Multimedia Systems
CS384M
Assignment #2

Design of a Buffer Management Scheme for Video Servers

Submitted by

Chandresh Jain
Onur Mutlu

Buffer Management Techniques for Video Servers

1.Introduction

Design of video servers is a challenging task which requires careful attention to various number of issues. Some of the most important of these issues are disk bandwidth management, disk array management (storage space management), and buffer space management. The choices of the policies chosen for each of these will significantly affect the performance of the video server, where performance can be measured as the number of clients serviced by the video server without violating a significant amount of deadlines. In this project we focus on "buffer space management" for video servers and design an efficient buffer space management algorithm that adjusts to dynamic changes in the load on the system.

2. Buffer Space Management Problem

In a video server, typically finite amount of memory is used as a buffer for the disk array. The available buffer space can be used for several purposes. The three most important main purposes of the buffer are the following:

1. Buffering of data retrieved in round i and to be sent to clients in round i+1 (Demand requests)
2. Buffering of data that is prefetched (Prefetch requests)
3. Buffering of data that is cached (Cached data)

In this document, we will refer to these three types of data that are to be stored in the buffer as Demand requests, Prefetch requests, and Cached data, respectively.

As suggested by the above classification of data, the buffer space management scheme in a video server must be aware of the existence of these different data types and optimize the use buffer space based on the inherent difference between these three different types. We now discuss the characteristics of these different data types. Later we will discuss different methods of allocating buffer space to each of these request types.

3. Characteristics of Different Request Types

3.1. Demand Requests

In this project, we have simulated a video file system, which employs round-based scheduling on a server-push architecture. Hence, the client requests are always serviced directly from the buffer in a given round.
In round i the server fetches the data needed by all clients in round i+1. This data is buffered in the buffer space once it is fetched. In round i+1, the space allocated for this data is claimed back after the requests are sent to the clients. Hence, the video server needs to keep the demand data in the buffer space for at most 2 rounds (Note that this is the worst-case assumption for the duration of time demand data stays in the buffer. Most demand data will stay in the buffer for the duration of 1 round on average).

In some basic level, the demand requests are the same as prefetch requests. The server-push architecture prefetches the data needed by the client one round before it is needed. Hence, the server-push architecture exploits the sequentiality present in the video stream by prefetching. However, we would like to make a distinction between "demand" requests and "prefetch" requests.

A "demand" request is not a prefetch in the sense that it must be satisfied one round before it is needed by the client given that the video server would like to service all client requests from the memory (buffer). If the video server did not have the requirement of servicing all client requests from the buffer, then a demand request would be no different from a prefetch request.

The reason we would like to distinguish a demand request from a prefetch request is to be able to make intelligent decisions in allocation and deallocation of buffer space for these two requests. Demand requests are in some sense higher-priority requests compared to the prefetch requests. They have to be serviced immediately because the deadline of the demand request is the end of the round.

3.2. Prefetch Requests

These requests are ones that are created by the video server by exploiting the sequentiality of the video streams. The video server knows the exact access pattern of every client once the client issues its first request. Hence, in an ideal world, the video server can fetch from the disk array into the buffer all the data required by the client in the first round and service the client from the buffer for the rest of the video stream the client is accessing.

We can see that the prefetching scheme a video server employs depends heavily on the size of the available memory buffer. If there is a big enough buffer, it is useful to generate prefetch requests. Generating prefetch requests for every disk would uniformly reduce the load balance across the disks in later rounds. The load balance of the disks will increase in the round when the prefetch requests are inserted into the scheduler. However, this increase is not very crucial because we would hope that those requests are inserted intelligently such that they do not delay any demand requests. We will describe such a scheme later in this report.

3.3. Cached Data

Caching is another method that exploits sequential nature of video access to save disk bandwidth. Due to the streaming nature of video accesses, traditional caching schemes that exploit temporal locality are not useful to employ in multimedia servers. Different caching mechanisms, such as ones that exploit the fact that a video file is accessed by multiple clients are more desirable.

One of the most effective methods for caching video frames is interval caching [1]. In interval caching, the interval between two temporally spaced clients is stored in the cache. Such a mechanism requires the following:

1. Detection of two clients accessing the same file back to back
2. Decision on which intervals to cache
3. Decision on which intervals to replace when a new interval is formed

Decision of which intervals to cache and which intervals to replace will have very significant effects on the disk bandwidth saved and number of clients serviced.

3.4. Bandwidth Savings of Different Buffer Data Types (Buffering Schemes)

We can immediately see that interval caching of video data is very powerful because it saves disk bandwidth. On the other hand, demand requests or prefetch requests do not really save disk bandwidth. They just shift the load on the disks that would be observed in a later round to previous rounds. This is a very important difference between caching and prefetching that needs to be considered during buffer space allocation for these different types of requests.

3.5. Disk Utilization of Different Buffering Schemes

Based on the described characteristics of the different buffered data types, we can see that demand requests and prefetch requests increase the disk utilization in the video server. Especially prefetching increases the utilization of the disks by inserting prefetch requests during the times disk is free of demand requests. On the other hand, we note that interval caching actually decreases the disk utilization by preventing many of the demand requests from going to the disk. The main benefit of interval caching comes from savings of disk bandwidth. Hence, we conclude that disk utilization is not a very good measure of the performance of the video server especially when there are many temporally spaced clients accessing the same data files.

4. Partitioning of the Buffer Space

An important policy decision to be made in buffer space management is the decision of how to divide the buffer into different data types. The amount of space allocated to each request type can affect the performance (number of clients supported without violating a specified percentage of deadlines) of the video server significantly. Hence, this policy decision should be made very carefully.

This decision of how to allocate buffer space among different request types bears many similarities to the allocation of disk space to different data types when designing an integrated file system [2]. Especially the differences between static and dynamic partitioning of the buffer space are analogous to the differences between a logically integrated file system and a physically integrated file system in the design of integrated file systems [2].

4.1. Static Partitioning

One simple policy for allocation is statically partitioning the buffer space among the three different data types. The server can be configured during initialization with d% of the buffer space allocated to demand requests, p% allocated to prefetch requests, and c% of it allocated to cached data. In any round, if the insertion of a block of type demand causes the percentage of demands in the buffer

exceed d% then that request will not be inserted into the buffer and hence simply dropped (even though rest of the buffer (p% + c%) may have free space).

Hence the disadvantage of the static buffer partitioning policy. This policy is not flexible and it results in higher frequency of deadline violations especially for variable bit rate video streams. In variable bit rate video streams, the size of data accessed in one round is not fixed. It varies from round to round. Hence, the size of demand data that needs to be accessed in a given round may exceed the amount of buffer space statically allocated to demand data. If this happens, deadlines will be violated.

The second disadvantage of static buffer partitioning is that it results in low utilization of the buffer space. Consider the case in which the demand requests do not fill up the portion of the buffer space allocated to demand requests. In that round, the unfilled portion of that portion of buffer space will not be used at all. Prefetch requests or cache requests cannot claim and use that portion of the buffer, because the buffer space is strictly partitioned. Hence the low utilization of buffer space.

In some sense, low utilization of buffer space implies low utilization of the disk array. If buffer space is available for prefetch requests, then the scheduler will schedule more prefetches to the disk array and will keep the disk array busy. By statically partitioning the buffer space, the design will limit t he buffer space available for prefetching and hence lower the disk utilization (This discussion assumes that the round time is enough to issue more prefetches).

4.2. Dynamic Partitioning

The discussion of static partitioning makes it clear that we would like to dynamically partition the buffer to maximize buffer utilization and minimize the number of deadlines missed. By using dynamic partitioning we can employ a greedy prefetching algorithm (bound by the service time of disks and round duration) which prefetches into the buffer until the buffer space is fully utilized. Of course, this is not the best algorithm to use for prefetching, although it maximizes the utilization of buffer space. Hence, we see that buffer space utilization also may not be the best performance metric of a video server.

The advantage of dynamic partitioning of the buffer space is that it maximizes the buffer space utilization and results in a high disk utilization if aggressive prefetching is employed.

One disadvantage of dynamic partitioning is the complexity involved in managing the buffer space. Decisions of what to allocate and what to deallocate will become more complicated in a dynamically-partitioned buffer. Policy decisions need to be made about what type of data and which data block needs to be deallocated upon the fetch of another data block.

In our project, we have implemented dynamic partitioning of buffer space. We will discuss the policy decisions we made in the later parts of this paper.

5. Description of the Simulation Environment

We have implemented our buffer management mechanism in a video server disk array simulator called DiskSim that is built on top of the event-driven simulation infrastructure csim. DiskSim models an integrated file system, which stores text files and continuous media files. Some clients send requests for text files and some send requests for continuous media files. The integrated file system modeled is a variant of Symphony [2].

DiskSim supports many of file system parameters to be specified. These parameters include the number of disks in the array, RAID level, top-level scheduling policy, disk scheduling policy, disk placement policy, base disk block size, continuous media block size, striping width, round time, number of video files on the array, number of text files on the disk array, number of clients accessing the file system, number of text clients accessing the file system, parity group size, etc.

We will briefly describe the important features of the simulation environment, which will help understand our implementation of buffer management mechanism on top of the DiskSim model.

5.1. Scheduling Mechanisms in DiskSim

DiskSim implements the Cello Disk Scheduling algorithm [3] used by Symphony integrated file system [2]. The scheduler is divided into two layers: class-specific schedulers and a class-independent scheduler. The class independent scheduler determines when and how many requests from each application should be inserted into the scheduled queue (Scheduled queue is a First Come First Serve queue which sends the requests to the disk array). Class-independent scheduler also notifies the class independent schedulers as to where in the scheduled queue they should insert their requests. The class-independent scheduler performs these functions by allocating weights to each class of application. This scheduler is work-conserving in the sense that if a class used up all of its allocation and still has requests pending, it will insert those requests one by one into the scheduled queue if no other class has pending requests [3].

The class-specific schedulers determine the position to insert requests of the application class they are supporting. This position is determined for each request. There are three class-specific schedulers: one that schedules real-time requests, one that schedules interactive best effort requests, and one that schedules throughput-intensive best effort requests. These schedulers take advantage of the properties of their application class. For example, real-time requests can be delayed until their deadlines. Hence, they have some slack in scheduling. However, interactive requests require good response time from the server. Therefore, interactive requests are inserted into the scheduled queue before any other request class as long as real-time requests' deadlines are met. After insertion into the scheduled queue, each class of requests is serviced in SCAN order from the disk.

5.2. Layout of Files in DiskSim

The number of files on the disk array is a parameter to the DiskSim simulator. Disksim takes in a trace of a file and randomly selects a disk in the disk array to start placing the file. In our

simulations, placement is done in fixed size blocks. To simulate multiple files, DiskSim places the same file starting from different disks. Starting positions of each placement is selected randomly. When a client accesses a file, it starts the access from a random position in the file and continuous to access until the whole file is complete. Once the whole file is complete, it starts reading the file from the beginning again.

We modified this feature of DiskSim to support reading of multiple trace files into the disks. Most of our simulations were conducted with 10 trace files fed into the simulator.

6. Design of Our Buffer Management Algorithm

Current version of DiskSim does not include a buffer management scheme. Infinite amount of buffering is assumed in the disk array. Besides, no prefetching or caching algorithm is implemented in DiskSim. We have implemented a buffer management algorithm that models finite amount of buffering that is dynamically allocated between prefetch and demand requests. We have also implemented an intelligent prefetching scheme that reduces the load across all disks in the disk array. The key features of our algorithm are:

1. Dynamic partitioning of the buffer space between demand and prefetch requests.
2. A priority mechanism used for deciding what to allocate in and what to deallocate from the buffer.
3. A prefetching mechanism that tries to distribute the load evenly among all disks.
4. A simple optional caching mechanism.

Here we describe the specifics of our buffer management algorithm.

6.1. Buffer Space

The buffer space in our algorithm is global to all disks. There is finite amount of buffer, which contains space that is a multiple of the basic disk block size. This buffer is used both for demand requests and prefetch requests. The buffer is not statically partitioned among the different types of requests. At any given time there might be any combination of demand and prefetch requests in the buffer. Due to the variable bitrate nature of the video streams simulated, the buffer space devoted to client demand (server-push) requests will vary from round to round. The remaining space will be allocated for any prefetch request that is issued in the same round.

6.2. Prefetching Scheme and Scheduling of Prefetch Requests

The prefetching scheme we have implemented can prefetch ahead for n rounds for all clients, where n is a parameter that is supplied to the simulator. Prefetch requests are inserted into a queue separate from the queue that contains client demand requests. During the scheduling of requests on the disk array, client requests are given priority over prefetch requests. Hence the scheduler never schedules a prefetch request if the already inserted client requests use up the whole round time. If the client requests do not use up the whole round time, the scheduler inserts prefetch requests into the scheduled queue such that the total time to service all requests does not exceed the round

duration. The demand requests and prefetch requests are then each serviced by the disk in SCAN order.

The generation of prefetches is done on a client basis. The video server knows which disk blocks each client will access in the next couple of rounds. Hence, it issues prefetch requests for those blocks for each client in the current round. In our current implementation, the video server tries to predict the most heavily loaded disk in the next round and starts to issue prefetches for that disk first. Hence, the prefetches issued would alleviate the load in the most heavily loaded disk in the next round. We can also order the disks based on the probability of a disk being the most heavily loaded one in the next round. Then we can issue the prefetches in this order, the ones that are going to a disk that has a higher probability of being the most heavily loaded first, and the ones that are going to a disk that has a low probability of being the most heavily loaded last.

The prediction of the load on a disk is currently done on a history based mechanism. The server keeps track of the most heavily loaded disks in every round. It calculates which disk will be most heavily loaded based on the access patterns of the clients and placement of data on the disks (the server knows exactly which frames will be accessed by the clients in round x and the location of those frames in the disk array). Hence, the server tries to schedule prefetch requests to the predicted most heavily loaded disk first.

6.3. Dynamic Allocation/Deallocation of Buffer Space

We use a priority-based mechanism to decide what to allocate/deallocate in the buffer space. Demand requests have higher priority than prefetch requests. The buffer allocates and deallocates on the granularity of disk block size. It keeps a list of unallocated (free) blocks. This list contains all buffer blocks in the beginning. When a client demand request is fetched from the disk, the buffer manager checks the *free list* to find free blocks to allocate for the client request. If free blocks are found they will be removed from the free list and client's demand data will be allocated in the corresponding addresses in the buffer. If no free blocks are found or if the client's data does not fit the found free blocks, the buffer manager checks the *prefetch list*, which contains the buffer blocks that contain data that is prefetched. This list is maintained in the order of prefetch request generation. The buffer manager frees enough of the blocks from the tail prefetch list (most recently inserted prefetches will be replaced with the anticipation that they will be the ones used latest in the future) and allocates these blocks for the incoming client demand request. Hence, prefetches are replaced in favor of demand requests so that deadlines will be less likely to be violated. If both the prefetch list and free list are empty on a demand fetch, the fetched block for the client is discarded.

Upon the fetch of a prefetched block from the disk, the buffer manager first checks if there are blocks in the free list. If blocks exist in the free list the prefetched data is placed in those blocks and those blocks are moved from the free list into the prefetch list. If there are no blocks available in the free list then the prefetched block is simply discarded. A prefetch request cannot deallocate a demand request from the buffer under any conditions.

As can be seen from this description, our mechanism is very flexible in dynamic allocation of requests and tries to maximize buffer space utilization and minimize deadline violations.

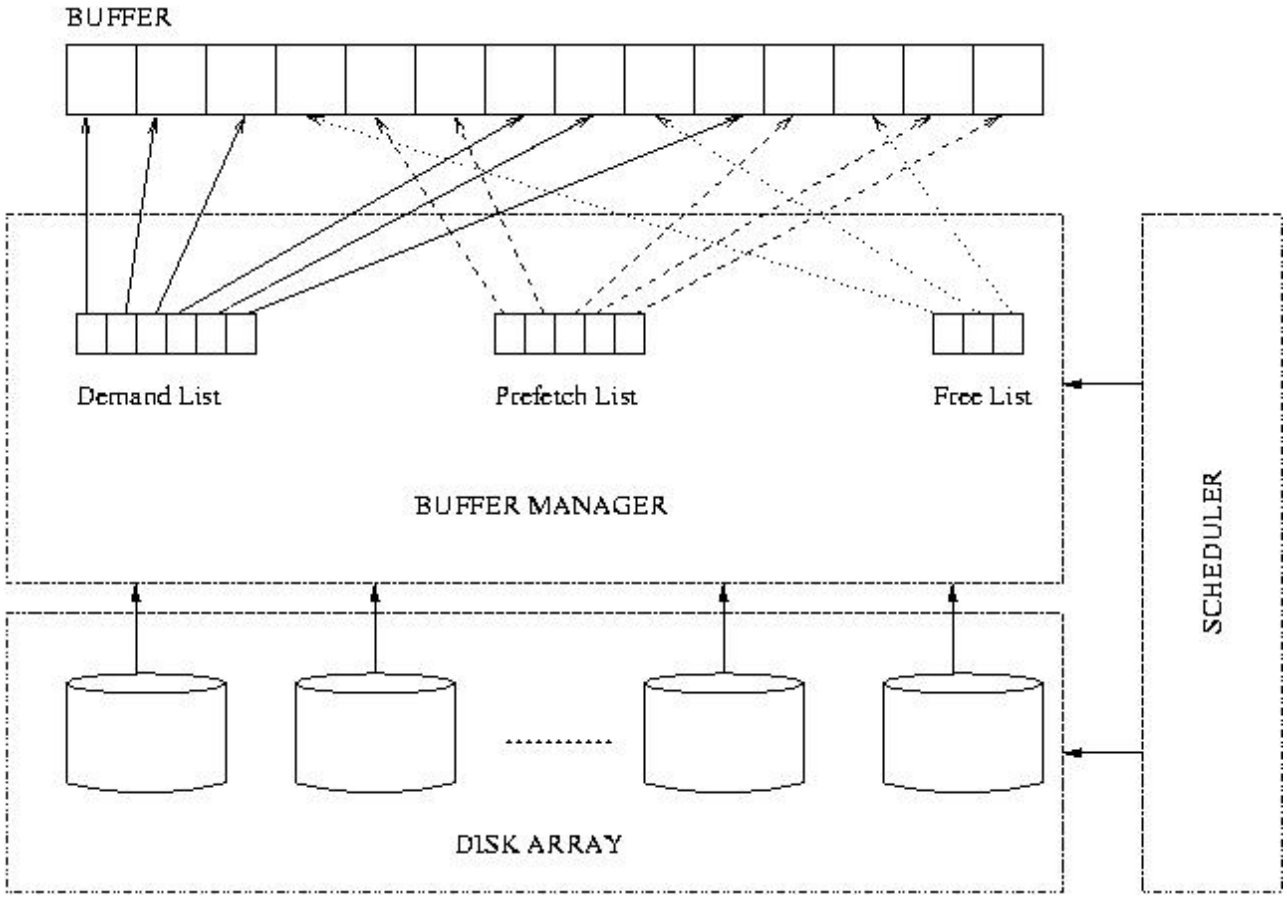


Figure 1. High Level Organization of the Video Server and Lists Used by Buffer Manager

6.4. Simple Interval Caching

Our buffer management scheme also employs a simple interval caching algorithm by allowing some demand requests not to be deallocated once their deadline has passed. Normally, once we service a demand request from the buffer, we free the blocks corresponding to that request. Alternatively, if we want to form links between two temporally spaced clients we can extend the deallocation time of the blocks fetched by the first client until the round they are read by the following client. Hence the interval caching algorithm will save disk bandwidth. Cached data has lower priority than demand requests in this scheme. However it has equal priority with prefetch requests. Hence, if a demand block is fetched and there is no free buffer space available either a cached block or a prefetched block is deallocated in the buffer to make space for the demand request. The choice of which block will be deallocated depends on which one of the block types is fetched latest in the future.

6.5. Putting It All Together

Figure 1 shows the high-level organization of the video server with a focus on buffer management. The scheduler on the right is a general logical scheduler that determines whether to send requests to

disks or to the buffer. Therefore it has connections to both the disk array controller and the buffer manager. If a request hits in the buffer, the scheduler will not schedule that request to the disk.
As mentioned before, the buffer manager is in charge of managing the three lists: free list, prefetch list, and the demand list. It is also in charge of communicating with the scheduler about what is contained in the buffer.

There are simple main rules used by the scheduler as to whether or not to generate/schedule a prefetch request:

1. If there are still client requests to be serviced, the scheduler will not schedule any prefetch request.
2. If there is no buffer space left, the scheduler will not schedule a prefetch request.
3. If the completion time of a prefetch request will exceed the round duration the scheduler will not schedule a prefetch request.
4. If none of the above conditions are true, the prefetch request can be scheduled.

The buffer space manager also uses simple rules to decide what to allocate/deallocate:

1. If the incoming data block is prefetched, never deallocate a block that is fetched by a client request (demand). Insert the prefetch into the buffer only if there are free blocks. The deadline of the prefetch is determined by the scheduler. Prefetch can stay in the buffer up to n rounds.
2. If the incoming data block is a demand request allocate a free block. If a free block does not exist in the buffer find the youngest prefetched block and deallocate it.

The server also uses a simple metric to determine what to prefetch:
1. It keeps a history of most heavily loaded disks over rounds.
2. Based on that history, it sorts the disks in order of predicted heavy load for the next round (This prediction is not always accurate).
3. It starts issuing prefetch requests for those disks that are predicted to be heavily-loaded in the next round.

7. Implementation of Our Buffer Space Manager in DiskSim

The implementation of our buffer space management algorithm follows closely what we have described in the previous section. We allocate a buffer structure, which contains n number of basic disc blocks where n is the size of buffer as supplied by the user of the simulator. This struture has three lists associated with it: free list, prefetch list, and demand list. Free list points to the unused blocks in the buffer. Prefetch list points to the blocks that are allocated for prefetched blocks. Demand list points to blocks that are allocated for server-push demand requests.

We implement the priority-based allocation/deallocation mechanism by tagging each block in the buffer as prefetch or demand request. The blocks that are tagged as demand requests are deallocated at the end of each round when their deadlines have passed. The blocks that are tagged as prefetch requests are deallocated when a demand request needs a free block and no free block is available. A prefetched block that is later used by a client is promoted to demand status and deallocated at the end of the round. Our server employs a simple kind of interval caching algorithm by supporting the demand requests to stay in the buffer space for n rounds where n is input by simulator user.

Prefetching is supported by the addition of a new prefetch queue. When the server-push requests for a client is generated for the periodic queue, prefetch requests are also generated for the prefetch queue. These requests are then inserted to the scheduled queue, which services the requests in SCAN order for each class (demand vs. prefetch). As mentioned before demand (server-push) requests have higher priority than prefetch requests. Hence, prefetches are not inserted into the scheduled queue if a demand request is outstanding. Once all demand requests are inserted into the scheduled queue, scheduler inserts prefetch requests in SCAN order and makes sure that the deadline of no demand request will be violated by inserting the new prefetch into the scheduled queue.

The following diagram shows the scheduler queues used to control demand and prefetch requests. The periodic demand queue and prefetch request queue are each kept in SCAN order. Demand requests are first inserted into the scheduled queue and prefetches are inserted when no requests are ready in the periodic queue.

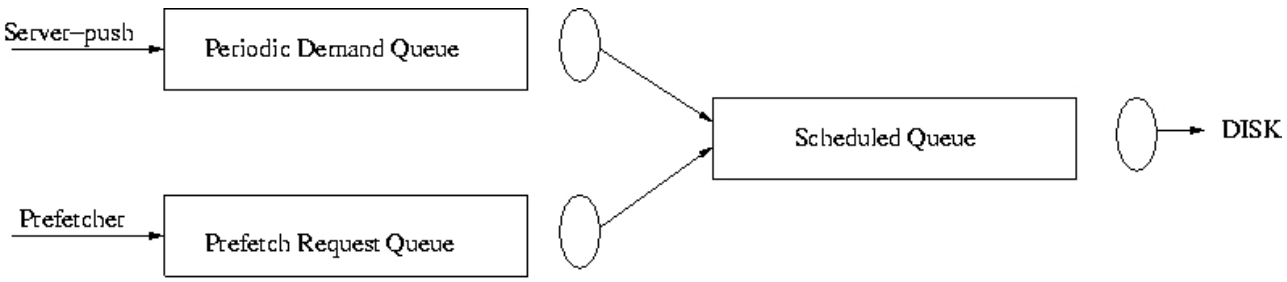


Figure 2. Scheduling of Prefetch and Demand Requests

8. Simulation Parameters Used in Experiments

Although DiskSim models an integrated file system, we did not experiment with text clients in this assignment. Our goal was to maximize the number of video clients supported by a video server. Hence, we did not place any text files in the disk array. Nor did we simulate the access of text clients. Therefore, the best effort queue was never used in our experiments. Besides, we did not experiment with aperiodic real-time requests. All real-time requests seen by our server were periodic.

Here is a listing of parameters we used to conduct our experiments. Some of the parameters are varied in experiments. We will talk about how those are varied in the evaluation section.

Number of disks in the disk array: Variable in experiments
RAID level: 0
Top level scheduling policy: PLOOKEDF
Disk Scheduling policy: FCFS
Disk Placement Policy: Round robin
Base block size: 4 KB
Use fixed block size: Yes

Continuous media block size: Variable in experiments
Stripe across all disks: Yes
Use default disk (Seagate Elite 3): Yes
Round duration: Variable in experiments (default is 1000 ms)
Data rate: 30 frames per second
Number of continuous media files on disk: 10
Number of text files on disk: 0
Size of text files on disk: 0
Assume all clients arrive at time = 0: Yes
Number of video clients: Variable in experiments
Number of text clients: 0
Use server push for all continuous media clients: Yes
Simulation length: Variable number of rounds
Starting round for collecting statistics: 1
Partition disk bandwidth: No
Prefetch for the next n rounds n = 1.
Deallocate prefetches after n rounds n = 4.

We do not use failure recovery mechanisms in our simulations. We assume a fault-free environment.

We have performed extensive experiments to evaluate the effectiveness of our buffer management scheme. We will now talk about these experiments and comment on the effectiveness of our mechanism.

9. Evaluation of Our Mechanism and Answers to the Questions

9.1. Evaluation Mechanism

In this section we describe the evaluation of our implementation of buffer management scheme in DiskSim. The evaluation of our scheme is tied closely with the project questions given in the project handout. While answering each question we will show that our buffer management scheme provides more flexibility and increases the performance of the video server under certain conditions.

9.2. Answers to Project Questions

9.2.1. Question 1

“For a given a server configuration (i.e., number of disks, amount of buffer space, and characteristics of client requested files), how will you select a round duration? “

Selection of Round Duration

The selection of round duration depends on many parameters of the server. Most important of these are:
1. Number of disks in the server (This affects how the load is distributed on disks).

2. Amount of buffer space available in the server
3. If the server employs prefetching or caching

Given a server configuration, to determine the optimal round duration, we need to find the round duration that supports maximum number of clients without violating more than 0.1% of the deadlines. Hence, we need to plot the graph of "Round duration vs. Maximum number of clients supported" for a given server configuration. The maximum point in this graph will correspond to the optimal round duration we would like to select.

Graph 1 shows one such graph for a given server configuration. The configuration corresponding to Graph 1 is depicted in Table 1.

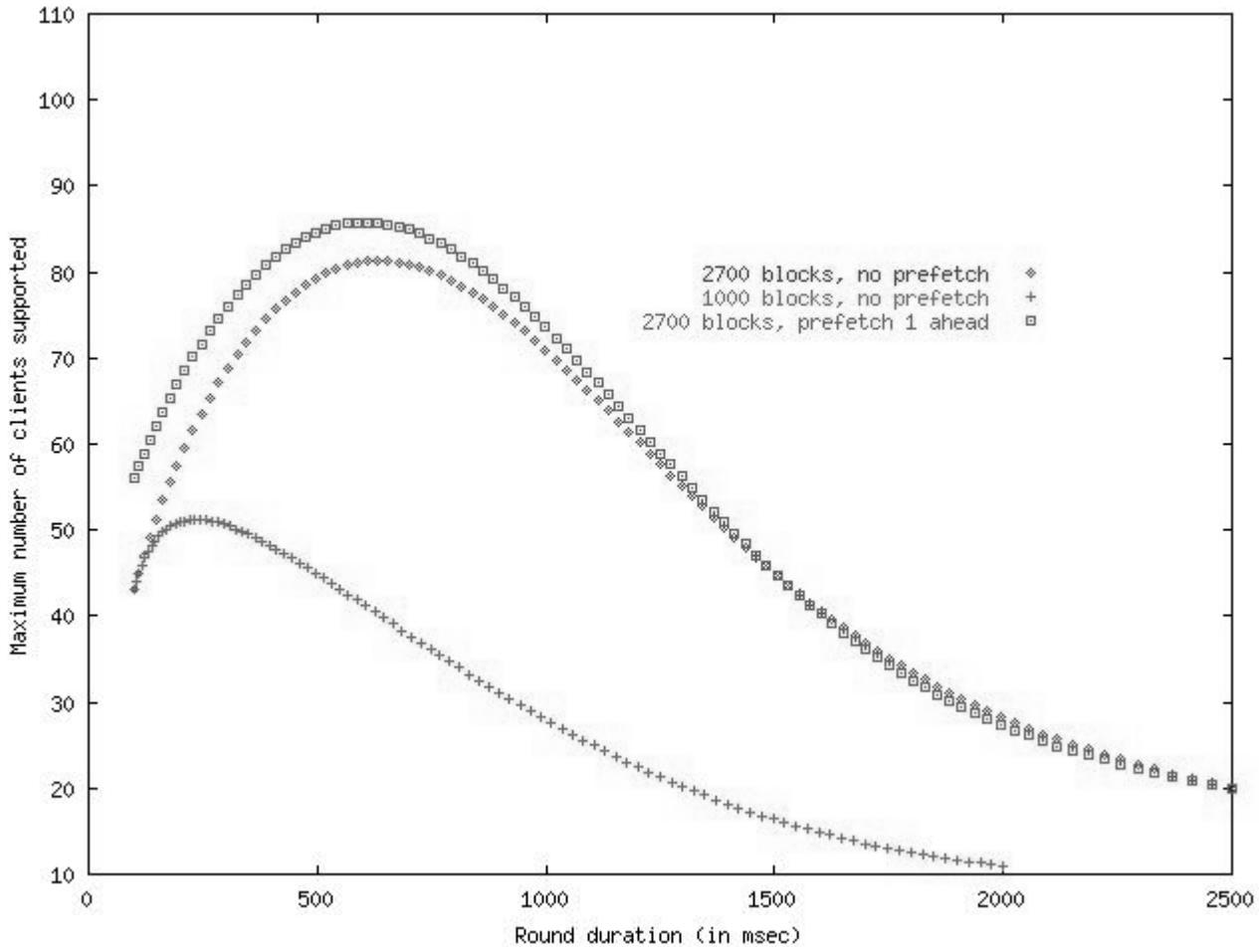
Number of disks	20
Amount of buffer space available	2700 blocks (10.8 MB)
Buffer allocation policy	Dynamic partitioning
Number of files on server	10
Data rate of files	30 frames per second
Type of files	8 variable bit rate, 2 constant bit rate
Disk block size	4 KB
Continuous media block size (stripe unit size)	16 KB
Number of simulation rounds	500

Table 1: Server and simulation configuration used for data shown in Graph 1.

Interpretation of Graph 1

Graph 1 shows several important characteristics of our scheme. It also shows how the optimal round time changes based on server configuration. The line marked "2700 blocks, no prefetch" shows a server which does not employ our prefetching algorithm. It has a buffer space of 2700 blocks (10.8 MB). All of the space is available for client requests (i.e. buffer space is not partitioned). For this configuration, we see that the optimal round duration is around 750 ms. The shape of the curve is quite interesting in the sense that it portrays the limitations of the server in extreme cases. If the round duration is very low (100 ms), the server is round-duration-limited. This means that buffer space is not a bottleneck. However, the short round duration is a bottleneck because it does not permit the addition of more clients. If we add more clients to the system, the service time will start to go up and frames will start to get dropped, because the server does not have enough time in a round to service all client requests. On the other hand, when the round duration is too high, buffer space becomes a bottleneck and frames will get dropped because they will not fit in the buffer. We can clearly start seeing this effect for round durations greater than 1000 ms.

The second configuration, which is marked "1000 blocks, no prefetch" shows a server configuration that also does not employ prefetching but has a buffer size of 1000 blocks (4 MB). All other parameters of the server are the same as the previous one. There are two observations we can make based on this configuration:



Graph 1: Round duration vs. Number of supported clients for a given server configuration

1. We can see that small buffer size seriously limits the number of clients that can be serviced. The maximum number of clients that can be supported using a 2700-block buffer is around 80, whereas the maximum number of clients that can be supported using a 1000 block buffer is around 50.

2. We can also see that buffer size can have a significant impact on the optimal round duration. Optimal round duration for a server with 2700-block buffer is 750 ms, whereas optimal round duration for a server with 1000-block buffer is 200 ms.

The third configuration employs our prefetching algorithm as described in Section 6. In rounds where client requests do not consume the whole round duration, prefetch requests are sent to the disk array to utilize the round duration better and possibly reduce the load on the disks in the next few rounds (if enough buffer space to hold the prefetched data exists). Note that dues to the variable bit-rate nature of the media stored in disks, client requests require variable amount of data in any given round. We exploit this characteristic of the video data and round-based scheduling in our prefetching algorithm.

Third configuration shows that our prefetching algorithm is especially effective in small round durations. This makes sense because in short round durations, the buffer space devoted to client requests will be quite small. Hence, whenever prefetch requests have time to be inserted, they will always be allocated in the buffer. Besides, our prefetching scheme also acts like a caching scheme in the sense that we do not deallocate prefetches from the buffer unless a client request requires space in the buffer. Due to these effects, prefetching increases the maximum number of supported clients for small round durations. In larger round durations, this effect is not visible, because the server becomes buffer-limited and prefetch requests are not issued, because almost all of the buffer space is devoted to client requests.

Hence, the conclusions we draw from this graph are:

1. Our prefetching scheme works effectively.
2. Our modeling of the buffer management works reasonably (As the buffer size changes, the maximum number of clients we can support also changes).
3. Optimal round duration for a server depends on whether or not prefetching is employed on the server. It also depends on the buffer space available.

9.2.2. Question 2

"How does the total buffer size required increases with the round duration? How does the number of clients supported by the server increase with the round duration? What is the average utilization of the buffer space (i.e., what is the average of the percentage of buffers utilized during each round) with your buffer management scheme? What is the average buffer space utilization if you had partitioned your available buffer space statically between the buffering and prefetching functions?"

Round Duration vs. Buffer Size

Graph2 shows the relationship between buffer size and round duration for several server configurations. Parameters of each configuration are determined so that each can support 50 clients. Hence, the buffer size (y axis) is the minimum buffer size that is required to be present in the server so that the server can support 50 clients (with frame drop probability less than 0.1%). The configuration of the server is the same as the one given in Table 1, except for the buffer size, which is variable in this case. The server employs prefetching. Graph 2 depicts two configurations:

1. Dynamic partitioning: The server partitions the buffer space between prefetches and client requests dynamically. There is no fixed or predetermined allocation for each type of request. If space is available, the server will prefetch aggressively.

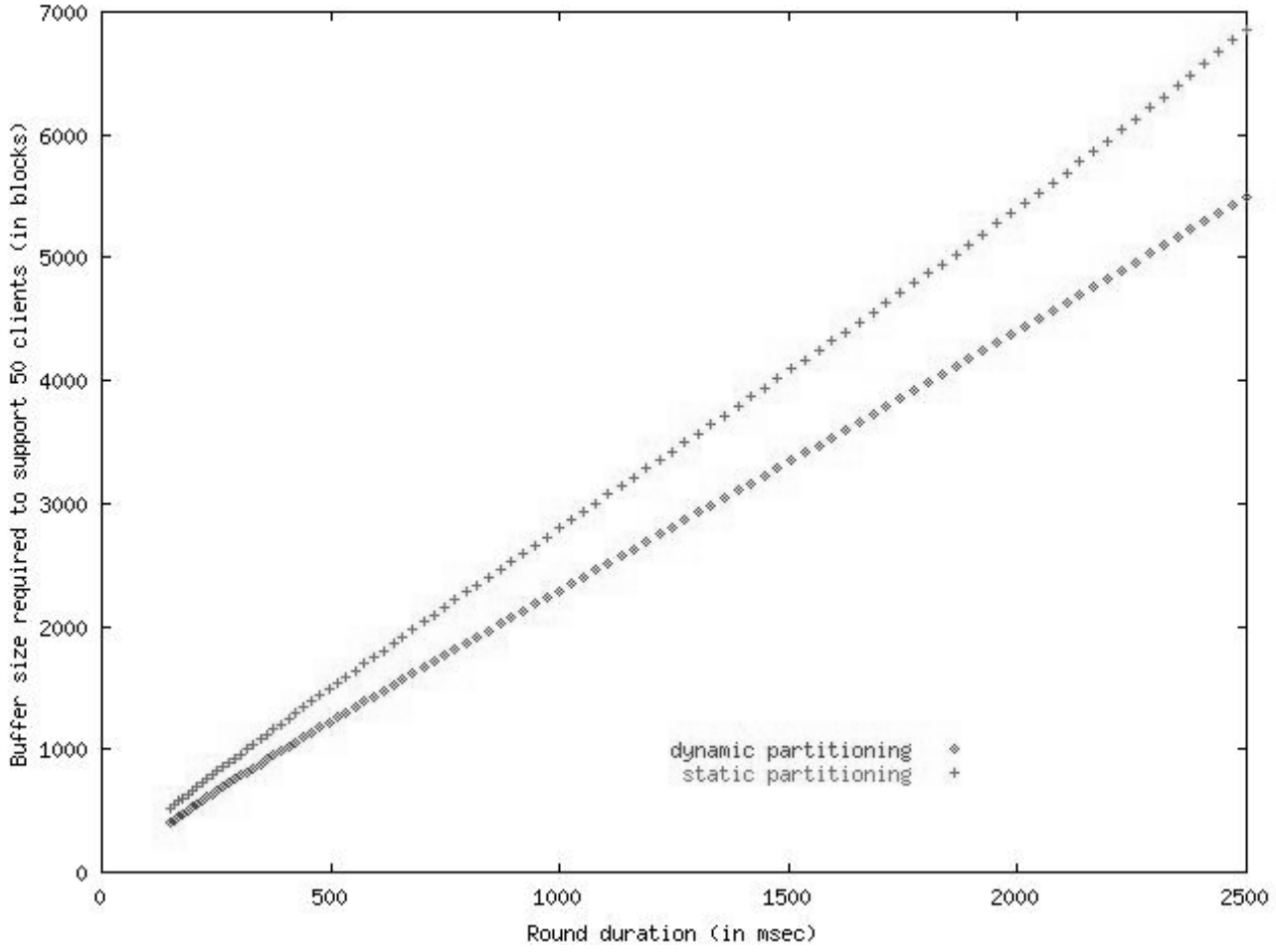
2. Static partitioning: Buffer space is statically partitioned between prefetches and client requests. In this experiment 20% of the buffer space is allocated for prefetches. Remaining 80% of the buffer space is used for client requests.

Graph 2 shows a linear relationship between round duration and buffer size. As we increase the round duration, the buffer size required to support 50 clients increases linearly. This is expected, because the buffer space required should at least be able to hold data that will be fetched by 50

clients in one round duration. As round duration increases, the amount of data that needs to be fetched by the client also increases. In fact an idealized equation for the amount of buffer space required is:

minimum blocks in buffer = (frames / second) * (disk blocks / frame) * (rounds / second)

Hence, it is reasonable to expect a linear relationship between round duration and required buffer size. The above equation is ideal because due to the variable bitrate encoding nature of the stored video data (disk blocks / frame) is variable.



Graph 2. Required buffer size to support 50 clients vs. round duration of the server based on buffer management policy.

Note that our prefetching algorithm will be less likely to be effective in this experiment because we are simulating a buffer space-limited server. Hence, we are less likely to prefetch data (or store the prefetched data) due to unavailability of free space in the buffer.

Graph 2 clearly shows the advantage of dynamic partitioning over static partitioning of the buffer. Dynamic partitioning increases the utilization of buffer. However, with static partitioning the buffer needs to be bigger to be able to service same number of clients without violating more than 0.1% of the deadlines. We see that the difference between static and dynamic partitioning in the required buffer size becomes magnified as round duration increases. This is due to the fact that bigger buffers are required for larger rounds (Because bigger buffers are required static partition that contributes to inefficiency becomes larger in size.).

Round Duration vs. Number of Clients Supported

We discussed about the relationship between number of clients supported vs. round duration when answering Question 1. As shown in Graph 1 this relationship is a concave down function. As the round duration increases, the number of clients supported increases up to some point. After that point, buffer space becomes the bottleneck and the number of clients supported starts to decrease. Hence, for a given server configuration we can find a round duration that supports the maximum number of clients. Further explanation of the relationship between round duration and number of clients supported is given in the answer to the previous question.

Average Buffer Utilization and the Effect of Static vs. Dynamic Partitioning

Graph 3 shows the relationship between the round duration and the buffer utilization. The size of the buffer in this experiment is the size required to support 50 clients. The configuration of the server is exactly the same as the one used for Graph 2 (prefetching server). We make several important observations in this graph:

1. Buffer utilization does not really depend on round duration.

2. Buffer utilization of the static partitioning scheme is much lower than the buffer utilization of the dynamic partitioning scheme. This explains the difference between static and dynamic partitioning observed in Graph 2.

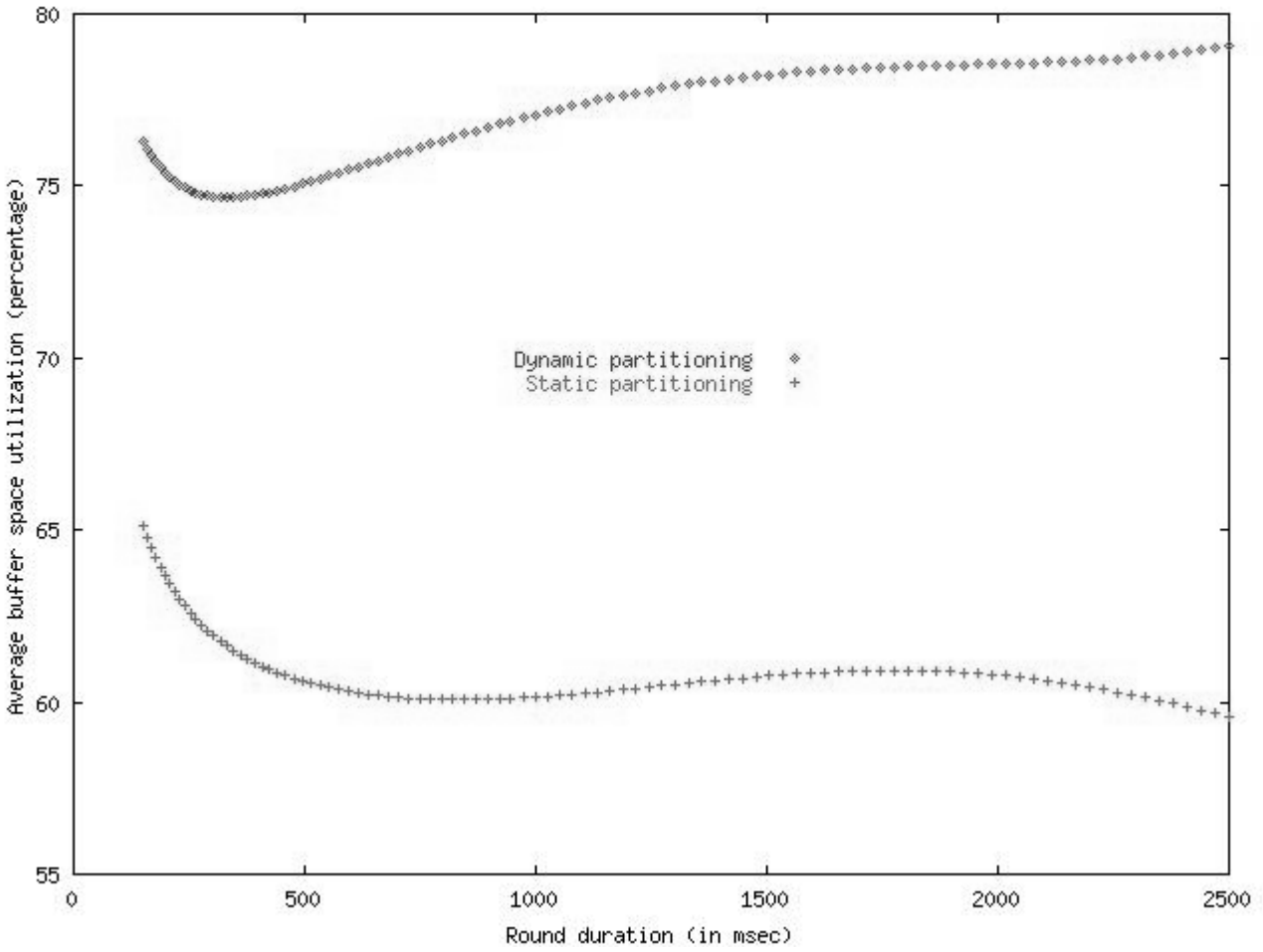
3. Even for dynamic partitioning, the buffer utilization is around 80% although the buffer size is selected such that it is the minimum size that can support 50 clients. This is due to the variable bitrate nature of the video files stored in the disk array. The required amount of buffer space varies every round. Hence, in many of the rounds we do not require the maximum buffer space. Therefore the utilization is on the lower side. However, if the stored media files were constant bitrate, the utilization of the buffer space would be much higher (perhaps 100%).

We conclude that our dynamic partitioning and prefetching scheme works very well because:

1. It increases the buffer utilization compared to static partitioning.

2. Compared to a non-prefetching configuration, it increases the number of clients supported by the video server for a given buffer size.

3. Compared to static partitioning it needs a smaller buffer to be able to support a given number of clients.



Graph 3. Average buffer utilization when the buffer is dynamically or statically partitioned.

9.2.3. Question 3

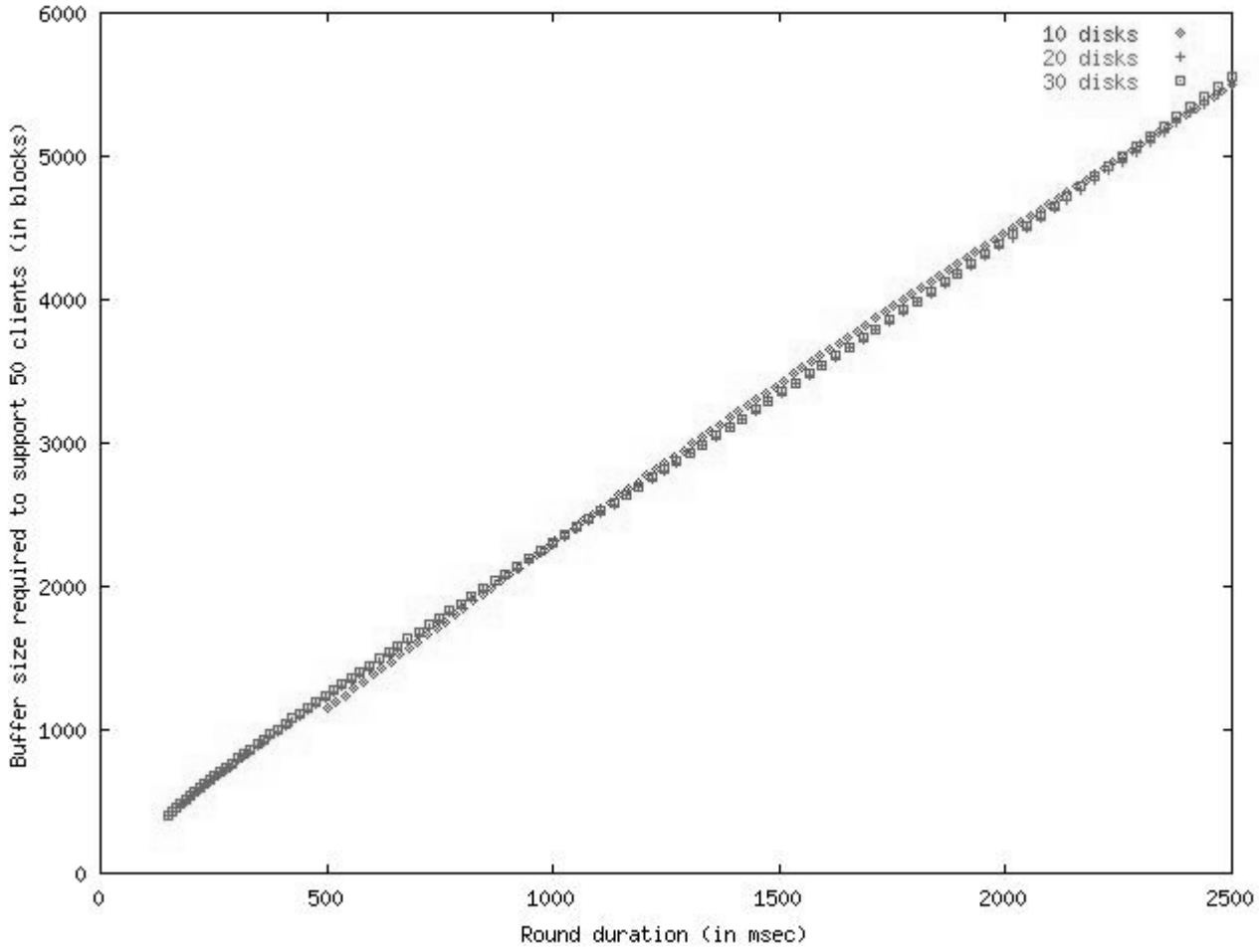
"How does increasing the number of disks in the system affect your answer to Question 2?"

In this question we analyze the same issues in Question 2. We now look at how these issues change when we decrease/increase the number of disks in the system.

Round Duration vs. Buffer Size

As the number of disks in the system increases, the linear relationship between the minimum buffer size required and the round duration does not change. In fact, as we can see from Graph 4, the number of disks in the system does not affect the required buffer size for a given round duration.

This is expected because our buffer management scheme accommodates a “global” buffer which sits on top of the disk array. The requests coming from all disks are placed in this global buffer. It is not the case that each disk has its own “local” buffer. If that were the case, then we would expect an increase in the buffer size requirement as we increased the number of disks in the disk array. In our buffer management scheme, the factor that determines the required buffer size is the amount of data that is required to be fetched in one round and the effectiveness of prefetching.



Graph 4. Relationship between buffer size required vs. round duration based on number of disks present in the system.

Round Duration vs. Number of Clients Supported

Intuitively, the number of disks present in the system should affect the number of clients supported by the system. Having a larger number of disks increases the possibility that request load in a given round will be distributed among more disks and hence the load on a given disk will decrease. This

means the service time of each disk will increase. Hence the server will be able to service more clients in a fixed round duration.

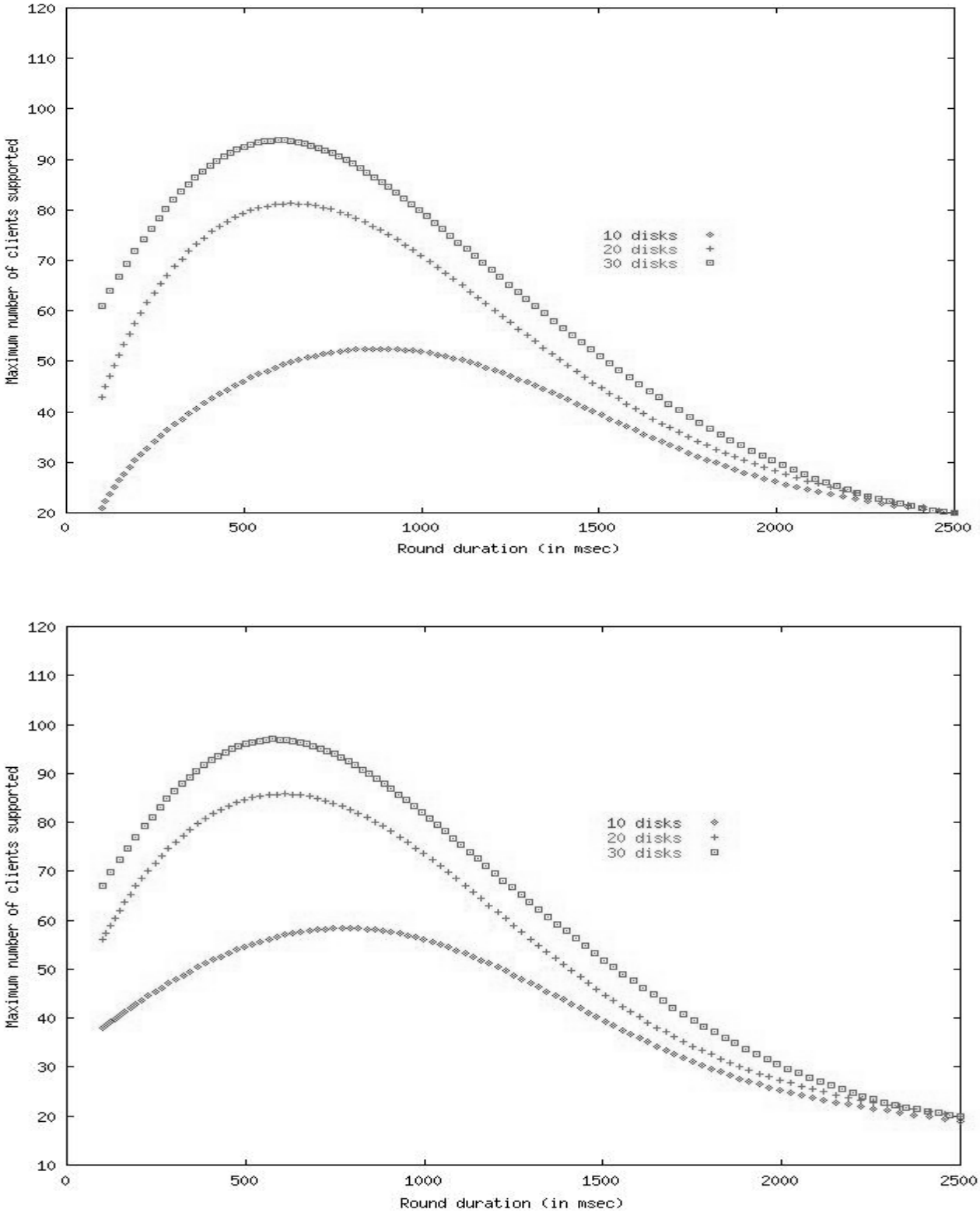
Graphs 5 and 6 confirm this intuition. As number of disks in the system increases, the number of clients supported by the system increases. The configuration of the server is the same as given in Table 1. Graph 5 shows the simulation results for a non-prefetching server. Graph 6 shows the results for a server that employs our prefetching algorithm.

We can see that greater number of disks results in greater number of supported clients for a given round duration. This is due to the distribution of load across more disks. We can also see that this effect diminishes for larger round durations. This is mainly due to the fact that the server becomes buffer-space-limited (In this case we assume a fixed buffer size of 2700 blocks). As buffer space becomes a bottleneck, it really does not matter how many disks are on the server because regardless of the number of requests that can be serviced in a round (round duration and service time are not bottlenecks) many requests will be dropped due to the small buffer size.

Another observation from Graphs 5 and 6 is that our prefetching scheme consistently outperforms the non-prefetching server configuration. This is true for smaller round durations where buffer size is not a bottleneck. We can see that our prefetching scheme is effective especially when there is a small number of disks in the system. For example, for a 100 ms round duration, the number of clients supported by the server that employs our prefetching scheme is 36 whereas this number is 21 for a server that does not employ any form of prefetching.

Average Buffer Utilization and the Effect of Static vs. Dynamic Partitioning

This interaction of this relationship with the number of disks in the system is very similar to the interaction of “round duration vs. buffer size” with the number of disks in the system: A change number of disks does not affect the average buffer utilization of the system. This is due to the fact that we are working in a buffer-space limited environment that can support 50 clients. The utilization of the buffer depends on the number of requests coming from the 50 clients. It does not depend on the number of disks present in the system. Even if the environment is not buffer space limited we would not expect the average buffer utilization to change with a change in the number of disks.



Graphs 5 and 6. Variation of the relationship between clients supported vs. round duration with a change in number of disks. Above figure shows a non-prefetching server. Below figure shows a prefetching server

9.2.4. Question 4

"What is the minimal server configuration that you would need to design a video server that can service up to n clients? Provide your answer for n=32, 64, 128, 256, and 512."

To answer this question we needed to reduce the simulation design space of the video server. Hence we ordered the resource requirements based on a priority scheme. We assume that the round duration is fixed to 1000 ms. Other configuration parameters of the server is the same as described in Table 1 except for the ones we modify below.

Our first aim was to get the minimum number of disks that can support n clients given infinite buffer size. After finding this minimum number we found the minimum buffer size that can support n clients. For n=256 and n=512 we encountered many problems with DiskSim. The maximum number of disks we were able to simulate using DiskSim was 39. Hence, for n=256 and 512, we followed a different algorithm to determine the minimum server configuration. We tried to choose a minimum round duration that can support 256 or 512 clients on a system that has 39 disks. The simulations for these configurations were quite slow hence the reported value may not be the minimum configuration.

The following are the minimal configurations that support n clients:

n = 32:
disks = 5
buffer size = 1380 blocks (5.5 MB)
round duration = 1000 ms

n = 64:
disks = 10
buffer size = 2800 blocks (11.2 MB)
round duration = 1000 ms

n = 128:
disks = 22
buffer size = 6000 blocks (24 MB)
round duration = 1000 ms

n = 256:
disks = 39
buffer size = 40000 blocks (160 MB)
round duration = 4000 ms

n = 512:
disks = 39
buffer size = 80000 blocks (320 MB)
round duration = 5000 ns

10. Addition of Caching Support to the Proposed Buffer Management Scheme

As we already described our scheme supports caching to some level (and unintelligently) by allowing prefetches to stay in the buffer even after they are used by a client. Clearly this is not the best was of caching objects and instead of this we would like to use an interval-based caching scheme that probably would work well due to the characteristics of client accesses.

Here we describe a simple mechanism in which interval caching can be added to our buffer management scheme. First of all, a mechanism that detects temporally spaced accesses to the same file by different clients needs to be implemented. This is easy to implement. The server can keep track of accesses to each individual file and figure out which files are accessed by temporally-spaced clients. One implementation decision is to decide what is the largest possible temporal-spacing that can be handled. The server can either set a predetermined limit to this spacing (for example, it can say that if the spacing is greater than n frames the interval will not be cached.). Or this can be determined dynamically based on the buffer space allocated for the cache.

A more important decision is how to allocate space for the cache. We propose several schemes:

1. One simple but static scheme is to allocate a cache space separate from the buffering mechanism we discussed in this report. When the server is started some memory space can be allocated to the cache and solely used for caching purposes. This mechanism guarantees that at least some data will be cached at any given time and surely saves bandwidth. However, if we are memory-limited, statically allocating this cache space may lead to violation of deadlines of client requests because they may not fit in the rest of the buffer. This is a problem with all static allocation schemes and needs to be addressed using a more flexible dynamic approach.

2. A second scheme is to extend our buffering mechanism to include cache requests. To be able to do this we need to determine the priority of cache requests. It might make sense to make cache requests higher priority than prefetch requests but lower priority than demand (client) requests. However, in this case we need to make sure that prefetch requests are also being issued. Hence, we might want to limit the buffer space occupied by cached data. But this will be going back to static cache partitioning in some sense, therefore we propose a better mechanism.

The mechanism dynamically allocates space in the buffer between prefetch and cache requests. We value cache requests more because, as discussed in the beginning of this report, they save disk bandwidth. However, prefetch requests are also useful because they are effective in reducing the load on the disks.

Our scheme works as follows:
- Client requests have higher priority over all requests so they can evict a prefetch request or a cached block. Eviction is done based on the cost of eviction. When trying to allocate a demand block the server calculates the predicted cost of evicting each prefetch block or cache block. Prefetched blocks that are used to reduce the load on the predicted most heavily loaded disk are the less likely ones to be evicted. Cache blocks that have the least time to reaccess are also least likely to be evicted.

- Prefetches are inserted if round time permits for them to be inserted. Hence, they will not always be inserted. Taking advantage of this fact, the server will pick an n value for temporal spacing limit for what data to cache (as discussed above). If not many of the prefetches are getting inserted the server will limit the number of buffer blocks consumed by cache requests by reducing n (meaning that only frames of clients that are following each other very closely in time will be cached). The details of this algorithm are dependent on the implementation.

We have not implemented any of these schemes due to lack of time. Both schemes have advantages and disadvantages. The first scheme is advantageous in the sense that cached data will never be flushed from the buffer because it has its own space. However, due to static partitioning, buffer utilization will be low and possibly some deadlines will be violated. The second scheme employs a more dynamic approach, increasing buffer utilization, but it does not guarantee anything for the cached data. Some of the cached data may be useless due to the fact that it might be flushed before it is used (This reduces the "effective" buffer utilization). Both schemes need to be implemented and evaluated to see if any of them is more advantageous.

11. Conclusion

We believe that the most important learning experience we had in this project is the realization of the fact that a dynamic buffer management scheme performs much better than a static buffer management scheme which statically allocates buffer space between prefetches and client requests. Due to the limitations of DiskSim and lack of documentation we were not able to perform all the experiments we would have liked to perform to gain a better understanding of our mechanism and the video servers in general, nor were we able to implement caching. We think that the number of clients supported by our buffer management scheme will increase dramatically when caching support is incorporated into the scheme.

References:

[1] Asit Dan and Dinkar Sitaram. Buffer Management Policy for an On-Demand Video Server. IBM Research Report RC 19347, T.J. Watson Research Center, Yorktown Heights, New York, January 1994.

[2] P.J. Shenoy, P.Goyal, S.S. Rao, and H.M. Vin. Symphony: An Integrated Multimedia File System. Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking, San Jose, CA, pp. 124-138, January 1998.

[3] P. Shenoy and H.M. Vin. Cello: A Disk Scheduling Framework for Next-generation Operating Systems, In Proceedings of ACM SIGMETRICS'98, The International Conference on Measurement and Modeling of Computer Systems, June 1998.