Cache Management Algorithms for Flexible Filesystems*

Silvano Maffeis
maffeis@ifi.unizh.ch
Computer Science Department University of Zurich
Winterthurerstr. 190, CH-8057 Zurich, Switzerland

IFI TR 93.03†

Abstract

Cache management in flexible filesystems deals with the problem of determining a cached file to be replaced when the local cachespace is exhausted. In analogy to virtual memory management, several different algorithms exist for managing cached files. In this paper we simulate the behavior of *First-In-First-Out (FIFO)*, *Least Recently Used (LRU)*, *Least Frequently Used (LFU)* and a variation of LFU we call the *File Length Algorithm (LEN)* from the viewpoint of file access times, cache hit ratios and availability. The results of several simulation runs are presented and interpreted.

Keywords: Cache Management, File Access, Flexible Filesystems, Replication

1 Introduction

For virtual storage management, Denning [4] developed a view of program paging activity called the *workingset model*. The workingset model is based on the assumption of *locality of reference*. The idea is to examine the most recent Δ page-references of a process $\mathcal{P}$. The set of pages accessed by the most recent Δ page-references is the *workingset* $\mathcal{W}_{\mathcal{P}}$ of process $\mathcal{P}$.

In this paper we simulate and analyze some well known *page replacement algorithms* [1] and the workingset model from the viewpoint of a flexible filesystem (FFS) [13]. In this context, we will use the term *FFS cache replacement algorithm (FFS-CRA)*. In FFS, client workstations are attached to several file servers from which they obtain replicas of the needed files. We will assume that clients always obtain a *full replica* and not only a part of a file. This in analogy to filesystems like ANDREW [12] or CODA [5] where only entire files are cached. In FFS problems similar to those encountered in virtual memory management arise, since the decision must be made which locally cached copy of a file to discard when the cache is full and an uncached file is referenced.

Replacement strategies and the workingset model are two well known research areas in the field of virtual memory management and have extensively been studied for many years. In conjunction with cache management in FFS, following novel aspects must be considered by a FFS-CRA:

Coarse granularity: Objects referenced in FFS may encompass few kilobytes to several megabytes of data. In contrast, virtual memory management systems handle much smaller objects, usually memory pages

*This work was supported by *Siemens AG, ZFE*, Germany and *Schweizer Bundesamt für Konjunkturfragen*, Grant No. 2255.1

†to appear in ACM SIGMETRICS Performance Evaluation Review, Vol. 21, No. 3

with sizes of a few kilobytes only. Within FFS, it is possible to realize replacement algorithms in software, since the resulting overhead is negligible compared to the time needed to transfer a file from a remote server to the client's cache.

Variable object sizes: Virtual memory management has to cope with fixed-size pages in most cases. Replacement algorithms in FFS have to respect differently sized objects. Since small files are more frequently referenced than large ones [7], but large files need more time to be transferred, the dilemma arises whether to discard a small or a large file from the cache. Two important input values to a FFS-replacement algorithm will thus be the reference frequencies and the sizes of the files in the current workingset.

Lower and variable transfer rates: FFS are normally built upon communication technology providing much lower and more variable transfer rates and longer roundtrip-delays. Therefore, it is mostly impossible to predict access times to FFS-objects exactly enough to make a formal analysis of FFS replacement algorithms feasible. In this work we will use discrete event simulation for analyzing FFS cache-replacement algorithms.

2 A Simulator for Cache Replacement Algorithms

For this study the RESIM [6] (REplication SIMulator) package developed at the University of Zurich was used. RESIM is a discrete event simulator appropriate for analyzing file access, transfer and replication in heterogeneous networks. These networks are supposed to be made up by several interconnected clusters [11], each containing a file server and several clients. Such a cluster could look like in figure 1.

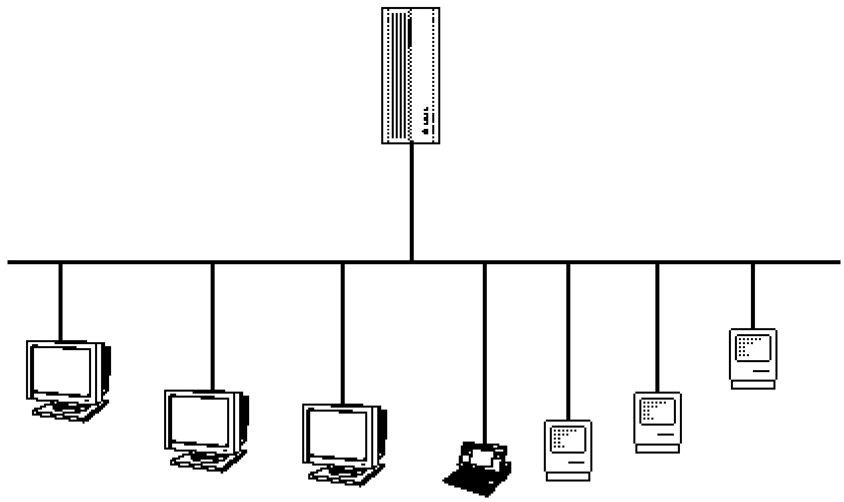

Figure 1: A small cluster of workstations.

A client may be a cluster server itself, leading to a hierarchical structure of the FFS. In this paper we shall restrict our studies to a two-level hierarchy, meaning a set of interconnected cluster servers each being attached to several client workstations.

We further assume that FFS-CRA primarily run on the *cluster servers*. This way, every client workstation attached to a cluster server may access the files on it. A cluster itself is supposed to be made up by fast LAN technology, therefore obviating a complex caching mechanism at the client workstations. Further advantages of this model are that diskless clients may easily be integrated into a cluster and that scalability of the FFS is better ensured. The cluster servers are supposed to be interconnected by slower technology, MAN- or WAN-links for example. Figure 2 exemplifies a network of six workstation clusters interconnected by WAN links. In the following we shall restrict our attention to the file transfers taking place between cluster servers and the cacheing at the cluster servers. Whenever a client references a file which is not provided by its cluster server, the server will contact a naming entity to find out which other remote cluster servers hold a replica of the file in their cache and will then obtain an own replica from there. If several replicas of the needed file are provided at different sites, the client will have to choose one along some predefined polling rule [8]. Future file requests of the attached workstations will then quickly be satisfied using the new replica generated on their cluster server, and availability of the file is also enhanced.

2.1 Simulation Input Parameters

For studying FFS-CRA, the simulator was provided with input data defining the topology of the simulated system, information about the files stored on the servers and the duration of the simulated situation. These parameters are described in what follows.

2.1.1 Topology

We report about a simulated scenario consisting of six interconnected cluster servers and 100 files initially stored on them. The simulated network topology could look as depicted by figure 2.

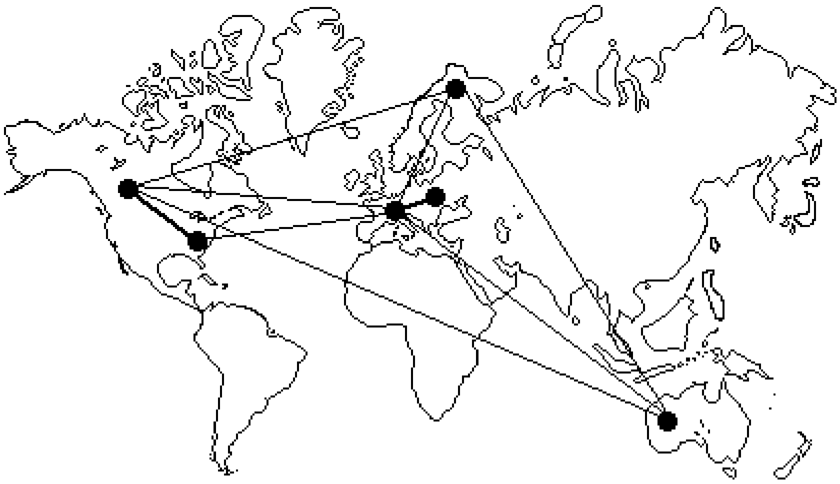

Figure 2: Possible topology of the simulated system.

To interconnect the servers, three different link types were defined. They provided an average throughput of 35.2 kb/sec, 6.7 kb/sec and 3.1 kb/sec. These values were chosen in accordance with measurements of real throughput rates for file transfer (`FTP`) in the `Internet` network. The throughput rates were found to not strongly influence the outcome of the cache replacement experiments described in this work. During the simulation, throughput rates and roundtrip-delays associated with the links were varied between predefined boundaries.

The cluster servers were characterized by following values:

- The size of the local cache in kilobytes. Every server had an equally sized cache. We experimented with cache sizes ranging from 500 kb to five megabytes. 500 kb is the size of the largest file in the simulated scenario. The cache size was continuously increased by half a megabyte for every new simulation run which was started.
- A performance value denoting how much system time the server needs to perform the local computations associated with a file request. The value we set for the simulations was obtained from measurements of file accesses on a `SUN SPARCstation 2`. Disk access times are reflected by this value.

- A factor denoting the load induced by activities other than the client requests, for example the burden caused by the FFS-CRA, the operating system overhead etc.
- A value defining the amount of access to remote servers a server generates.
- A list of local *master files*. These files belong to the server and are not stored in its cache. Cached files may get deleted from the cache when space is needed for storing a new file. Master files may *not* disappear from the host they belong to.

  Initially, the 100 master files were unevenly distributed over the six servers, with one server holding the largest part of the files.

2.1.2 Files

The most important part of the parametrization concerns the files used in the simulation. For the simulator, a file is characterized by its *size* and *access-rate*, the latter standing for the "popularity" of the file. Both parameters were set for the 100 files in accordance with the results we report in [7]. The range of the file sizes covered 50 kilobytes to 500 kilobytes, with an average file size of 60 kilobytes. It was further assumed, according to the cited study, that 40% of the file transfers (file accesses) relate to 76% of the files. File requests are generated by each server along the file popularity pattern depicted by figure 3. The time between two subsequent file-accesses from a certain server follows an exponential distribution.

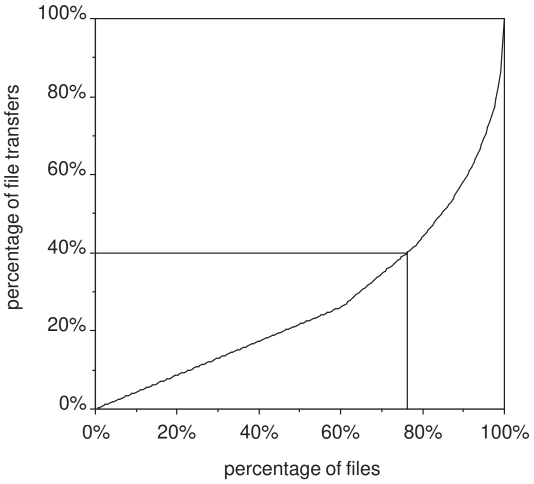

Figure 3: File popularity graph. 40% of the file accesses relate to 76% of the files.

The cached files are assumed to be stored locally on disk. Consistency among the replicated file copies was not addressed, meaning that read-only files are assumed.

2.1.3 Duration

A scenario of 30 minutes was simulated. During this time up to 70 megabytes of data were transferred between the servers and more than 1000 file accesses performed. All files referenced during these 30 minutes by a server are part of its actual *workingset.*

2.2 Simulation Output Parameters

Three types of information were produced by the simulator during every run, namely access times to files, cache hit ratios and availability.

Access Times to Files An access to a non-cached, remote file comprises the time to send a request packet to the server containing a replica and the time for transferring the file over the network. These values are computed by RESIM using the parameters associated to the links the file must pass through, namely throughput and roundtrip delay. If several alternative paths in the network exist between sender and receiver, *shortest-path routing* is performed and the route granting the largest throughput is chosen.

The simulator produces a list of the durations the accesses to the objects lasted. The average access time can be computed thereof, and the primary goal of our FFS-CRA will be to minimize the average access time to files. The standard deviation computed over the access times depicts how predictable access times to objects are. We will report about access times to objects in section 4.1.

Cache Hit Ratios It is also interesting to analyze FFS-CRA from the viewpoint of the cache hits a CRA produces. We define a cache hit ratio as $\frac{H}{H+M}$ with H being the number of cache hits over all nodes occurred during the simulation and M the number of overall cache misses. We will report about hit ratios in section 4.2.

Availability By measuring the average number of replicas in the caches we can assess the availability of the files. For simplicity, we assume that the availability of a file proportionally increases with the number of its replicas. Availability is addressed in section 4.3.

3 Analyzed CRAs

The replacement algorithms FIFO, LRU, LFU and LEN were analyzed in our work. The algorithms are defined as follows.

3.1 First-In-First-Out (FIFO)

The simplest replacement algorithm is the *first-in-first-out (FIFO)* algorithm. A FIFO replacement algorithm always replaces the oldest file in the cache, independently from the popularity (access-frequency) of the file. The FIFO algorithm is easy to realize, but its performance may be weak. FIFO is subject to the *Belady anomaly* [2], which states that for certain replacement algorithms, the cache hit-ratio may *decrease* as the cache size increases. Thus, in a FFS relying on FIFO, one would increase the cache sizes of the attached computer systems and, at the same time, obtain a worse performance in certain cases.

The Plan 9 filesystem [10, 9] uses the FIFO rule to determine which of the possible cache map blocks is used when blocks are added to the cache. The authors of the current version of the Plan 9 filesystem assumed that the extra complexity and storage requirements to implement other algorithms such as LRU (see next section) was not justified. In this work, we will compare the performance of LRU, FIFO and other algorithms and try to assess whether the extra complexity of implementing LRU is worth the effort.

3.2 Least Recently Used (LRU)

This algorithm claims that the object in the cache which has not been referenced for the longest time must be replaced. Here we rely on the locality heuristic that the recent past is a good indicator of the near future, meaning that an object not referenced for a long time will probably not so soon be referenced.

In virtual memory management units, LRU could require substantial overhead since it requires that each memory page be timestamped whenever it is referenced. Therefore, LRU is not implemented in most of the current memory management units, but is approximated by other lower overhead strategies [3].

In the field of file systems and file transfer protocols, LRU can be realized in software with less effort and less overhead. Filesystems like ANDREW [12] take advantage of this fact by providing LRU cache management for client workstations. In contrast to the FIFO algorithm, LRU does not suffer from *Belady's anomaly.*

3.3 Least Frequently Used (LFU)

The LFU algorithm discards the object which has been referenced for the least amount of times. LFU relies on the heuristic that an object which got infrequently referenced in the recent past will not so soon be referenced in the near future. Intuitively, at the beginning of our investigations we supposed that the LFU policy could ensure shorter access times to objects than LRU. We therefore simulated LFU and compared the results with LRU.

3.4 The File Length (LEN) Algorithm

We developed a modification of the LFU algorithm which we supposed could efficiently perform the management of cached files. Upon a full cache, LEN discards the object with the least *number of references* $*$ *object size* factor. The rationale behind this strategy is that LFU and LRU do not consider the size of an object to discard. If, for example, a small and a large object both have associated a very high access frequency, the small object being confronted with slightly more accesses than the large one, LFU would always replace the large object, even thought that refetching the large object is considerably more expensive. Such anomalies can be avoided using an algorithm like LEN.

LFU and LEN have the problem that a large file which received a burst of references may remain in the cache for too much time, even when the file would not be referenced again for a long time. In this case, the space for many small, also frequently accessed files is wasted. For example, imagine a flight-reservation system. When a certain flight becomes actual, the large file containing the reservation data for this flight may get referenced intensively during a short time period. Afterwards, the activity may cease and remain low for a long time, until the flight becomes actual again. LFU and LEN would perform poorly in such a situation.

4 Simulation Results

This section reports about the results obtained by simulating the CRAs for the scenario described in section 2.1.

4.1 Access Times to Files

cache size (kb)	*FIFO*	*LRU*	*LFU*	*LEN*
500	46.7 sec	46.6 sec	46.2 sec	46.4 sec
1000	45.6 sec	45.4 sec	45.0 sec	45.2 sec
1500	44.7 sec	44.5 sec	44.1 sec	44.3 sec
2000	44.1 sec	43.8 sec	43.4 sec	43.6 sec
2500	43.5 sec	43.3 sec	43.0 sec	43.3 sec
3000	43.1 sec	42.8 sec	42.7 sec	42.8 sec
3500	42.7 sec	42.6 sec	42.5 sec	42.6 sec
4000	42.5 sec	42.4 sec	42.4 sec	42.4 sec
4500	42.3 sec	42.3 sec	42.3 sec	42.3 sec
5000	42.3 sec	42.3 sec	42.3 sec	42.3 sec

Table 1: Average access times to files for the simulated CRAs.

cache size (kb)	*FIFO*	*LRU*	*LFU*	*LEN*
500	32.1 sec	31.9 sec	32.0 sec	32.0 sec
1000	31.7 sec	31.7 sec	31.6 sec	31.7 sec
1500	31.7 sec	31.6 sec	31.6 sec	31.5 sec
2000	31.5 sec	31.5 sec	31.3 sec	31.4 sec
2500	31.4 sec	31.4 sec	31.4 sec	31.5 sec
3000	31.4 sec	31.3 sec	31.5 sec	31.2 sec
3500	31.3 sec	31.3 sec	31.2 sec	31.4 sec
4000	31.4 sec	31.3 sec	31.2 sec	31.2 sec
4500	31.2 sec	31.2 sec	31.2 sec	31.2 sec
5000	31.2 sec	31.2 sec	31.2 sec	31.2 sec

Table 2: Standard deviation for the access times in table 1.

cache size (kb)	*FIFO*	*LRU*	*LFU*	*LEN*
500	79.1	78.7	74.6	76.5
1000	68.2	67.1	61.2	64.0
1500	57.9	56.2	48.6	52.1
2000	49.2	46.2	39.8	43.8
2500	41.5	37.5	34.4	37.6
3000	35.2	31.5	30.0	32.1
3500	29.8	27.6	26.7	28.6
4000	26.4	25.0	24.9	25.8
4500	24.1	24.2	24.2	24.0
5000	23.7	23.8	23.8	23.7

Table 3: Megabytes transferred between the servers.

In table 1 we report about the average access times to the files in relation to the simulated CRAs. As may be seen from the results, a relatively small cache considerably increases the performance of a flexible filesystem by decreasing server load. This is due to the usually high locality of reference for file accesses.

Access times decrease slowly with growing cache sizes. In our experiment, increasing the cache size by a factor of 10 leads to a 10% lower access time. As expected, we also remark a decrease in the standard deviation of the access times as cache size increases (see table 2). Thus, caching also induces better predictable access times to files, which is important for real time systems.

The amount of data transferred between the cluster servers is reported in table 3. LFU seems to cause less overall network load than the other algorithms.

4.2 Cache Hit Ratios

In table 4 we present the cache hit ratios of the analyzed algorithms. The cache hit ratio was defined in section 2.2. LFU shows a higher hit ratio than FIFO, LRU and LEN. With growing cache size, the hit ratios become more and more equal and gravitate towards the maximal possible hit ratio, which is around 82% for the simulated system.

Now, the question arises why the CRAs show different cache hit ratios but similar access times at the same time. The answer lies in the distribution of the cache hits per object (figure 4). LRU shows more cache misses for smaller files but less misses for larger files, with their size in the range of 150 kilobytes to 270 kilobytes. Thus, LRU has to refetch more objects with smaller size, but less large objects. This results in a smaller cache hit ratio but higher performance at the same time.

The fact that FFS-CRAs have to cope with variable object sizes asks for a separated analysis of FFS

cache size (kb)	*FIFO*	*LRU*	*LFU*	*LEN*
500	29.7%	30.1%	34.0%	32.2%
1000	39.9%	40.9%	46.6%	43.9%
1500	49.4%	51.1%	58.3%	55.0%
2000	57.4%	60.4%	66.5%	62.7%
2500	64.6%	68.4%	71.4%	68.3%
3000	70.3%	73.9%	75.4%	73.2%
3500	75.0%	77.3%	78.2%	76.4%
4000	78.2%	79.6%	79.7%	78.7%
4500	80.2%	80.3%	80.3%	80.3%
5000	80.6%	80.5%	80.5%	80.6%

Table 4: Hit-ratios of the simulated CRAs.

replacement algorithms. Results obtained in the field of virtual memory management can therefore not always be applied to filesystem cache management without adaptation.

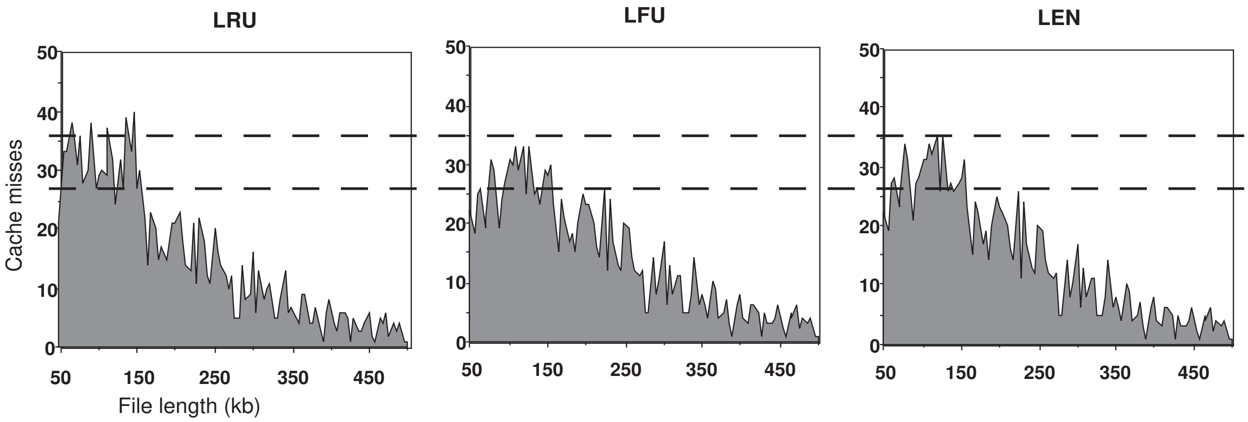

Figure 4: Number of cache misses during the simulation in relation to the file size.

4.3 Availability

LRU and LFU showed the highest availability during our experiments. Neither of the two CRAs respects object sizes, but the files with the highest access-rates are maintained in the cache, which happen to be the smallest ones. Many more short files fit into a cache than in the case where also large files obtain a high priority, and therefore the availability is enhanced by having many more replicas around in the distributed file system. LEN showed the worst availability, since in this algorithm large files have a higher priority than in LRU and LFU. The availability ensured by FIFO is also lower, since the round-robin management of the cache gives the equal priority to short and large files, decreasing the average number of files in the caches.

5 Conclusions

In this study we have simulated the performance of the FIFO, LRU, LFU and LEN replacement algorithm. The simulated scenario consisted of six interconnected cluster servers with 100 active files initially distributed over them. The simulator was parametrized using real values obtained from a previous investigation of file access patterns in public file archives [7]. The results of the simulation show that from the viewpoint of average access time to files, the performance differences between LRU, LFU and LEN are small. Of the simulated algorithms, FIFO shows the worst performance. This relationship is depicted in figure 5.

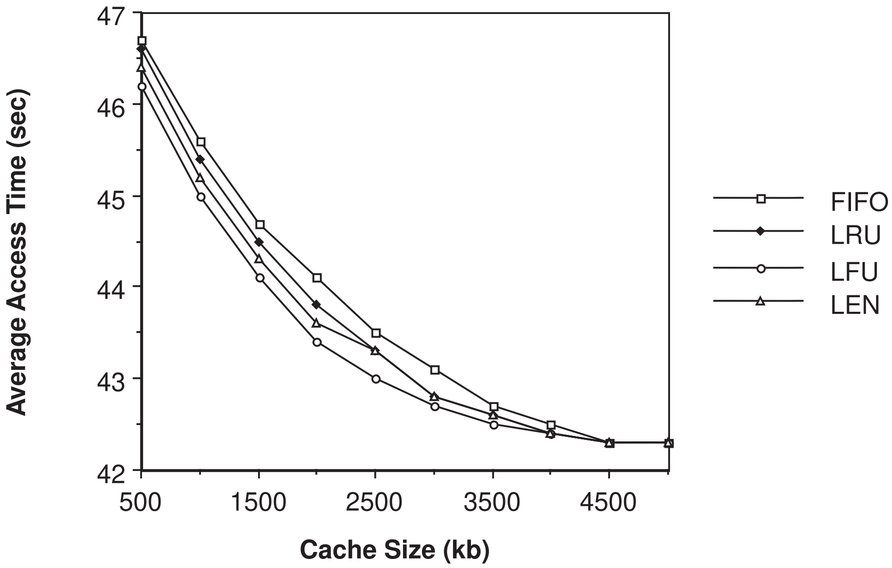

Figure 5: Comparison of average file access time in relation to cache size.

LRU has the advantage of less cache misses for large objects, but shows a smaller overall cache hit ratio than LFU and LEN. LFU and LEN have to refetch more large objects and therefore lose their advantage of a better overall cache hit ratio. Furthermore, LFU and LEN are subject to a caching anomaly, where once frequently accessed objects may remain in the cache for too much time. FIFO suffers from *Belady's anomaly* as was remarked in section 3.1. LRU is not affected by these anomalies and has the further advantage of ensuring a good availability of the files. Simulations with other file sizes, number of files and topologies showed that LRU is more stable than the other algorithms, in the sense that access times are better predictable. LFU and LEN perform slightly better in some situations, like the one reported in this paper, but worse under other conditions due to the described caching anomaly.

We also have shown that a moderated cache size can considerably increase the performance of a distributed filesystem by balancing the load on the servers holding popular files and by reducing network load. In this way, better response times and scalability is ensured. Due to the fact that many accesses to files can be satisfied from the local cache, file access times become better predictable as the cache size increases.

Algorithms like LEN taking into account the filelengths give a priority to longer files, since refetching such files causes long delays and high network load. This advantage must be compared with the disadvantage of losing cache space for many small files when a large file is retained in cache. From the viewpoint of access times, availability and networkload, LEN did not outperform LFU. The advantages of LEN were mostly compensated by its disadvantages.

From the viewpoint of the results gathered in our work it is a good choice to use LRU as cache replacement algorithm in flexible filesystems, since it ensures a good and stable performance of the file system. More sophisticated and complex algorithms often show an only slight performance advantage over LRU and may be subject to caching anomalies.

References

[1] A.V. Aho, P.J. Denning, J. U. Principles of Optimal Page Replacement. *Journal of the ACM 18*, 1 (Jan. 1971).

[2] Belady, L. A., Nelson, R. A., and Shedler, G. S. An Anomaly in Space-Time Characteristics of Certain Programs Running in a Paging Machine. *Communications of the ACM 12*, 6 (June 1969).

[3] Deitel, H. M. *Operating Systems*, second ed. Addison Wesley, 1990.

[4] Denning, P. The Working Set Model for Program Behavior. *Communications of the ACM 11*, 5 (May 1968).

[5] Kistler, J. J., and Satyanarayanan, M. Disconnected Operation in the CODA File System. *ACM Transactions on Computer Systems 10*, 1 (Feb. 1992).

[6] Maffeis, S. RESIM – A REplication SIMulator. CS Dept. University of Zurich, Internal Report, Jan. 1992.

[7] Maffeis, S. File Access Patterns in Public FTP Archives and an Index for Locality of Reference. *ACM SIGMETRICS Performance Evaluation Review 20*, 3 (Mar. 1993).

[8] Maffeis, S., and Cap, C. H. Replication Heuristics and Polling Algorithms for Object Replication and a Replicating File Transfer Protocol. Tech. Rep. 92.06, CS Dept. University of Zurich, June 1992.

[9] Presotto, D., Pike, R., Thompson, K., and Trickey, H. Plan 9, A Distributed System. In *Proceedings of the Spring 1991 EurOpen Conference* (May 1991).

[10] Quinlan, S. A Cached WORM File System. *Software Practice and Experience 21*, 12 (Dec. 1991).

[11] Sandhu, H., and Zhou, S. Cluster-Based File Replication in Large-Scale Distributed Systems. *ACM Sigmetrics and Performance 20*, 1 (June 1992).

[12] Satyanarayanan, M., et al. Scale and Performance in a Distributed File System. *ACM Operating Systems Review 21*, 5 (Sept. 1987).

[13] Siegel, A. *Performance in Flexible Distributed File Systems.* PhD thesis, Faculty of the Graduate School, Cornell University, May 1992.