

Evaluation of caching strategies for an internet server *

A. L. Narasimha Reddy
Texas A & M University
214 Zachry
College Station, TX 77843-3128
reddy@ee.tamu.edu

Abstract

In this paper, we look at a number of issues in organizing a file system cache in an internet multimedia server. Traces from NCSA World-Wide-Web (WWW) server are used in this study. We study several caching strategies for improving the overall performance of such a server and show that request response time can be improved by some of the replacement policies that take size of the request into account. The new policies perform better as the video/audio content increases in the file system workload. We also show that previously proposed cache sharing strategies improve performance significantly in these multimedia servers.

1 Introduction

With the popularity of WWW servers, many organizations are trying to build servers that can handle the increased data traffic. Currently, most of these servers are big traditional file servers. Many sites employ distributed file systems across multiple server machines and distribute the requests across these machines to support the request load [1]. These servers typically employ large amounts of memory as cache space for supporting the high request rates on such systems. It is essential that the available memory be used as efficiently as possible to reduce the traffic to disks and also to improve the response time to the user. In this paper, we study several strategies for improving the performance of caching in such systems.

Studies on earlier file system request behavior [2] have motivated the design of later file systems [3], specially the caching algorithms and data organization on disks. Most of the earlier studies included none or very few applications that retrieved video or audio data. New forms of data such as audio and video are being used more widely now. Also, the current internet servers support wide use of images in the form of icons. It is not clear if the traditional caching strategies would work as well in the new systems where images, audio and video accesses are significant. In this paper, we study cache organizations of a

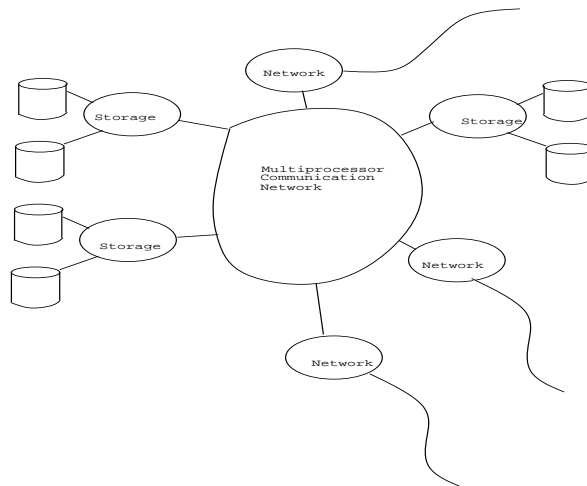


Fig. 1. System model of a multimedia server.

multimedia server that supports text, images, audio and video.

A multimedia server organization is shown in Fig. 1. The server has two sets of nodes, *storage nodes* and *network nodes*. Magnetic disks are attached to the storage nodes and users are connected to the server through network nodes. Such organizations are employed at NCSA [1], in many videosevers, for example [4, 5]. The separation of functions into network and storage nodes, either physically or logically, has a number of benefits: (i) easier management of storage and network bandwidth requirements and (ii) easier adaptability of the system to different network interfaces.

Network nodes and storage nodes both may employ caches for storing recently accessed blocks. If the functions of a network node and a storage node are physically separated, data may be cached at a storage node and at several network nodes at the same time. Caches are normally managed locally at a node. If a requested block is missing at the network node cache where the request originates, that request is routed to the storage node of the requested block. What is the best way to manage the different caches in such a server?

*This work is supported in part by an NSF Career Award and by a grant from State of Texas Higher Education Board.

LRU is used widely as a replacement policy. LRU policy treats every cache block equally and replaces least recently used blocks to make room for new blocks. LRU doesn't make a distinction between data blocks based on their data type (text, images etc.) or based on the size of the request. It may be possible to exploit the information about data blocks, such as the size of the request, the type of data, and the access behavior of different data types, to improve the request response time. It may also be possible to improve the overall response time by not caching the larger requests. We explore these possibilities by examining a number of replacement policies that take the size of the request or the data type into account. We will describe these policies in detail in a later section.

Cooperative caching [6, 7], where requests can be satisfied by caches of other clients on the network as well as the server caches is shown to be quite effective in improving the block response time. The cache organization in such systems resembles that of the organization of a multimedia server as described above. The client caches correspond to network node caches and the server caches correspond to the storage node caches.

In [6, 7], several strategies for sharing data across clients caches are discussed and evaluated. We will use greedy-forwarding in this study as an example technique that uses such sharing. Greedy forwarding allows a client's request to be satisfied by either the cache at the server or by a cache at another client, on a miss at its own cache. Among the many policies discussed in [6, 7], greedy-forwarding is fair to individual clients while improving the performance on the whole.

With some new interconnect technologies such as FCS (Fibre Channel), it is possible to build a server by attaching disks directly to the interconnect (as opposed to being connected to a node that is attached to the interconnect) as shown in Fig. 2. In such systems, the storage node is simply a disk and the storage node cache is the 1 or 2 Mbytes of memory used as a cache on the disk. Since the storage node (in this case, disk) cache is small, it is expected that the size and the utilization of the network node caches would determine the performance of caching. It is likely that cooperative caching (by sharing the network node caches) can improve performance substantially over traditional caching techniques in such systems. We explore how various policies do in such systems where the storage node caches are small in size.

We will use traces obtained from NCSA's web server to evaluate several cache replacement policies. We will also evaluate the cache sharing approaches for multimedia workloads.

The rest of the paper is organized as follows. Section 2 discusses the various caching algorithms considered and how they were implemented in the simulator. Section 3 discusses the traces that we collected and how they are used in the simulations. Section 4

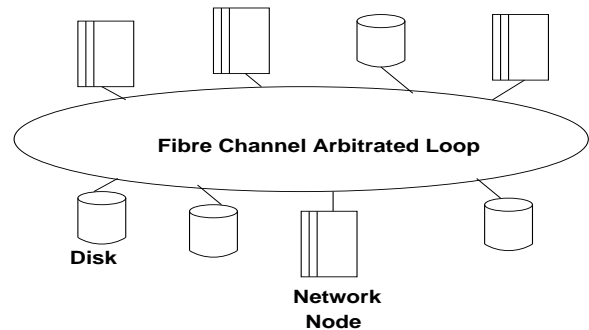


Fig. 2. An FCS-based server.

presents the results of simulations. Section 5 presents a discussion of the results and Section 6 concludes the paper with directions for future research.

2 Caching algorithms

LRU is widely used as a replacement policy. It is well understood and shown to perform well in a wide variety of workloads. LRU replaces the least recently used block in the cache to make room for a new block. In this section, we consider algorithms that take size of the request and the data type into account.

In the first algorithm we consider, text and image data are managed by an LRU policy and audio and video are always left at the LRU end of an LRU chain, i.e., audio and video are managed by an MRU policy. Audio and Video data tend to be large objects and by using an MRU policy to handle them, we expect to limit the impact on other cache contents. Using MRU policy to handle audio and video is equivalent to stealing some of the cache space for buffering the data when audio or video data is accessed. This buffer space is returned to the cache if the audio/video data is not immediately accessed again. We call this policy AVI-MRU.

The second algorithm we consider is based on the size of the request. Any request larger than a certain preset threshold (32 KBytes in our simulations) is managed by an MRU policy. Requests smaller than this threshold are managed by an LRU policy. All the requests share the same LRU chain. But blocks of the larger requests, after an access, are placed at the LRU end of the LRU chain. Again, it is expected that this policy would limit larger requests from evicting too many smaller requests from the cache. We call this policy SIZE-MRU.

The third policy we consider is a variation of SPACExAGE algorithm. SpacexAge algorithm is shown to be quite effective in minimizing misses in a mass storage system [8, 9]. In these systems, the product of size and age (since last use) of the file is used in determining the candidates for migration to tertiary storage. The variation we considered maintains a number of LRU chains instead of the usual

single chain. The chains are partitioned based on request size. In our implementation, requests with sizes up to 2 blocks were put on chain 1, requests between 3 and 8 blocks on chain 2, requests between 9 and 16 blocks on chain 3, and requests above 16 blocks were put on chain 4. A block is 4 KB. Each of these chains are managed by an LRU policy. When a block needs to be replaced from the cache, the space X age product of the four LRU blocks (on the four different chains) are evaluated and the block with the largest space X age product is replaced. It is to be noted that the cache space is not partitioned based on sizes and the cache space can be used by requests of any size. However, larger requests tend to occupy cache space for less time than smaller requests. We call this policy SpacexAge.

At this point, it is to be pointed out that in current systems, information about the size of the request or the type of data is not part of the request header passed along to the file (or I/O) system caches. We study these replacement policies to see if the request performance can be improved if such additional information were available at the file system cache.

When a request arrives at a network node, the cache at that network node is checked. If the requested block is found, it is returned to the user. If it misses at that network node, the request is forwarded to the storage node. If the block is found at the storage node, the block is copied to the network node and forwarded to the user. If the block misses at the storage node also, it is read from the disk system, copied to the storage node cache and the network node cache and then forwarded to the user.

In addition to the evaluation of the above replacement policies, we also evaluated the cooperative caching policy called greedy forwarding. In this policy, if a request misses at a network node cache, it is forwarded to the storage node cache. If the request results in a miss at the storage node as well, it is forwarded to one of the network node caches if it is cached at any other network node. That network node cache then satisfies the request and the requested block will be cached at the originating node. This is somewhat similar to how caches are managed in cache-coherent shared memory machines [10] and for further details please see [6, 7]. If the requested block is not present at any network node, then it is accessed from the disk.

In this paper, we evaluate the following caching strategies: (i) different replacement policies (compared to LRU), (ii) cooperative sharing and (iii) combination of (i) and (ii).

3 Simulations

We simulated a storage system with a number of network nodes and a number of storage nodes. For the results reported here, the number of network nodes is fixed at 7. The traces of request activity on the NCSA's server are collected in a different file

Table 1. Service times for different accesses.

Access	Latency ms	Per-block Transf. time ms
Network node hit	0	0.125
Storage node hit	0.65	0.210
Storage node miss	14.80	0.730

at each of the 7 network nodes of NCSA's machine. It is possible to pool these requests together and randomly distribute them over a different number of network nodes (to vary the number of network nodes). However, the cache behavior is closely tied to the order in which these requests arrive at the network nodes. For this reason, we kept the number of network nodes constant at 7 to minimize the impact of our assumptions on the results. We varied the number of storage nodes from 1 to 2 to 4. Each storage node is assumed to be responsible for serving the requests for a random set of files on the machine. The number of storage nodes does not have any impact on the relative order of arrival of requests at the server. The size of a cache at network nodes and storage nodes is varied from 16 Mbytes to 256 Mbytes.

In our simulations, the data objects/files are equally distributed among the storage nodes at random. The file distribution among the nodes is determined at the beginning of the simulations and is fixed across all the simulations for that configuration of nodes. Entire files are stored at a single storage node in our simulations. Disk queuing times at storage nodes are not considered in this study. This assumption is made such that the effectiveness of caching strategies can be studied without being impacted by the disk queuing times.

Table 1 lists the costs of reading a 4 KB block from memory, across the network and from the disk. These numbers are obtained from [6]. The first column indicates the time for the first byte of the first block to be available. These numbers are based on the assumption that the latency of transfer across the server's interconnection network is 650 us and the latency of disk access (on an average) is 14.8 ms. Transfer times per block (second column in the table) are obtained by assuming that the network can transfer data at 19 MB/sec (152 Mbit/sec ATM) and that the disk bandwidth is about 5.5 MB/sec.

3.1 Traces

NCSA records every access made to its web server using a httpd daemon [1]. For details about the trace collection, please see [1]. We obtained traces of the NCSA server for the week of 17th June to 21st June. Each record in this trace contains: the name of the accessed object/file, time of access, the size of the object along with other information. From the description of the objects, we can deduce information about the type of data being accessed whether it is text, images, audio or video. Occasionally, the WWW

Table 2. Statistics about file accesses.

Type of request	# of req.	# of blocks	Avg. Req. size in blocks	# of files accessed
Text	390,329	1,645,016	4.21	14,361
Images	544,389	1,247,139	2.29	8,137
Audio	703	8,939	12.72	106
Video	749	89,040	118.88	90
Total	936,170	2,990,134	3.19	22,694

server may experience errors and these were thrown out of the trace. Similarly, aborted accesses to files were dropped from the trace. We used traces of three days of 18th, 19th and 20th of June as input to our simulator. Each record in our final trace has the following information: day and time of access, name/id of the object, size of the object, type of the object and the origin of the access. The origin of the access is the id of the network node where the request is received. The traces contain accesses to files and not individual blocks within a file. It is assumed that all the blocks within a file are accessed. All requests are read accesses.

The traces of the three days were combined into one long trace. The final trace had 936,170 requests for a total of 2.99 million block (4KB blocks) accesses. The statistics about different accesses are given in Table 2. Based on the number of requests, the current usage of audio and video over the internet is still minimal. However, these data types on an average access many more blocks per request than text or images. It is noted that there are more requests for images than for text, but the images access fewer data blocks. Frequent use of small icons on web pages is the main reason for this behavior. About 170,000 requests from the trace were used to warm up the caches and the rest of the trace was used to gather performance statistics.

3.2 Performance measures

File system caches and interfaces are organized such that requests are handled one block at a time. This provides a simple and uniform way of handling all the requests. When a request retrieves multiple blocks from disk, they need not be located contiguously in the cache. A block interface to disk makes it easier to pass a pointer to the cache block (rather than a list of pointers) where this data may be stored. On a miss, a block is requested at a time from the disk.

Now, let's consider what happens at the disk system. Disk subsystems typically maintain some amount of buffer space and retrieve more data than a single block at a time. Larger block accesses are more efficient than multiple smaller block accesses. Spatial locality in disk accesses and the seek and latency penalties to be paid for a disk access favor such larger

accesses.

Given current file system cache structures, number of block misses is important. However, given current disk subsystem organizations every block miss at the file system cache doesn't result in a seek and latency penalty at the disk subsystem. Block response time overestimates the response time.

It can be argued that the user is more concerned with the request response time than the block response time. Transaction response time is used as a measure in database systems than block response time based on similar arguments. We used both the measures of performance, request response time and block response time. For calculating the block response time, it is assumed that each missed block results in paying seek and latency overheads at the disk subsystem.

A request may consist of multiple blocks. Request response time is the time taken to service all the blocks in the request. Data organization on the disk will determine the number of seeks that may be required to satisfy a single request. If all the blocks of a given request are contiguous on the disk, a single seek may suffice. If the blocks are not contiguous, the number of seeks may equal the number of blocks requested. We used a parameterized model for measuring request response time. The parameter specified the number of blocks that can be fetched on an average with one seek (this would depend on the data organization on the disk and the request access behavior). This parameter is varied among 1, 2, 4, 8 and ∞ . For example, with a parameter of 4, a request for 9 blocks would pay seek and latency overheads three times. With a parameter of ∞ , every request pays seek and latency overhead only once. We call this parameter disk-layout parameter.

Table 3 gives the results of analyzing the layout of files on one of our local machines. These results are generated by using the 'fileplace' utility provided as part of IBM's AIX operating system. The fileplace utility reports various statistics about logical and physical placement of blocks of a file. The data shows that 63% of the files are single-block files, but a large percentage of the blocks, 92.4%, belong to multi-block files. The blocks of 75.8% of these multi-block files are allocated contiguously on the disk. The blocks of the remaining 24.2% of the multi-block files are not all allocated contiguously on the disk. On an average, these files had 7.755 blocks allocated contiguously on the disk. Nearly 80% of the space on this disk was allocated. The results depend on how much of the space is allocated. File systems are more successful at allocating space contiguously on the disk when the disk space is not highly utilized. With increasing practice of allocating disk space in larger units (for example 64K in IBM's Shark video server [4]), the blocks of larger files will be more contiguously allocated. We will assume a disk-layout factor of 4 later on as a conservative estimate.

Table 3. Disk placement statistics.

Characteristic	Value
% of single-block files	63.0%
% of Multiple block files	37.0%
% of blocks in multi-block files	92.4%
% of multi-block files allocated contiguously	75.8%
Average # of contiguous blocks in broken multi-block files	7.755

4 Results

In this section, we will present results obtained through simulations.

Fig. 3. shows the request response time (with a disk_layout parameter of ∞) as a function of storage node memory for various cache replacement policies. Among the policies considered, SpacexAge and SIZE_MRU policies have the least request response times at different sizes of storage node memory. For this set of experiments, the network node memory is fixed at 16MB. SpacexAge improves response time by 20% when compared to LRU policy. AVI-MRU exhibits slightly better performance than LRU. In our experiments, we found that the SIZE_MRU policy is very sensitive to the size threshold parameter. Based on these results, it seems that the request response time can be improved by policies that take size into account (SpacexAge and SIZE_MRU) when compared to LRU.

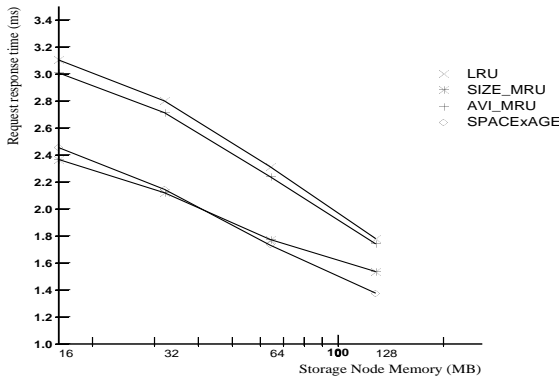


Fig. 3. Request response time for different replacement policies.

Fig. 4. shows the block response time with different replacement policies. Block response time is measured by assuming that each block that requires disk access pays the penalty of seek and latency at the disk system i.e., a disk-layout parameter of 1 is assumed. SpacexAge and LRU policies have the best block response time depending on the memory at the storage

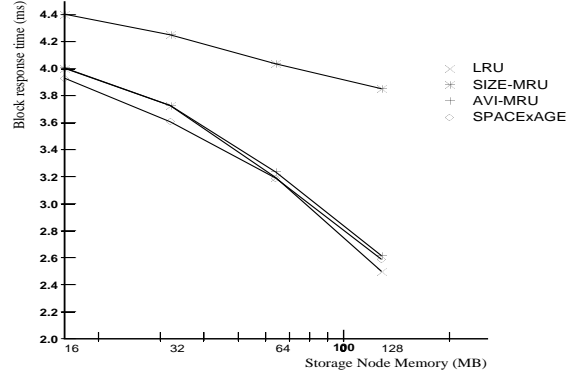


Fig. 4. Block response time for different replacement policies.

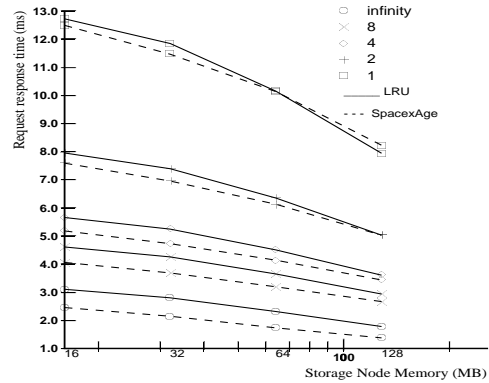


Fig. 5. Response times for different disk-layout parameters.

node. SIZE_MRU has the worst block response time. Again, for these experiments, we kept the network node memory fixed at 16MB.

From figures 3 and 4, SpacexAge policy has better response times than the other policies that take size of the request or data type into account. Hence, we will compare this policy with LRU for the rest of the paper.

Fig. 5. shows the request response times as a function of storage node memory and the disk-layout parameter. The response times increase when spatial locality decreases on the disk (with decreasing disk-layout parameter values). It is observed that LRU performs better than SpacexAge only when response time is considered with a disk-layout parameter of 1 (i.e., seek and latency overheads are paid for every block access). The trends in performance do not change significantly as the disk-layout parameter is changed even though the relative differences may change.

For the rest of the paper, we will use request response time with a disk-layout parameter of 4 as

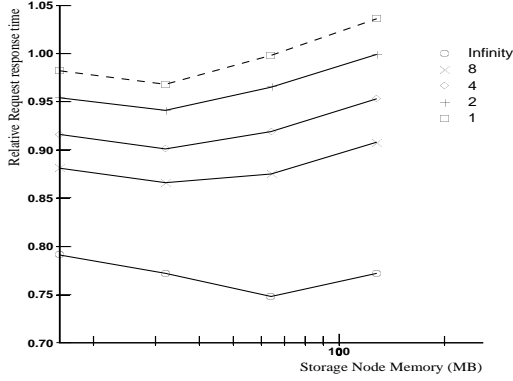


Fig. 6. Relative response times for different disk-layout parameters.

a performance measure. Fig. 6. presents the relative request response time of SpaceXAge policy when compared to LRU. It is observed that with a disk-layout parameter of 4, we get a fairly conservative estimate of the possible improvements of request response time by using SpaceXAge policy. Most file systems try to allocate space contiguously on the disk and the importance of such allocation has been recognized by a number of researchers, both in traditional file systems e.g., [3] and in multimedia systems e.g., [14]. Performance studies on multimedia systems have recommended employing large blocks (64K -256K) for allocation of video files [15].

4.1 Cooperative Caching

Fig. 7. shows the request response time with and without cooperative caching. It is observed that cooperative caching improves request response time considerably independent of the replacement policy. It is also noted that SpaceXAge algorithm continues to have better request response times than LRU even when cooperative caching is employed. As we increase the storage node memory from 16MB to 128 MB, keeping the network node memory fixed at 16MB, the impact of cooperative caching on request response time diminishes. A relatively larger storage node cache results in fewer chances for finding a block at another network node on a miss at the storage node. This is the reason for the diminishing returns. Cooperative sharing improves request response times by up to 35%.

Fig. 8. shows the impact of size of the network node cache on performance. For this set of experiments, the storage node memory is fixed at 128 MB. Larger the network node cache, higher the hit ratio at the network node for an arriving request. When cooperative caching is employed, the size of the network node cache improves performance more significantly. A larger network node cache implies higher chances of finding a block that misses at the

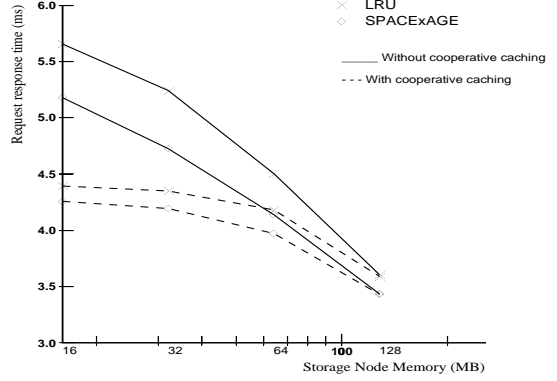


Fig. 7. Request response time with and without cooperative caching.

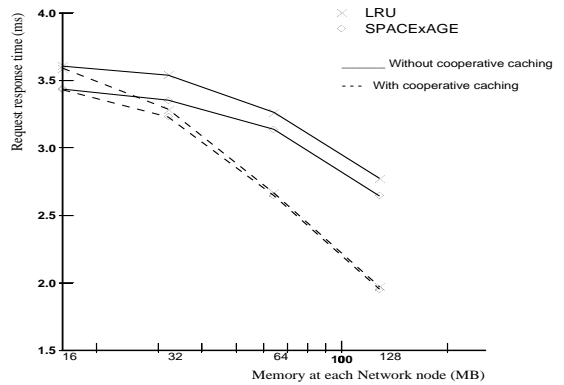


Fig. 8. Request response time as a function of network node cache size.

storage node cache. SpaceXAge continues to have better request response times than LRU. But the relative contribution of replacement policy diminishes as the size of the network node cache is increased.

From figures 3 and 8, it is observed that both cache replacement policy and cooperative caching can have a significant impact on performance (whether or not they are implemented together).

Fig. 9. shows the impact of organizing the storage node cache space over 1 or 2 or 4 or 8 nodes. For example, 64MB of total storage node cache space can be organized over 4 nodes with 16MB each or over 2 nodes with 32 MB each or over 1 node with 64MB (16MB is the minimum amount of memory we considered for a storage node). From the figure, it is seen that the impact of splitting the storage node cache space over several storage nodes is minimal. The amount of total storage node cache space determines the performance and the number of storage nodes only has a marginal impact.

As explained earlier, FCS interconnect based systems may have small amounts of cache memory at the

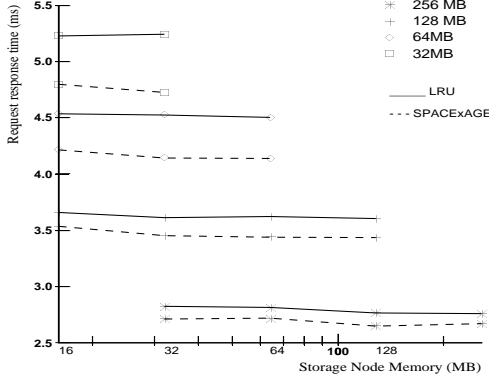


Fig. 9. Performance as a function of number of storage nodes.

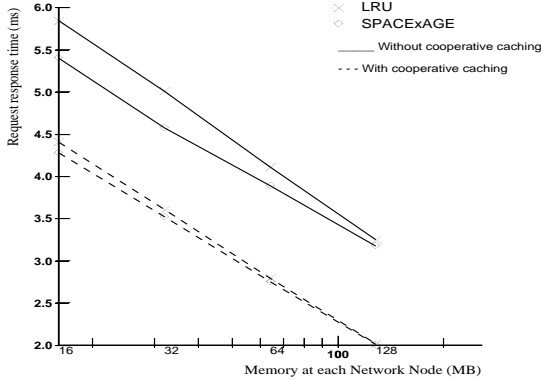


Fig. 10. Response times in an FCS-based system.

storage nodes. Fig. 10 shows the performance of the storage system with 1MB storage node caches. It is clear that cooperative caching is much more important in such systems because of insignificant caching at the storage nodes. Cooperative caching improves request response time by 25-40%.

4.2 Video enhanced workloads

To study the impact of increased video usage, we created video-enhanced workloads by modifying the traces used earlier. Some of the files in the trace were randomly marked as video files at the beginning of the simulation. Correspondingly, we increased the size of these requests to match the average size of a video request (120 blocks). By doing so, we have not altered the request behavior (order of arrival and frequency of access) but only the size (and type) of the requests. These results are expected to give an indication of how well the various policies perform as the video usage increases.

Results based on these video-enhanced workloads are presented in Fig. 11. We calculated the relative (block and request) misses of SpacexAge policy

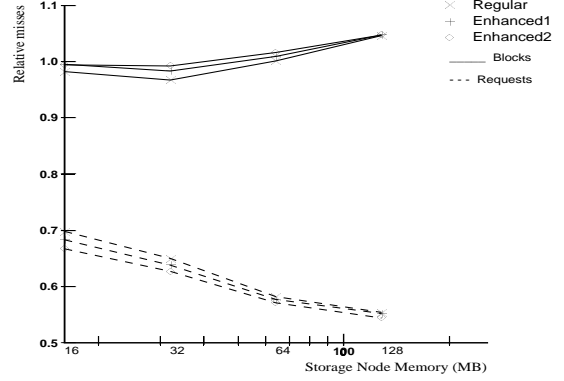


Fig. 11. Relative block and request misses.

compared to LRU policy. A request is considered to miss if any block of that request requires a disk access. The relative request misses are below 70% for all the workloads. Relative request miss rate decreases as we increase the number of video requests in the workload. Larger video requests evict many smaller requests from the cache. In LRU, these larger requests stay in the cache longer and hence impact the requests over a longer period of time. SpacexAge, by reducing the time larger requests spend in the cache, improves the cache utilization by the smaller requests. The more video requests, the more impact on cache contents and hence the better relative request miss rates of SpacexAge. The relative block miss rate is close to 1.00 (i.e., nearly same number of blocks are missed in both cases) and can be slightly higher than 1.0. The relative block miss rate however increases as the workload has more video requests. This is because of the fact that on an average the size of a missed request is larger in SpacexAge compared to LRU. This clearly shows that SpacexAge policy decreases the number of request misses and when coupled with an efficient disk allocation policy can deliver better response times.

5 Analysis of SpacexAge & LRU

In this section, we will present a simple analysis of SpacexAge and LRU performance. Given a request stream, each policy transforms the request stream into a different set of requests going to the disk system. Let's assume that the original request stream R is transformed into R_{lru} by LRU and R_{sa} by SpacexAge. In this modified request stream, each request can ask for multiple contiguous blocks on disk. Let n_{lru} , n_{sa} denote the average number of blocks accessed by a request in LRU and SpacexAge respectively. The access cost of LRU is then $= R_{lru}(seek + latency + n_{lru} * block_read_time)$. Similarly, for SpacexAge, total access cost $= R_{sa}(seek + latency + n_{sa} * block_read_time)$. For SpacexAge

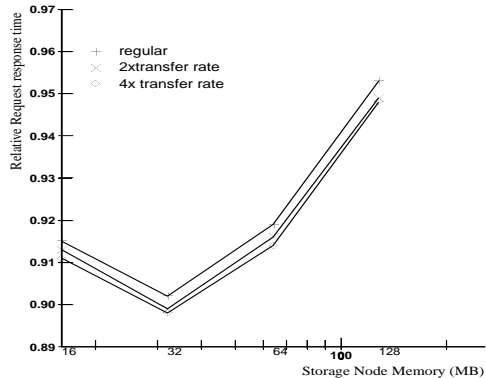


Fig. 12. Effect of increasing disk transfer rates.

to perform better than LRU, $R_{sa}(seek + latency + n_{sa} * block_read_time) \leq R_{lru}(seek + latency + n_{lru} * block_read_time)$. This implies $(R_{sa} * n_{sa} - R_{lru} * n_{lru}) \leq (R_{lru} - R_{sa}) * (seek + latency) / block_read_time$. But, $R_{policy} * n_{policy}$ is the total number of blocks accessed from the disk by that policy. Hence, the above equation implies that number of extra blocks retrieved by SpacexAge (compared to LRU) from disk \leq the number of extra requests generated by LRU * $(seek + latency \text{ overheads}) / block_read_time$.

The current trends in disk technology are such that the seek and latency overheads are decreasing at a slower rate than the rate at which the block read times are decreasing. This trend favors SpacexAge policy as the above equation indicates. To verify this, we ran a set of experiments where the disk transfer rate is improved by factors of 2 and 4. The results of trace-driven simulations are shown in Fig. 12. It is observed that the increasing transfer rates favor SpacexAge policy over LRU.

6 Conclusions and Future Work

We presented an evaluation of cache replacement policies that take account of the size of the request or the type of data being accessed. We showed that among the policies considered, SpacexAge algorithm performed the best. We showed that SpacexAge can have better request response times than LRU. SpacexAge is shown to minimize the number of request misses considerably. We also showed that as the video usage grows, SpacexAge does better at minimizing the number of request misses.

With efficient disk allocation strategies, this decreased request miss rate (with possibly higher block request rate) can translate into better response times. The new trends of higher video content, larger disk allocation units, and faster disk transfer rates all favor SpacexAge policy relative to LRU.

We presented a comparison of LRU and SpacexAge with or without cooperative sharing. Cooperative

sharing is shown to improve performance significantly for all replacement policies. Cooperative sharing has a significant impact on performance in FCS-based systems where disks are directly attached to the network. SpacexAge continues to perform better than LRU in all these cases.

In this study, we used the same cache management algorithms both at the storage nodes and network nodes. We plan on evaluating the impact of employing different cache management algorithms at storage nodes and network nodes.

Acknowledgments: Thanks to Robert McGrath and Randy Sharpe of NCSA for making the WWW traces available.

References

- [1] T. Kwan, R. McGrath, and D. Reed. NCSA's world wide web server: Design and performance. *IEEE Computer*, pages 68–74, Nov. 1995.
- [2] J. K. Ousterhout, H. DaCosta, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson. A trace-driven analysis of the unix 4.2 bsd file system. *Proc. 10th Symp. on Operating System Principles*, pages 15–24, Dec. 1985.
- [3] M. McKusick, W. Joy, S. Leffler, and R. Fabry. A fast file system for UNIX. *ACM Trans. on Comp. Systems*, pages 181–197, Aug. 1984.
- [4] R. Haskin. The shark continuous-media file server. *Proc. of IEEE COMPCON*, Feb. 1993.
- [5] Microsoft. The tiger video server. *Microsoft Press Release*, Apr. 1994.
- [6] M. Dahlin, R. Wang, T. Anderson, and D. Patterson. Cooperative caching: Using remote client memory to improve file system performance. *Proc. of 1st Symp. on Oper. Sys. Design and Implementation*, pages 267–280, Nov. 1994.
- [7] A. Leff, P. Yu, and J. Wolf. Policies for efficient memory utilization in a remote caching architecture. *Proc. of 1st Int. Conf. on Par. and Dist. Info. Sys.*, pages 198–207, Dec. 1991.
- [8] D. H. Lawrie, J. M. Randal, and R. R. Barton. Experiments with automatic file migration. *IEEE Computer magazine*, pages 45–55, July 1982.
- [9] A. J. Smith. Long term file migration: development and evaluation of algorithms. *Comm. of ACM*, pages 521–532, Aug. 1981.
- [10] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy. The directory-based cache coherence protocol for the DASH multiprocessor. *Proc. of 17th Ann. Symp. on Computer Architecture*, pages 148–159, May 1990.
- [11] E. Hagersten, A. LAndin, and S. Haridi. DDM - A cache-only memory architecture. *IEEE Computer*, pages 45–54, Sep. 1992.
- [12] P. M. Chen and D. Patterson. Maximizing performance in a striped disk array. *Proc. 17th Ann. Int. Symp. on Computer Architecture*, June 1990.
- [13] D. P. Anderson, Y. Osawa, and R. Govindan. Real-time disk storage and retrieval of digital audio/video data. *Tech. report UCB/CSD 91/646, Univ. of Cal., Berkeley*, Aug. 1991.
- [14] H. M. Vin and P. V. Rangan. Designing file systems for digital video and audio. *Proc. of 13th ACM Symp. on Oper. Sys. Principles*, 1991.
- [15] A. L. Narasimha Reddy and Jim Wyllie. Disk scheduling in a multimedia I/O system. *Proc. of ACM Multimedia Conf.*, Aug. 1992.