Run-Time Optimization of Heterogeneous Media Access in a Multimedia Server

Tsun-Ping J. To and Babak Hamidzadeh

***Abstract*—Discrete media (DM) data throughput is vital to systems that need to support the heterogeneity and variety of data found in interactive hypermedia and digital library applications. Therefore, a multimedia server's ability in delivering a high DM data throughput without degrading its *CM* data throughput deserves more attention. In this paper, we address this issue by optimizing the use of disk bandwidth spent on CM service at run-time and redirecting the saved bandwidth to service DM requests. Based on a new cost model of disk access incurred in CM service, we formulate strategies to control the size of each disk read at run time. These strategies improve the efficiency of each disk access and improve the DM data throughput of a multimedia server, without jeopardizing the CM throughput. Through experimental evaluations, the improvements achieved are demonstrated. Furthermore, efficient control of the allocation of disk bandwidth between DM and CM services over a wide range is also demonstrated.**

***Index Terms*—Discrete media, multimedia server, readsize optimization.**

I. Introduction

VIDEO and audio programs are referred to as *continuous media* (CM) due to their real-time delivery requirement, whereas text, graphics and image data are referred to as *discrete media* (DM). A server designed to provide real-time storage and retrieval of CM data streams is known as a *CM server* [1]–[4]. While the main design objective of CM servers is to maximize CM data throughput, DM data throughput has often been neglected or undermined. In the near future, applications for interactive television systems and distributed multimedia systems such as the World Wide Web will require real-time delivery of both CM and DM data. In particular, upcoming multimedia formats such as HyTime [5] and MHEG [6] combines related CM and DM information in temporal and hyperlinked compositions. *Hypermedia* (HM), a superset of CM, are hypergraph-structured programmes that consist of video branches, as well as loosely time-tied DM data, separated by user-interaction nodes [7], [8]. As a multimedia user navigates, the data requested may switch frequently between DM and CM. The ratio of DM to CM loads imposed onto a multimedia server can then vary considerably over time. Consequently, the performance of multimedia servers in the delivery of heterogeneous data requires more attention.

In many multimedia and hypermedia systems, the application and its server may require transparent storage and retrieval of both types of data. In other words, the application and the server may regard the data of different types as an integrated and monolithic logical entity. So, the server may be designed to keep heterogeneous data associated with the same application onto the same physical storage unit. Further, if there is a significant mismatch in the amount of data available from each type, storing the two types on separate storage units will result in wastage of storage space. If, for example, we support a small amount of DM data compared to CM data, storing CM and DM on separate storage units would result in shortage of space for CM and in waste of space on the unit where DM data resides. As the performance and storage capacity of workstations/PC's keep on growing, they may eventually play the role of peer multimedia servers to others. Multimedia server capabilities will then migrate into future PC operating systems. Placing CM and DM data on separate devices may then not be cost effective. Efficient techniques to retrieve heterogeneous data from the same device is, therefore, needed.

Typically, a CM server exercises admission control and in a predetermined repetitive sequence services the admitted CM requests by refilling memory buffers used to hold the data read ahead for each CM data stream. In each round of service, a server provides one refill service to each CM stream [1]–[4]. DM data requests can only be serviced during times at which a server can safely pause real-time stream services [1], [7], [9], [10]. However, a server fully engaged in precommitted CM data delivery may not be able to yield substantial time to serve pending DM data requests. To improve DM throughput without degrading CM throughput, disk access efficiency needs to be reduced dynamically at run-time.

In the past, seek and rotational disk overheads have been modeled to occur before reading of data. These overheads are assumed to be independent of the size of the data read. In this paper, we shall introduce a new cost model, referred to as ***post-reading overhead*** model, in which we view the overhead associated with a disk access to occur after the read, instead of before the read. Based on this overhead model, we shall introduce three readsize control strategies to maximize reading time and minimize rotational overheads. The first strategy, MAX, maximizes the ratio of time spent on reading to the time spent on overheads. The second, ZROT, eliminates as far as possible the rotational latency incurred after a read. The third strategy, OPT, dynamically maximizes the disk efficiency. Through analytical and experimental evaluation, we shall demonstrate the

Manuscript received April 22, 1999; revised November 1, 1999. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Jan-Ming Ho.

T.-P. J. To is with the Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong (e-mail: enjimmy@polyu.edu.hk).

B. Hamidzadeh is with the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, B.C., V6T 1Z4 Canada (e-mail: babak@ece.ubc.ca).

Publisher Item Identifier S 1520-9210(00)01701-6.

positive effects of these readsize control strategies on efficiency and throughput. Furthermore, we shall also show how a server, by means of a mechanism called ***buffer discounting***, can efficiently exercise control over the ratio of disk bandwidth shared between real-time CM streams and sporadic DM requests.

The remainder of this paper is organized as follows. In Section II, we shall review the related work. In Section III, we shall describe our cost model of disk access in servicing CM streams and formulate our readsize control strategies. In Section IV, we shall discuss mechanisms for improving DM data throughput. We shall present our experimental results in Section V, and draw our conclusions in Section VI.

II. Related Work

The historical development and various issues in designing multimedia servers is well covered in [2] and [3]. Anderson's pioneering work [1] on CM server gives a good analytical background in admission control and buffer management. In his work, the need to serve DM requests (referred to as sporadic services) is already well recognised. Anderson's file server, named as CMFS [1], supports DM delivery by slack times left over from CM services, or, at a priority lower than CM services. He also shows that DM service can be increased by tightening the admission of CM service. Later, Gopal foresees the need for a hypermedia server to support the delivery of a substantial amount of DM data under relaxed time constraints [8]. His experiments on CMFS reveal that DM throughput and response time can become unacceptable under high CM load commitments. Reddy's SCAN-EDF [11] then attempts to safeguard the throughput of DM delivery by giving DM requests a higher priority of service over CM delivery. However, if ongoing CM data streams are lengthy and cannot be preempted, DM requests may still be starved. Therefore, SCAN-EDF's approach in DM delivery may not be very effective unless CM data streams are typically short.

Recent work by Shenoy and Vin proposes that access to different media types can be scheduled separately [12], [13]. In the file system of Symphony [12], data type specific techniques are used to support the storage and retrieval of heterogeneous data types. In the Cello disk scheduling framework [13], the disk bandwidth is first partitioned to serve specific application classes. Bandwidth for each application class is then scheduled by its class-specific scheduler at a finer level to serve data requests of the same class. These approaches are more effective when multiple storage devices are present, and when the ratios of load imposed do not fluctuate frequently. While the redirection of disk bandwidth is shown to be feasibly in the experimental evaluation of Cello, the direct tradeoff between throughputs of CM and DM data has not been clearly revealed.

In the attempt to reduce the sum of seek and rotational latency components of disk accesses, a number of techniques have been proposed early in the literature [14]–[16]. However, these techniques are targeted for servicing requests of small discrete data chunks typically found in data processing environments. The order of servicing arrived requests in a long queue is manipulated to reduce the total latencies. The effectiveness of these techniques often relies on assumptions (such as long request queue, small size of requests, and sometimes the ability of the disk drive to report its rotational position) which are not valid in multimedia systems. Vin et al. [17] explored the possibility of reducing the cumulative seek and rotation disk overheads by reordering the sequence of service in a VOD server at run-time. This approach involves searching for an optimal reordering of the ongoing service sequence. While reordering the service sequence can reduce the total overhead, the seek component of the total overhead often increases because the original seek sequence (SCAN) is most likely disturbed. This technique is meant to improve CM throughput, instead of addressing DM throughput.

Apart from techniques that allocate disk bandwidth among heterogeneous data accesses, techniques that aims to boost DM throughput without sacrificing CM throughput has not been proposed. We believe that such objective can be achieved, through improving the efficiency of each CM disk access, and redirecting the saved disk bandwidth to maximize DM throughput.

III. Improving the Efficiency of Continuous Media (CM) Service

Consider a server that exhaustively allocates disk-sector-sized memory buffers to CM streams and issues nonpreemptable disk reads to refill a variable amount of a CM stream's vacant buffers in each service. The server performs admission tests to control the number of CM requests it can accommodate. In each round of service, it services the CM streams successively in the order depicted by the service sequence (e.g. round-robin or CSCAN). Physical placement of CM data on the disk may be interleaved, scattered, or contiguous [16]. Contiguous placement, by which a complete CM file is stored contiguously on the disk, is known to be the most efficient placement method to utilize disk bandwidth [18]–[20] because long seeks will not be incurred within a transfer. Therefore, we shall employ the contiguous placement model in our subsequent analysis.

A. Post-Reading Overhead Model

Seek latency and rotational latency are incurred in switching service between CM streams. Independently of the seek time, the rotational latency that follows a seek is on the average half of a rotation time. In a multimedia server that employs seek-reducing techniques, the rotational latencies constitute a significant disk access overhead. In most previous studies, the seek and rotational overheads, before a read, are normally treated as the overhead cost of servicing a stream. In our ***post-reading overhead*** model, the stream-switching overhead is incurred after, rather than before, the read, as shown in Fig. 1.

The symbols used in Fig. 1 are defined as follows:

- f readsize;
- X disk data transfer rate;
- t_f actual reading time to read data of readsize $f = f/X$;
- t_{sw} actual software and controller overhead;
- t_{seek} actual seek overhead spent on switching streams;
- t_{rot} actual rotational overhead;
- t_{OH} actual stream switching overhead $= t_{\mathrm{sw}} + t_{\mathrm{seek}} + t_{\mathrm{rot}}$;

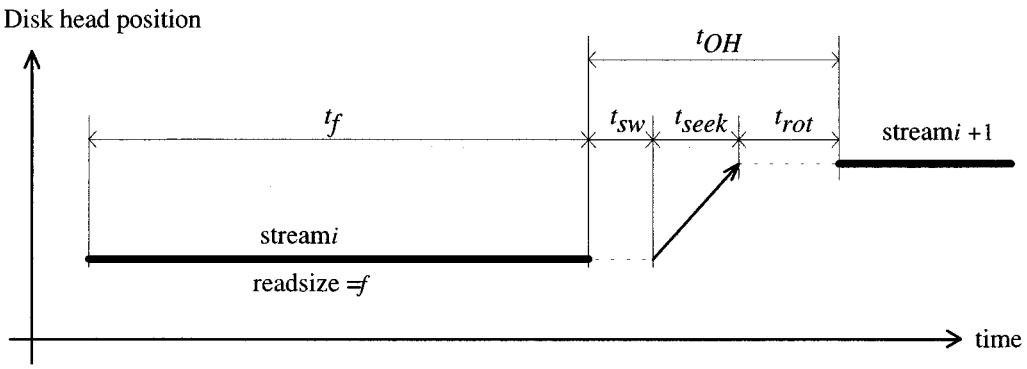


Fig. 1. Post-reading overhead model.

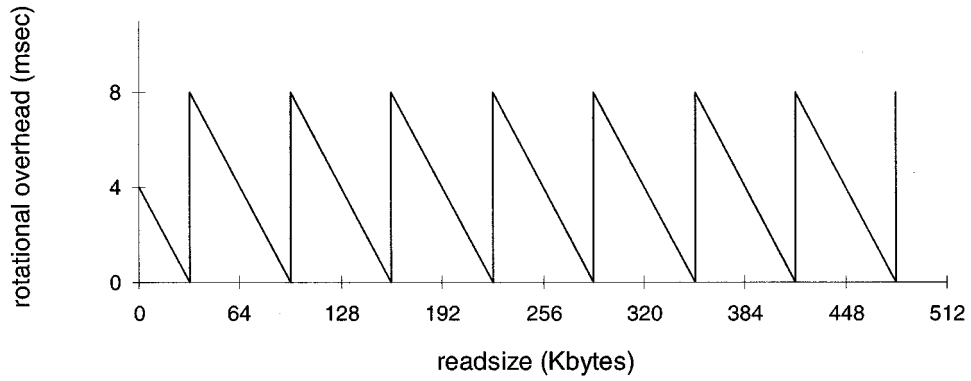


Fig. 2. Rotational overhead versus readsize.

The service time for a stream is the sum of its reading time and the following seek and rotational overheads incurred in switching the service to the succeeding stream. The physical rotational position (within value ranging from 0 to 2π radians) of a disk sector will be referred to as its *physical phase*, or *phase* in short. The physical phase at which reading must be continued for the next stream $(i+1)$ is the same physical phase at which reading stopped in the previous service to stream $(i+1)$. The post-reading rotational overhead $t_{\rm rot}$ is not constant and is dependent on the readsize f. Since contiguous placement is assumed in our model, the rotational overhead (as well as the total overhead) will decrease when the readsize f is increased. This is true except where the readsize is increased to a point where the rotational overhead reaches zero. At such points the rotational overhead will rebound to one full rotation time because the sector of the succeeding stream where reading is to begin will be missed when the seek completes. Therefore, the rotational overhead is a sawtooth function of the readsize, as shown in Fig. 2, for a disk with a track size of 64 Kbytes and a rotation time of 8 ms.

The *service time* $t_{\rm svc}$ to read data of readsize f is

$$t_{\rm svc} = t_f + t_{\rm sw} + t_{\rm seek} + t_{\rm rot}.$$

Using the above equation, the service time is plotted against readsize in Fig. 3. It can be seen that, the service time $t_{\rm svc}$ is a raised staircase function of the readsize f. More precisely, the service time increases stepwise by one rotation time for each increment of the readsize by one track (64 Kbytes). For each CM service, we can compute the ***efficiency*** η of the associated disk access as the ratio of the reading time to the service time, that is

$$\eta = t_f/(t_f + t_{\rm sw} + t_{\rm seek} + t_{\rm rot}).$$

The efficiency as a function of readsize is shown in Fig. 4. In a coarse view, we can see that the efficiency increases with the readsize. In each rising segment of the curve, the efficiency increases almost linearly with the readsize. However, when the readsize is increased beyond values that will incur another full rotation time, the efficiency drops abruptly. From the above observations, we can see that the efficiency of each disk access can be maximized by careful control of the readsize. Clearly, the highest efficiency can be achieved by executing readsizes that correspond to the peaks of the efficiency curve. However, buffer vacancy and real-time constraints may not allow us to always execute readsizes that correspond to the efficiency peaks, as we shall discuss next.

B. Readsize Constraints

Buffer vacancy and real-time constraints may not allow a server to arbitrarily vary readsizes. Therefore, we need to establish bounds within which a server can safely vary the readsize upon servicing a CM stream. First, let us define some terms and symbols:

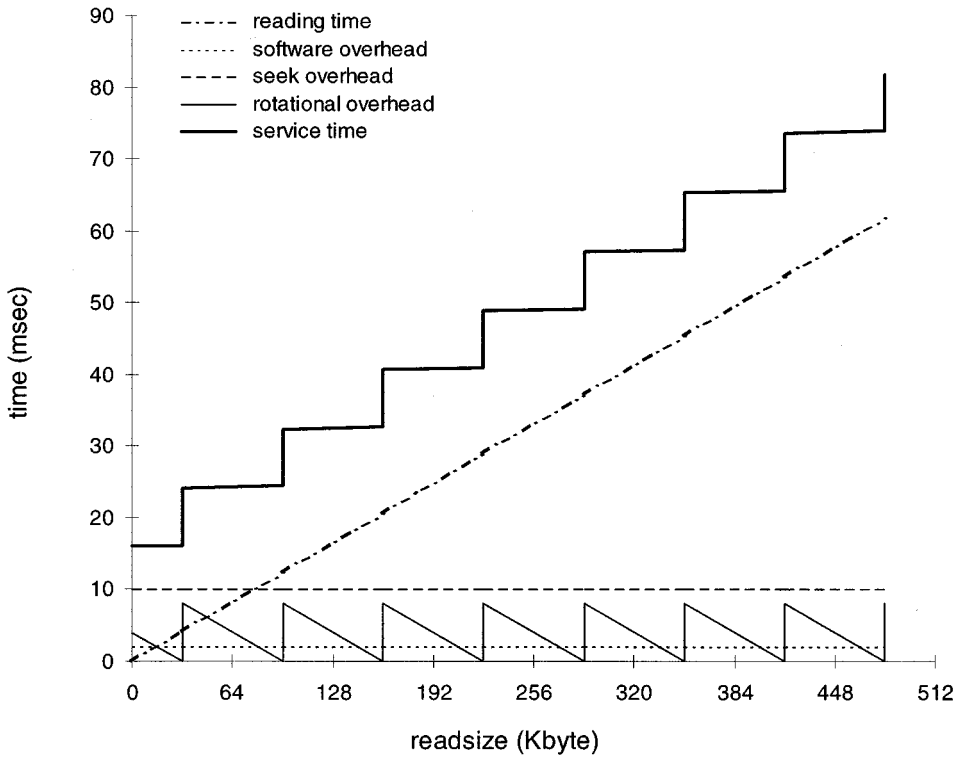


Fig. 3. Service time as a function of readsize.

Definition 1: A readsize is ***sufficient*** if the associated stream, after being serviced, will not suffer from starvation within one maximum service period, or before receiving another service.

Definition 2: A readsize is ***safe*** if its execution will not cause starvation to other streams.

Definition 3: A readsize is ***feasible*** if it does not cause buffer overrun.

Definition 4: A readsize is ***executable*** if it is sufficient, safe and feasible.

Definition 5: The ***nominal readsize*** $F_{\rm nom}$ (determined by the admission control algorithm) of a stream is the maximum amount of data that the stream will consume in one maximum service period.

Definition 6: The ***minimum buffer size*** $B_{\rm min}$ (determined by the admission control algorithm) of a stream is the minimum amount of buffers needed by a stream to hold data that can sustain consumption of the stream over the time spanned by its service time and one maximum service period. $(B_{\rm min} > F_{\rm nm})$.

Definition 7: The ***minimum readsize*** $f_{\rm min}$ is defined as the non-negative readsize at or above which a stream's consumption can be sustained for until the same stream receives another service. That is, $f_{\rm min} = MAX(0, B_{\rm min} - n)$ where n is the amount of filled (nonvacant) buffers of that stream just prior to receiving service.

Definition 8: The ***vacancy bound*** $f_{\rm vac}$ is defined as the amount of vacant buffers a stream has upon its turn to receive service. For a stream allocated with B sector-sized buffers out of which n buffers are filled, $f_{\rm vac} = B - n$.

We then establish Lemmas 1 to 6 (proofs are given in the appendix).

Lemma 1: Any readsize no smaller than the minimum readsize is sufficient.

Lemma 2: Any readsize that refills buffers to a level at or above the minimum buffer size is sufficient.

Lemma 3: Any readsize no larger than the Vacancy bound is feasible.

Lemma 4: A readsize equal to the nominal readsize is sufficient and safe, but may not be feasible.

Lemma 5: A nominal readsize truncated to match buffer vacancy is executable.

Lemma 6: A readsize that refills all vacant buffers of a stream is sufficient and feasible, but may not be safe.

By Lemma 6, an excessively large readsize of a stream can cause starvation to other streams. Therefore, we need to further impose a safety bound on the readsize. Let us denote the rate of consumption and the number of filled buffers of stream i as r_i and n_i, respectively. The data exhaustion time E_i is given by: $E_i = n_i/r_i$. At time t, the deadline D_i for the completion of the next service to stream i is given by $D_i = t + E_i$. Among the set of streams S, the earliest deadline D_e, is $D_e = {\rm MIN}(D_j : j \in S)$. Before servicing stream i, the server can first estimate the amount by which reading time can be safely extended. The slack time $T_{\rm SLACK}$, signifying the maximum amount of time by which all stream services can be safely postponed, is computed as: $T_{\rm SLACK} = D_e - (t + P)$, where P is the maximum service period computed in the admission test.

Definition 9: The ***Nonstarvation bound*** $f_{\rm ns}$ is defined as the maximum amount that can be read without causing starvation to other streams. For a sector-reading time of $T_{\rm sec}$, a nominal readsize of $F_{\rm nom}$ and a slack time of $T_{\rm SLACK}$, the nonstarvation

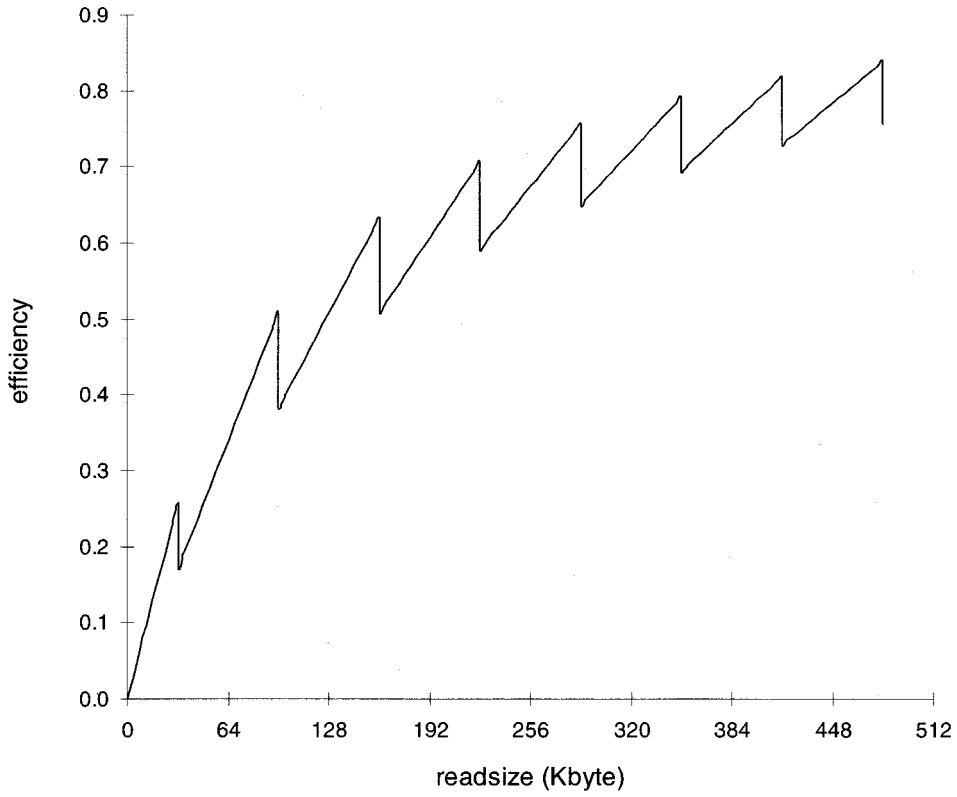


Fig. 4. Disk efficiency as a function of readsize.

bound is

$$f_{\rm ns} = \lfloor F_{\rm nom} + \max(\lfloor T_{\rm SLACK}/T_{\rm sec} \rfloor, 0) \rfloor.$$

Lemma 7: A readsize no larger than the Nonstarvation bound defined by Definition 9 is safe.

Using Lemma 3 and result of Theorem 1, we shall define a single upper bound for the readsize.

Definition 10: The ***maximum readsize*** $f_{\max}$ is defined as: $f_{\max} = \min(f_{\rm vac}, f_{\rm ns})$.

Lemma 8: Any readsize no larger than the maximum readsize is both safe and feasible.

Combining Lemma 1 and Lemma 8, we establish Theorem 1.

Theorem 1: Any readsize between the minimum and the maximum readsizes is executable. That is, readsize f is executable if $f_{\min} \leq f \leq f_{\max}$.

By Theorem 1, the server is free to vary the readsize within the minimum and maximum readsize bounds. However, these bounds are not static and depend on the real-time load committed by the server, as discussed below. Under full CM load, the amount of extra buffers given to each stream is often quite limited. The maximum readsize will likely be bounded by buffer vacancy. That is, $f_{\max} = f_{\rm vac}$. Therefore, under full load the server can vary the readsize only within a limited range. Under light CM load, each stream will receive a substantial amount of extra buffers, above that of a stream's minimum need computed by the admission control algorithm. So, the maximum readsize bound can be quite large. Furthermore, the minimum readsize can often be zero because with substantial data accumulation it is safe even if service to the stream is skipped in the current round of service. We expect $f_{\min} \cong 0$, and $f_{\max} \gg f_{\min}$. Therefore, under light load the server enjoys a large freedom in varying the read size.

C. Readsize Control Strategies

With exhaustive allocation of buffers, the actual amount of buffers allocated to a stream can be significantly more than its designated need. Using the extra buffers to extend the readsizes, the fraction of time the disk spends on uninterrupted productive reading will be increased. However, unconditionally increasing the readsize may increase the actual rotation overhead. In the following subsections, we shall introduce three readsize control strategies that can improve the disk efficiency at run-time.

1) The MAX Strategy: Under the **MAX** strategy, the maximum readsize $f_{\max}$ is always chosen for execution. With the MAX strategy, improvement in efficiency is very significant when the server is under light CM load and abundant buffer room exists due to exhaustive buffer allocation. Under full CM load, the advantage of this strategy becomes less prominent due to tighter constraints in the buffer vacancy. However, the total CM load imposed onto a multimedia server by interactive applications will likely fluctuate over time. Therefore, we do not expect a server to be fully loaded by CM deliveries most of the time.

2) The ZROT Strategy:

Definition 11: A ***zero-rotation readsize*** $f_{\rm ZR}$ is one which will incur no rotational latency in switching service to the next stream. After reading with a zero-rotation (ZR) readsize for

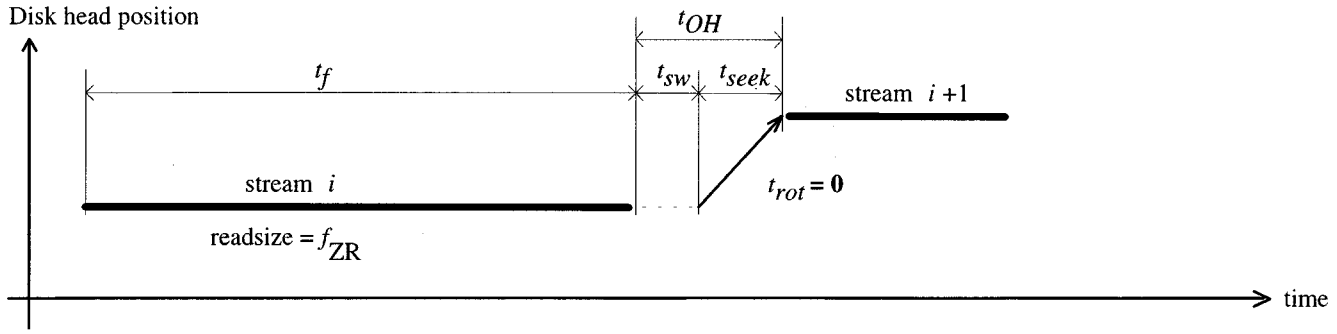


Fig. 5. A ZR readsize.

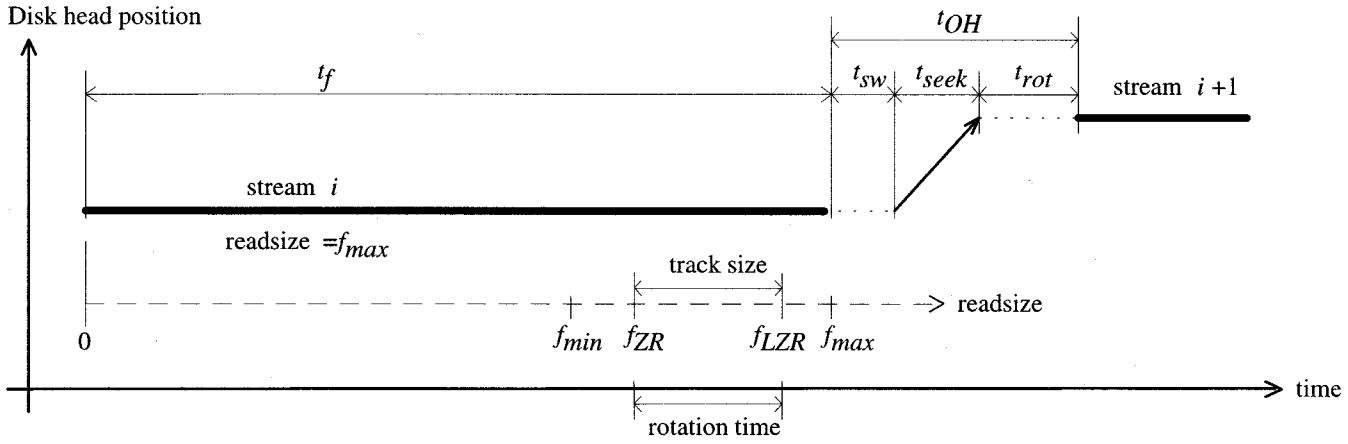


Fig. 6. Locating the ZR readsizes.

stream i, by the time the disk head seeks to the track for stream $i+1$, the sector to begin reading for stream $i+1$ also arrives under the head. This condition is graphically shown in Fig. 5.

It would not be safe to violate the nonstarvation bound, and it is also not feasible to read beyond the vacancy bound. However, between the minimum readsize $f_{\min}$ and the maximum readsize $f_{\max}$, there may exist one or more ZR (ZR) readsizes (the largest one denoted as the largest zero-rotation readsize f_{LZR}) with incremental differences of one track or one rotation time, as shown in Fig. 6.

Lemma 9: If the difference between two readsizes is no less than one track size, there must exist at least one ZR readsize between them.

Lemma 10: Between the minimum and the maximum readsizes, there exist $\lfloor f_{\max} - f_{\min}/N_{\mathrm{trk}} \rfloor$ ZR readsizes, with incremental difference of one track size (N_{trk}, in units of disk sectors) between successive pairs.

Definition 11: The ***largest-zero-rotation readsize*** f_{LZR} is defined as the largest of the ZR readsizes between the minimum and the maximum readsize bounds (see Fig. 6).

Theorem 2: The largest-zero-rotation (LZR) readsize is executable and has the highest efficiency among the ZR readsizes.

Under the **ZROT** readsize control strategy, the server always executes the LZR readsize whenever it exists, otherwise it executes the maximum readsize. In Fig. 7, it can be seen that while the maximum readsize incurs a post-reading rotational latency of t_{rot}, cutting the maximum readsize $f_{\max}$ down to the LZR readsize f_{LZR} eliminates the rotational latency. The service time spent in executing the LZR readsize is shorter than that of the maximum readsize by exactly one rotation time. From this we observe that, without violating safety constraints, ZROT cuts down the service time by one rotation time whenever it executes the LZR readsize instead of the maximum readsize. Under the ZROT strategy, the server uses Lemma 9 to confirm the existence of at least one ZR readsize. If Lemma 9 holds, the server can compute and execute the LZR readsize, instead of the maximum readsize.

3) The OPT Strategy: The negative effect in cutting down the maximized readsize to the LZR readsize is that the fraction of time spent on productive reading is also reduced. So, there is a tradeoff in efficiency. This is illustrated by the magnified efficiency curve in Fig. 8. The peaks in the efficiency curve correspond to the ZR readsizes f_{ZR}. In Fig. 8, It can be seen that the efficiency at point A is lower than that at point R, and the efficiency at point B is higher than that at point R. In the case where the maximum readsize is $f_{\max\_A}$, cutting $f_{\max\_A}$ down to the LZR readsize f_{LZR} will improve the efficiency. However, if the maximum readsize is $f_{\max\_B}$, replacing $f_{\max\_B}$ with f_{LZR} will degrade, rather than improve, efficiency.

To maximize efficiency, the server can choose whether to go with the LZR readsize, or fall back onto the maximum readsize. To make this decision, the server only needs to compute the efficiency of each of the two readsizes:

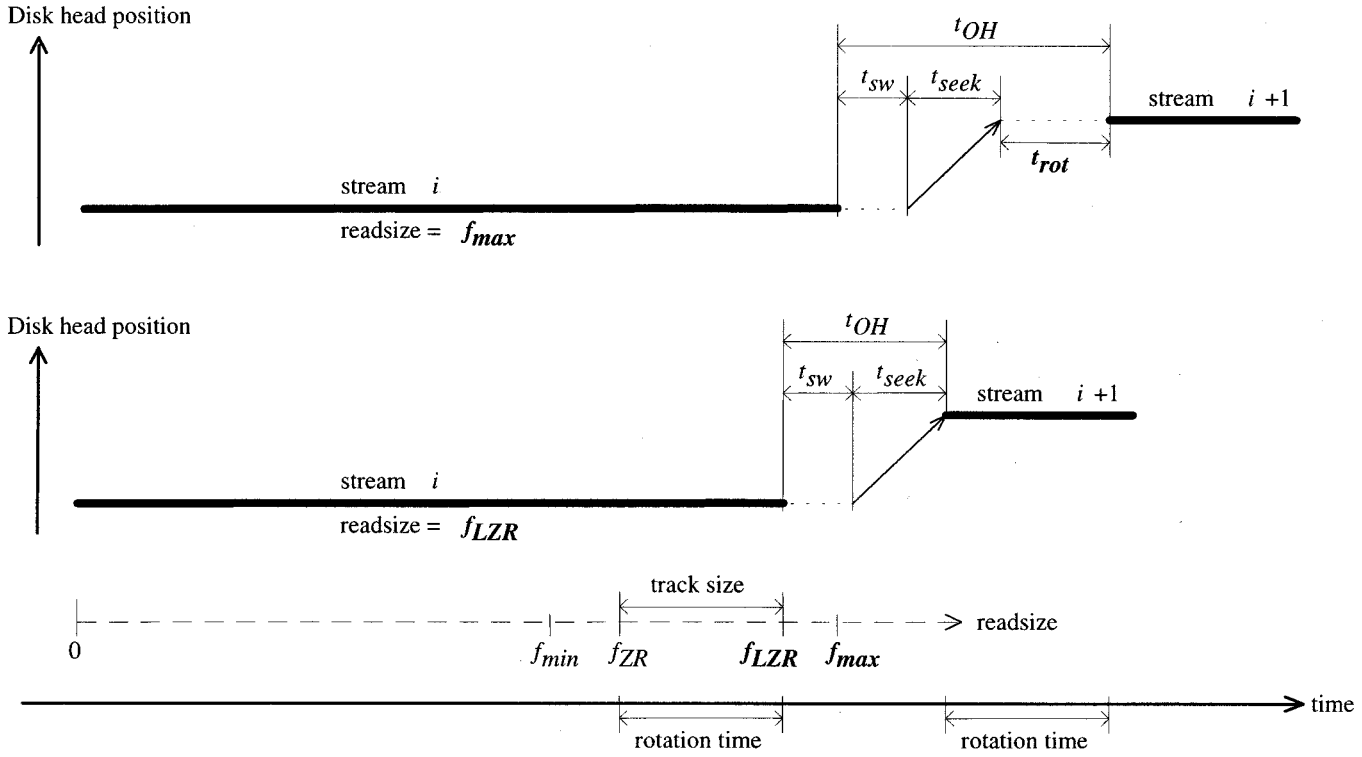


Fig. 7. Rotational overhead eliminated by ZROT.

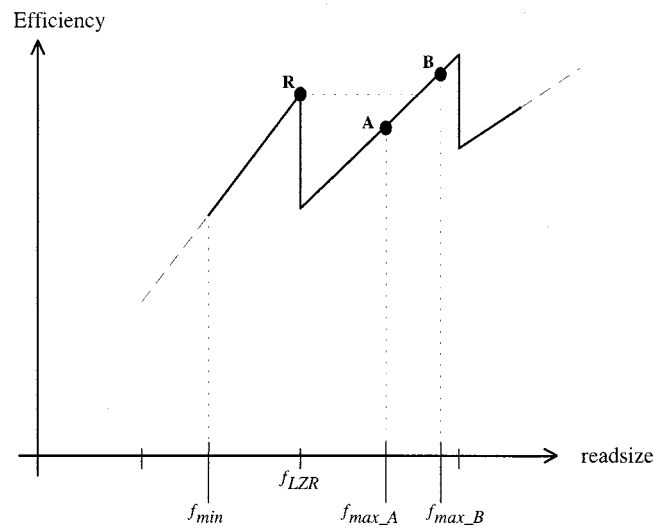


Fig. 8. Locating the optimal readsize.

- The efficiency of the maximum readsize is

$$\eta_{\max} = t_{f\_\max}/(t_{f\_\max} + t_{\rm sw} + t_{\rm seek} + t_{\rm rot}),$$

where $t_{f\_\max}$ is the reading time of $f_{\max}$.
- The efficiency of the LZR readsize is

$$\eta_{\rm LZR} = t_{f\_\rm LZR}/(t_{f\_\rm LZR} + t_{\rm sw} + t_{\rm seek}),$$

where $t_{f\_\rm LZR}$ is the reading time of $f_{\rm LZR}$.

Definition 13: The ***optimum readsize*** $f_{\rm OPT}$ is defined as follows:

$$\text{If } \eta_{\rm LZR} > \eta_{\max}, \quad f_{\rm OPT} = f_{\rm LZR}, \quad \text{otherwise } f_{\rm OPT} = f_{\max}.$$

Theorem 3: The optimum readsize is the most efficient executable readsize.

Under the **OPT** strategy, a server always executes the optimum readsize as defined above. Looking more carefully at the efficiency curve in Fig. 4 we can see that, with a small buffer capacity the readsizes will be small. The server will be operating on the lower part of the efficiency curve in Fig. 4, and the chances that $\eta_{\rm LZR} > \eta_{\max}$ will be small. On the other hand, at larger read sizes, the efficiency peaks are closer in value. That is, the chance of $\eta_{\rm LZR} > \eta_{\max}$ is high. So, it is more likely that the LZR readsize is more efficient than the maximum readsize, and the LZR readsize will be taken most of the time. Reasonably configured servers are commonly equipped with moderate buffer capacity, and are expected to operate in the middle range of the efficiency curve.

IV. Discrete Media (DM) Service Mechanisms

Precommitted real-time guarantees on CM stream deliveries do not allow a server to perform DM services at arbitrary times because CM service is necessarily paused whenever a server services DM requests [1], [9], [10], [20]. Thus, the server should observe deadlines of CM services, or monitor the buffer filling levels on CM streams, to determine the amount of time it can safely pause CM service. In Section III, we have established

theoretical grounds by which readsize control strategies can improve CM service efficiency. In this section, we shall further discuss how to improve DM service.

A. Inter-Round versus Intra-Round DM Service

Pausing CM service too frequently to service DM requests need not be undesirable when efficiency is considered. If a server services DM requests whenever CM service can be safely paused between successive services to CM streams, the server is said to perform DM service in an ***Intra-round*** fashion. Alternatively, if a server performs DM service only at the end of each round of CM service, it is said to perform DM service in an ***Inter-round*** fashion.

Insertion of DM service in the Intra-round fashion does have detrimental effects on efficiency. In seek-reducing schedulers, insertion of DM service into a service round essentially disturbs the originally assumed disk seek sequence. As a result, the actual seek overheads incurred will increase. Inter-round DM service, on the other hand, will not disturb the original CM service sequence or increase seek overhead. Furthermore, servicing a batch of DM requests between CM service rounds can open up opportunities for the server to reduce seek overheads of DM service (by reordering), if FCFS service order is not mandatory.

Between service rounds, time available for DM service is also considerably longer than the time available between individual CM services in a service round. Therefore, large DM requests will less likely be spliced when they are serviced in the Inter-round fashion. Since efficiency is our main concern, DM requests should be serviced in an Inter-round fashion so that CM service sequence will not be disturbed and the need to splice large DM requests can be minimized.

B. Bandwidth Reservation

Since DM service can only be performed using the time left after CM service, the simplest approach to reserve more time for DM service is to admit less CM streams. Mechanisms based on this principle are referred to as ***bandwidth reservation*** mechanisms by which the server can tighten admission of CM streams. How well the server can efficiently use the reserved bandwidth to improve its DM throughput is crucial. We do not wish to sacrifice a large amount of CM throughput to trade for a small improvement in DM throughput. In this respect, we can compare the performance of servers by their CM-to-DM ***throughput redirection*** curves. An example is shown in Fig. 9, where the DM throughputs of three servers are plotted against their respective CM throughputs in a decreasing order (decreasing CM throughput corresponds to increasing bandwidth reservation). If the entire range spanned by the curves in Fig. 9 is considered, Server 3 is the best because the major part of its curve is higher than that of the others. Since it is unlikely that a server would need to trade-off a large fraction of its CM throughput for DM throughput, our attention should be focused on the lower left region of the figure. In this region, Server 1 is the best. With only a small tradeoff in CM throughput, Server 1 can already deliver a substantial amount of DM throughput.

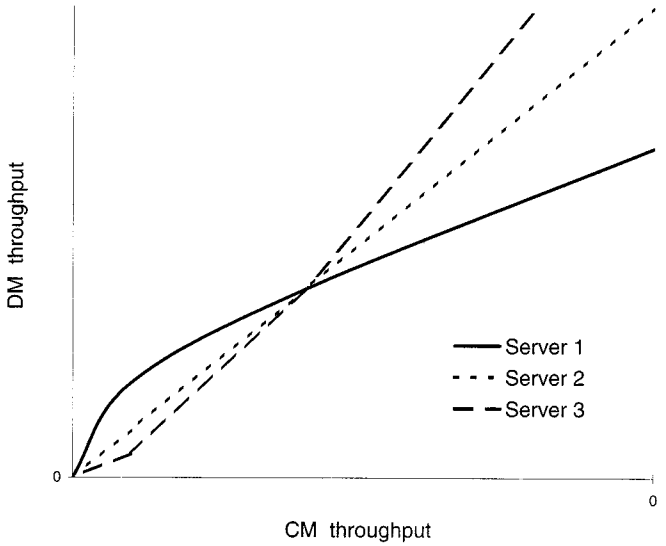


Fig. 9. CM-to-DM throughput redirection curves.

C. Buffer Discounting Technique

Unnecessarily reserving bandwidth permanently for nonexistent DM service demands can result in under-utilization of the CM throughput capacity of a server. A good bandwidth reservation mechanism should not impose a permanent degradation on the potential CM throughput of the server. Such a mechanism should allow the server to deliver a substantial amount of DM throughput when the need arises.

Memory buffers are needed to hold CM data read ahead by a server. For any given set of streams, the total buffer requirement $B_{\rm req}$ is a mathematical function of stream data rates, disk data rate and estimates on disk overhead [1], [9]–[11], [17], [18]. Admission control can always be formulated as a test on whether the buffer capacity $B_{\rm SYS}$ of a server can satisfy $B_{\rm SYS} \geq B_{\rm req}$. To flexibly exercise bandwidth reservation only on a need basis, we propose a simple mechanism called ***Buffer Discounting*** by which the server performs its admission test as if it has less buffer memory. That is, applying a *buffer discount* of D_b means using $D_b \cdot B_{\rm SYS} \geq B_{\rm req}$ as the admission criteria, instead of $B_{\rm SYS} \geq B_{\rm req}$. Imposing a *buffer discount* of 25% on 16 Mbytes of buffers means that the server performs its admission control test with 75%, or 12 Mbytes, of its buffers. Using this mechanism, fewer streams will then be admitted. The uncommitted bandwidth will automatically be directed to DM service. As long as all the buffers are exhaustively allocated, more buffer headroom will be given to each stream for data accumulation and maximizing readsizes. The probability of finding ZR readsizes will then be increased also. As a consequence, the server will be able to service more DM requests.

V. Experimental Evaluation

We employ a discrete-event approach to model and simulate the system. We model three types of resources in our hypermedia server: the disk, the memory buffers and the CPU. Each active entity in the system is modeled by one process. Active entities include a CPU scheduler, a disk scheduler, a buffer manager, an admission controller, a CM request generator, a DM

request generator, and the admitted CM streams. The server exhaustively allocates disk-sector sized memory buffers. In each round of service, the server services the set of admitted CM streams in a CSCAN order (increasing cylinder positions) and refill buffers by nonpreemptable disk reads. DM data requests are serviced in an Inter-round fashion using slack times as computed in [10]. The server services DM requests in a first-come-first-served (FCFS) order, possibly splicing large DM request into one or more disk reads of size up to one cylinder. The disk parameters used are set according to specifications of the latest high-performance high-capacity units. The seek times are modeled as a linear function of cylinder span, and range from 2 to 20 ms. Data of a CM file are stored in contiguously allocated disk sectors. Disk access latencies are modeled accurately by the disk process with head position, calculated seek delays, calculated rotation delays, and calculated reading delays. The rotation position of the disk is maintained as a modulo function of the simulated time. Rotation time of the disk is set to 8 ms.

Both real-time CM and DM requests are generated by Poisson arrivals. The server performs admission test upon arrival of a CM request or completion of an ongoing stream (as in [10]). The DM request queue of length 256 is kept filled with DM requests of size uniformly distributed between 1 to 512 Kbytes. The DM requests remain in the DM request queue until they are serviced. Each of our simulation runs realistically spans the system life cycle from start-up to shutdown over a period of two hours of simulated time. We measure CM throughput as the total amount of CM data actually retrieved within each simulation run. Since the size of DM requests is also variable, we measure DM throughput as the total amount of DM data delivered in each simulation run. We also record the rotational latency, disk efficiency and read ratio (ratio of the actual readsize to the nominal readsize) for each stream service.

To compare the performance of different readsize strategies, we shall simulate the MAX, ZROT and OPT strategies as well as a strategy called NOM. NOM represents typical servers which execute truncated nominal readsizes without extending readsize beyond the nominal value F_{nom}.

A. Performance Under Full Load

In general, the CM throughput increases with buffer capacity, and is quite independent of the readsize strategy used because it is mainly determined by admission control. In this experiment, request queues for both CM and DM requests are kept full. The server can always find a CM request waiting to be admitted and can always find a DM request to service.

Fig. 10 shows that the MAX strategy can already deliver about twice as much DM throughput compared to the NOM strategy. Across the major range of buffer capacities, the ZROT and OPT strategies can deliver at least three times of NOM's DM throughput, without detectable differences in the CM throughput recorded. The average rotational overhead of each strategy is shown in Fig. 11.

The reduction in rotational overhead experienced by the ZROT and the OPT strategies can be clearly seen. The effects of each readsize control strategy on the average readsize can be seen in Fig. 12. The read ratio of MAX is always higher than NOM because of read maximization. However, as ZROT and

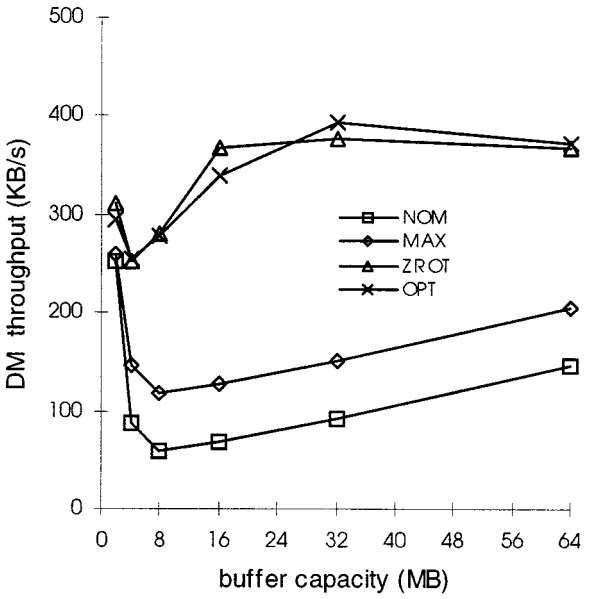


Fig. 10. DM throughput.

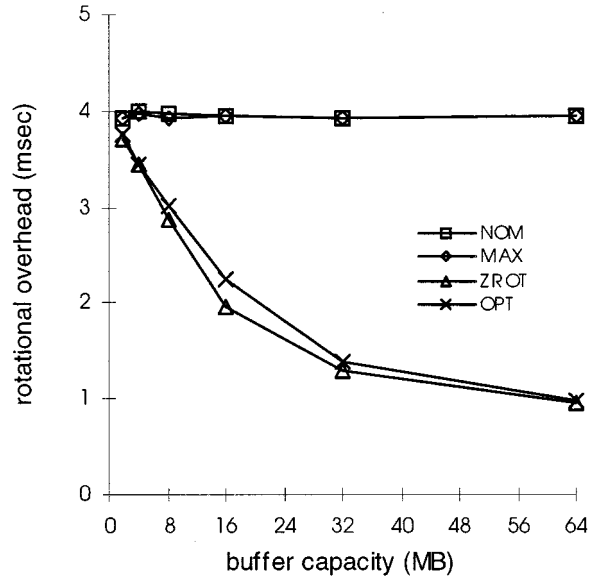


Fig. 11. Rotational overhead reduced by ZROT and OPT.

OPT cut down the maximized readsize to eliminate rotational overhead, their average read ratio is always lower than MAX. From the above, we can see how the underlying readsize control strategies boost the DM throughput capacity of a busy server.

B. Performance under Reduced CM Loads

Recorded for a buffer capacity of 8 Mbytes, the disk efficiency of each readsize strategy, at different CM request arrival rates, is shown in Fig. 13. The disk efficiency of the ZROT and OPT are higher across arrival rates.

In Fig. 13, we can see the ability of ZROT and OPT in reducing rotational overheads. At higher arrival rates, tight buffer headroom prevents them from cutting down readsizes to eliminate rotational overhead. At low arrival rates, the set of CM streams being serviced becomes smaller and less CM streams

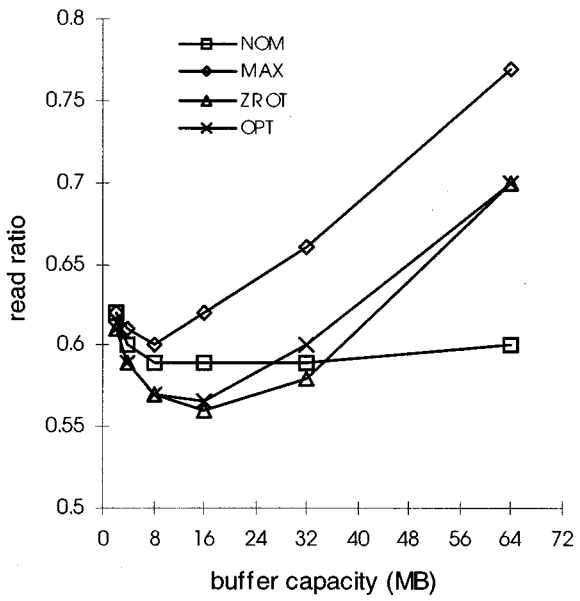


Fig. 12. Read ratio versus buffer capacity.

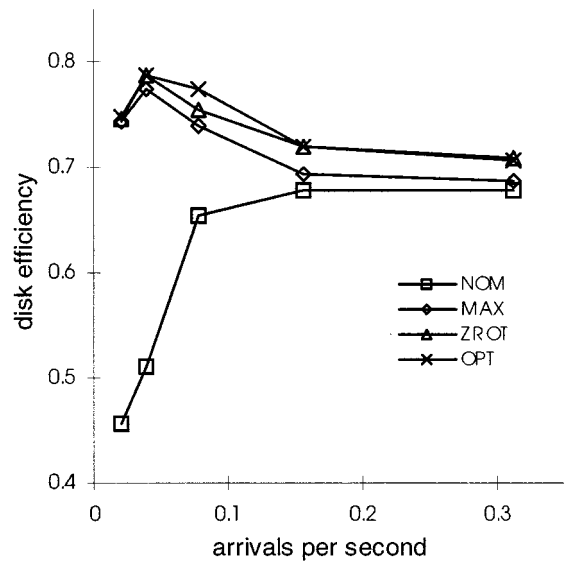


Fig. 13. Efficiency under different CM load.

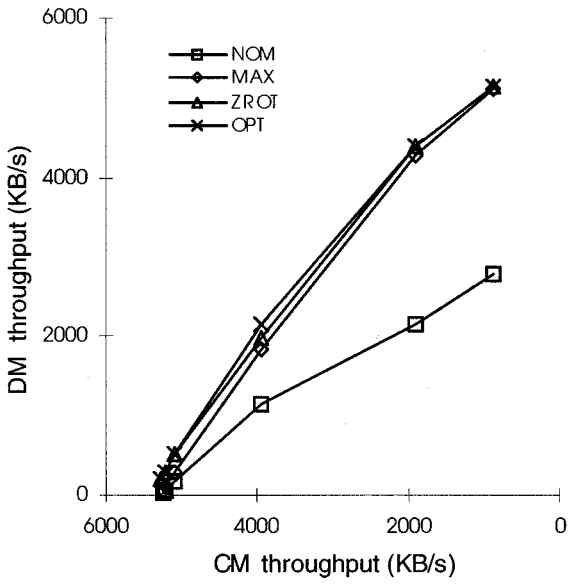


Fig. 14. Loading-induced throughput redirection.

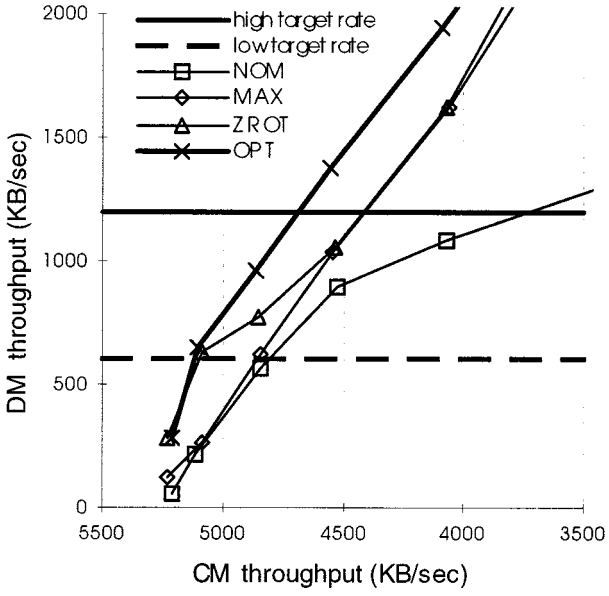


Fig. 15. Throughput redirection under server control.

can have the rotational overhead removed. This is why the observed rotational overhead increases gently slightly when arrival rates become extremely low.

As we decrease the arrival rate of real-time CM requests in our experiments, the demand on the throughput of real-time streams also decreases. With more accumulation of slack time, the disk bandwidth will be redirected to serve DM requests. Ideally, all the disk bandwidth unconsumed by stream service should be made available to DM service. Fig. 14 shows the DM throughput against decreasing stream throughput, as the arrival rate of stream requests is decreased. It can be seen that MAX, ZROT, and OPT are all efficient in their CM-to-DM throughput redirection. They can deliver significantly higher DM throughput than NOM across the full range of CM load levels.

C. Throughput Redirection Under Explicit Control

If the server needs to ensure certain levels of DM data traffic or improve response time to DM data requests, it can always exercise some form of control over stream admission. Under high arrival rates of both CM requests and DM requests, we conduct our last set of experiments with the buffer capacity set to 8 Mbytes. In this set of experiments, we employ the Buffer Discounting mechanism and increases buffer discount from zero to 60% in 15% steps. The DM throughput of each strategy is plotted against its corresponding CM throughput in Fig. 15.

With buffer discounts, the extra buffers allow more room for data accumulation and give the server more freedom in cutting down the readsize to reduce rotational overhead. Notably, the slope of the initial rising part of the OPT and ZROT curves are

TABLE I
CM THROUGHPUT SACRIFICED TO ACHIEVE A DM THROUGHPUT OF 1200 Kbytes/s

Strategy	Loss in CM throughput	
	(KB/sec)	(percentage)
NOM	1500	28.5 %
MAX	850	16.2 %
ZROT	850	16.2 %
OPT	510	9.8 %

greater than unity. This indicates that, by trading off only a small amount of CM throughput, both OPT and ZROT can achieve a substantial increase in the amount of DM throughput.

Our main concern here is the amount of CM throughput each technique needs to sacrifice in order to achieve the same target level of DM throughput. The amount of CM throughput each technique must sacrifice to deliver a moderate level of DM throughput can be easily measured in Fig. 15. To deliver a target DM throughput at 1200 Kbytes/sec (15% of the disk's raw bandwidth), we tabulate the measured tradeoff in CM throughput needed by each strategy in Table I.

To achieve this target DM throughput, the CM throughput given up by NOM is three times that of OPT. We can also see that OPT is now significantly better than ZROT. Summing up, we conclude that the OPT readsize control strategy can provide a high level of DM throughput at a very low cost in CM throughput.

VI. CONCLUSION

In this paper, we present solutions to improve the throughput of heterogeneous data in a hypermedia (HM) server. A dynamic approach is taken to improve DM throughput without sacrificing any CM throughput. Based on our Post-Reading Overhead model for disk accesses, we show that the efficiency of CM service is closely related to the readsize and, thus, can be improved through explicit control of readsize at run time. We derive constraints for varying readsize and establish theoretical grounds for three readsize control strategies, namely MAX, ZROT and OPT to improve the disk efficiency in servicing CM streams. We have analyzed the effects of readsize on the efficiency of disk accesses in a multimedia server. The efficiency improvements achieved by optimizing readsize is well demonstrated by our experiments. The reclaimed disk bandwidth resulting from the improved efficiency can be redirected to improve the DM data throughput of a multimedia server significantly, without disturbing the service sequence or sacrificing any CM throughput. Thus, our techniques can easily be used with round-robin and seek-reducing disk access sequencing algorithms. If the server needs to support a substancial level of DM throughput, bandwidth can be efficiently balanced between CM and DM service using the Buffer Discounting mechanism, once any of our readsize control strategies is employed.

APPENDIX I

PROOFS FOR LEMMAS AND THEOREMS

Lemma 1: Any readsize no smaller than the minimum readsize is sufficient.

Proof:: Let n' be the amount of filled buffers after a refill. Using the Minimal readsize $f_{\min}$ to perform a refill, $n' = n + f_{\min}$. By Definition 7, $f_{\min} = \mathrm{MAX}(0, B_{\min} - n)$. If $f_{\min} = 0$, then $n \geq B_{\min}$ and $n' \geq B_{\min}$. On the other hand, $f_{\min} > 0$ if and only if $B_{\min} > n$. Then, $n' = n + B_{\min} - n = B_{\min}$. Therefore, within the range of possible values for $f_{\min}, n' \geq B_{\min}$. By Definition 6, $n' \geq B_{\min}$ will guarantee nonstarvation of the stream until it receives its next service. By Definition 1, the minimum readsize is sufficient. □

Lemma 2: Any readsize that refills buffers to a level at or above the minimum buffer size is sufficient.

Proof:: Let n' be the amount of filled buffers after a refill. If $n' \geq B_{\min}$, by Definition 6 starvation of the stream will not occur for one maximum service period. By Definition 1, such readsize is sufficient. □

Lemma 3: Any readsize no larger than the Vacancy bound is feasible.

Proof:: If a refill is no larger than the Vacancy bound f_{vac}, the amount of filled buffers after the refill will not exceed the amount of buffers allocated to the stream. By Definition 3, such readsize is feasible. □

Lemma 4: A readsize equal to the nominal readsize is sufficient and safe, but may not be feasible.

Proof:: By Definition 5, the nominal readsize is sufficient. It is safe because admission control ensures that the total time taken by a round of service using the nominal readsizes will not be longer than the maximum service period. It is not feasible whenever the vacant buffers of a stream are smaller than its nominal readsize, or $B - n < F_{\mathrm{nom}}$. Since admission is conservative, the actual inter-service time experienced by a stream is always shorter than the estimated maximum service period. By virtue of such nonstarvation guarantee, between receiving successive services a stream will never vacate more buffers than the nominal readsize. That is, $B - n \leq F_{\mathrm{nom}}$ is guaranteed. This implies that $B - n < F_{\mathrm{nom}}$ is possible. Therefore, the nominal readsize may not be feasible. □

Lemma 5: A nominal readsize truncated to match buffer vacancy is executable.

Proof:: If the read is no greater than the nominal readsize, it will not stretch the time to complete a service round. Therefore, it is safe. Since the readsize is truncated to match buffer vacancy, by Definition 3 it is feasible. If truncation is performed, the amount of filled buffers after the refill will be at least $B_{\min}$, which is sufficient according to Lemma 2. Therefore, Lemma 5 follows. □

Lemma 6: A readsize that refills all vacant buffers is sufficient and feasible, but may not be safe.

Proof:: Let B be the amount of sector-sized buffers allocated to the stream, out of which n buffers are filled. Let n' be the amount of filled buffers after a refill. If all vacant buffers are refilled, $n' = B$. Since a stream will always be allocated no less buffers than its minimum buffer size $B_{\min}, B \geq B_{\min}$ and $n' \geq B_{\min}$. By Lemma 2, such readsize is sufficient. Refilling all vacant buffers cannot cause buffer overrun and is, by Definition 3, feasible. On the safety side, if the buffers returned by a large number of completed streams are exhaustively distributed to the incomplete streams. This abruptly creates a large buffer vacancy to all the remaining streams in the system. If the next

round of service to these streams refills all their vacant buffers, the service time of each stream will be stretched. This introduces an increasing delay in servicing successive streams and there is no guarantee that such delay in service cannot cause starvation and is, therefore, not safe. □

Lemma 7: A readsize no larger than the Nonstarvation bound defined by Definition 9 is safe.

Proof:: Even if the server idles for a period of time equal to T_{SLACK}, the remaining time being P, is sufficient to service every stream once before any deadline is missed. The server can thus use the slack time for extending the reading time of the immediate stream to be serviced. The number of extra sectors that can be read with a positive slack time T_{SLACK} is $\lfloor T_{\mathrm{SLACK}}/T_{\mathrm{sec}} \rfloor$. Therefore, it is safe to extend the read size up to $F_{\mathrm{nom}} + \lfloor T_{\mathrm{SLACK}}/T_{\mathrm{sec}} \rfloor$. □

Lemma 8: Any readsize no larger than the maximum readsize is both safe and feasible.

Proof:: By Definition 10, the maximum readsize cannot be larger than either of the Nonstarvation bound and the Vacancy bound. By Theorem 1, the maximum readsize is safe. By Lemma 3, it is feasible. □

Theorem 1: Any readsize between the minimum and the maximum readsizes is executable ($f_{\min} \leq f \leq f_{\max}$).

Proof:: Since $f \geq f_{\min}$, by Lemma 1, readsize f is sufficient. Since $f \leq f_{\max}$, by Lemma 8, readsize f is safe and feasible. By Definition 4, readsize f is executable. □

Lemma 9: If the difference between two readsizes is no less than one track size, there must exist at least one ZR readsize between them.

Proof:: The time to read one track is exactly one rotation time of the disk. If two readsizes differ in size by more than one track, the difference between their reading times is at least one rotation time. Within one rotation time, we can always find a time to stop reading such that rotation latency can be eliminated. Since such time exists between the two reading times, a ZR readsize must also exist. □

Lemma 10: Between the minimum and the maximum readsizes, there exist $\lfloor f_{\max} - f_{\min}/N_{\mathrm{trk}} \rfloor$ ZR readsizes, with incremental difference of one track size (N_{trk}, in disk sectors) between successive pairs.

Proof:: By Lemma 9, at least one ZR readsize exists between two readsizes which differ by more than one track size. Applying this to successive track-size intervals, a ZR readsize must exist in each successive track-size interval between the two readsizes. Therefore, the number of ZR readsizes existing between two readsizes is exactly the number of integral track-size differences between them, with reading times differ by one rotation time and their sizes differ by one track. □

Theorem 2: The LZR readsize is most efficient among the ZR readsizes.

Proof:: By Theorem 1, all ZR readsizes in between the maximum and minimum readsize bounds are executable. Since rotational overhead is eliminated by all the ZR readsizes, efficiency η becomes $\eta = t_f/(t_f + t_{\mathrm{sw}} + t_{\mathrm{seek}})$. While software and seek overheads remain the same, the difference in efficiency among ZR readsizes is determined by the reading time t_f which is proportional to the readsize. A larger reading time, or readsize, will clearly achieve a higher efficiency. Therefore, among the ZR readsizes, the largest one has the highest efficiency. □

Theorem 3: The Optimum readsize is the most efficient executable readsize.

Proof:: The Optimum readsize can only assume either the maximum readsize or the LZR readsize. Since both the maximum readsize and the LZR readsize are executable, the Optimum readsize is executable. By Theorem 1, efficiency of the maximum readsize is higher than the minimum and any value of truncated nominal readsize. By Theorem 2, efficiency of the LZR readsize is highest among all ZR reasizes. By Definition 13, efficiency of the Optimum readsize is never lower than the maximum or the LZR readsizes. Therefore, the Optimum readsize is the most efficient. □

REFERENCES

[1] D. P. Anderson, "A file system for continuous media," *ACM Trans. Comput. Syst.*, vol. 10, no. 4, pp. 311–337, Nov. 1992.

[2] D. J. Gemmell, H. M. Vin, D. D. Kandlur, P. V. Rangan, and L. A. Rowe, "Multimedia storage servers: A tutorial," *IEEE Computer*, pp. 40–49, May 1995.

[3] D. J. Gemmell, "Disk scheduling for continuous media," in *Multimedia Information Storage and Management*, S. M. Chung, Ed. Norwell, MA: Kluwer, 1996, pp. 1–21.

[4] P. V. Rangan, H. Vin, and S. Ramanathan, "Designing an on-demand multimedia service," *IEEE Communi. Mag.*, vol. 30, no. 7, pp. 56–64, July 1992.

[5] *Hytime: Hypermedia Time-Based Structuring Language*, Aug. 1992.

[6] *MHEG: Base Object Encoding*, Dec. 1995.

[7] J. F. Buford, C. Gopal, and L. Rutledge, "Storage server requirements for delivery of hypermedia documents," presented at the IS&T/SPIE High-Speed Networking and Multimedia Computing 95, Feb. 1995.

[8] C. Gopal and J. F. Buford, "Delivering hypermedia sessions from a continuous media server," in *Multimedia Information Storage and Management*, S. M. Chung, Ed. Norwell, MA: Kluwer, 1996, pp. 209–235.

[9] T. P. J. To and B. Hamidzadeh, "Dynamic real-time scheduling strategies for interactive continuous media servers," *ACM/Springer Multimedia Syst. J.*, vol. 7, no. 4, pp. 91–106, Mar. 1999.

[10] ——, "Dynamic scheduling techniques for interactive hypermedia servers," *IEEE Trans.Consum. Electron.*, vol. 45, no. 2, pp. 46–56, Feb. 1999.

[11] A. L. Reddy and J. Wilie, "Disk scheduling in a multimedia I/O system," in *Proc. First ACM Conf. Multimedia*, 1993, pp. 289–297.

[12] P. J. Shenoy, P. Goyal, S. S. Rao, and H. M. Vin, "Symphony: An integrated multimedia computing," in *Proc. SPIE/ACM Conf. Multimedia Computing and Networking (MMCN' 98)*, San Jose, CA, Jan. 1998, pp. 124–138.

[13] P. J. Shenoy and H. M. Vin, "Cello: A disk secheduling framework next generation operating system," in *Proc. ACM SIGMETRICS Conf.*, Madison, WI, June 1998, pp. 44–55.

[14] M. Seltzer, P. Chen, and J. Outsterhout, "Disk scheduling revisited," in *Proc. Winter 1990 USENIX Conf.*, 1990, pp. 313–324.

[15] S. P. Ng, "Improving disk performance via latency reduction," *IEEE Trans. Comput.*, vol. 40, no. 1, pp. 22–30, Jan. 1991.

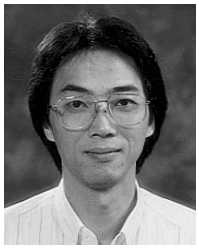
[16] K. Hwang and H. Shin, "New disk scheduling algorithm for reduced rotational latency," in *Proc. Third Int. Symp. Datebase Systems for Advanced Applications*, 1993, pp. 395–402.

[17] H. M. Vin, A. Goyal, A. Goyal, and P. Goyal, "An observation-based admission control algorithm for multimedia servers," in *IEEE Int. Conf. Multimedia Computing and Systems*, 1994, pp. 234–243.

[18] D. J. Gemmell and S. Christodoulakis, "Principles of delay-sensitive multimedia data storage and retrieval," *ACM Trans. Inform. Syst.*, vol. 10, no. 1, pp. 51–90, Jan. 1992.

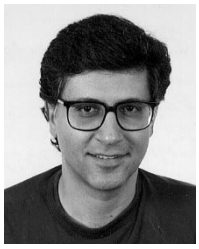
[19] P. Lopher and D. Shepherd, "The design of a storage server for continuous media," *Comput. J.*, vol. 30, no. 1, pp. 32–42, 1994.

[20] H.-J. Chen and T. D. C. Little, "Storage allocation policies for time-dependent multimedia data," *IEEE Trans. Knowl. Data Eng.*, vol. 8, no. 5, pp. 855–864, Oct. 1996.



Tsun-Ping Jimmy To received the B.Sc. degree in electrical engineering from the University of Manitoba, Winnipeg, Man., Canada, and the M.Sc. degree in computer engineering from the University of Southern California, Los Angeles. He received the Ph.D. in computer science from the Hong Kong University of Science and Technology in 1997.

He spent four years on product development in the electronic and computer industry in Hong Kong, prior to joining the Hong Kong Polytechnic University (HKPU) as a Lecturer. Currently, he is an Assistant Professor in the Department of Electronic and Information Engineering at the HKPU. His current research interests are in multimedia storage systems and real-time systems. He is a member of IEE and ACM.



Babak Hamidzadeh received the M.S. and Ph.D. degrees in computer science and engineering from the University of Minnesota, Minneapolis, in 1989 and 1993, respectively.

During that period, he also worked as a Research Associate at The Systems and Research Center, Honeywell, Inc., and as a Research Scientist at The Research and Technology Center, Alliant Techsystems, Inc. for over three years. From 1993 to 1996, he was an Assistant Professor of computer science and computer engineering at The Hong Kong University of Science and Technology. Currently, he is an Assistant Professor of electrical and computer engineering at The University of British Columbia, Vancouver, B.C., Canada. His areas of research include real-time computing, parallel and distributed processing, multimedia, and communication networks.

Dr. Hamidzadeh is a member of IEEE Computer Society and a Research Fellow of the BC Advanced Systems Institute.