Chaining: A Generalized Batching Technique for Video-On-Demand Systems

Simon Sheu Kien A. Hua Wallapak Tavanapong
Department of Computer Science, University of Central Florida
Orlando, FL 32816-2362, U. S. A.

Abstract

Although the bandwidth of the storage I/O typically dictates the performance of a conventional DBMS, network-I/O bandwidth limitation is the main operating constraint of most multimedia database systems. In spite the throughput of a public network (e.g., ATM) can be huge, the network-I/O bottleneck limits the number of client stations a media server can support simultaneously. A possible solution to this problem is to batch requests for the same video and multicast the data to these requesters to save the network-I/O bandwidth. A disadvantage of this scheme is that it unfairly forces requests arriving early in a batch to wait for the latecomers. As a result, the reneging rate can be high in a system which employs this technique. To reduce the long access latency, we examine in this paper a new batching mechanism called Chaining. *This approach allows the server to serve a "chain" of client stations using a single data stream. The idea is to pipeline the data stream through the chain of stations. Requests arriving early in a chain (virtual batch), therefore, do not have to experience long delays as in conventional batching. Our simulation results based on ATM networking environment indicate that very significant performance improvement over batching can be obtained.*

1 Introduction

Recent advances in computer technology have made practicable *video-on-demand* (VOD) applications. In this system, the subscribers can flexibly watch their favorite movies via the modern communication networks. To avoid jitters, the retrieval and delivery of videos must be carefully designed to meet their playback criteria [17, 18]. As a result, a *video stream* corresponding to an *I/O stream* and an *isochronous channel* is typically reserved for a VOD service. Dedicating a stream for each viewer, however, will quickly exhaust the network-I/O bandwidth of the video server. This problem limiting the number of users a VOD server can support at a time is observed in many existing systems. As an example, the bottleneck is encountered in the Time Warner Cable's *Full Service Network* project in Orlando. In spite each of the SGI Challenge servers used in this project can sustain thousands of I/O streams, the network-I/O bottleneck limits its performance to less than 120 MPEG-1 video streams. As a result, eight servers had to be used to serve the 4,000 homes significantly increasing the hardware and administration costs. Similar problem was also observed in Microsoft's *Tiger Video Fileserver* [3].

To address this issue, *Batching* [7, 1] has been proposed. In this scheme, requests waiting for the same video are served in a batch by a single video stream to allow resource sharing. Typically the arriving requests are first queued until their associated batch is initiated. The duration between its arrival time and the batch start time is known as its *service latency*. Once a batch is launched, an I/O stream is set up to retrieve the video from the storage subsystem. Then with the network multicast facility, the data is streamed to the group of clients by the packet store-and-forward mechanism [16, 4]. It was shown in [7] that batching demands much less server capacity (in the order of 60%.)

Nevertheless, batching has the following limitations: (1) The number of batches can be served at a time is still constrained by the network-I/O bandwidth. (2) Requests arriving early in a batch is unfairly made waiting for the latecomers. As a result, many users of such a system will likely experience long delay in the services. (3) The interest of the viewers may be quenched and they would cancel their requests (or *renege*) if they have been waiting too long. The achievement of batching schemes will be leveled off by the reneging behaviors. In [13], we treated the above issues using a *generalized batching* technique called *Chaining*. As memory buffers have been used to effectively reduce the demand on the storage bandwidth, Chaining lessens the demand on the network I/O by caching data in the local storage of the client stations. This local storage is currently used in several designs (*e.g.*, [2, 5, 9, 14]) to facilitate VCR functions. We propose also using these storage systems to pipeline video data to stations playing back the same video.

Chaining can be seen as "generalized batching" due to the following reason. In batching, requests arriving temporally apart is batched together regardless of the temporal distance among the arrival times. Contrar-

0-8186-7819-4/97 $10.00

ily, chaining allows the request arriving first in a chain (*virtual batch*) to receive its service right away (*i.e.*, its arrival time is respected). The next request for the same video can then receive its data from the first one provided it has not discarded the first data block of the video. Similarly, the third request arriving later can receive its data from the second one, and so on. Since requests arriving early do not have to wait for the latecomers, it was shown in [13] that latency and throughput can be vastly improved. This is because most of the batches are actually served by a multicast from a client workstation in the chaining environment. In this way, a service request arriving at the server can be seen as a contributor, rather than just a burden to the video server. This feature allows chaining to scale far beyond the limit of regular batching.

We note that the idea of allowing workstations to share the network storage has been studied in [10, 6, 8]. However, their concerns are cache consistency and/or transaction processing issues. Especially, handling continuous media requires Chaining to address entirely different matters. In fact, the notion of chaining is not applicable to conventional distributed applications. In this paper, we fine tune the Chaining strategy to include more comprehensive features. In particular, we will introduce a *delay* notion which can further improve the performance of standard Chaining. We also use a more realistic simulation model for the performance study. It simulates the topology of the ATM switches [16, 4]. The simulation results indicate that the new features vastly improve the performance. The remainder of this paper proceeds as follows. We discuss some batching schemes in Section 2. The Chaining technique and new features are presented in Section 3. In Section 4, we present our performance study. Finally, we give our concluding remarks in Section 5.

2 Multicasting

Several batching policies have been proposed. FCFS (*First Come First Served*) [7] designates the released video streams according to the longest waiting time of requests among the batches. Whenever a video stream is available, it will be assigned to initiate the batch with the request hankering longest. Regardless of the popularity of videos, the requesting batches are fairly treated in this scheme. Unlike FCFS, MQL (*Maximum Queue Length first*) [7] is designed to minimize the number of reneging customers by first choosing the batch with the largest number of waiting clients. Thus, this scheme prefers requests of more popular videos rendering less service latency for these requests. However, the overall average latency of all requests including the reneging ones is worse than that of FCFS [1], while the entire reneging rate is better.

On the other hand, FCFS-n [7] assumes certain knowledge on the users' reneging behavior. It reserves a fraction of the server capacity for batching requests for the n most popular videos, while the remaining cold videos are served by FCFS policy. Therefore, the popular videos are served by dedicated streams, *i.e.*, each of these videos is multicast every B minutes (batch interval). This approach provides more flexibility to balance the fairness and the reneging probability. Nevertheless, the optimal value of n depends on many factors, and is difficult to determine. MFQ (*Maximum Factored Queue length first*) [1] schedules the videos according to the queue length weighted by the *best* factor, $(\text{the associated access frequency})^{-\frac{1}{2}}$, derived from an optimization model. This policy can generally reduce the reneging rate more effectively than both FCFS and MQL can. Besides, it is much more fair than MQL.

3 Chaining

Currently, the most common standard to represent video objects is MPEG. In this scheme, a set of frames, either I-, P- or B-frame, can be defined to form a *group of frames* (GOF). In this research, we group every k consecutive GOFs into a data unit called *retrieval block* (R-block) [11], which is the minimum unit of data we refer to in terms of disk accesses.

3.1 Standard Chaining

Suppose R-blocks of a video are numbered from 1 to the length of the video file according to their temporal position in the video. Let $v(i)$ be the i-th R-block of video v. Let D_m and D_n be two stations currently playing back $v(i)$ and $v(j)$, respectively. The *playback distance* of D_m and D_n, denoted by $d(m,n)$, is computed as $j-i$, assuming $j > i$. In this case, D_m and D_n can be served by the same *video chain* if D_n has at least $d(m,n)$ units of disk space to momentarily cache data for D_m and the isochronous channel is available from D_n to D_m. Fig. 1 illustrates this strategy, where each station has four R-blocks (denoted as b_d=4.) A number i in a buffer indicates that it currently has a copy of $v(i)$. The shaded block indicates that it is being refreshed. In this example, stations D_a, D_b, D_c, D_d, and D_e are served by video chain A, while chain B is serving D_f, D_g, and D_h. The relevant playback distances are $d(b,a) = 4$, $d(c,b) = 0$, $d(d,c) = 3$ and $d(e,d) = 4$ in Chain A; $d(g,f) = 2$ and $d(h,g) = 1$ in Chain B. Since the distances are less than or equal to b_d $(= 4)$, these stations can share the corresponding video stream. The two video chains, however, cannot be combined because $d(f,e) = 7 > b_d$.

To illustrate the data flow in Chaining, we consider *Chain A* shown in Fig. 1. The video server needs only retrieve the R-blocks for D_a, the *head* of this video chain. D_a is responsible for forwarding the R-blocks to D_b and D_c which were admitted to the video chain as a batch [7, 1]. However, the data is multicast from a client, instead of from the server as in conventional batching. Eventually, each R-block traversing

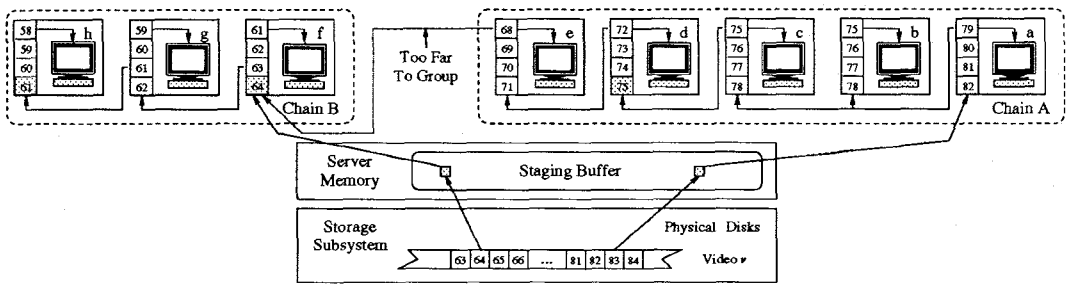


Figure 1: Chaining strategy.

the chain will arrive at D_e, the *tail* of the chain. D_e then discards the R-blocks after they are played out.

A major advantage of Chaining is that it can reduce the demand on the network-I/O bandwidth. Comparatively, requests arriving temporally far apart must be served in different batches by conventional batching schemes. Chaining offers the opportunity to service these requests in a single chain (*i.e.*, virtual batch), and therefore conserve the server resources for more batching. We note that if the length of each chain is constrained to only one client station, Chaining degenerates into the conventional batching approach. As multicast has served many types of applications, Chaining offers a new communication strategy for a new class of applications involving multimedia data.

The cost of Chaining is the small disk space used at each client station to pipeline data through the chain. This disk space, however, is minimal. For instance, a disk space of only 56 MBytes can cache five minutes of MPEG-1 video. Such a tiny disk space costs less than $10 today. The high cost of a networked VOD business is due mostly to the network costs. For instance, the cost of networking contributes to more than 90% of the hardware cost of the Full Service Network project in Orlando. Therefore, it is essential for the design to take advantage of the aggregate bandwidth of the network, even at a small additional cost. In the case of Chaining, this cost is negligible, and can be offered to subscribers for free. The significant increase in the number of subscribers who can receive the services simultaneously should easily make up for the nominal cost. Furthermore, this small disk space is required in many designs to implement the VCR functions anyway. In which case, this cost is considered as a part of the original cost, and not an add-on cost.

3.2 Extended Chaining

In Standard Chaining, as the arrival of the requests momentarily becomes sparse, the temporal distances between the arrival time become large (*i.e.*, bigger than $t(b_d)$, the playback duration of b_d R-blocks) and render Chaining unachievable. More comprehensive chaining designs are necessary to handle this condition. A heuristic approach is presented in this paper which is similar to FCFS-n [7]. This scheme makes use of the minimal time each requester is willing to wait, and is denoted as W_{min}. Each user is assumed to wait for at least W_{min} time units. The additional wait time follows an exponential distribution. To improve the performance of standard chaining, whenever a set C of candidate requests are considered for joining an existing chain as a batch, the algorithm HEC (*Heuristic Extended Chaining*), given in Fig. 2, is used to determine the following two values: (1) *kept*: The number of candidates in C that should be delayed; (2) t: The time when the oldest delayed request must be served to avoid closedown of the chain. As an example, let

INPUT
$kept_{max}$ (> 0): maximum number of requests to retain
$D_1, D_2, \ldots, D_n$ (in arriving order): pending requests for the same video
$t_1, t_2, \ldots, t_n$ ($t_i \leq T$; $t_i \leq t_j$, if $i \leq j$): corresponding arriving time
T: current time
W_{min}: minimum wait time of any request
b_d: amount of storage of each client
OUTPUT
kept: actual number of requests retained
t: time to service the first retained requests

```
1.  t = t_n + W_min;
2.  IF (t ≤ T) THEN RETURN(kept = 0, t = T);
3.  kept = 1;
4.  WHILE (kept < n) AND (kept < kept_max) DO
5.      t = MIN (t_{n-kept} + W_min, t - t(b_d));
6.      IF (t ≤ T)
7.          THEN
8.              t = t + t(b_d);
9.              RETURN(kept, t);
10.         ELSE
11.             kept = kept + 1;
12.     ENDIF
13. ENDWHILE
14. IF kept = n THEN kept = kept - 1;
15. RETURN(kept, t);
```

Figure 2: Heuristic Extended Chaining algorithm.

$kept = 1$ and $t = 38$. These two numbers indicate that the latest arrival among the candidates in C should not join the chain in the current batch; instead, it should wait until time t (*i.e.*, 38.) The motivation of delaying the last request is to keep the chain open until at least time $38 + t(b_d)$. We note that a longer chain is desirable as it allows more clients to share the same video stream. Therefore, there is better utilization of the throughput in the public network.

Although the algorithm HEC computes only the ser-

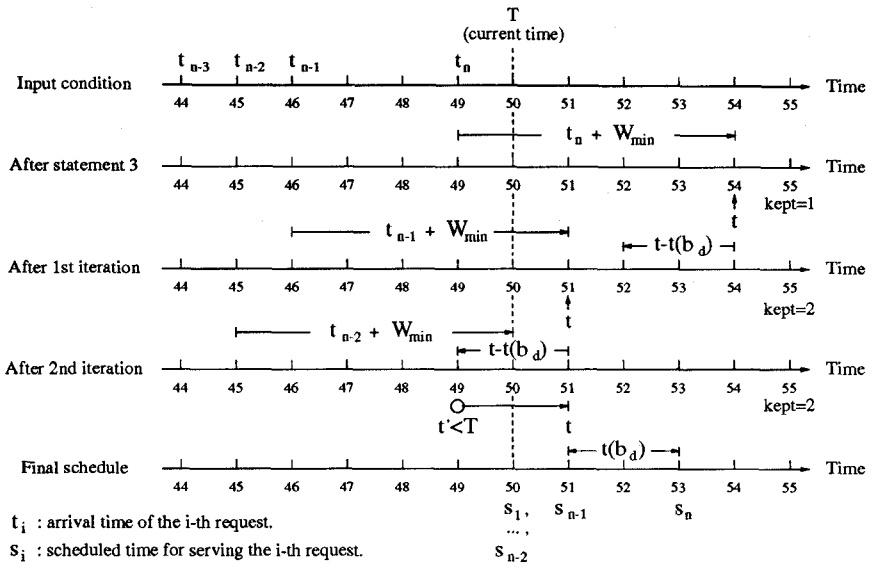


Figure 3: Illustration of HEC algorithm.

vice time t for the oldest of the *kept* delayed requests, the remaining $kept - 1$ delayed requests are scheduled at times $t+t(b_d)$, $t+2 \cdot t(b_d)$, $\cdots$, and $t+(kept-1) \cdot t(b_d)$, respectively. An example is given in Fig. 3 to illustrate this algorithm: (1) The first time line shows that at a particular time T, n candidate requests $D_1, \cdots, D_n$ with arrival times $t_1 < t_2 < \cdots < t_n$, respectively, are ready to join a chain in a batch. We assume that $kept_{max} = 3$, $W_{min} = 5$ and $t(b_d) = 2$. (2) The second time line shows the intermedia values for *kept* and t after statement 3 is executed. Note that we want D_n to receive its service at time "$t_n + W_{min} = 54$" since each user is only "guaranteed" to wait for at least W_{min} (= 5) time units. (3) The third time line shows the values of *kept* and t after the first iteration of the WHILE loop. The service time for D_{n-1} is set at time 51 for the same reason discussed for D_n in the second time line. With this decision, however, it is implied that the service time for D_n must be moved forward from 54 (according to the second time line) to 53 to ensure that it can join the chain before D_{n-1} discards the first R-block of the video file. Note that in this case, the client station corresponding to D_{n-1} can hold only two R-blocks in its disk buffer. The strategy used to determine t as discussed above is implemented in statement 5. (4) The fourth time line illustrates a different behavior of the WHILE loop during its second iteration. In this case, we want to service D_{n-2} at time 50. However, since we have decided in the last iteration to start D_{n-1} at time 51, we can move the starting time of D_{n-2} forward by one time unit to 49 and still have enough time for D_{n-1} to chain to D_{n-2}. The motivation for scheduling D_{n-2} at an earlier time is to reduce its access latency. Nevertheless, the current time is 50, and we cannot schedule D_{n-2} at time 49 which is in the past. The execution, therefore, terminates at the THEN clause after restoring t to the start time of D_{n-1} at time 51. In this iteration, *kept* is not increased. It remains at the value 2 indicating that only the two latest requests D_{n-1} and D_n should be delayed. (5) The final schedule is illustrated in the last time line. It shows that requests $D_1, D_2, \cdots, D_{n-2}$ should be serviced immediately in a batch. The service times for D_{n-1} and D_n should be delayed until time 51 and time 53, respectively. The advantage of introducing these two delays is to allow future arrivals to connect to this chain at time as late as 55. Without these delays, the chain will become permanently terminated if no new request for the same video will arrive by time 52.

To employ the Extended Chaining technique, any batching policy can be adapted to provide admission control. A batch is admitted if it can join an existing chain. Otherwise, it can be admitted only if server resources are available to support a new chain headed by this batch. Once a batch for a video v has been admitted, HEC is called to determine the values *kept* and t. The service for this batch, without the *kept* delayed requests, is started right away. The *kept* delayed requests will join the chain for video v by the schedule. Nevertheless, if new requests for video v arrive before the schedule is finished, a new batch is created to contain the new arrivals and all the remaining delayed requests. The function HEC is then called to handle this new batch.

To explain the advantage of Extended Chaining, we discuss the state diagrams illustrated in Fig. 4. There is a state diagram for each video file in the system. The state diagram for Standard Batching is shown in Fig. 4(a). It shows that a video file can be in one of two possible states: (1) **I State**: The video file is in I state if there is currently no pending request for this video;

(2) *Q* **State**: The arrival of a new request causes the video file to transit from the *I* state to the *Q* state, and a new batch be created to hold the new arrival. Subsequent requests for this video file are accumulated in this batch while the video file remains in the *Q* state. Eventually, a video stream is allocated to serve this batch according to some queuing policy, and the video file returns to the *I* state. The state diagram for Standard Chaining is shown in Fig. 4(b). It is similar to that of Standard Batching except for one additional state, *C* **State**: Unlike regular batching, when a video stream is allocated to serve the current batch, the video file does not return to the *I* state. Instead, it goes into the *C* state indicating that a new chain has been established for the respective video. As long as new requests continue to arrive to keep the chain open, the video file remains in the *C* state. When the arrival of requests momentarily discontinue for an extended period of time, the chain is closed down and the video file returns to the *I* state. The state diagram for Extended Chaining is shown in Fig. 4(c). This scheme has one additional state, *E* state. Unlike Standard Chaining, the allocation of a video stream does not immediately move the video file into the *C* state. Instead, it transits into the *E* state. In *E* **State**, the scheduler retains a number of requests for future batches according to the HEC algorithm. As long as new requests continue to arrive, the video file remains in the *E* state. If the arrival of requests momentarily discontinue for an extended period of time, the video file transits into the *C* state after the service is initiated for the final delayed request. We note that if new requests begin to arrive again at this time, the video file can return to the *E* state; otherwise, it returns to the *I* state after a period of missing new arrivals.

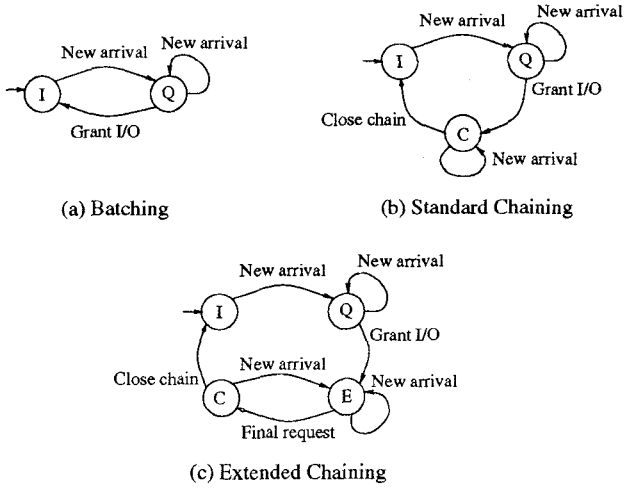


Figure 4: State transition diagrams.

Comparing the three state diagrams, we observe that Standard Batching returns to the *I* state much more frequently than the other two techniques. As a result, it is a lot more demanding on the server bandwidth. Standard Chaining addresses this problem by introducing an additional state *C* to reduce the frequency of transition into the *I* state. Extended Chaining adds yet another state *E* to further reduce this frequency. We note that the state diagrams presented in this paper are simplified versions which do not capture the reneging behavior of the subscribers. From this perspective, essentially all subscribers of Standard Batching must wait when the video file is in the *Q* state. Therefore, this technique can experience high reneging rates. On the other hand, subscribers of the chaining techniques (except for a small number of requests retained by the scheduler) do not have to wait when the requested video file is in the *C* state or the *E* state. Hence very low reneging rates are achievable under chaining. The reduction in the reneging rate is the key to the performance of the chaining techniques. The detailed version of the state diagrams were used to implement our schedulers.

4 Performance Study

To facilitate our simulation study, we implemented three detailed simulators for FCFS [7], MFQ [1], and Extended Chaining, respectively. FCFS and MFQ represent the conventional batching approach. In spite FCFS is known to offer less performance, it serves as a good benchmark for fairness measure. MFQ, on the other hand, has been shown to provide the most performance among the existing batching policies [1]. Therefore, it is a good basis for performance comparison. To be fair, MFQ is also used for the Extended Chaining environment. This scheme is denoted as MFQ-C*kept*, where *kept* signifies the maximal number of pending requests to be delayed for a given video. Thus, MFQ-C2 represents an extended chaining scheme using MFQ as the admission control policy, and the maximum number of delays allowed for any video is two. We note that MFQ-C0 represents the Standard Chaining in [13].

4.1 System Environment

The VOD system is assumed to have one server and 1,048,570 ($\approx 2^{20}$) stations interconnected through an ATM-based MAN (Fig. 5). We assume that the connection topology forms a 20-dimensional hypercube.

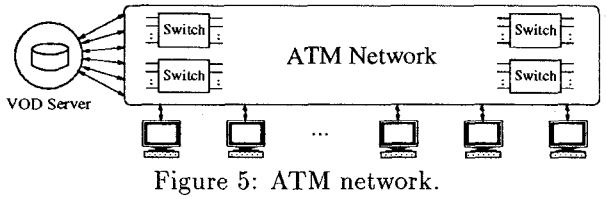
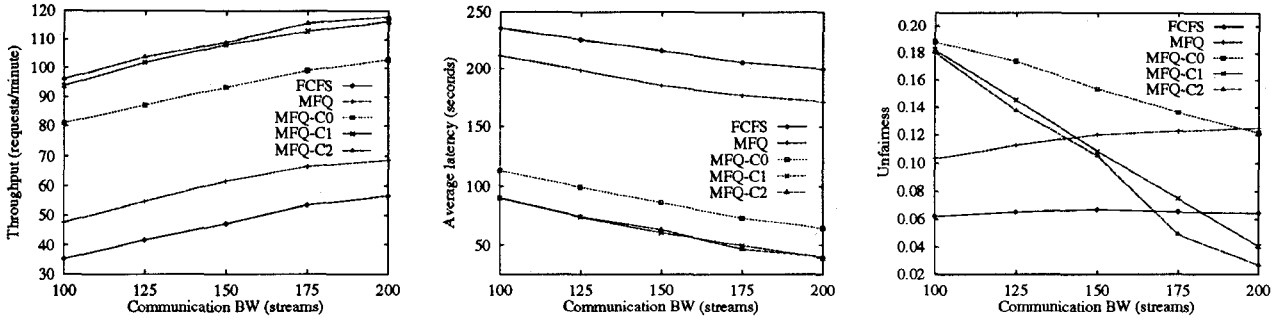


Figure 5: ATM network.

The multicast facility of the ATM network is implemented following a method similar to the greedy algorithm presented in [15]. Such a network is still too expensive today for most applications, but it offers a platform to study how well batching and chaining can overcome the network-I/O bottleneck in order to take advantage of the network resources. 1,048,570 clients,



(a) Throughput (b) Latency (c) Unfairness

Figure 6: Performance under different communication bandwidths.

used in our study, represent the typical population of a medium size city. Every client has a connection to a distinct node of the hypercube. The server is assumed in this case employ six communication ports. Each port is connected to a distinct hypercube node. We also assume that each subscriber submits one request and views only one video at a time. Each station is equipped with a small storage system to facilitate chaining. The VOD server contains 100 videos for selection and each video is 2 hours long.

PARAMETER	VALUE
Storage I/O BW	1,000 (400, 600, 800, 1000, 1200) streams
Client storage size	1.5 (0.0, 0.5, 1.0, 1.5, 2.0, 2.5) minutes of video data
BW of each Comm. port	150 (100, 125, 150, 175, 200) streams
BW between cube nodes	200 streams
Request rate	120 (60, 90, 120, 150, 180) requests/minute
Skew factor	0.7 (0.0, 0.1, ..., 0.9, 1.0)

Table 1: Simulation parameters.

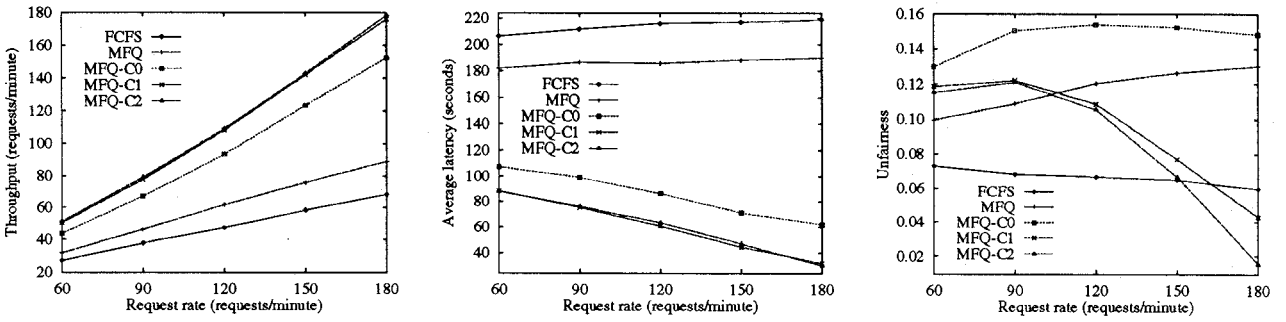
Each request is modeled by interarrival time, client ID, and choice of a video. Interarrival times follows a Poisson process. The client is randomly chosen from inactive stations. The selection of videos follows a Zipf-like distribution [12]. That is, the probability of choosing $video_i$ is $\frac{1}{i^z \cdot \sum_{j=1}^{N} 1/j^z}$, where N is the total number of videos in the system, and z is the *skew factor*. A larger z value corresponds to a more skew condition, where some videos are chosen more frequently than others. This factor has default value 0.7, which is typical for VOD applications [7]. We also assume that each user is willing to wait at least two minutes plus some amount of time exponentially distributed with a mean of three minutes as in [7]. Each simulation runs for 24 simulated hours to achieve the steady-state performance. The default values for the system and workload parameters are given in Table 1. The values in the parentheses were used to do various sensitivity analyses. The performance metrics are defined as: (1) $Throughput = \frac{number\ of\ requests\ finished}{total\ simulated\ time}$, (2) *Average latency*=mean of the individual latencies of all requesters, (3) $Unfairness$=the standard deviation of r_i over all videos, where r_i is the *reneging probability* of $video_i$. Note that r_i measures the percentage of requests for $video_i$ reneging due to the long wait. A larger value of unfairness corresponds to a less fair policy: the requests of some videos gets more attention from the system than those of others.

4.2 Effect of Communication Bandwidth

The effect of the communication bandwidth on the various schemes is shown in Fig. 6. It shows that standard Chaining offers substantial improvement over the conventional batching schemes (*i.e.*, FCFS and MFQ). The improvement due to Extended Chaining is even more significant. It is about 70% better than MFQ in terms of throughput, and is four times better in terms of access latency. From another perspective, if we want to design a system to achieve a throughput of 70 services per minute, Extended Chaining requires a network-I/O bandwidth only one quarter that of MFQ. Extended Chaining also provides excellent fairness when the bandwidth is sufficiently large. Fig. 6(c) indicates that Extended Chaining outperforms even FCFS in fairness when the bandwidth is larger than 160. This result is due to the fact that Chaining does not require as much bandwidth as conventional batching does. The amount of bandwidth saved is used to service the cold videos rendering excellent fairness for the new schemes. We also observe in Fig. 6 that retaining one request to facilitate future chaining (*i.e.*, MFQ-C1) is sufficient to ensure good performance.

4.3 Effect of Request Rate

We observe that the new approach offers very sizable savings in Fig. 7. For instance, when the arrival rate is 180 requests per minute, it is 100% better than MFQ in terms of system throughput, and 6.3 times better in terms of average latency. The decreases in the latency of the new schemes as we increase the request rate are due to the fact that a higher request rate results in more opportunity for chaining. As a result, most of the requests for popular videos can be served immediately by existing chains rendering more server resources available for the less popular videos. This



(a) Throughput (b) Latency (c) Unfairness

Figure 7: Performance under various request rates.

phenomenon has a positive effect on the fairness of the proposed scheme (Fig. 7(c)).

4.4 Effect of Storage I/O Bandwidth

The effect of the storage-I/O bandwidth is very similar to that of network-I/O bandwidth as discussed before (Section 4.2). In this study, the network-I/O bandwidth is fixed at 900 concurrent streams (6 communication ports with 150 concurrent streams per port). This explains why all the curves become flat when the storage-I/O bandwidth is increased beyond 900 concurrent streams. This phenomenon is very common in today's media servers. For instance, the network-I/O bottleneck in the SGI Challenge server severely limits the amount of its outstanding storage-I/O bandwidth available for VOD applications. Most of servers

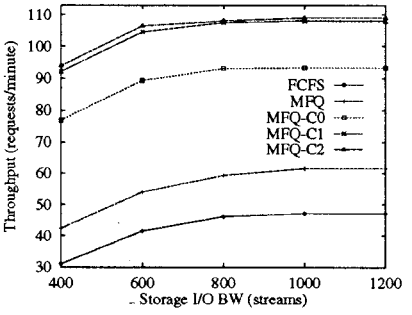


Figure 8: Effect of storage I/O bandwidth.

today are designed for conventional database applications. They are not suitable for multimedia applications. In conventional database applications, the server requires a lot of storage I/O to support query processing [12]. However, the network-I/O bandwidth can be small as the result returned to the client is typically a very small fraction of the data examined by the query. On the contrary, multimedia applications, like VOD, have to deliver the entire video file to the client. Therefore, it is essential that the media server has sufficient network-I/O bandwidth to make its storage bandwidth available to clients for retrieving video data.

4.5 Effect of Client Storage Size

To demonstrate the fact that only a tiny storage space on each client is sufficient to facilitate the chaining operation, we varied the client buffer size from zero to a size large enough to cache 2.5 minutes of video. The plots for this study are shown in Fig. 9. When the buffer size is zero, no chaining is possible and the

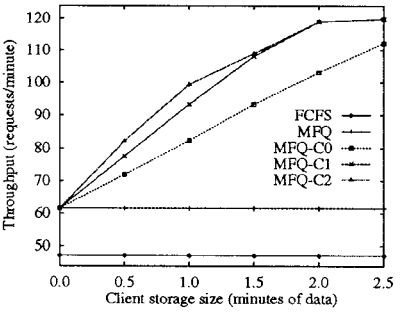


Figure 9: Effect of client storage sizes.

proposed methods degenerate into conventional batching. As a result, they share the same performance with MFQ. However, as soon as we increase the client buffer size, the performance differences increase rapidly. At a buffer size large enough for 2 minutes of video, for instance, it shows that Extended Chaining doubles the performance of MFQ. We note that the curves for MFQ and FCFS are flat because they do not use the client storage and are not effected by this factor.

4.6 Effect of Reference Skew

The effect of the skew factor is plotted in Fig. 10, which shows Chaining and Extended Chaining outperform MFQ and FCFS by a very significant margin for all the skew conditions. The throughput is constant

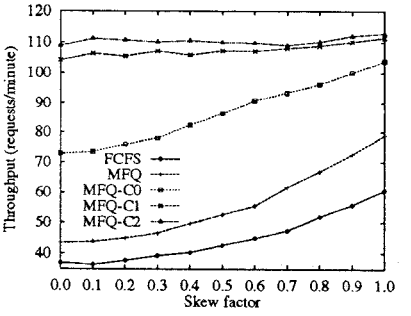


Figure 10: Performance under various skew factors.

for Extended Chaining. When the skew condition is mild, all the videos are about equally popular and they

compete fairly for the limited server resources. Under this condition, both Standard Chaining and Extended Chaining have about the same number of chains under steady state. However, the chains generated by Extended Chaining are much larger due to the delay features. Therefore, each of its chains can accommodate many more clients than those of Standard Chaining. This explains the better performance observed for Extended Chaining. In fact, since the chains formed by Extended Chaining are large enough to sustain the arrival rate used for this study, its performance is essentially unaffected by the various skew conditions.

5 Concluding Remarks

Multicast is a very useful feature for VOD applications. This approach allows a group of client stations viewing the same video to share a single video stream. The scarcity of the network-I/O bandwidth, however, severely constrains the number of multicasts which can be supported simultaneously. To extend the performance beyond the level allowed by multicast, we proposed in this paper a generalized multicast technique called *Chaining*. This scheme is similar to conventional multicast, except that the *multicast tree* of a conventional multicast is statically determined before the actual multicast can take place. On the contrary, *chaining trees* are grown dynamically to accommodate late requests for the corresponding video. This is achieved by caching data in the local storage of clients to facilitate future multicasts to the late comers. Thus, data are actually pipelined through the client stations residing at the nodes of the respective chaining tree. A network storage management technique was presented in this paper to provide the pipelining mechanism.

The advantage of Chaining is that not every request has to receive its data directly from the server. A large amount of video also becomes available from clients located throughout the network. This scheme is very scalable because each client station using the service also contributes its resources to the community. Hence the larger the chaining trees, the more effective the application can utilize the aggregate bandwidth of the public network. The sizes of the chaining trees are maximized in our technique by strategically deferring the service times for some of the requests. These delayed requests are served at a later time to extend (prolong the "life" of) the tree just in case no new request will arrive in the near future. Our simulation results indicate that this extension significantly improves the performance of standard chaining. Our performance study also demonstrates that the extra storage cost required for chaining is minimal and the dramatic increase in the performance is definitely well worth it.

References

[1] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. On optimal batching policies for video-on-demand storage servers. In *Proc. of IEEE ICMS*, Japan, June 1996.

[2] K. C. Almeroth and M. H. Ammar. The use of multicast delivery to provide a scalable and interactive video-on-demand service. *IEEE Journal In Communications*, 14(6):1110–1122, August 1996.

[3] W. J. Bolosky, J. S. Barrera, R. P. Draves, R. P. Fitzgerald, G. A. Gibson, M. B. Jones, S. P. Levi, N. P. Myhrvold, and R. F. Rashid. The tiger video fileserver. In *Proc. of the Int'l Workshop on Network and Operating System Support*, April 1996.

[4] J. Y. L. Boudec. The Asynchronous Transfer Mode: A tutorial. *Computer Networks and ISDN Systems*, 24:279–309, 1992.

[5] M.-S. Chen and D. D. Kandlur. Stream conversion to support interactive video playout. *IEEE Multimedia magazine*, 3(2):51–58, Summer 1996.

[6] M. Dahlin, R. Wang, T. Anderson, and D. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In *Proc. of USENIX OSDI*, pp. 267–280, Nov. 1994.

[7] A. Dan, D. Sitaram, and P. Shahabuddin. Dynamic batching policies for an on-demand video server. *Multimedia Systems*, 4(3):112–121, June 1996.

[8] M. Feeley, W. Morgan, F. Pighin, A. Karlin, H. Levy, and C. Thekkath. Implementing global memory management in a workstation cluster. In *Proc. of ACM SOSP*, Dec. 1995.

[9] W. Feng, F. Jahanian, and S. Sechrest. Providing VCR functionality in a constant quality video-on-demand transportation service. In *Proc. of IEEE ICMS*, Hiroshima, Japan, June 1996.

[10] M. Franklin, M. Carey, and M. Livny. Global memory management in client-server DBMS architectures. In *Proc. of VLDB*, Aug. 1992.

[11] C. S. Freedman and D. J. DeWitt. The SPIFFI scalable video-on-demand system. In *Proc. of ACM SIGMOD*, pp. 352–363, California, May 1995.

[12] K. A. Hua, C. Lee, and C. M. Hua. Dynamic load balancing in multicomputer database systems using partition tuning. *IEEE Trans. on Knowledge and Data Engineering*, 7(6):968–983, December 1995.

[13] K. A. Hua, S. Sheu, and J. Z. Wang. Earthworm: A network memory management technique for large-scale distributed multimedia applications. In *Proc. of IEEE INFOCOM*, Japan, April 1997.

[14] K. A. Hua, J. Z. Wang, and S. Sheu. BiHOP: A *bi*directional *h*ighly *o*ptimized *p*ipelining technique for large-scale multimedia servers. In *Proc. of IEEE INFOCOM'97*, Kobe, Japan, April 1997.

[15] Y. Lan, A. Esfahanian, and L. M. Ni. Multicast in Hypercube Multiprocessors. *Journal of Parallel and Distributed Computing*, 8:30–41, 1990.

[16] D. J. Marchok, C. Rohrs, and M. R. Schafer. Multicasting in a growable packet (ATM) switch. In *IEEE INFOCOM*, pages 850–858, 1991.

[17] H. M. Vin and P. V. Rangan. Designing a multiuser HDTV storage server. *IEEE Journal in Communications*, 11(1):152–164, Jan. 1993.

[18] P. Yu, M. Chen, and D. Kandlur. Grouped sweeping scheduling for DASD-based multimedia storage management. *Multimedia Systems*, 1:99–109, 1993.