8. References


Plot 9: Response Time, MET with Multicasting and MET without Multicasting

Plot 10: Throughput, MET with Multicasting and MET without Multicasting
Plot 7: Response Time, MFP, MFP with MFM, and MFP with MFM and Multicasting

Plot 8: Throughput, MFP, MFP with MFM, and MFP with MFM and Multicasting
Plot 5: Response Time, MFP with Two Media Types and Four Media Types

Plot 6: Throughput, MFP with Two Media Types and Four Media Types
Plot 3: Response Time, MIP with Four Media Types, Four Media Types with MTM, Two Media Types, and Two Media Types with MTM

Plot 4: Throughput, MIP with Four Media Types, Four Media Types with MTM, Two Media Types, and Two Media Types with MTM
7. Plots of Simulation Results

Plot 1: Response Time, HiP

Plot 2: Throughput, HiP
verify this by running simulations with an increased number of processors. Finally, as previously mentioned, all of our simulations were run with uniform access and uniform distribution. We did this to simplify the analysis of our results. We would have liked to run an entire suite of simulations skewing access and distribution in order to clearly define which system configurations work best in conjunction with each of the partition strategies, their appropriate merging techniques and multicast capabilities.
ticasting increases throughput at a much greater rate as the number of users increases. This would suggest that multicasting is a viable alternative to merging in attempts to improve partitioning strategies. This simulation would seem to suggest the benefits of multicasting to be infinitely scalable. It should be noted, however, that these simulations do not account for additional scheduling overhead of multicasting. As the number of users increase, so will the size of the request queue and therefore the time needed to search for identical requests. This overhead could be controlled by using smarter insertion and searching algorithms when interacting with the request queue.

5.2.2. MET
Plot 9 and Plot 10 show our results of simulating MET with multicasting. MET limits sets of processor groups to exactly one media type. It would lead to the second smallest number of replicas, behind MFP. As expected, the system throughput flattens out for normal MET, but increases linearly for MET with multicasting. Response time, on the other hand, does not significantly improve with multicast. This is due to the extreme level of disk contention in this particular example. Here exactly half of the processors are allocated to the largest media type, and the other half are divided between the remaining three media types. Multicasting will only improve response time in systems where requests for objects are being satisfied. If extreme disk contention leads to the majority of the requests being queued, multicasting can do little to help system response until disks become available. Due to the limiting replication nature of MET, the best way to improve response time would be to add processors to the system.

6. Conclusion
While we were able to explore many issues through our modifications to the simulator, there were a few of our hypotheses we did not get a chance to investigate. We assumed the degree of both internal and external fragmentation would greatly increase due to the addition of media types. If we had been able to compare response time and throughput of the largest media type in a system with both two and four media types, we had hoped to be able to illustrate degradation in system performance due to external fragmentation. If we had been able to collect statistics on processor utilization and request queue length, we had hoped to be able to illustrate an increase in internal fragmentation when running a system with four media types compared to that of two. Many of our results with simulations of MET we hypothesized to be inconclusive. We would have liked to
5.1.4. MET

For this partitioning strategy we discovered that a comparison of our results with the results from two media types would not be valid due to the media types we chose and the number of processors in the system. MET allocates processors to each media type by the frequency of access to that media type, with the degree of fragmentation of the media type as the lower bound for the number of processors to allocate. In our case with 70 processors, four media types, and uniform access to all media types, the largest media type, with a degree of fragmentation of 35, occupies half the processors even though it is only being accessed approximately 25% of the time. This was not discovered until we saw the simulation results and realized that we should’ve compare both systems with at least 140 processors instead of 70. Due to lack of time we were unable to get good results for this partitioning strategy for comparison of a four media type system with a two media type system.

5.2. Experiment II

Ideally, we would have liked to simulate all four partitioning strategies, with and without merging using our multicast extension. This would allow us to thoroughly explore the performance of multicast. Unfortunately, we had to limit the number of simulation runs, due to timing constraints. We chose to simulate multicasting with the partitioning strategies that have the least overall number of replicas and therefore would stand to gain the most from multicasting.

5.2.1. MFP

MFP has exactly one replica of each object in each processor group. This yields the least number of replicas of any partitioning strategy. As can be seen from Plot 7, multicasting significantly decreases the system response time when compared to normal MFP as well as MFP with MFM merging. Multicasting satisfies multiple requests with a single object retrieval, thus decreasing overall system response time.

Plot 8 graphs the system throughput comparing MFP to MFP with merging and finally MFP with merging and multicasting. Notice that the throughput flattens out for normal MFP, indicating system saturation. Both merging and multicast allow for linear improvements to throughput, but mul-
lower than the response time with two media types (Plot 1 and Plot 2). This can be attributed to
the fact that we still have the same number of disk partitions in our system but it is less likely to
have disk contention with the largest media object. With four media types in the system, only
approximately 25% of the requests will be for the largest media object instead of 50% of the
requests in the case of two media types. The throughput of the four media type system is also
higher than the two media type system for the same reason.

5.1.2. MIP
Similar results were found for the Media Independent Partition strategy (Plot 3 and Plot 4). Again
we have the case where both the four media type system and the two media type system have the
same number of partitions for the media type with the maximum bandwidth. By adding two lower
bandwidth media types to the system, the requests for the maximum bandwidth objects decreases
from approximately 50% to 25%. The system performance is improved because the likelihood for
partition contention with the maximum bandwidth object is decreased in the four media type sys-
tem. Merging (MTM) also improves the performance of both the four media type system and two
media type system with this partitioning strategy.

5.1.3. MFP
The results for the Mixed Fragmentation Partitioning strategy for four media types is nearly iden-
tical to the case for two media types (Plot 5 and Plot 6). We think this is due to the fact that with
this partitioning strategy, all objects have the same degree of fragmentation as the media type with
the maximum fragmentation and the number of available disk partitions are identical in both
cases. With uniform access and distribution, the performance of the system is limited by the max-
imum media fragmentation. With merging we should expect the system performance to increase
since there are more possible merged objects with more media types of lower bandwidths. We ran
a simulation on MFP with MTM merging technique and found that the system performance did
improve slightly. We think this is due to the fact that the object type with the largest bandwidth
cannot merge with any other object type, since in our case it takes exactly 35 disks to satisfy its
bandwidth requirement. The rest of the objects can be merged but there is still disk contention
with objects with the maximum bandwidth since when they are being played no other objects can
be played from the same partition.
objects, since they were reinitialized in the beginning of each run. Therefore we replaced them with normal objects in an attempt to reduce any I/O they might incur. This modification turned out to require more changes to the code than we expected because we had to ensure that these two objects are passed as parameters to all the methods and functions which expected to retrieve them as global persistent objects. The second modification we made to speed up the simulator is to increase the period of each simulated clock tick from one millisecond to two millisecond. This should effectively speed up the simulator by a factor of two. The drawback is some loss in the accuracy of the results; for our simulation runs, however, the difference of one millisecond is not critical and is well worth the speed-up we gained. The last modification in this category was an attempt to change the type of the object used to simulate media strands. The original implementation used the Bitmap type in $O_2$ to simulate each frame of the video object. This also turned out to be unnecessary for this version of the simulator and we tried to replace it with dummy integers instead to try to improve the simulator performance. We finally decided to keep it as Bitmap after we spent much time debugging the modification and it became obvious to us that this would take longer to change than we had thought. Instead we made the bitmaps as small as possible to decrease the size of the objects.

5. Results
The results from the simulator runs are the overall system throughput, measured in number of media objects played per minute, and average response time, measured in milliseconds. The response time measures the time when a requests for an object is made to the time when the object begins to be played.

5.1. Experiment I
The system performance with four media types is in general similar to the system performance with two media types. For some of the partitioning strategies the performance is slightly better. We will go through each case and explain the results.

5.1.1. HiP
With the Hierarchical Partitioning strategy, the response time with four media types is slightly
cycles, the system first updates the displays of users currently being serviced, and removes requests for objects that have completed playing. It then proceeds through the request queue in FCFS order, beginning service of any requests for which there are available processors. This continues for 5 minutes of system service. The simulation ends by gathering and outputting statistics on the system throughput and total system response time.

4.3.2. Modifications

We have made a number of modifications to the simulator implementation. These include:

1. More flexible media type and distribution specification: In order to specify the exact bandwidth of the media types we want to store in the system, we modified the simulator to accept two lists of integers. The original implementation accepted the maximum and minimum bandwidths of the media types to be stored in the system and calculated the bandwidths of intermediate objects, based on the number of media types to be stored in the system. The intermediate bandwidths derived by the simulator are spaced evenly between the maximum and the minimum bandwidths specified. The modified simulator now accepts one list which specifies the bandwidth of each media type, and the other list specifies the number of objects of that media type to be stored in the system. This modification was the first one we made to the simulator and most of the time was spent on understanding the code and O₂’s development environment. We also spent some time verifying simulator results by comparing output from the modified version to that of the original simulator, using the same input parameters.

2. Multicasting: We also added the option to use multicast in the simulation. To implement this modification we went into the main scheduling loop and added code to one of the inner loops which scans for merged objects. Each time we serve a request in the queue we also serve any additional requests for that same object. Thus the object will be played out at the same time to all users who have requested it.

3. Simulator Performance: The next set of modifications attempt to increase the performance of the simulator itself. The original simulator took anywhere from 4 hours per run to 67 hours! As we looked at the simulator code we noticed that the simulator does not depend on the fact that the display objects (multimedia stations) and the media objects were implemented as persistent
tiple requests for that object from multiple users. Multicasting can be seen as a more general case of merging. In merging, retrieval of a single media object will incidentally lead to the retrieval of any other merged objects. If any of these merged objects happen to be requested, the retrieval of a single object will lead to the servicing of other requests. Given these similarities, we hoped to compare the gains of merging to that of multicasting. To minimize complications due to confounding variables, we hypothesized our results assuming both a uniform distribution and a uniform access pattern. We assumed that performance gains of multicast would be proportionate to the gains of merging, for a small number of users. As the number of users increased, the throughput of a multicast system should increase linearly, exceeding that of merging.

4.3. Simulator

The simulator was developed using O₂, an object-oriented database system. The development language is an O₂ specific extension of C++. The simulated system was a shared-nothing media server consisting of 70 processors, each having a dedicated 20 mb/s magnetic disk. The simulation covered everything from the scheduling of the user requests to the storage of media objects onto specific disk sectors. It simulates the system running for 5 minutes, whereby a specified number of users make requests for media objects, all of which run for 12 seconds. There was a command-line interface to the simulator that allowed you to input the following parameters: number of users, type of partitioning strategy, type of merging technique, number of media objects, number of media types, media type bandwidths, distribution skew and access skew.

4.3.1. Simulation

The flow of control in the simulator is rather straightforward. First the simulator creates the media objects and the media stations based on the user specified variables. Then based on the partitioning strategy it allocates certain processors for certain media objects. Using the appropriate merging technique it simulates the placement of all media objects on the appropriate disks. It is at this point that the actual simulated system begins to run. Initially, each user makes a request for a specific media object. If there are processors available to satisfy that request, the system begins to retrieve and deliver media data to that user’s display. If there are not available processors, the user request is placed on a queue. The simulator runs in cycles of one millisecond. In all subsequent
4. Experiments

For our class project we worked with Al Watkins’ simulator of the four partitioning strategies and two merging techniques. Al’s previous work included extensive comparisons of the performance of these four partitioning strategies, with and without merging.[2] His simulations were done using only two media types, assuming that results from all four partitioning strategies would be scalable to multiple media types. He also studied the effects of skewed access and distribution patterns of these two media types. His simulation was of a system that did not have the capability to multicast a single object to multiple displays. Our goal was to test his assumptions about scalability to multiple media types as well as compare the performance gains of multicast capability.

4.1. Experiment I

In the first part of our project we wanted to compare the results from [2] with simulations of additional media types. The media types used in the previous study[2] were an HDTV video type, with a bandwidth of 700 Mbps [1,2] and a motion JPEG compressed video type with a bandwidth of 1.5 Mbps[2]. We wanted to compare the results using two additional media types and look at their effects on the system performance. The two media types we added were a medical imaging object type (1280x1024 pixels per image using lossless compression) with a bandwidth of 350 Mbps, and CD quality audio type with a bandwidth of 705.6 Kbps. Because we are introducing two new media types with lower bandwidths than the HDTV object, we expect the overall throughput to increase. If the access and distribution of the objects are uniform across all media types, then the system should be able to serve more requests. On the other hand, introducing more media types may decrease the system performance for Media Independent Partitioning since there are more sets of partitions in the system thus increasing the likelihood of external fragmentation. More media types may also increase the degree of internal fragmentation for Hierarchical Partitioning, which would also decrease the system performance.

4.2. Experiment II

Multicast is the ability to serve separate streams from a single disk buffer to multiple display devices. This would mean that the retrieval of a single media object could be used to satisfy mul-
This eliminates the chances of both internal and external fragmentation and allows for the amount of replication to be based on object access patterns.

Mixed Fragmentation Partitioning (MFP) groups objects by media type and separates the processors into groups based on the media type with the highest degree of fragmentation. All objects of remaining media types are then replicated exactly once for each processor group. This approach is similar to HiP, but produces less replicas. Consider the example in Figure 3d. There are two media types with degrees of fragmentation five and two respectively. All ten processors are split into two processor groups. There are two replicas of the first media type one for each processor group. There are exactly two replicas of the second media type as well. Note that this partitioning strategy actually modifies the degree of fragmentation for all media types to be equal to that of the largest media type.

3. Merging Techniques

Merging techniques attempt to interleave media blocks of different objects across multiple disks. This would allow multiple objects to be retrieved from disks at the same time, without violating the playback requirements of any one object. A merging technique may alter a partitioning pattern, in order to improve the amount of merging, as long as this does not violate the playback requirement of an object and the buffer size is adjusted accordingly. We studied two merging techniques: Media Type Merging (MTM), which merges only objects of the same media type, and without modifying partitioning patterns and Mixed Fragmentation Merging (MFM), which merges objects of various media types and may modify the partitioning pattern.

Not all partitioning strategies make sense when combined with both merging techniques. Since MFM modifies the partitioning strategy, it can not be combined with MIP or HiP, both of which do not allow modifications to their partitioning pattern. MTM is too restrictive to be a useful merging technique for MFP, since there is only one object of each media type in a processor group. Since all processors, and therefore all disks, are dedicated to a specific media type in MET, attempts to use MFM in combination with MET will default to MTM with MET. This leaves for valid partitioning and merging combinations MIP_MTM, MIP_MTM, MET_MTM and MFP_MFM.
external fragmentation. This results in low processor utilization caused by internal fragmentation.

Media Type Partitioning (MET) dedicates groups of processors to specific media types. Each processor partition is dedicated objects of the same media type, as is illustrated in Figure 3c.
There are four different partitioning strategies as well as two merging techniques which we will consider. We will also discuss which merging techniques work best with the various partitioning strategies. It is assumed that these merging and partitioning strategies are employed by a shared-nothing media server made up of multiple processing nodes. Each node is attached to a dedicated disk or disk array. A set of nodes (partition) is dedicated to the retrieval of data to satisfy a given display request. It is also assumed that the system will be required to service multiple users concurrently. The server will need to store and retrieve various types of media objects.

2. Partitioning Strategies

Media Independent Partitioning (MIP) groups objects by media type. It takes all the objects of one type and allocates each object its necessary number of processors. It then duplicates each object the maximum number of times for all the remaining processors. It does this for all objects of a given type and then continues on in the same manner for all remaining media type groups. While this partitioning strategy provides the maximum number of replicas of each object, regardless of media type, it also results in the highest degree of interaction between objects of different media types. This interaction increases the chances of a specific type of disk contention, known as external fragmentation. Consider the example in Figure 3a. Here we have two media types with degrees of fragmentation five and three, respectively. An access to replica 2b only allocates three processors but invalidates access to both 1a and 1b, possibly leaving seven processors idle for the full playback of 2b.

Hierarchical Partitioning (HiP) groups objects by media type and sorts each group in descending order based on the degree of fragmentation. It first takes the media type with the highest degree of fragmentation and splits up all processors into groups based on its fragmentation level. All objects in this first media type group are fully replicated across all processor groups. All remaining objects are fully replicated within each processor group. This strategy eliminates external fragmentation, but introduces the notion of internal fragmentation, as illustrated in Figure 3b. Here we have three media types with degrees of fragmentation five, three and two, respectively. There are two processor groups, yielding two replicas of the first type, two of the second type and four of the third. Objects of media type three require only two disks, yet replicas 3b and 3d were given three processors each. This is due to the fact that an object can only be replicated within a processor group, thus avoiding
Sue Long and Will Luo

As shown in Figure 2, this would maximize disk utilization and minimize service delay due to disk contention. The trade-off of this would be an increased storage requirement. We will describe various replication or partitioning strategies in Section 2.

The actual disk blocks read in during the gap between data blocks of a media fragments may not necessarily contain useless data. It is possible to interleave blocks of one object in between gaps of another object. Since these gap blocks are going to be read anyway, if they are buffered and happen to be contain data required by another data stream during a given playback interval, they could be sent to another display device. This would allow the simultaneous request of two objects to be fulfilled, due to the retrieval of one media object. This technique of interleaving multiple media objects is known as merging.

Figure 1: (a) Non-uniform media object fragmentation. (b) Uniform media object fragmentation
onto a set of disks to be accessed in parallel. The object must be placed onto the separate disks in such a way as to guarantee that the overall rate of retrieval is equal to or greater than the playback rate of the display device. If the retrieval rate exceeds the playback rate, the display device must have an adequate amount of buffer available.

In most cases fragmentation will lead to some degree of wasted bandwidth. Consider the example in Figure 1a. The media object requires a bandwidth of 13 Mb/sec. Each disk has a bandwidth capacity of 4 Mb/sec. This would require fragmenting the object into four separate parts. If we fill each disk to its maximum capacity all disks will be fully utilized except the last disk, in which only 1 Mb of its bandwidth will be used and the rest will be wasted. A more desirable approach is uniform fragmentation where the wasted bandwidth is uniformly distributed over all the disks. This can be seen in the example shown in Figure 1b. The wasted bandwidth on each disk allows us to insert gaps between data blocks in a fragment. Assuming a uniform data distribution, the more we fragment an object, the larger the gap size can be without violating the media object’s playback requirement, but the more bandwidth we will waste.

It is assumed that a set or a partition of disks are dedicated to a requested object for the duration of its playback. If multiple objects span over a given disk, this could lead to disk contention. Previous studies have shown the most effective way to deal with disk contention is to replicate

---

**Figure 2**: Object x and y are both fragmented with a degree of two and each fragment occupies a disk. When either object is retrieved both fragments must be accessed at the same time.
The Effect of Multicasting and Multiple Media Object Types on Parallel and Merging Storage Strategies

by
Will Luo  Sue Long
spine@cc.gatech.edu  suelong@cc.gatech.edu

Abstract
This paper examines the effect of multicasting and storing multiple media object types on a storage system. We have extended Watkins and Omiecinski’s study[2] on the performance of continuous media storage strategies using parallelism with merging, by extending the simulator created by Watkins to include multicasting capability as well as accepting a general list of media object types to be stored in the system. We hypothesized that adding more media types would affect external and internal degrees of fragmentation and multicasting would affect the average wait queue size. We then designed and ran a number of experiments using additional media types as well as multicasting and compared our results with results from previous experiments.

1. Introduction
Recent technological advances have made it possible to store, retrieve and manipulate multimedia data, allowing for the development of a wide variety of applications including Video-On-Demand systems, Medical Imaging, On-Line Encyclopedia and Teleconferencing. Since there have been significant advances in processing power, network capacities and compression techniques, it has become increasing obvious that the bottle neck in these multimedia systems is the retrieval of media objects from the storage system due to a lack of adequate bandwidth. The bandwidth of a single hard drive is limited by current magnetic disk technology. Assuming this is a relatively fixed limit, the use of multiple disks in parallel is a viable technique to increase overall system bandwidth.

The use of parallelism creates the illusion of increased disk bandwidth by accessing a set of disks simultaneously and presenting data to the display device immediately as they are read off the disks. There are many issues involved in the decoupling or fragmentation of a media object