Evaluating the Effect of Online Data Compression on the Disk Cache of a Mass Storage System

Odysseas I. Pentakalos and Yelena Yesha
Computer Science Department
University of Maryland Baltimore County
Baltimore, Maryland 21228
and
Center of Excellence in Space Data and Information Sciences
Goddard Space Flight Center
Greenbelt, Maryland 20771

A trace driven simulation of the disk cache of a mass storage system was used to evaluate the effect of an online compression algorithm on various performance measures. Traces from the system at NASA's Center for Computational Sciences were used to run the simulation and disk cache hit ratios, number of files and bytes migrating to tertiary storage were measured. The measurements were performed for both an LRU and a size based migration algorithm. In addition to seeing the effect of online data compression on the disk cache performance measure, the simulation provided insight into thecharacteristics of the interactive references, suggesting that hint based prefetching algorithms are the only alternative for any future improvements to the disk cache hit ratio.

I. Introduction

Mass storage systems are used in research environments for storing data generated by scientific simulations and satellite observations in amounts on the order of terabytes. The cost of storage devices of that capacity is still very high while the rate of increase in disk space requirements by the users grows continuously. This problem is especially evident in scientific research centers where enormous amounts of data are generated on a daily basis which must be archived so that they can be analyzed at a later time [1],[2].

In this study the actual system under consideration is the Unitree Mass Storage System (UMSS) used at NASA's Center for Computational Sciences (NCCS). The system administrators are experiencing a situation where they constantly need to purchase additional storage devices which are filled to capacity in a decreasing amount of time. The main resource whose utilization must be optimized in this case is storage capacity. Removing the redundancy in the data stored in the file system, by inserting an online compression/decompression module, is one method of increasing the effective capacity of the system without the addition of expensive hardware devices.

After considering various alternative locations in the system at which the compression algorithm could be placed we determined that the user interface would be the best choice. Some of the advantages of placing compression at the user interface are: a) does not impose an additional load on the storage servers CPU, b) reduces the amount of data that flows

through the network, and c) does not require modifications to the Unitree code.

To evaluate the performance of compression on the specific data stored at NCCS, the ftp clients were modified to implement Ziv-Lempel and LZW compression transparently [3],[4],[5]. Sequential and pipelined implementations were tested against two sets of files and the performance of each implementation was compared based on file compression ratio and compression rate. An earlier paper describes the implementations and the results in detail [6].

In this study we examine the effect of compression on the disk cache of the mass storage system. A simulation is used to determine the effect of compressing data on the hit-ratio of the disk cache, the number of migrations of files from the disk cache to robotic storage, and the total number of bytes migrating to robotic storage. We also look at two different migration algorithms and their effect on the hit ratio and the file migrations.

Section II gives a description of the system under consideration and reviews terminology that will be used throughout the rest of the paper. Section III describes the simulation used in this study. Section IV describes the simulations performed and analyzes the results. Section V concludes the paper and discusses future work.

II. System Overview

The UMSS is a hierarchical mass storage management system which runs as a centralized application program on top of the Unix operating system and manages a hierarchical mass storage file system. The specific installation offers three levels in the storage hierarchy. Figure 1 shows the typical storage pyramid provided by most hierarchical mass storage systems.At the higher level it provides a disk array, with a total capacity of 150 Gbs, which serves mainly as a cache for the lower levels. The second level has a capacity of 4.8 terabytes provided by four near-line robotic tape storage units. The third level is the off-line storage vault which has the slowest transfer rate serving as the long-term repository.

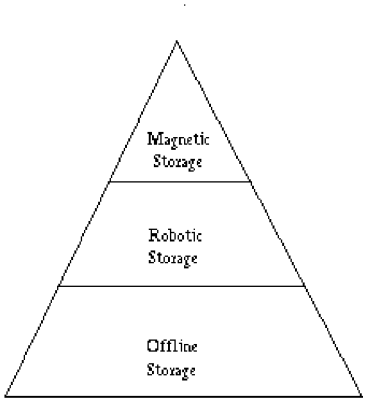


Fig.1 Hierachical Storage Pyramid

Users access files stored in the UMSS using the ftp protocol from their local workstations via a local area network. In addition to the ftp protocol, UMSS also provides an NFS interface to the file system but due to performance and security reasons the NFS protocol is not used by many installations including the one at NCCS. The UMSS was designed in a modular fashion in order to make possible its distribution over multiple host machines. Figure 2 shows a block diagram of the UMSS components [7].

Each of the components shown in figure 2 is represented by one or more independent daemon processes and is responsible for certain tasks.

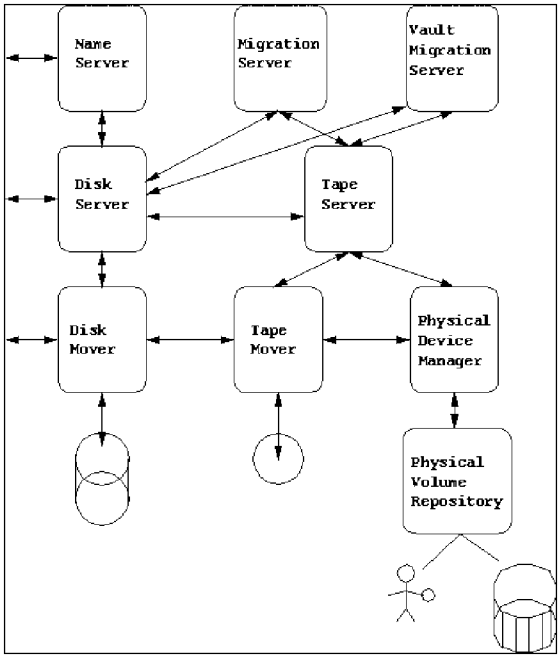


Fig 2. UMSS Block Diagram

The ‘‘Name Server’’ resolves string file names used by the users, into unique integer identifiers, used internally by all the other components of the UMSS. The ‘‘Disk Server’’ keeps track of the files stored in the disk cache, providing the view of a Unix file system to the user. The ‘‘Disk Mover’’ is responsible for all transfers to and from the disk cache. The ‘‘Migration Server’’ controls the migration of files from the disk cache to lower levels in the disk hierarchy to ensure that the disk cache always has sufficient free space to operate efficiently. The ‘‘Tape Server’’ keeps track of the files stored in the tape storage units whether online or off-line. The ‘‘Tape Mover’’ performs all file transfers to and from a tape device. The physical device manager is responsible for managing the tape mounts, scheduling them in an order which maximizes the utilization of the system resources. Finally, the ‘‘Physical Volume Repository’’ is responsible for mounting and dismounting both automated online and off-line storage physical volumes [8]. Any files retrieved from the UMSS are first placed in the disk cache, if they are not already there, and then are transferred to the user. Likewise, any files stored into the UMSS are first stored in the disk cache and then they are moved to a lower level of the hierarchy through migration.

In an earlier paper we investigated the effectiveness of an online data compression algorithm placed at the user interface of a mass storage system [6]. For a sequential implementation the following inequality describes the trade-off in time of compressing the data online.

$$\frac{S}{R_t} > \frac{S}{R_c} + \frac{S(1-r_c)}{R_t}$$

$$\frac{1}{R_t} > \frac{1}{R_c} + \frac{1-r_c}{R_t} \quad (1)$$

$$R_t < r_c R_c$$

where S is the size of the file, R_t is the file transfer rate, R_c is the compression rate and r_c is the compression ratio normalized to the range [0,1]. The left hand side is the time it takes to transfer the file without compression and the left side with compression. If the compression rate of the compression algorithm used is faster than the transfer rate of the network between the client and the server then the embedded compression increases the effective capacity of the storage server at no additional cost. Note that by cost here we mean the amount of time it takes to store a file into the mass storage system. If this inequality does not hold, the online compression algorithm increases the effective capacity of the system at the expense of added time when storing the file. The above inequality applies only to the sequential implementation. Assuming that the communication time between the parent and child processes is negligible we can derive a similar relation for the

pipelined implementation as shown in inequality 2.

$$\frac{S}{R_t} > \max\{\frac{S}{R_c}, \frac{S(1-r_c)}{R_t}\} \qquad (2)$$

The total time of the pipeline is bounded by the maximum of each of its components. Which of the two components prevails will depend on the particular client making the request and on the network topology. If the client is connected locally relative to the server but is a slow machine then the compression component will prevail whereas on a fast machine which is a few hops from the server the transmission component will prevail.

III. Disk Cache Simulation

A trace-driven simulation of the disk cache was used to ascertain the effect on the hit ratio and on the migration of files caused by file compression and migration algorithm. A discrete event simulator was developed using the ftp request traces to drive the simulation.The disk cache size was varied from 150GB, which is the actual disk cache size at the NCCS site, to 250GB. Initially the cache was assumed to be empty. The disk cache was represented by a doubly linked list of structures which described each file entry. The information stored for each file were a unique file identifier, the file size, a timestamp of the time the file entered the disk cache, and an indicator of whether the file is stored in the disk cache or in the lower levels of the hierarchy.

Put requests were placed in the disk cache. If the file already resided in the cache or lower in the hierarchy the operation was processed as an update, ensuring that only one copy of the file existed in the entire mass storage system. For get requests, if the file existed in the disk cache then the request was considered a hit. If the file existed lower in the hierarchy it was staged in the disk cache. If the file requested did not exist in the hierarchy, it was processed as if it was in the lower levels of the hierarchy and a new entry was created for the file in the disk cache.

Migration in simulated time was performed using a high water mark as in the UCFM. If the amount of free space in the cache went below the high water mark of 75 the total disk cache capacity, files were migrated to the lower levels of the hierarchy to create more space. Two different migration algorithms were tested. The first one, was LRU based, selecting files to migrate which had resided in the cache the longest without being referenced. The second algorithm was based on the file size, migrating larger files first.

Since it would be impractical to collect the compression ratios for each of the files in the mass storage system each simulation run used a fixed compression ratio. The simulation was run for various compression ratios ranging from 0% to 60% compression.

IV. Results

The ftp interactive request logs for a period of three months were used to run the simulation. The total number of references in that three month period was approximately 106,000. The references from the first two months were used for bringing the disk cache to a warm state. Then the number of hits, the hit ratio, the number of files migrating to tertiary storage, and the total number of bytes migrated were measured for fixed values of compression ratio.The simulation was run

also for two different migration algorithms. The first migration algorithm, which selected files to migrate if they had resided on the disk cache the longest without being referenced, will be referred to as the *LRU based algorithm*. The second algorithm which selected files to migrate based on their file size will be referred to as the *Size based algorithm*. The hit ratio was computed as the number of hits per day over the number of get requests on that specific day.

One important observation that was made about the reference patterns used in this mass storage system was that the requests do not exhibit significant temporal locality. Users do not tend to re-use their files very frequently as in a typical file system. This implies that this specific mass storage system is used more as an archive than as a typical file system. Since the working set of the get request stream continuously changes, only low hit ratios are possible regardless of size increases to the disk cache.

In order to be able to compare the hit ratios measured with some sort of an optimal hit ratio we run the simulation on the same trace data setting the compression ratio to a value very close to zero. This allowed all the files to fit within the disk cache, imitating a disk cache of an enormous size, generating no migrations. This experiment was used to generate the optimal (OPT) disk cache hit ratios. The same method was used to compute the hit ratio of this cache as in the other cases. Table I summarizes the effect of compression on the number of hits for each of the experiments. The table is divided in three major column groups for each of the migration algorithms. The first column group shows the results for the LRU based migration algorithm, the second column group for the Size based migration algorithm, and the last column shows the results for the OPT disk cache. The first two column groups consist of three columns, one for each of three different compression ratios attempted. Comparing the results from the two migration algorithms against the results under OPT we see that the number of hits for both algorithms are very close to the optimal. Compression does not affect the hit ratio very much and this is because the disk cache is large enough to support the hits in the reference patterns. It should be noted that the LRU based algorithm exhibits the inclusion property as expected since the number of hits is non-decreasing with increases in the disk cache size. On the other hand, the size based algorithm in certain cases decreases with a larger effective disk cache size.

The hit ratios were also plotted in figure 3 for various compression ratios. The plot on the top shows the hit ratio variation with respect to the compression ratio for the size based migration algorithm and the bottom plot shows the variation for the LRU based migration algorithm. It is apparent from these figures that size based migration provides higher hit ratios than the LRU based algorithm. The variation in compression ratio does not have significant effect on the hit ratio and the reason for this is the same as discussed in the previous paragraph. This implies that adding additional disks to the disk cache will not have any effect on the hit ratio based on the references analyzed. Also any further effort in improving the hit ratio by varying the migration algorithm will not generate any significant improvement on the hit ratio. The only possible method of increasing the hit ratio would be to develop a prefetching

	LRU Based			Size Based			
r_c	0.0	0.2	0.4	0.0	0.2	0.4	OPT
1	285	285	286	285	286	286	286
2	87	87	87	104	104	104	105
3	186	186	186	186	186	186	202
4	342	342	342	343	343	343	352
5	235	241	242	435	435	435	493
6	1086	1087	1088	1089	1088	1087	1130
7	1323	1323	1323	1500	1500	1500	1698
8	143	143	143	145	145	145	153
9	60	60	60	63	63	61	63
10	250	250	250	248	248	248	252
11	321	321	321	317	317	318	324
12	422	422	422	434	434	434	464
13	371	371	371	354	355	355	409
14	376	381	381	376	376	377	436
15	1249	1249	1251	1244	1243	1244	1256

algorithm that is based on hints provided by the user.

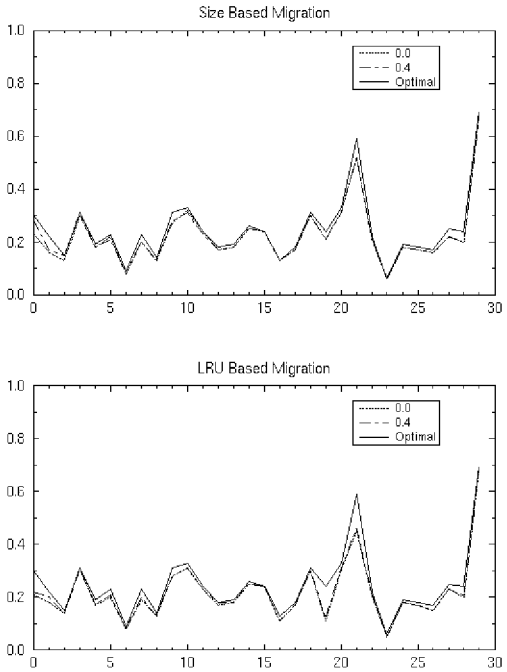


Fig. 3. Hit-Ratio versus Compression Ratio

The second part of the simulation analysis focused on the migrations. Since migration involves the use of tape drives from the robotic silos it is an expensive operation. Thus, reducing the number of migrations or the total number of bytes migrating to the tape will improve the mass storage system's performance. Figure 4 shows the number of files migrating versus compression ratio for the two migration algorithms. The LRU based algorithm maintains a consistent number of migrations and tends to smooth the migration operations over time. It appears that the effect of file compression is minimal. Looking at the peaks in the LRU based algorithm it appears that compression simply shifts the migration effects but does not reduce their number. The size based migration algorithm decreases significantly the number of migrations but it has the negative effect of generating on certain days tremendous migration traffic. Analyzing the file sizes for both get and put requests we found that the mean file size of files stored in the storage system is an order of magnitude larger than the mean file size of files

retrieved. Since the size based algorithm removes larger files first, eventually it runs out of large files and it has to remove a huge number of small files to free space in the disk cache.

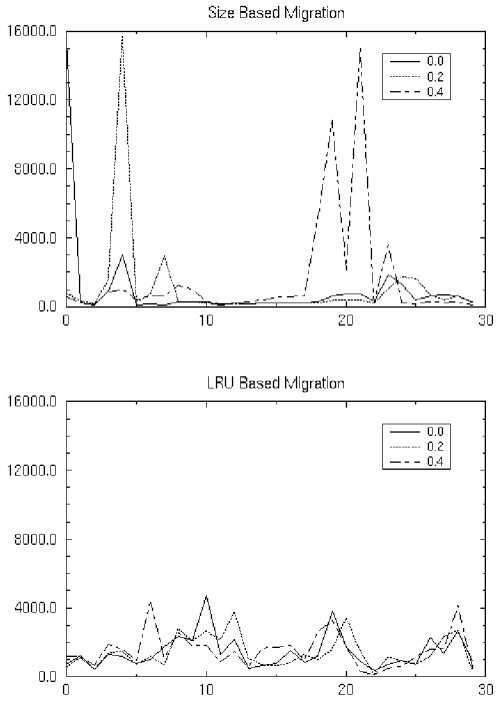
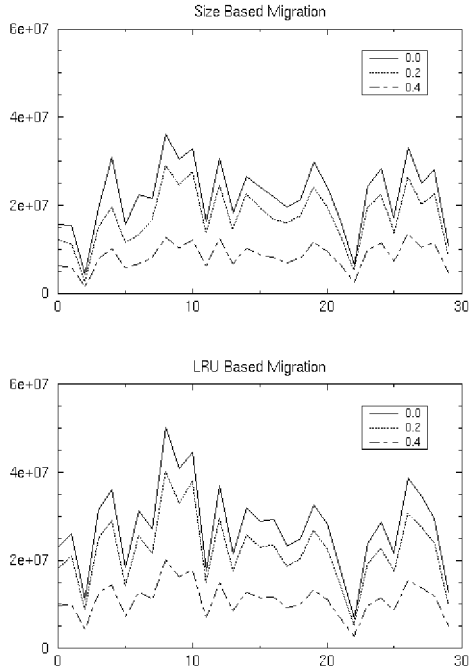


Fig. 4. Number of Migrations versus Compression Ratio

Figure 5 shows the number of bytes migrating to robotic storage for various compression ratios. It is apparent that for both migration algorithms the higher compression ratio provides significant reduction in the number of bytes that need to migrate. The size based migration algorithm provides better performance throughout the simulation period. The time it takes the system to process a migration involves an overhead time and a data transfer time. The overhead time consists of mounting the tape on a tape drive, a seek time to place the tape drive heads at the proper location, a rewind time after the data have been written, and an unmount time. Reducing the number of migrations from the disk cache affects the overhead time while reducing the number of bytes migrating to robotic storage reduces the data transfer time.



V. Conclusion

We evaluated the performance of an online compression algorithm on the disk cache of a mass storage system. A trace driven simulation of the disk cache was used for the evaluation. The traces used to drive the simulator were collected from the ftp logs of the system. The simulation was configured to match the disk space and migration algorithm of the system at NCCS. The effect of compression was simulated by uniformly reducing the file size of the get and put requests. Various compression ratios were used in the simulation. The simulation also evaluated two different migration algorithms, specifically an LRU based and a size based algorithm.

One important observation that was made about the references at this mass storage

system was that the working set continuously changes. This implies that the disk cache hit ratio cannot be improved significantly by increasing the disk cache size since get operations are usually to files that were stored in the mass storage system a very long time in the past. This effect was evident by comparing the two migration algorithms against a disk cache which was large enough to store all files stored during the three month evaluation period. As a result both algorithms attained hit ratios very close to the optimal hit ratios of the huge cache. Comparing the two migration algorithms we found that the size based algorithm decreases the total number of bytes migrating to tertiary storage at the expense of causing occasional peaks in the number of files migrating. Both algorithms were not affected by the compression ratio due to the fact that the disk cache is of large enough size to cover the intereference pattern of the requests.

Future work will focus on evaluating various prefetching algorithms.The current simulation suggested that only the use of user hints and an appropriate prefetching algorithm can improve the hit ratio of this system. The use of transparent informed prefetching could be applied to improve the hit ratio of the disk cache by exploiting application level hints about future file accesses [9]. Another area of future research is the implementation and evaluation of migration algorithms based on a combination of file size and cache residency time as described in [10],[11]. This simulation analysis showed that size based migration reduces the number of bytes that migrate to tertiary storage but occasionally it produces a large number of migration loads. By using a migration algorithm based on the space time product we expect that the migration peaks will disappear, while maintaining the lower number of bytes migrating.

Acknowledgements

We would like to thank Adina Tarshish, Ellen Salmon and George Rumney from NASA's Center for Computational Sciences at Goddard Space Flight Center for providing the ftp traces and the NCCS file set used for testing our ideas.

REFERENCES

[1] Randy H. Katz, Thomas E. Anderson, John K. Ousterhout, and David A. Patterson, ''Robo-line Storage: Low Latency, High Capacity Storage Systems over Geographically Distributed Networks'', Tech. Rep. UCB/S2K-91-3, University of California, Berkeley, March 1994.

[2] Ethan L. Miller and Randy H. Katz, ''An Analysis of File Migration in a Unix Supercomputing Environment'',Tech. Rep. UCB/CSD-92-712, University of California, Berkeley, March 1993.

[3] J. Ziv and A. Lempel, ''A Universal Algorithm for Sequential Data Compression'', IEEE Transactions on Information Theory, vol. 23, no. 3, pp. 337-343, 1977.

[4] Debra A. Lelewer and Daniel S. Hirschberg, ''Data Compression'', ACM Computing Surveys, vol. 19, no. 3, pp. 261-296, September 1987.

[5] Terry A. Welch, ''A Technique for High-Performance Data Compression'', IEEE Computer, vol. 17, no. 6, pp. 8-19, June 1984.

[6] Odysseas I. Pentakalos and Yelena Yesha, ''Online Data Compression for

Mass Storage File Systems’’, Tech. Rep. TR-CS-95-05, University of Maryland Baltimore County, July 1994.

[7] Adina Tarshish and Ellen Salmon,‘‘The Growth of the Unitree Mass Storage System at the NASA Center for Computational Sciences’’, 3rd NASA GSFC Conference on Mass Storage Systems and Technologies, College Park, Maryland, October 1993, pp. 19-21.

[8] Convex Computer Corporation, Unitree++ System Administration Guide, First Edition, Convex Press, Richardson, Texas, 1993.

[9] Hugo R. Patterson, Garth A. Gibson, and M. Satyanarayanan, ‘‘A Status Report on Research in Transparent Informed Prefetching’’, Operating Systems Review, vol. 27, no. 2, pp. 21-34, April 1993.

[10] Alan Jay Smith, ‘‘Analysis of Long Term File Reference Patterns for Application to File Migration Algorithms’’, IEEE Transactions on Software Engineering, vol. SE-7, no. 4, pp. 403-417, July 1981.

[11] Alan Jay Smith, ‘‘Long Term File Migration: Development and Evaluation of Algorithms’’, Communications of the ACM, vol. 24, no. 8, pp. 521-532, August 1981.