrieval

It may be noted that the caching problem can be trivially implemented within a single host together a mature cache product (e.g, Squid [150]), as far as the experimental verification is concerned, and does not pose any challenge in an experimental venture. On the other hand, realizing the retrieval strategies poses a considerable challenge as a number of factors influence the performance on a network-based system. Consequently, it becomes more meaningful to implement our retrieval strategies. In this chapter, we precisely attempt on implementing our retrieval algorithms by testing on real-life network conditions. Specifically, we consider implementing the single-channel/single-server and multiple-channel/multiple-server retrieval strategies, and compare their performances.

5.1 Hardware and Software

Figure 5.1 shows the network designed for our retrieval experiments, where there are two video servers (a local one and a remote one) and a client host. On the one hand, the

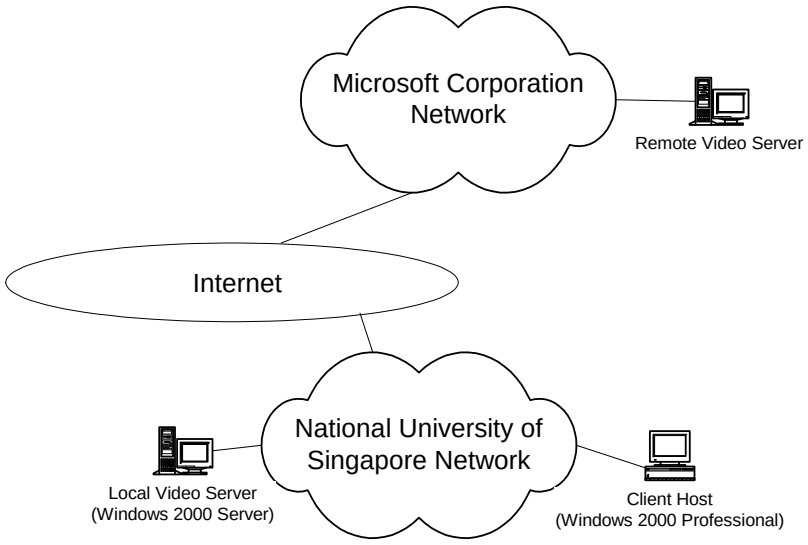


Figure 5.1: Network diagram for retrieval experiments

retrieval from a remote server usually has inferior QoS (mainly in terms of the continuous playback) because of congested network conditions. Thus, in this case, our multiple-channel retrieval strategies can clearly show their advantage in improving QoS. On the other hand, we can easily control the workload on the local server so that we can easily compare different retrieval strategies with the same workload conditions.

In the local video server, the operating system is Windows 2000 server, in which the Windows media services are running inherently. Windows media services can send the prerecord or live audio and video data to the client computer via unicasting or multicasting [109]. The remote server has also been installed with similar software products. When the local server and/or the remote server receives a request from the client for retrieving a CM document, the server starts to cast data of the requested CM document to the client. Next, the client can playback the content of the document while data of the document are arriving at the client. The server can control the quality of every sent stream by setting the retrieval bandwidth, typically such as, $56k, 100k$, and $300k$. However the change of the

retrieval bandwidth is discrete and not continuous. Furthermore, we cannot program to control the retrieval bandwidth of the streams, as the source code of the Windows media server is not *open*.

At the client end, we adopt a CM data retrieval software - ASFRecorder and ASFR+ [12]. As they are open sources, we can customize these software to meet our requirements. The function of ASFRecorder is to retrieve and playback CM documents (with the extensions of ASF, WMA, and WMV) or CM redirector documents (with the extensions of ASX, WMX, and WVX). The function of ASFR+ is to retrieve a CM document in a multi-threaded fashion from a single server. We combine ASFRecorder and ASFR+ so that the CM data can be retrieved and played back from multiple channels and multiple servers. Moreover, we include our retrieval algorithm within the client software.

5.2 Implementation Detail

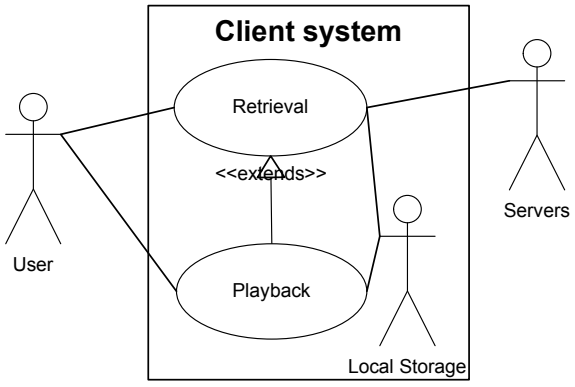


Figure 5.2: Use case diagram of the system on the client computer

The entire client system consists of a retrieval sub-system and a playback sub-system

as shown in Figure 5.2. The retrieval sub-system retrieves ASF files from media servers using TCP or HTTP protocol. The playback sub-system plays the received ASF file.

5.2.1 Playback sub-system

The playback sub-system applies the DirectShow, which is a part of Microsoft DirectX. DirectShow provides APIs in the high-quality capture and playback of multimedia streams. Also, it supports a variety of formats, including Advanced Streaming Format (ASF), Motion Picture Experts Group (MPEG), Audio-Video Interleaved (AVI), MPEG Audio Layer-3 (MP3), and WAV files. The main process of playing a video document using DirectShow is in two steps.

1. A filter graph, which is *IGraphBuilder* type, is created using *CoCreateInstance* function.

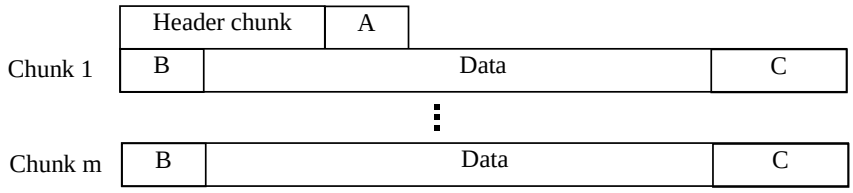
2. The filter graph responds to user events and controls the playback process until the playback is completed.

The beginning time of a playback is determined by the retrieval scheduling.

5.2.2 Format of ASF file

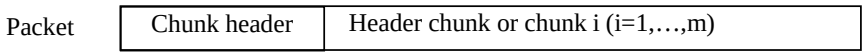
In Figure 5.3, we present the format of the ASF (version 1.0) file. ASF 1.0 is one of the most popular streaming video formats. An ASF file consists of a header chunk and multiple fixed-size chunks.

All actual data of ASF file, such as compressed audio or video, is delivered from the server to the client, in the format of "packets". All packets are of fixed sizes. Figure 5.4 is the structure of every packet.



Header chunk contains the following information: the total byte size, the time duration of this ASF file, the byte size of every chunk, etc.
A block is reserved, and its size is 50 bytes.
B block is the time code of this chunk, i.e., the time location of data in an entire file.
C block is unused bytes and is used so that every chunk has the same length.
Data block contains the available video data.

Figure 5.3: Format of ASF 1.0



Chunk header contains the following information: chunk type (which identifies header chunks or ordinary chunks), chunk size, sequence number of the chunk.
Chunk in the packet does not contain unused bytes. It is the difference between a chunk in an ASF file and a chunk in a packet.

Figure 5.4: Format of a packet in streaming ASF Files

5.2.3 Retrieval bandwidth

In a Windows media server, no configurations are needed for delivering on-demand ASF files. After a server receives a request, the server automatically sends the requested ASF file to the client using a fixed bandwidth, i.e., the playback rate. However, because of the network congestion, the client often receives the data in the data rate which is *smaller* than the playback rate of the requested file. Therefore, the *actual* retrieval bandwidth, or receiving rate in the client, is usually less than the playback rate. As the actual retrieval bandwidth is one of input parameters of the retrieval scheduling, we shall estimate the actual bandwidth retrieval according to the actual network traffic before the scheduling.

The multiple-channel retrieval will be implemented using the multi-threaded retrieval. In

detail, every thread retrieves data in a *Round-Robin* fashion. The actual retrieval bandwidth of every thread is indeed the channel bandwidth.

In our experiment, we observed that two threads, which are involved in the data retrieval from the same server, cannot be started simultaneously within a very small interval of time. If this is attempted, the server treats them as a single thread and sends them with the same data. In our experience, we note that the minimum time interval required to initiate two threads is 1.5 seconds.

5.2.4 Number of installments

As a rule of thumb, in order to achieve a smaller access time, we tend to keep the number of installments large. Here, we calculate m_{max}, the maximum value of m.

When the number of installments increases, $a_{1,1}$ decreases, consequently t_0 will decrease as well. Note that a *chunk* is the minimum amount of retrieved data in every response from the server. It means that the minimum value of $a_{1,1}$ is the chunk size s_c. Therefore, the m_{max} is the maximum value of m that satisfies the following condition: when $m = m_{max}$, $a_{1,1} \leq s_c$; when $m = m_{max} - 1$, $a_{1,1} > s_c$. If we assume $n = 2$ and $bw = cr$ (r is the playback rate and c is a constant), then we can obtain the following equations from (3.7)-(3.10).

$$\begin{cases} a_{1,2} = \frac{a_{1,1}}{(1-c)} \\ a_{i,1} = \frac{a_{1,2}c^{2(i-2)+1}}{(1-c)^{2(i-2)+1}}, \quad i = 2, .., m \\ a_{i,2} = \frac{a_{1,2}c^{2(i-2)+2}}{(1-c)^{2(i-2)+2}}, \quad i = 2, .., m \\ \sum_{i=1,j=1}^{i=m,j=2} a_{i,j} = L \end{cases} \tag{5.1}$$

From (5.1), we can calculate m_{max} and $a_{1,1}$.

Now we consider a special case. Let $B = \frac{c}{1-c}$. If $B < 1$ or $c < 0.5$, then we obtain the

following result from (5.1).

$$\frac{L(1-c)}{a_{1,1}} = (1-c) + 1 + B + B^2 + ...B^{2(m-1)} < (1-c) + \frac{1}{(1-B)}$$

So,

$$a_{1,1} > \frac{L(1-2c)}{2(1-c)}.$$

It means that when $c < 0.5$, with the increase of the installments, $a_{1,1}$ gradually decreases until it reaches a lower limit given by, $\frac{L(1-2c)}{2(1-c)}$. It is reasonable as we use two channels now. When the total size of channels is less than the playback rate, the access time has a lower limit. The simulation in Chapter 3 also demonstrates this fact.

In addition, a switch from an installment to another installment also causes overhead. This kind of overhead is noticeable and about $1-3$ seconds according to our observation in experiments.

5.2.5 Retrieval sub-system

In the retrieval process, the client uses the socket to communicate with the video server. The main process of retrieving a video document is shown in Figure 5.5.

5.3 Experimental Results and Analysis

In this section, based on the above discussions, we carried out the following experiments.

1. Retrieval of a CM document from the remote server using single-channel and multiple-channel strategies

2. Retrieval of a CM document from the local server using single-channel and multiple-channel strategies

Step 1. The client sends the first HTTP or TCP request to the server to retrieve the meta information of the requested document. Consequently, the response from the server will provide two kinds of information.

- The response header. From this header, we can know whether the required document exists or not, and whether the request document is a live stream or a pre-recorded content.
- The chunk header (refer to Figure 5.4) and the header chunk (refer to Figure 5.3). From them, we can know the chunk length and the document length (the byte size and the time duration).

Step 2. If the required ASF file exists and the file is a pre-recorded file, then we can continue the following procedures.

- Firstly, the actual retrieval bandwidth is estimated according to the retrieval in Step 1.
- Secondly, we determine the number of channels, n, and the number of installments, m, according to Section 3.4 and section 5.2.4, respectively.
- Thirdly, the client software carries out our scheduling strategy to determine the size of every portion retrieved from every channel. As the data are transmitted in chunks, the size of every portion is approximated to a constant multiple of the chunk size.

Step 3. The client sends n requests to retrieve the CM data in n threads.

- When a thread j ($j = 1, .., n$) finishes retrieving a portion $a_{1,j}$, the thread j begins to retrieve $a_{2,j}$,..., until $a_{m,j}$. Note that the thread j needs to send different requests for different portions.
- If a thread j finishes the task of retrieving $a_{m,j}$, then this thread will aid another thread that may probably miss the deadline (refer to the strategy in Section 3.5.1).
- If a thread j finishes the task of retrieving $a_{m,j}$ and none of the threads will miss their deadlines according to the current retrieval bandwidth, then thread j will help to shorten the retrieval duration (refer to the strategy in Section 3.5.3).

Figure 5.5: Retrieval process of an ASF file

Table 5.1: Input parameters of the retrieval scheduling

File byte length (L)	File time length (H)	Chunk byte length
3959258 bytes	102.4 seconds	3861 bytes
Ave. playback rate (r)	**Channel bandwidth (bw)**	**Maximum Number of installment**
38664.6 bytes/s	$< 0.5r$	∞
	$0.7r$	5
	$0.9r, 0.8r$	3

3. Retrieval of a CM document from the local server and the remote server using multiple-channel and multiple-servers strategies

Figure 5.1 shows the input parameters, which are the meta information of the requested document, used for the retrieval scheduling. We also calculate the maximum number of installments for the case of the 2-channel retrieval according to our analysis in Section 5.2.4.

Figure 5.2 shows the results of the retrieval scheduling, where m is the number of installment and n is the number of channels. At the beginning of experiment, we carry out the single-channel retrieval (in which no scheduling is needed) from the local server and the remote server, respectively, so that we can know the actual retrieval bandwidth (i.e., channel bandwidth) of the local server and the remote server at that time. They are approximately r and $0.3r$, respectively. The schedule in Figure 3.2 is used in the multiple-channel retrieval from a local server, since the channel bandwidth is r. The schedule in Figure 3.1(b) is used in the multiple-channel retrieval from a remote server, since the channel bandwidth is less than r. The schedule in Figure 3.2 is used in the multiple-channel retrieval from both the local server and the remote server, since the first channel, which is from the local server, is r.

Table 5.2: Results of the retrieval scheduling

Server	m	n	Portion size
Local	1	2	$a_{1,1} = 0.5L, a_{1,2} = 0.5L$
Local	1	3	$a_{1,1} = 0.33L, a_{1,2} = 0.33L, a_{1,3} = 0.33L$
Local	1	4	$a_{1,1} = 0.25L, a_{1,2} = 0.25L, a_{1,3} = 0.25L, a_{1,4} = 0.25L$
Remote	1	2	$a_{1,1} = 0.41L, a_{1,2} = 0.59L$
Remote	1	3	$a_{1,1} = 0.22L, a_{1,2} = 0.32L, a_{1,3} = 0.46L$
Remote	1	4	$a_{1,1} = 0.14L, a_{1,2} = 0.19L, a_{1,3} = 0.28L, a_{1,4} = 0.39L$
Remote	2	2	$a_{1,1} = 0.30L, a_{1,2} = 0.43L, a_{2,1} = 0.19L, a_{2,1} = 0.08L$
Remote	3	2	$a_{1,1} = 0.29L, a_{1,2} = 0.41L, a_{2,1} = 0.18L$ $a_{2,2} = 0.08L, a_{3,1} = 0.03L, a_{3,2} = 0.01$
Remote	4	2	$a_{1,1} = 0.286L, a_{1,2} = 0.409L, a_{2,1} = 0.175L, a_{2,2} = 0.075L$ $a_{3,1} = 0.032L, a_{3,2} = 0.014L, a_{4,1} = 0.006L, a_{4,2} = 0.003L$
local+remote	1	1+1	$a_{1,1} = 0.77L, a_{1,2} = 0.23L$
local+remote	1	2+2	$a_{1,1} = 0.38L, a_{1,2} = 0.12L, a_{2,1} = 0.38L, a_{2,2} = 0.12L$

Table 5.3 shows the results of retrieval experiments. In this table, the access time (or start-up delay) is the interval between the instant when initial data arrive at the client and the instant the playback begins at the client. The retrieval duration is the interval between the instant when initial data arrive at the client and the instant when an entire requested document has been retrieved. We calculate the access time and the retrieval duration for accessing the remote server using the channel size of $0.3r$. The computed value of the retrieval time of accessing the remote server has two values. The greater value is obtained according to the multiple-channel retrieval scheduling. When a variable-size channel retrieval strategy (refer to Section 3.5) is adopted, the retrieval duration will be reduced,

Table 5.3: Results of the retrieval experiment

Server	m	n	Access time (calculation)(s)	Retrieval duration (calculation)(s)	Retrieval duration (experiment)(s)
Local	1	1	0	N/A	104
Remote	1	1	238	N/A	378
Local	1	2	0	51	54
Local	1	3	0	34	37
Local	1	4	0	26	29
Remote	1	2	98	[170,200]	179
Remote	1	3	53	[113,155]	137
Remote	1	4	32	[85,134]	99
Remote	2	2	72	[170,174]	191
Remote	3	2	68	[113,171]	186
Remote	4	2	68	[85,170]	201
local+remote	1	1+1	0	78	113
local+remote	1	2+2	0	39	67

which is shown as the smaller value. The calculation of this value is demonstrated by the following example. A 2-channel retrieval from the remote server has a total bandwidth $2 \times 0.3r$, thus, the duration is $H \times r/(2 \times 0.3r) = 170(secs)$.

5.3.1 Result analysis

The above experiment shows that the multiple-channel and multiple-server retrieval strategies can be very well realized in a real environment. We summarize the experiment as follows.

- With the single-channel retrieval from the remote server, a media player (e.g., Windows media player) buffers some data before the playback begins. However, the player does not try to guarantee a continuous playback during the entire playback process. Hence, the user frequently needs to re-buffer after the playback starts. To take care of this situation, more data should be buffered or a multiple-channel retrieval must be adopted. Our experiment has obviously shown that the multiple-channel retrieval has shorter access times (the time used to buffer data) and retrieval durations than the single-channel retrieval. Furthermore, by carefully using scheduling strategies, the multiple-channel retrieval strategy minimizes the possibilities of data starvation during the playback.

- On our campus intranet, even the single-channel retrieval has a short access time as the retrieval bandwidth is usually close to the playback rate. In this case, the advantage of the multiple-channel retrieval is shown by reducing the retrieval duration. A short retrieval duration reduces the possibilities of network-disconnections. Besides, the client usually cannot command the server to send the CM data using the *user-defined* retrieval bandwidth in the *single-channel* retrieval. In comparison, using the multiple-channel retrieval, the retrieval bandwidth is well controlled by determining the number of concurrent channels.

- In general, the calculation value of the retrieval duration is close to the corresponding experiment value. However, more installments cause more overheads as starting a new installment means multiple operations: stop an old thread, generate a new thread, and send the request for data. Therefore, the experiment value of the multiple-installment retrieval from the remote server departs from the calculation value.

- In the experiment, the actual bandwidth of every channel from the remote server is

greatly affected by the variable network traffic. The strategies introduced in Section 3.5 prevent certain slow threads from missing their deadlines.

- Finally, from our experiments, we observe that the performance of using multiple servers will be effective only when participating servers have more-or-less identical retrieval bandwidths. If this is not the case, much of the time is wasted in actual retrieval of the data by remote servers, thus clearly defeating the purpose of employing multiple-server strategy.

5.4 Concluding Remarks

Our experiments demonstrate the following facts.

1. We test the multiple-channel and variable-size channel retrieval strategies (excluding the variable-channel control by the servers). Proposed retrieval strategies and algorithms presented in Chapter 3 are applicable in real-life environments.

2. The actual retrieval bandwidth of a channel is an important input of static scheduling. However, the actual retrieval bandwidth is greatly affected by real network traffics, and we cannot determine its exact value beforehand. Our solution is that we estimate the retrieval bandwidth before the static scheduling, and then we adopt variable-size channel retrieval strategies to adjust the retrieval in every channel.

Our experiments do not include the following issues, as they have been simulated in Chapter 3. We do not consider implementing a general access pattern, i.e., a group of client accesses a group of servers in a duration. Also, we do not consider a variable-size channel retrieval controlled by servers, i.e., the server adjusts the retrieval bandwidth according to its workload. In order to achieve this, a special server software/component should be

developed, e.g., Darwin Streaming Server [40], which is an open source streaming server for QuickTime by Apple. The work is beyond this thesis.

Chapter 6

Conclusions and Extensions to the Current Work

In this thesis, we have addressed two major important issues in a network-based multimedia servicing system. They are the problems of the *CM data retrieval* and *caching*. We have presented the multiple-server retrieval strategies, which are used when a service provider attempts to employ *multiple servers* in a coordinated fashion to retrieve the data and to service the user requests. Similarly, when several requests arrive at a server repeatedly for a CM document, we have considered designing caching strategies that efficiently handle *variable bit rate* situations. Our strategies are very important for the performance of network-based multimedia services. Moreover, our strategies are applicable for various kinds of distributed multimedia systems. The impact of these strategies have been clearly demonstrated via several real-life simulation experiments, as presented in respective chapters.

One of the significant contributions in this thesis is in presenting a generalized approach for handling multiple client requests using the multiple-server retrieval. We have ana-

lyzed and designed multiple-server retrieval strategies for the streaming (i.e., play-while-receive) mode. In the case of the play-after-download mode, the multiple-channel retrieval/download has been widely used, and it can greatly reduce the *downloading time* in comparison with the single-channel retrieval. However, play-while-receive is significantly different with play-after-download in constraints and performance metrics. In this thesis, we have presented a complete theoretical and experimental study for the multiple-server retrieval using the play-while-receive mode. We realized this strategy using the Multiple-Channel Retrieval (MCR) algorithm, whose main contents can be summarized as follows.

- The channel partition strategy is proposed to allocate the channels for the retrieval. In partitioning channels, we try to achieve both the minimum access time and load balancing among servers.
- The static scheduling is adopted before the playback begins. In the static scheduling, we use the constant retrieval bandwidth as the channel bandwidth. We presented the multiple-channel, multiple-installment, and synchronous/asynchronous-channel scheduling strategies.
- The dynamic scheduling is adopted during the playback. In the dynamic scheduling, we use the variable-size channel retrieval to prevent the slow channel from missing its deadline and improve the system capacity.

By using all the above strategies, the multiple-server retrieval and the multiple-channel retrieval have been shown better performances than the single-server retrieval and the single-channel retrieval in satisfying the continuous playback, the access time, the block ratio, and the retrieval duration.

Furthermore, we have shown that the experiment clearly testified the applicability of our retrieval algorithm to a practical situation. With our experimental results, our strategies

indeed provide more performance improvements for the play-while-receive mode. It will make the play-while-receive mode be widely applied to view CM documents on a network-based system as opposed to the conventional play-after-download mode.

We now present several issues that have not been explored in our studies of the multiple-server retrieval. Theoretically, CM data can be retrieved from FTP, HTTP, MMS, or RTSP servers using UDP, TCP, or HTTP protocol [154]. In our experiments, we only consider retrieving data from MMS server using HTTP and TCP protocols. Thus, as a first step it would be interesting to consider using our strategies on more protocols and servers and to see the actual performance. Also, the server software supporting the variable-size channel retrieval should be developed so that the multiple-server/multiple-channel retrieval can be controlled by servers. Besides, the client software should be configured to intelligently select the transport protocol among UDP, TCP, and HTTP. HTTP has the largest and worst overhead but can be used through a firewall. UDP has no error correction mechanism and therefore the (picture) quality usually has glitches, but it does have the fastest data rate. UDP does not work through a firewall, because network administrators tend to disable UDP for security reasons. TCP is the medium, with error correction facilities, giving less errors but a slightly slower data throughput than UDP.

As far as caching CM data are concerned in the view of providing a faster service, our strategies in all our simulation experiments exhibited a superior performance with respect to a variety of influencing parameters. In the conventional caching, cached documents are usually smaller-size documents, e.g., memory page and web page. Thus, caching an entire document costs little overheads. However CM documents are usually larger in sizes, and hence, caching a part of a document (so-called "interval") is preferred. In past caching algorithms for CM documents, GIC and RBC strategies are adopted in the memory caching

and the disk caching, respectively. Our design of strategies were fundamentally based on GIC and RBC. We now summarize our contributions as follows. Firstly, as an interval is usually formed by two streams, the latter stream reads the cache data that are written by the former stream. If the former stream stops writing data into the cache, then the latter stream will meet the switching operation (refer to Chapter 4). Switching operations probably cause an abrupt stop of a continuous playback, and hence we propose some strategies to avoid/minimize the switching operation. Secondly, the interval size is often a variable because of variable network traffics or the control operation from the server and the client. A variable-size interval means variable resource requirements in the storage space and bandwidth. Therefore, we proposed to fine-tune the resource allocation in every service cycle. Those operations in every service cycle has been detailed in Chapter 4. Finally, the resource allocation in RBC is basically a reservation scheme. We adopted a *just-in-time* scheme to allocate the resource only when the resource is actually required. This strategy clearly has been conclusively shown to improve the performance of disk caching algorithms.

In comparison with GIC and RBC, our algorithm VBRC is more appropriate in a practical application because we consider variable-size intervals and strategies by which we avoid/minimize switching operations. Furthermore, our simulation testified our analysis and showed the proposed caching algorithm VBRC has a much better performance than RBC.

Although we use the single-channel retrieval, which is the most common case, in the research of caching strategies, our caching strategies can be easily adapted to the multiple-channel retrieval and other kinds of retrieval, e.g., batching, patching, and multicasting. While so many cases of the retrieval cannot be analyzed without omitting, the idea of proposed caching strategies are broadly applicable. It is an essential significance of our caching

strategies.

We can envision some direct extensions that can be considered readily, to the caching problem addressed in this thesis. We may note that VBRC does not completely eliminate the possibility of switching operations while handling streams that demand variable retrieval bandwidths. Hence, as a first step, it will be beneficial if we fine-tune the VBRC algorithm to further reduce the possibilities of the switching operation for the case of the variable retrieval bandwidth. In the context of distributed systems, the distributed version of the caching algorithm must determine certain vantage sites to cache the data, depending on the arrival rates and on the activities at the sites. A recent work by Tang *et al.*[155] has demonstrated caching web objects in a hierarchical architecture, while we already considered caching web objects in a distributed architecture [42]. It will be an interesting topic to study how to combine them.

Bibliography

[1] M. Afonso, A. Santos, and V. Freitas, "QoS in Web Caching," *Computer Networks and ISDN Systems*, Vol. 30(22-23), pp. 2093-2103, 1998.

[2] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "On Optimal Piggyback Merging Policies for Video-on-Demand Systems," *Proc. SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1996.

[3] C. Aggarwal, J. L. Wolf, and P. S. Yu, "Caching on the World Wide Web," *IEEE Trans. Knowledge and Data Engineering*, Vol. 11(1), pp. 94-107, 1999.

[4] T. Ahmed, A. Mehaoua, and R. Boutaba, "Interworking between SIP and MPEG-4 DMIF for Heterogeneous IP Video Conferencing," In *Proc. IEEE International Conference on Communications (ICC)*, 2002.

[5] K. C. Almeroth and M. H. Ammar, "On the Performance of a Multicast Delivery Video-On-Demand Service with Discontinuous VCR Actions," In *Proc. IEEE International Conference on Communications (ICC)*, 1995.

[6] K. C. Almeroth and M. H. Ammar, "On the Use of Multicast Delivery to Provide a Scalable and Interactive Video-on-Demand Service," *IEEE Journal on Selected Areas in Communication*, Vol. 14(6), 1996.

[7] P. M. Anselone and J. W. Lee, "Multivariable Calculus with Engineering and Science Applications," Upper Saddle River, NJ : Prentice Hall, 1995.

[8] M. F. Arlitt and C. L. Williamson, "Internet Web Servers: Workload Characterization and Performance Implications," *IEEE/ACM Trans. Networking*, Vol. 5, pp. 631-645, 1997.

[9] M. F. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin, "Evaluating Content Management Techinques for Web Proxy Caches," In *Proc. 2nd Workshop on Internet Server Performance*, May 1999.

[10] M. F. Arlitt, R. Friedrich, and T. Jin, "Performance Evaluation of Web Proxy Cache Replacement Policies," *Performance Evaluation*, Vol. 39, pp. 149-164, 2000.

[11] T. Asaka, H. Miwa, and Y. Tanaka, "Distributed Web Caching Using Hash-based Query Caching Method," In *Proc. IEEE International Conference on Control Applications*, Vol. 2, 1999.

[12] ASFR+ ASFFix Homepage, http://yaan2.linux.net.cn/.

[13] Yun-Cheol Baek and Kern Koh, "Real-time Scheduling of Non-preemptive Periodic Tasks for Continuous Media Retrieval," In *Proc. IEEE Region 10 Technical Conference (TENCON)*, 1994.

[14] Hyokyung Bahn, K. Koh, S. H. Noh, and Sang Lyul Min, "Efficient Replacement of Nonuniform Objects in Web Caches," *IEEE Computer*, Vol. 35(6), Jun 2002.

[15] S. A. Barnett and G. J. Anido, "Performability of Disk-array-based Video Servers," *ACM/Springer-Verlag Multimedia Systems*, Vol. 6(1), pp. 60-74, 1998.

[16] P. Basu, R. Krishnan, and T. D. C. Little, "Optimal Stream Clustering Problems in Video-on-Demand," In *Proc. Parallel and Distributed Computing and Systems*, Oct.1998.

[17] P. Basu, A. Narayanan, R. Krishnan, and T. D. C. Little, "An Implementation of Dynamic Service Aggregation for Interactive Video Delivery," In *Proc. SPIE Multimedia Computing and Networking*, Jan.1998.

[18] A. Bestavros, "WWW Traffic Reduction and Load Balancing through Server-Based Caching," *IEEE Concurrency*, Vol. 5(1), pp. 56-67, 1997.

[19] S. Bhattacharjee, K. Calvert, and E. W. Zegura, "Self-Organizing Wide-Area Network Caches," In *Proc. INFOCOM*, 1998.

[20] J.-C. Bolot, S. M. Lamblot, and A. Simonian, "Design of Efficient Caching Schemes for the World Wide Web," In *Proc. 15th International Teletraffic Congress*, 1997.

[21] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "On the Implications of Zipf's Law for Web Caching," In *Proc. INFOCOM*, 1999.

[22] P. Brucker, "Scheduling Algorithms," Edition 3rd, New York : Springer, 2001.

[23] J. F. K. Bufford (ed), "Multimedia Systems," New York : ACM Press, 1994.

[24] Y. Cai, K. A. Hua, and K. Vu, "Optimizing Patching Performance," In *Proc. SPIE Conference on Multimedia Computing and Networking (MMCN)*, 1999.

[25] W. Cai, P. Xavier, S. J. Turner, and Bu-Sung Lee, "A Scalable Architecture for Supporting Interactive Games on the Internet," In *Proc. 16th Workshop on Parallel and Distributed Simulation*, 2002.

[26] Cache Digest Specification - version 5, http://www.squid-cache.org/CacheDigest/cache-digest-v5.txt.

[27] P. Cao and S. Irani, "Cost-aware WWW Proxy Caching Algorithms," In *Proc. 1st USENIX Symposium on Internet Technologies and Systems*, pp. 193-206, 1997.

[28] Cache Array Routing Protocol (CARP) and Microsoft Proxy Server 2.0, http://msdn.microsoft.com/library/en-us/dnproxy/html/carp.asp.

[29] A. Carzaniga and A. L. Wolf, "Content-Based Networking: A New Communication Infrastructure," In *Proc. NSF Workshop on an Infrastructure for Mobile and Wireless Systems*, 2002.

[30] S.-H. Chan and S.-H. Yeung, "Client Buffering Techniques for Scalable Video Broadcasting over Broadband Networks with Low User Delay," *IEEE Trans. Broadcasting*, Vol. 48(1), 2002.

[31] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A Hierarchical Internet Object Cache," In *Proc. 1996 USENIX Technical Conference*, Jan. 1996.

[32] S. Chen, J. A. Stankovic, J. F. Kurose, and D. Towsley, "Performance Evaluation of Two New Disk Scheduling Algorithms for Real-Time Systems," *Journal of Real-Time Systems*, Vol. 3(3), 1991.

[33] Tien-Fu Chen and Jean-Loup Baer, "Effective Hardware-Based Data Prefetching for High-Performance Processors," *IEEE Trans. Computers*, Vol. 44(5), May 1995.

[34] A. Cohen and W. A. Burkhard, "Segmented Information Dispersal (SID) for Efficient Reconstruction in Fault-Tolerant Video Servers," In *Proc. ACM Multimedia*, 1997.

[35] R. Cucchiara, M. Piccardi, and A. Prati, "Temporal Analysis of Cache Prefetching Strategies for Multimedia Application," In *Proc. 20th IEEE International Performance, Computing, and Communications Conference*, Apr. 2001.

[36] A. Dan and D. Sitaram, "Buffer Management Policy for an On-demand Video Server," *Technical Report RC 19347*, IBM Research Report, 1994.

[37] A. Dan, P. Shahabuddin, D, Sitaram, and D. Towsley, "Channel Allocation under Batching and VCR Control in Video-On-Demand Systems," *Journal of Parallel and Distributed Computing (Special Issue on Multimedia Processing and Technology)*, Vol. 30(2), 1995.

[38] A. Dan, D. Sitaram, and P. Shahabuddin, "Dynamic Batching Policies for an On-demand Video Server," *ACM/Springer-Verlag Multimedia Systems*, Vol. 4, pp. 112-121, 1996.

[39] A. Dan and D. Sitaram, "Multimedia Caching Strategies for Heterogeneous Application and Server Environments", *Multimedia Tools and Applications*, Vol. 4(3), May 1997.

[40] Darwin Open Source Project, http://developer.apple.com/darwin/.

[41] J. K. Dey-Sircar, J. D. Salehi, J. F. Kurose, and D. Towsley, "Providing VCR Capabilities in Large-Scale Video Servers," In *Proc. ACM Multimedia*, 1994.

[42] Li-gang Dong and B. Veeravalli, "Design of Object Replacement Strategies for Cooperative Web Proxy Caching," Revision version, To appear in *Multimedia tools and applications*, Kluwer Academic, Jun. 2002.

[43] S. G. Dykes, C. L. Jeffery, and S. Das, "Taxonomy and Design Analysis for Distributed Web Caching," In *Proc. 32th Hawaii International Conference on System Sciences*, Jan. 1999.

[44] S. Erfani and M. Malek,"Issues in Networked Multimedia Services," In *Proc. IEEE 39th Midwest Symposium on Circuits and Systems*, Vol. 3, 1996.

[45] H. Fahmi, S. Baqai, A. Bashandy, and A. Ghafoor, "Dynamic Resource Allocation for Multimedia Document Retrieval over High Speed LANs," *Multimedia Tools and Applications*, Vol. 8(1), pp. 91-114, 1999.

[46] L. Fan, P. Cao, and Q. Jacobson, "Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance," In *Proc. Joint International Conference on Measurement and Modeling of Computer Systems*, May 1999.

[47] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *IEEE/ACM Trans. Networking*, Vol. 8(3), 2000.

[48] Z. Fei, I. Kamel, S. Mukherjee, and M. H. Ammar, "Providing Interactive Functions for Staggered Multicast Near Video-on-Demand Systems," In *Proc. International Conference on Multimedia Computing and Systems (ICMCS)*, 1999.

[49] A. P. Foong, Yu-Hen Hu, and D. M. Heisey, "Logistic Regression in an Adaptive Web Cache," *IEEE Internet Computing*, Vol. 3(5), pp. 27 -36, 1999.

[50] P. Frossard and O. Verscheure, "Batched Patch Caching for Streaming Media," *IEEE Communications Letters*, Vol. 6(4), 2002.

[51] B. Furht, D. Kalra, F. L. Kitson, A. A. Rodriguez, and W. E. Wall, "Design Issues for Interactive Television Systems," *IEEE Computer*, Vol. 28, May 1995.

[52] B. Furht, S. W. Smoliar, and HongJiang Zhang, "Video and Image Processing in Multimedia Systems," *Boston: Kluwer Academic Publishers*, 1995.

[53] S. Gadde, M. Rabinovich, and J. Chase, "Reduce, Reuse, Recycle: An Approach to Building Large Internet Caches ," In *Proc. 6th Workshop on Hot Topics in Operating Systems (HotOS-VI)*, 1997.

[54] S. Gadde, J. Chase, and M. Rabinovich, "Directory Structures for Scalable Internet Caches," *Technical Report CS-1997-18*, Duke university, Nov.1997.

[55] J. Gemmell and S. Christodoulakis, "Principles of Delay-sensitive Multimedia Data Storage and Retrieval," *ACM Trans. Information Systems*, Vol. 10(1), pp. 51-90, 1992.

[56] D. Ghose and H-J. Kim, "Scheduling Video Streams in Video-on-Demand Systems: A Survey", *Multimedia Tools and Applications*, Vol. 11, pp. 167-195, 2000.

[57] G. A. Gibson, J. S. Vitter, and J. Wilkes, "Strategic Directions in Storage I/O Issues in Large-scale Computing," *ACM Computing Surveys*, Vol. 28(4), pp. 779-793, 1996.

[58] L. Golubchik, J. C. S. Lui, and R, R, Muntz, "Adaptive Piggybacking: A Novel Technique for Data Sharing in Video-On-Demand Storage Servers," *ACM/Springer-Verlag Multimedia Systems*, Vol. 4(3), 1995.

[59] P. Goyal, X. Guo, and H. M. Vin, "A Hierarchical CPU Scheduler for Multimedia Operating Systems," In *Proc. USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 1996.

[60] K. A. Hua and S. Sheu, "Skyscraper Broadcasting: a New Broadcasting Scheme for Metropolitan Video-on-demand Systems," In *Proc. ACM SIGCOMM'97 conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, Sep.1997.

[61] K. A. Hua, Y. Cai, and S. Sheu, "Patching: A Multicast Technique for True Video-on-Demand Services," In *Proc. ACM Multimedia*, 1998.

[62] D. M. Jacobson and J. Wilkes, "Disk Scheduling Algorithms Based on Rotational Position," *Technical Report of Hewlett Packard Labs*, February 1991.

[63] D. Jadav, A. N. Choudhary, and P. B. Berra, "Techniques for Increasing the Stream Capacity of a High-Performance Multimedia Server," *IEEE Trans. Knowledge and Data Engineering*, Vol. 11(2), 1999.

[64] Shudong Jin and A. Bestavros, "Popularity-aware Greedy Dual-size Web Proxy Caching Algorithms," In *Proc. 20th International Conference on Distributed Computing Systems*, pp. 254-261, 2000.

[65] Shudong Jin and A. Bestavros, "Greedy dual* Web Caching Algorithm: Exploiting the Two Sources of Temporal Locality in Web Request Streams," In *Proc. 5th International Workshop on Web Caching and Content Delivery*, 2000.

[66] Li-Shen Juhn and Li-Ming Tseng, "Harmonic Broadcasting for Video-on-Demand Service," *IEEE Trans. Broadcasting*, Vol. 43(3), 1997.

[67] Li-Shen Juhn and Li-Ming Tseng, "Fast Data Broadcasting and Receiving Scheme for Popular Video Service," *IEEE Trans. Broadcasting*, Vol. (44)(1), 1998.

[68] J. Jung, D. Lee, and K. Chon, "Proactive Web Caching with Cumulative Prefetching for Large Multimedia Data," *Computer Networks*, Vol. 33(1-6), pp. 645-655, 2000.

[69] M. Kamath, D. Towsley, and K. Ramamritham. "Buffer Management for Continuous Media Sharing in Multimedia Database Systems," *Technical Report 94-11*, University of Massachusetts, Feb. 1994.

[70] M. Kamath, K. Ramamritham, and D. Towsley, "Continuous Media Sharing in Multimedia Database Systems," in *Proc. 4th International Conference on Database Systems for Advanced Applications*, Apr. 1995.

[71] J. Kangasharju, F. Hartanto, M. Reisslein, and K. W. Ross, "Distributing Layered Encoded Video through Caches," *IEEE Trans. Computers*, Vol. 51(6), 2002.

[72] M. F. Khan, A. Ghafoor, and M. N. Ayyaz, "Design and Evaluation of Disk Scheduling Policies for High-Demand Multimedia Servers," In *Proc. International Conference on Data Engineering (ICDE)*, pp. 592-599, 1999.

[73] Sooncheol Kim, Cheolmin Kim, and Yookun Cho,"An Effective Resource Management for Variable Bit Rate Video-On-Demand Server," In *Proc. 23rd Euromicro Conference*, 1997.

[74] H. J. Kim and Y. Zhu, "Channel Allocation Problem in VoD System Using Both Batching and Adaptive Piggybacking," *IEEE Trans. Consumer Electronics*, Vol. 44(3), pp. 969-976, 1998.

[75] S. E. Kim, A. Sivasubramaniam, and C. R. Das, "Analyzing Cache Performance for Video Servers," In *Proc. International Conference on Parallel Processing Workshops*, pp. 38-47, 1998.

[76] S. Kim and Y. Choi, "An Efficient Cache Replacement Algorithm for Digital Television Environment," In *Proc. IEEE Region 10 Technical Conference (TENCON)*, Cheju, Korea, Sep.1999.

[77] B. Krishnamurthy and C. E. Wills, "Proxy Cache Coherency and Replacement - Towards a More Complete Picture," In *Proc. 19th IEEE International Conference on Distributed Computing Systems*, pp. 332-339, 1999.

[78] R. Krishnan, D. Venkatesh, and T. D. C. Little, "A Failure and Overload Tolerance Mechanism for Continuous Media Servers," In *Proc. ACM Multimedia*, 1997.

[79] P. Krishnan, D. Raz, and Y. Shavitt, "The Cache Location Problem," *IEEE/ACM Trans. Networking*, Vol. 8(5), 2000.

[80] J. Korst,"Random Duplicated Assignment: An Alternative to Striping in Video Servers," In *Proc. ACM Multimedia*, 1997.

[81] J. B. Kwon and H. Y. Yeom, "Providing VCR Functionality in Staggered Video Broadcasting," *IEEE Trans. Consumer Electronics*, Vo.48(1), 2002.

[82] S. Lau and J. C. S. Lui, "Scheduling and Data Layout Policies for a Near-line Multimedia Storage Architecture," *ACM/Springer-Verlag Multimedia System*, Vol. 5, pp. 310-323, 1997.

[83] S. Lau, J. C. S. Lui, and L. Golubchik, "Merging Video Streams in a Multimedia Storage Server: Complexity and Heuristics," *ACM/Springer-Verlag Multimedia System*, Vol. 6, pp. 29-42, 1998.

[84] Donghee Lee, Jongmoo Choi, Honggi Choe, S. H. Noh, S. L. Min, and Yookun Cho, "Implementation and Performance Evaluation of the LRFU Replacement Policy," In *Proc. 23th Euromicro Conference*, pp. 106-111, 1997.

[85] K. Lee, J. B. Kwon, and H. Y. Yeom, "Exploiting Caching for Realtime Multimedia Systems," In *Proc. Sixth IEEE International Conference on Multimedia Computing and Systems*, 1999.

[86] Chien-I Lee, Ye-In Chang, and Wei-Pang Yang, "An Efficient Strategy to Support Continuous Retrieval with Dynamic Bandwidths," In *Proc. Seventh International Conference on Parallel and Distributed Systems*, 2000.

[87] J. Y. B. Lee, "Parallel Video Servers - A Tutorial," *IEEE Multimedia*, Vol. 5(2), 1998.

[88] J. Y. B. Lee,"Concurrent Push-A Scheduling Algorithm for Push-Based Parallel Video Servers," *IEEE Trans. Circuits and Systems for Video Technology*, Vol. 9(3), 1999.

[89] J. Y. B. Lee and P. C. Wong, "Performance Analysis of a Pull-Based Parallel Video Server," *IEEE Trans. Parallel and Distributed Systems*, Vol. 11(12), 2000.

[90] J. Y. B. Lee, "Supporting Server-Level Fault Tolerance in Concurrent-Push-Based Parallel Video Servers," *IEEE Trans. Circuits and Systems for Video Technology*, Vol. 11(1), 2001.

[91] J. Y. B. Lee, "Buffer Management and Dimensioning for a Pull-Based Parallel Video Server," *IEEE Trans. Circuits and Systems for Video Technology*, Vol. 11(4), 2001.

[92] J. Y. B. Lee, "On a Unified Architecture for Video-on-Demand Services," *IEEE Trans. Multimedia*, Vol. 4, Mar.2002.

[93] J. Y. B. Lee and J. C. S. Lui, "Automatic Recovery from Disk Failure in Continuous-Media Servers," *IEEE Trans. Parallel and Distributed Systems*, Vol. 13(5), 2002.

[94] A. Leff, J. Wolf, and P. S. Yu, "Efficient LRU-based Buffering in a LAN Remote Caching Architecture," *IEEE Trans. Parallel and Distributed Systems*, Vol. 7(2), pp. 191-206, Feb.1996.

[95] M. Y. Y. Leung, J. C. S. Lui, and L. Golubchik, "Use of Analytical Performance Models for System Sizing and Resource Allocation in Interactive Video-on-Demand Systems Employing Data Sharing Techniques," *IEEE Trans. Knowledge and Data Engineering*, Vol. 14(3), 2002.

[96] V. O. K. Li and W. Liao,"Distributed Multimedia Systems," In *Proc. IEEE*, Vol. 85, Jul. 1997.

[97] B. Li, M. J. Golin, G. F. Italiano, X. Deng, and K. Sohraby, "On the Optimal Placement of Web Proxies in the Internet" In *Proc. INFOCOM*, Vol. 3, pp. 1282-1290, 1999.

[98] W. Liao and V. O. K. Li, "The Split and Merge Protocol for Interactive Video-on-Demand," *IEEE Multimedia*, Vol. 4(4), 1997.

[99] F. Y. -S. Lin, "Optimal Real-time Admission Control Algorithms for the Video-On-Demand (VOD) Service," *IEEE Trans. Broadcasting*, Vol. 44(4), Dec.1998.

[100] H. Lim and D. H. C. Du, "Protocol Considerations for Video Prefix-caching Proxy in Wide Area Networks," *Electronics letters*, Vol. 37(6), Mar.2001.

[101] C. Lindemann and O. Waldhorst, "Evaluating Hardware and Software Web Proxy Caching Solutions," *Report for Milestone 1 of the Project "Analysis of the Effectiveness of Web Caching in the Gigabit Research Network G-WiN"*, supported by the DFN-Verein with funds of the BMBF, Nov.2000.

[102] A. Luotonen, " Web Proxy Servers," *Prentice Hall PTR*, Upper Saddle River, NJ 07458, 1997.

[103] T. D. C. Little and D. Venkatesh, "Prospects for Interactive Video-on-Demand," *IEEE Multimedia*, Vol. 1, 1994.

[104] M. Makpangou, G. Pierre, C. Khoury, and N. Dorta, "Replicated Directory Service for Weakly Consistent Distributed Caches," In *Proc. International Conference on Distributed Computing Systems (ICDCS)*, May 1999.

[105] M. Mahran, M. Hashem, A. Mohamed, and A. Taha, "Design and Implementation of a Distance Educational System," In *Proc. Meditarranean Electrotechnical Conference (MELECON)*, 2002.

[106] E. P. Markatos and C. E. Chronaki, "A TOP-10 Approach to Prefetch on Web," In *Proc. International Networking conference (INET)*, 1998.

[107] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson, "Adaptive Web Caching: Towards a New Global Caching Architecture," *Computer Networks and ISDN Systems*, Vol. 30(22-23), pp. 2169-2177, Nov. 1998.

[108] Micron Crucial PC2100 128MB DDR-SDRAM Memory Review, http://www.a1-electronics.co.uk/Memory/DDR_CrucialPC2100.shtml.

[109] Microsoft Corporation Homepage, http://www.microsoft.com.

[110] P. V. Mundur, "An Intergrated Approach to End-to-End Analysis of Distributed Video-on-Demand Systems," *Ph.D thesis of George Mason Univerity of Virginia*, 2000.

[111] E. J. O'Neil, P. E. O'Neil, and G. Weikum."The LRU-K Page Replacement Algorithm for Database Disk Buffering," In *Proc. ACM SIGMOD International Conference on Management of Data*, 1993.

[112] B. Ozden, R. Rastogi, and A. Silberschatz, "A Framework for the Storage and Retrieval of Continuous media Data," In *Proc. International Conference on Multimedia Computing and Systems*, 1995.

[113] B. Ozden, R. Rastogi, and A. Silberschatz, "Buffer Replacement Algorithms for Multimedia Databases," *Multimedia Information Storage and Management*, ed by S.Chung, pp. 163-180, Kluwer Academic Publishers, Boston, MA, 1996.

[114] H. H. Pang, B. Jose, and M. S. Krishnan, "Resource Scheduling in a High-performance Multimedia Server," *IEEE Trans. Knowledge and Data Engineering*, Vol. 11(2), 1999.

[115] S. C. Park, Y. W. Park, and Y. E. Son, "A Proxy Server Management Scheme for Continuous Media Objects Based on Object Partitioning," In *Proc. Eighth International Conference on Parallel and Distributed Systems*, pp. 26-29, Jun. 2001.

[116] Sang-Hyun Park, Jae-Won Kim and Sung-Jea Ko, "MPEG I-Frame Arrangement and Admission Control for Video-on-Demand Systems," *IEEE Trans. Consumer Electronics*, Vol. 48(1), 2002.

[117] J. -F. Paris, "A Fixed-Delay Broadcasting Protocol for Video-on-Demand," In *Proc. ICCCN*, 2001.

[118] G. Pierre, M. V. Steen, and A. S. Tanenbaum, "Dynamically Selecting Optimal Distribution Strategies for Web Documents," *IEEE Trans. Computers*, Vol. 51(6), 2002.

[119] M. Pinedo, "Scheduling : Theory, Algorithms, and Systems," Englewood Cliffs, N.J. : Prentice Hall, 1995.

[120] B. Ping, B. Prabhakaran, and A. Srinivasan, "Retrieval Scheduling for Collaborative Multimedia Presentations", *ACM/Springer-Verlag Multimedia Systems*, Vol. 8(2), pp. 146-155, 2000.

[121] W. -F. Poon, K. -T. Lo, and J. Feng, "A Hybrid Delivery Strategy for a Video-on-Demand System With Customer Reneging Behavior," *IEEE Trans. Broadcasting*, Vol. 48(2), 2002.

[122] D. Povey and J. Harrison, "A Distributed Internet Cache," In *Proc. 20th Australasian Computer Science Conference*, Feb.1997.

[123] M. Rabinovich, J. Chase, and S. Gadde, "Not All Hits Are Created Equal: Cooperative Proxy Caching Over a Wide-area Network," *Computer Networks and ISDN system*, Vol. 30(22-23), pp. 2253-2259, 1998.

[124] P. V. Rangan, H. M. Vin, and S. Ramanathan, "Designing an On-Demand Multimedia Service," *IEEE Communications Magazine*, pp. 56-64, 1992.

[125] P. V. Rangan and H. M. Vin, "Efficient Storage Techniques for Digital Continuous Media," *IEEE Trans. Knowledge and Data Engineering*, Vol. 5(4), Aug. 1993.

[126] A. L. N. Reddy and J. Wyllie, "I/O Issues in a Multimedia System," *IEEE computer*, Vol. 27(3), pp. 67-74, 1994.

[127] R. Rejaie, H. Yu, M. Handley, and D. Estrin, "Multimedia Proxy Caching Mechanism for Quality Adaptive Streaming Applications in the Internet," In *Proc. INFOCOM*, 2000.

[128] "Relais: Cooperative Caches for the World-Wide Web," http://www-sor.inria.fr/projects/relais/index.html.en.

[129] L. Rizzo and L. Vicisano, "Replacement Policies for a Proxy Cache," *IEEE/ACM Trans. Networking*, Vol. 8(2), Apr.2000.

[130] L. Roberts, "Internet Growth Trends," *IEEE Computer*, Jan.2000.

[131] P. Rodriguez, C. Spanner, and E. W. Biersack, "Web Caching Architectures: Hierarchical and Distributed Caching," In *Proc. International Workshop on Web Content Caching and Distribution (WCW)*, 1999.

[132] Y. Rompogiannakis, G. Nerjes, P. Muth, M. Paterakis, and P. Triantafillou, "Disk Scheduling for Mixed-Media Workloads in a Multimedia Server," In *Proc. ACM Multimedia*, 1998.

[133] D. Rotem and J. L. Zhao, "Buffer Management for Video Database Systems," In *Proc. Eleventh International Conference on Data Engineering*, 1995.

[134] Y. Saito, "Optimistic Replication Algorithms," In *Proc. International Symposium on Distributed Computing*, 2000.

[135] J. R. Santos and R. Muntz, "Performance Analysis of the RIO Multimedia Storage System with Heterogeneous Disk Configurations," In *Proc. ACM Multimedia*, pp. 300-308, 1998.

[136] P. Scheuermann, J. Shim, and R. Vingralek, "A Case for Delay-conscious Caching of Web Documents," *Computer Networks and ISDN Systems*, Vol. 29(8-13), pp. 997-1005, 1997.

[137] SCSI Hard Disk Drives & SCSI Controller Cards, http://www.a1-electronics.co.uk/PcHardware/HardDrives/SCSI_diskdrives.shtml.

[138] S. Sen, J. Rexford, and D. Towsley, "Proxy Prefix Caching for Multimedia Streams," In *Proc. INFOCOM*, Vol. 3, pp. 1310-1319, 1999.

[139] C. Shahabi, S. Ghandeharizadeh, and S. Chaudhuri,"On Scheduling Atomic and Composite Continuous Media Objects," *IEEE Trans. Knowledge and Data Engineering*, Vol. 14(2), 2002.

[140] P. Shenoy and H. M. Vin, "Cello: A Disk Scheduling Framework for Next Generation Operating Systems," In *Proc. ACM SIGMETRICS, the Internation Conferenc on Measurement and Modeling of Computer System*, pp. 44-55, 1998.

[141] S. Sheu and K. A. Hua, "Virtual Batching: A New Scheduling Technique for Video-on-Demand Servers," In *Proc. Fifth International Conference on Database Systems for Advanced Applications*, Apr.1997.

[142] S. S. Y. Shim and Yen-Jen Lee, "Interactive TV: VoD Meets the Internet," *IEEE Computer*, Vol. 35(7), 2002.

[143] A. Sikeler, "Var-page-LRU: A Buffer Replacement Algorithm Supporting Different Page Sizes," In *Lecture Notes in Computer Science 303* ed by G. Goos and J. Hartmanis, pp. 336-351, Springer Verlag, 1988.

[144] W. D. Sincoskie, "Video on Demand: Is it Feasible?" In *Proc. GLOBECOM*, Vol. 1, pp. 201-205, 1990.

[145] M. Sinnwell and G. Weikum, "A Cost-Model-Based Online Method for Distributed Caching," In *Proc. 13th International Conference on Data Engineering*, 1997.

[146] D. Sitaram and A. Dan , "Multimedia servers : Design, Environments, and Applications ",*San Francisco, Calif. : Morgan Kaufman Pub*, 1999.

[147] Y. Smaragdakis, S. Kaplan, and P. Wilson, "EELRU: Simple and Effective Adaptive Page Replacement," *Measurement and Modeling of Computer Systems*, 1999.

[148] B. Sonah and M. R. Ito, "Cache Transparency in VoD system: Taking Advantage of Viewers' Flexibility," In *Proc. Third International Conference on Computational Intelligence and Multimedia Applications*, Sep.1999.

[149] B. Sonah and M. R. Ito, "Merging Interval Caching with Adaptive Viewers' Bias Based on Caching Strategy," In *Proc. Twenty-Third Annual International Computer Software and Applications Conference*, Oct.1999.

[150] Squid: Freely Available, Open Source Caching Software, http://www.squid-cache.org/.

[151] R. Steinmetz and K. Nahrstedt, "Multimedia: Computing, Communications, and Applications," Upper Saddle River, NJ : Prentice Hall, 1995.

[152] G. Stix, "The Triumph of the Light", *Scientific American*, Jan. 2001.

[153] "STREAM: Sustainable Memory Bandwidth in High Performance Computers," http://www.cs.virginia.edu/stream/.

[154] "Streaming MultiMedia Data," http://www.teamsolutions.co.uk/streaming.html.

[155] Xueyan Tang and S. T. Chanson,"Coordinated En-route Web Caching," *IEEE Trans. Computers*, Vol. 51(6), Jun 2002.

[156] R. Tewari, R. P. King, D. Kandlur, and D. M. Dias, "Placement of Multimedia Blocks on Zoned Disks," In *Proc. IS&T/SPIE Multimedia Computing and Networking*, Jan.1996.

[157] R. Tewari, H. Vin, A. Dan, and D. Sitaram, "Resource-based Caching for Web Servers," In *Proc. Proc. ACM/SPIE Multimedia Computing and Networking*, 1998.

[158] R. Tewari, M. Dahlin, H. M. Vin, and J. Kay, "Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet," *Technical Report CS98-04*, Department of Computer Sciences, UT Austin, Texas, USA, May 1998.

[159] Tsun-Ping J. To and B. Hamidzadeh, "Dynamic Real-time Scheduling Strategies for Interactive Continuous Media Servers, " *ACM/Springer-Verlag Multimedia Systems*, Vol. 7(2), 1999.

[160] J. Tse and A. J. Smith, "CPU Cache Prefetching: Timing Evaluation of Hardware Implementations," *IEEE Trans. Computers*, Vol. 47(5), May 1998.

[161] S. Uehara, O. Mizuno, and T. Kikuno, "An Implementation of Electronic Shopping Cart on the Web System Using Component-object Technology," In *Proc. Sixth International Workshop on Object-Oriented Real-Time Dependable Systems*, 2001.

[162] V. Valloppillil and K. W. Ross, "Cache Array Routing Protocol v1.0," http://icp.ircache.net/carp.txt.

[163] B. Veeravalli and G. D. Barlas, "Access Time Minimization for Distributed Multimedia Applications," *Multimedia Tools and Applications*, Kluwer Academic Publishers, Vol. 12(2/3), pp. 235-256, Nov.2000.

[164] H. M. Vin, P. Goyal, and A. Goyal, "A Statistical Admission Control Algorithm for Multimedia Servers," In *Proc. ACM Multimedia*, 1994.

[165] H. M. Vin, A. Goyal, and P. Goyal, "Algorithms for Designing Large-Scale Multimedia Servers," *Computer Communications*, Vol. 18(3), 1995.

[166] S. Viswanathan and T. Imielinski, "Metropolitan Area video-on-demand Service Using Pyramid Broadcasting," *ACM/Springer-Verlag Multimedia Systems*, Vol. 4, pp. 197-208, 1996.

[167] O. Verscheure, C. Venkatramani, P. Frossard, and L. Amini, "Joint Server Scheduling and Proxy Caching for Video Delivery," *Computer Communication*, Vol. 25, 2002.

[168] Y. Wang, J. C. L. Liu, D. H. C. Du, and J. Hsieh, "Efficient Video File Allocation Schemes for Video-on-Demand Services," *ACM/Springer-Verlag Multimedia System*, Vol. 5(5), 1997.

[169] Y. Wang, Z. L. 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 *Proc. IEEE INFOCOM*, Apr. 1998.

[170] D. Wessels and K. Claffy, "ICPv2 Protocol Specification," *IETF RFC 2186*, Sep.1997.

[171] D. Wessels and K. Claffy, "ICPv2 Application Specification," *IETF RFC 2187*, Sep.1997.

[172] P. White and J. Crowcroft, "Optimized Batch Patching with Classes of Service," *ACM Communications Review*, Oct.2000.

[173] S. Williams, M. Abrams, C.R.Standridge, G.Abdulla, and E.A.Fox, "Removal Policies in Network Caches for World-Wide Web Documents,", In *Proc. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Aug. 1996.

[174] R. Wijayaratne and A. L. N. Reddy, "Integrated QoS management for Disk I/O," In *Proc. IEEE International Conference on Multimedia Computing and Systems*, Vol. 1, pp. 487-492, 1999.

[175] J. L. Wolf, P. S. Yu, and H. Shachnai, "Disk Load Balancing for Video-on-Demand Systems," *ACM/Springer-Verlag Multimedia System*, Vol. 5(6), 1997.

[176] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy, "On the Scale and Performance of Cooperative Web Proxy Caching," In *Proc. 17th ACM Symposium on Operating Systems Principles*, Dec. 1999.

[177] R. P. Wooster and M. Abrams, "Proxy Caching that Estimates Page Load Delays," *Computer Networks and ISDN Systems*, Vol. 29(8-13), pp. 977-986, Sep. 1997.

[178] B. L. Worthington, G. R. Ganger, and Y. N. Patt, "Scheduling Algorithms for Modern Disk Drives," In *Proc. ACM SIGMETRICS*, 1994.

[179] Min-You Wu and W. Shu, "Efficient Support for Interactive Browsing Operations in Clustered CBR Video Servers," *IEEE Trans. Multimedia*, Vol. 4(1), 2002.

[180] Zhe Xiang, Zhun Zhong, and Yuzhuo Zhong, "A Cache Cooperation Management for Wireless Multimedia Streaming," In *Proc. International Conference on Information Technology and Information Networks (ICII)*, 2001.

[181] N. Young, "On-line Caching as Cache Size Varies," In *Proc. Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 241-250, Jan. 1991.

[182] P. S. Yu, M. S. Chen, and D. D. Kandlur, "Grouped Sweeping Scheduling for DASD-based Multimedia Storage Management," *ACM/Springer-Verlag Multimedie System*, Vol. 1(3), pp. 99-109, 1993.

[183] Q. Zhang and J. -F. Paris, "A Channel-Based Heuristic Distribution Protocol for Video-on-Demand," In *Proc. IEEE International Conference on Multimedia and Expo (ICME)*, 2002.

[184] A. N. Zincir-Heywood, M. I. Heywood, and C. R. Chatwin, "Object-Orientated Design of Digital Library Platforms for Multiagent Environments," *IEEE Trans. Knowledge and Data Engineering*, Vol. 14(2), 2002.

Appendix A: Author's Publication

[1] Li-gang Dong, B. Veeravalli and C.C. Ko, "Design and Analysis of Efficient Remote Caching Strategies for LAN based Architectures," In *Proc. IEEE International Conference on Networks*, Singapore, 2000.

[2] Li-gang Dong, V. Bharadwaj, and C. C. Ko, "Multiple Servers Retrieval Strategy for Movie On Demand Multimedia Services on Distributed Networks", In *Proc. Internet and Multimedia Systems and Applications (IMSA)*, Las Vegas, Florida, Nov. 2000.

[3] Li-gang Dong, B. Veeravalli and C.C. Ko, "Efficient Movie Retrieval Strategies for Movie-On-Demand Multimedia Services on Distributed Networks," Accepted by *Multimedia Tools and Applications*, Kluwer Academic, Aug.2001.

[4] Li-gang Dong and B. Veeravalli, "Design of Object Replacement Strategies for Cooperative Web Proxy Caching," Accepted by *Multimedia Tools and Applications*, Kluwer Academic, Jun. 2002.

[5] Li-gang Dong and B. Veeravalli, "Design and Analysis of a Variable Bit Rate Caching Algorithm for Continuous Media Data," submitted to *IEEE Trans. Circuits and System for Video Technology*, Jan. 2002.

[6] Li-gang Dong, B. Veeravalli, and Viktor K. Prasanna, "Design and Analysis of Interval-level Caching Strategies for Continuous Media Data," submitted to *ACM/Springer-Verlag Multimedia Systems*, Feb. 2002.