Effectiveness of Caching in a Distributed Digital Library System

Jochen Hollmann [a,*,1,3], Anders Ardö [b,2], Per Stenström [a]

[a] *Department of Computer Engineering, Chalmers, 412 96 Göteborg, Sweden*

[b] *Technical Knowledge Center of Denmark, P.O. Box 777, 2800 Lyngby, Denmark*

Abstract

Today independent publishers are offering digital libraries with fulltext archives. In an attempt to provide a single user-interface to a large set of archives, the studied Article-Database-Service offers a consolidated interface to a geographically distributed set of archives. While this approach offers a tremendous functional advantage to a user, the fulltext download delays caused by the network and queuing in servers make the user-perceived interactive performance poor.

This paper studies how effective caching of articles at the client level can be achieved as well as at intermediate points as manifested by gateways that implement the interfaces to the many fulltext archives. A central research question in this approach is: What is the nature of locality in the user access stream to such a digital library. Based on access logs that drive the simulations, it is shown that client-side caching can result in a 20% hit rate. Even at the gateway level temporal locality is observable, but published replacement algorithms are unable to exploit this temporal locality. Additionally, spatial locality can be exploited by considering loading into cache all articles in an issue, volume, or journal, if a single article is accessed. But our experiments showed that improvement introduced a lot of overhead. Finally, it is shown that the reason for this cache behavior is the long time distance between re-accesses, which makes caching quite unfeasable.

Key words: H.3 Information Storage and Retrieval, H.3.7 Digital Libraries, Document Caching, Performance

* Corresponding author. Tel: +46-31-772-1692; Fax: +46-31-772-3663
Email addresses: joho@ce.chalmers.se (Jochen Hollmann), anders@it.lth.se (Anders Ardö), pers@ce.chalmers.se (Per Stenström).

[1] Supported by NORDUnet2

[2] Present address: Dep. of IT, Lund University, 221 00 Lund, Sweden

[3] Present address: Google Norway A/S, Beddingen 10, 7014 Trondheim, Norway

1 Introduction

The World Wide Web has changed the way research literature is accessed. Most publishers put their fulltext archive onto the web as digital libraries (DL) containing downloadable files in the portable document format (PDF). Integrating multiple indexes of bibliographic data about the accessible documents into a single DL portal allows users to search all archives simultaneously. The result is the DL architecture shown in Figure 1.

The portals described above are implemented as gateways which constitute a confluence point for multiple users to get access to the multiple fulltext servers. While the portals can be brought close to the user to reduce delays, e.g. by locating them in a local on-campus network, the fulltext servers will remain remote. Even if it is technically feasible to replicate all fulltext archives, too, publishers will not allow this in fear of untraceable copyright violations. Due to the physical distance and network delays, users are hence facing poor performance according to the latency involved.

With respect to the fulltext archive, DL gateways act similarly to web proxy servers, which are confluence points where web caches can be located. Web caches are successfully used to reduce the latency for web downloads by using locality in the access stream. Those access streams are already filtered by the caches in the web browsers, but enough locality remains in order to get an advantage.

Portals are spreading fast as a standard technique of data representation on the Internet. Hence it is interesting to study if caching within portals could be

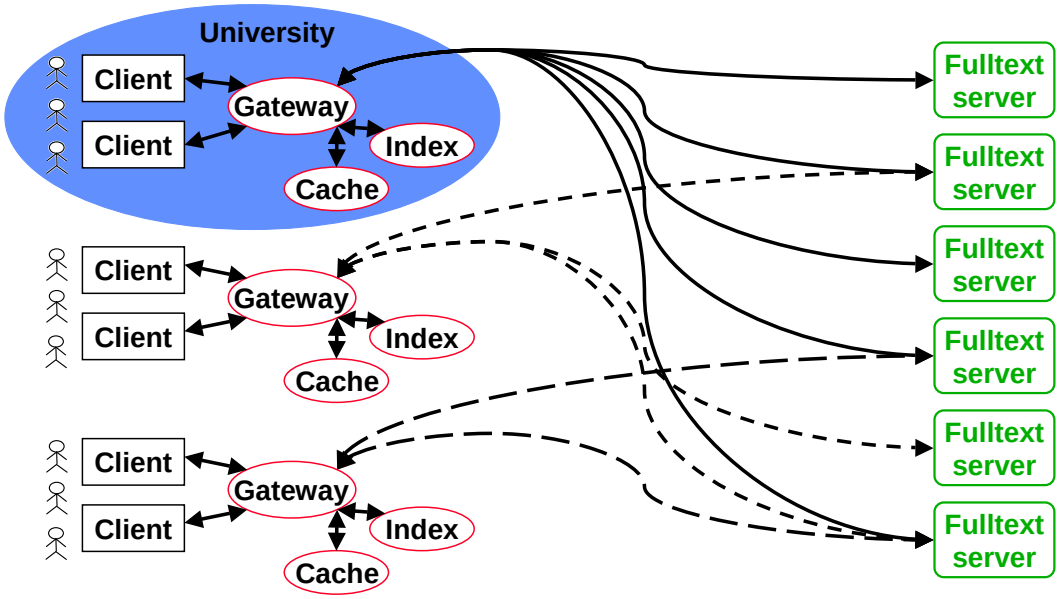

Fig. 1. Distributed Digital Library Architecture. Adapted from [1].

used to improve the user experience. One class of portals are DLs, which have been in use for some time now and thus allow long time studies of content access. Since additionally pure caching is allowed by most publishers, it is interesting to study the access stream to DLs for patterns suitable for caching.

This paper presents the first study on how effectively caching in a distributed DL can help reducing the access latency. A trace driven simulation model is used to compare various caching strategies published in the literature. The trace itself was gathered during a period of almost two years from a DL portal called DADS [2,3], which is used at the Technical University of Denmark.

The contributions of this paper are the following main observations: Web browser caches are either not used or are inefficient for DLs. Almost 1/5 of all document requests from any particular client were re-fetched during our observation. Once those are not considered, the remaining access stream does still contain temporal and spatial locality, but can be only used effectively by oracle strategies. Implementable history strategies like LRU perform hardly better than random and spatial based strategies need to cache large proportions of the document base. This leaves a large gap between the wish for implementation of an efficient caching scheme and practice. The reasons for this gap can be found in the large distance in time between re-accesses, which leads to a small amount of re-accesses within a reasonable time frame. In fact during the observation period most documents were only accessed once, and of the remaining ones most were only accessed twice. As a result history-based caching strategies, even long term, hardly exist, as the temporal locality is too sparse. As spatial locality based central strategies are ineffective, too, only re-use by the same user can be used effectively for caching. However we where unable to study strategies based on metadata such as references, which might open possibilities to improve caching.

As for the rest of the paper, in Section 2 the caching strategies used in this study are introduced and related to other application areas, in particular web caching. In Section 3 the analysis method is described and the results are displayed in Section 4. Finally the obtained results are summarized and concluded.

2 Caching strategies

This section introduces some important locality concepts that form the foundation for the experiments done in subsequent sections. Additionally, the studied replacement algorithms are introduced in 2.3. Finally, similarities and differences in replacement algorithms in related contexts are described in 2.4.

2.1 Reference locality

For an effective cache, the reference stream must exhibit locality either in the temporal or in the spatial domain. In the following, these concepts are put into perspective of a Digital Library system, where the reference stream consists of the user accesses to fulltext servers.

- *Temporal locality* is based on the observation that in many systems objects are re-accessed shortly after they have been accessed. For digital libraries, this would mean that a given document often is read within a short period of time but is then not re-accessed for a long period.

- *Spatial locality* is based on the observation that accesses are not spread out evenly, instead the probability to access nearby objects is high. In a digital library serving research articles, the objects to be cached may not only be articles, but whole issues, volumes, or journals. Bringing in these units can potentially serve upcoming accesses to other documents within the same object. Note that unit and object are used as interchangeable terms throughout the paper.

2.2 One-timers, Multi-timers, Reuse and Popularity

Caches can improve system performance if objects are accessed multiple times. In the following, objects that are accessed only once are referred to as *one-timers*. Other objects are called *multi-timers*. It is useless to cache one-timers as they can in fact even displace other objects, that will be re-accessed in the near future.

Re-accesses to an object come from two sources. The document can be reused from the same client machine. Assuming that there is a one-to-one relation between a client machine and a user, which is the case for most desktop machines, this means that the same user uses the document again. This type of re-access is called *reuse*. Another possibility is that different clients access the same object. Under the assumption that client machines are independent and used by different users, re-accesses of this type must be caused by *popularity* within a user group. Note in the popularity case the special meaning of one-timers, used by one person, and multi-timers, re-used by different persons.

To study how significant spatial locality is articles, issues, volumes, and journals are considered as units. These units are selected because it is a natural way of grouping articles. Besides this, the data is available from the databases used by DADS. So when an article is accessed it is possible to look up the issue, volume and journal from the meta data database.

The key question raised in this research is whether there is enough exploitable locality, temporal or spatial, to make caching effective in a digital library system. To study temporal locality, published history-based replacement algorithms are reviewed next.

2.3 Replacement strategies

The design space of proposed caching strategies fall in between two theoretically interesting strategies: *OPT* and *RANDOM*.

The OPT strategy is aware of the future and is an upper limit for the achievable performance, if the cached objects have a fixed size. OPT only brings objects into the cache, which will be accessed again, hence it avoids cache pollution. In case there is not enough room left, the object which will be accessed the longest in the future will be removed. Objects which will not be re-accessed in the future are also removed after they have experienced their last hit.

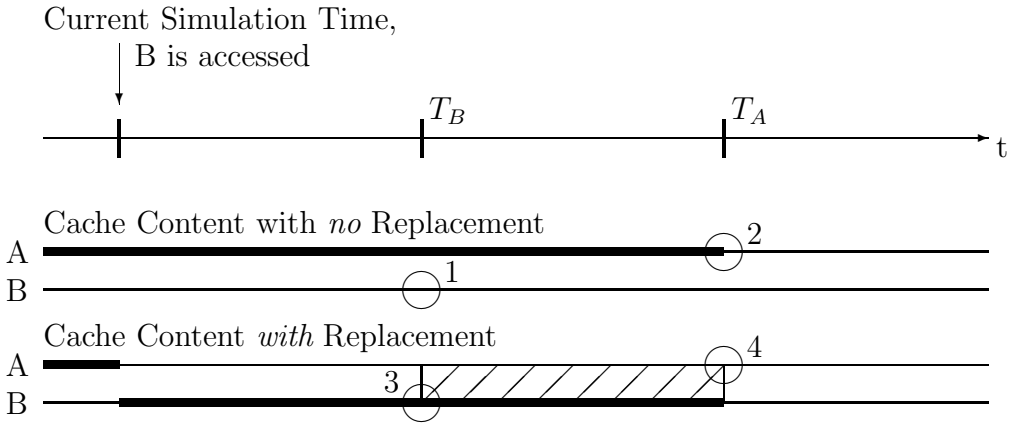

Fig. 2. Example for a Decision running the OPT Caching Strategy

In case of caching at the article level, where a single article is removed to make space for another one, OPT is the optimal strategy. Figure 2 shows an example of two objects A and B, which are marked with thick lines while being in the cache. Assume that object A resides in the cache and will be accessed the next time at T_A, the largest next access time-stamp of all objects in the cache. Assume that there is no space left in the cache and a not yet present object B is accessed and that the following access to B will take place at $T_B < T_A$. If A is not replaced by B, there will be a miss at T_B (1) and a hit at T_A (2). If A is replaced by B there will be a hit at T_B (3) and a miss at T_A (4). Hence there is no difference between the two approaches. But the second approach has the advantage that there may be hits to B in the time between T_B and T_A (the hatched area), which can improve the hit rate. Note that at point T_A there will be a new decision if A will be brought into or removed from the

cache or if the cache contents remains unchanged.

At the issue, volume and journal level, OPT applies the same scheme. But since such objects differ considerably in size, this strategy will only be a lower bound for the optimal strategy. A performance loss may occur when moving in an object replacing two or more other objects but having only one hit. In this case, two or more hits are traded for a single one, which is not optimal. If an object is larger than the cache size, it will not be put into the cache, hence the cache content remains unchanged.

The RANDOM strategy does not exploit the behavior embedded in the access pattern. Instead, if a miss occurs, the object which caused the miss is moved into the cache and space for it is made by removing a randomly selected object. RANDOM is often used as a baseline for comparison.

The well-known strategy Least Recently Used (LRU) [4] will be used in most experiments as an implementable strategy based on temporal locality. It is implemented as follows. On a miss, the available space in the cache is checked. If the object does not fit, a sufficient number of objects with the oldest access time-stamps are searched and removed to make enough space available before the new object is moved into the cache. LRU is widely used and often represents the baseline for comparing improved caching strategies.

LRU considers only the last access. O'Neil *et al.* [5] introduce LRU-K in the context of database disk buffering. LRU-K considers the K last accesses to an object. Based on these time-stamps the oldest object is purged. Objects, which have not yet been accessed K times, will have the k-age ∞. If two or more objects have the k-age ∞, the decision is done according to the (k-1)-age and so on. O'Neil *et al.* [6] also proved that LRU-K is optimal on the available information. It is of advantage to use LRU-K instead of LRU if regular accesses are mixed with patterns of infrequent burst accesses.

Another variant of LRU is Segmented LRU (SLRU), introduced by Karedla *et al.* [7]. For this strategy, the available cache space is divided into two parts, an unprotected and a protected segment. LRU runs on both segments. A hit in the unprotected segment moves this object into the protected area and moves the oldest object from the protected segment into the unprotected segment. Hence objects which have been accessed at least twice get always one further chance by being moved to the unprotected area. This strategy is advantageous if many objects are accessed only once, because these objects will never make it to the protected segment, which will be reserved for objects with more accesses. Arlitt[8] found that SLRU works best for web caching if 60% of the cache is protected area.

The Most Frequently Used (MFU) and Least Frequently Used (LFU) [9] algorithms represent two more history-based approaches. MFU removes the ob-

jects which have a high access frequency. All objects above a certain MFU threshold are removed from the cache. Hence this strategy is good for burst accesses with an upper limit in the number of re-accesses. LFU on the other hand removes objects which have a low access count first, hence it favors objects with many, frequent accesses. LFU-AGE, as considered here, reduces the access counts in regular intervals by 10% to give accesses less weight which had no accesses recently.

2.4 Caching in various contexts

Caching and cache management have been studied for a long time, mainly in the context of processor caches[10] and as page replacement strategies for virtual memory[11,12]. In both cases, access streams are characterized by a high degree of temporal locality and hence LRU strategies work well.

Data caches for databases[13] or storage systems experience less temporal locality. Still they contain enough temporal locality that LRU works reasonably well, but can be outperformed by frequency based strategies [14]. All of the above systems are normally based on fixed size caching units.

In contrast, web caching systems have to deal with a tremendous size range of cached objects — from a couple of bytes to hundreds of megabytes in the case of multimedia files. This adds another dimension to caching strategies[15], which are in this context often measured differently: By the *hit rate*, which is the relative number of hits to objects of the total number of object accesses, and by the *byte hit rate*, which is the number of bytes served from the cache compared to the total number of bytes requested.

Efficient algorithms for web proxy caches like GreedyDual-Size (GDS) proposed by Cao and Irani [16] or LRU-SP [17] by Cheng and Kambayashi are optimized to balance both, the hit rate by keeping many small units and the byte hit rate by keeping few frequently accessed large units. Williams *et al.* [18] found that for web proxy caches both size based and frequency based algorithms outperform pure LRU. For a more detailed overview of caching in the web context see Podlinig and Böszörmenyi[15]. While files storing research papers also differ in size, the difference in scale is much smaller and in fact for this study the sizes are not considered. See section 3.1 for the detailed reasons.

Caches can be placed everywhere along the access path. When two or more caches are in place, they form a multi-level or hierarchical cache system. Busari *et al.* [19] have studied the effects of such a system in web context and found that it is better to use different caching strategies on the various caching levels, because the client-near cache strategies work as filters removing the type of locality they are based on. Their findings are mirrored in some of our results.

3 Analysis Method

A trace of user accesses is used to drive a simulation model of the distributed digital library system introduced earlier. Now the details of trace capturing as well as the simulation model will be discussed in Section 3.1 and Section 3.2, respectively.

3.1 Logging Infrastructure

This analysis is based on log files from DTVs Article Database Service (DADS) [2,3], a digital library portal for journal and conference articles developed at Denmark's Technical Knowledge Center (Danmarks Tekniske Videncenter, DTV) of the Technical University of Denmark. DADS is implemented according to Figure 1. An HTTP server acts as a gateway running scripts, which communicates with an index database performing the searches on the users behalf. DADS index database is a superset of the databases from all major primary and secondary publishers. It contains around 20 million entries.

Once the user requests an article, it is fetched from the fulltext server given online availability. The user environment is a standard web browser, which interacts with the gateway HTTP server. No influence on the web browsers is possible, hence it is not known if and what kind of caching is implemented at this level. During the time the log files were recorded DADS itself did not have any cache implemented.

All incoming HTTP requests are recorded by the gateway of the Technical University of Denmark. The log files contain for incoming fulltext accesses among others the client's IP-address, time-stamp, the article number and the status code. This limits this study to the behavior of members of a university environment such as students, faculty and staff. No special test environment was set up to study specific user groups such as classes or researchers only. Hence the trace probably captured typical mixed access patterns. The user behavior was captured from March 2000 to January 2002, a total of 700 days.

This study only considers article access patterns. In particular, single articles that users try to access as well as when this happens is of interest. This does not mean that users will always navigate through the search interface to a single article. Instead, users can also browse through articles in the same issue, volume, or journal. It should also be noted that conference proceedings are often classified as a journal which contains only one issue.

Requesting an article in fulltext is a two-step process in DADS. The system first displays a delivery page with the choice of different access methods like

ordering a paper copy or downloading from the fulltext archive. Because fulltext is only available for a limited number of database records (approximately 20%), accesses to the above described delivery page is used as a replacement of the actual fulltext accesses. This is reasonable, because the user has to access this decision page for every single real fulltext access. The user does not know in advance, whether the fulltext will be accessible electronically or not. This way, the access traces are valid for both online articles as well as for articles available as paper copy only. For a more detailed description of the DADS user model see our previous paper [1].

As a result of the missing file size data for the paper-copy-only documents, the simulations are limited to be done at the article level. But even if file sizes would be available for all database entries, it is not clear if this would give more insights. Many old articles are scanned and hence often ten times as large as slim-line generated PDF-files with the same number of pages. As a result, the access patterns would change over time as scanning disappears as a standard production technique. By doing the study based on an article as the smallest unit, these artificial effects are avoided.

The gathered log files do not contain any information about the issue, volume or journal of the accessed articles. Instead, this data is available from the index database within the DADS system. Combining the information from the log files with the index database allows the creation of a suitable trace. Such a trace contains a time-stamp for any access as well as a unique identifier for the article, issue, volume and journal, and the size in articles for each unit.

3.2 Simulation

A trace-driven simulation model is used to study user access patterns and the effects of different caching strategies. The simulator takes a trace as described above as input and reads through it, line by line, modeling the caching strategies as schematically shown in figure 3.

The cache is implemented as a set of access unit IDs. It also counts how many articles it contains at every moment in time. Besides this, there is the possibility to store additional data for every unit accessed, like the last access time, total number of accesses and alike.

To implement strategies which need to be aware of the future, there is the option to read the whole trace twice. In a preparation phase only meta data such as future accesses are recorded on a per unit base, so that the OPT strategies have this data available later on. The simulator also got the possibility to discard all but the first access to a particular article from a particular client machine. This implements filtering as done by a perfect client cache.

Caching Simulator — gathering statistics of caching strategies

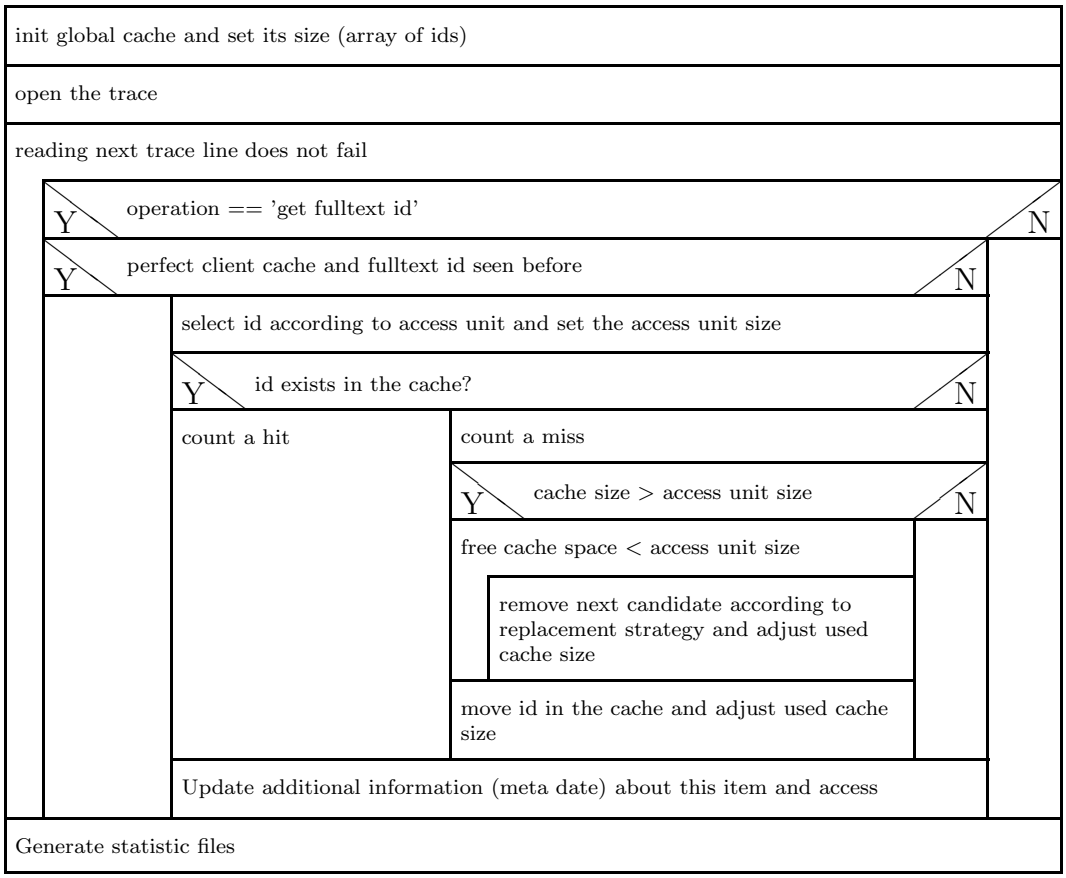

Fig. 3. Nassi-Shneiderman[20] diagram of the simulator
Program statements are executed from top to bottom. Loops are marked by indention with their conditions on top. If-statements are marked with their outcome (Yes or No) and the execution is divided into separate columns under Yes and No. For example, the "id exists in cache?" is an if-statement, where the execution either continues with "count a hit" or "count a miss" and the statements underneath.

When a new request, represented by a line in the trace, is read in, it is checked if its access unit ID is present in the cache. If it is, this is counted as a cache hit. In any case, the request itself is counted and the appropriate actions according to the caching strategy are performed.

For caching units larger than articles, it may be necessary to discard more than one unit in order to get space for the new larger unit. This is done in a loop by calculating new targets to be removed, until the cache has room for the new object. When a particular unit size exceeds the cache size, no replacement happens and the cache remains unchanged. The output of a simulation run will be the hit rate, computed over the total time of the trace.

4 Results

4.1 Reuse versus Popularity

The first experiment studies the effects of client-side caching versus central proxy caching. Ideally, a trace of a system where no caching is implemented should be used, but in practice this is not possible. Most users already have a local client cache integrated into their web browser, which can be assumed to keep recently accessed web documents in order to reduce latency. These caches only influence cache hits due to reuse, but not due to popularity.

Figure 4 shows the simulation results for implementing a central proxy cache, taking advantage of both reuse and popularity. On the x-axis, using a logarithmic scale, the cache size is varied from one article to a million articles, which represent 5% of the total archive size. The curves show the hit rate for OPT, LRU and RANDOM. OPT does significantly better than both LRU and RANDOM, with LRU being only slightly better than RANDOM. A cache size above 200 000 articles is sufficient to achieve the maximum performance of the cache with any replacement strategy. This is the point where the cache can hold all articles accessed. A hit rate of close to 35% is achievable, which is high

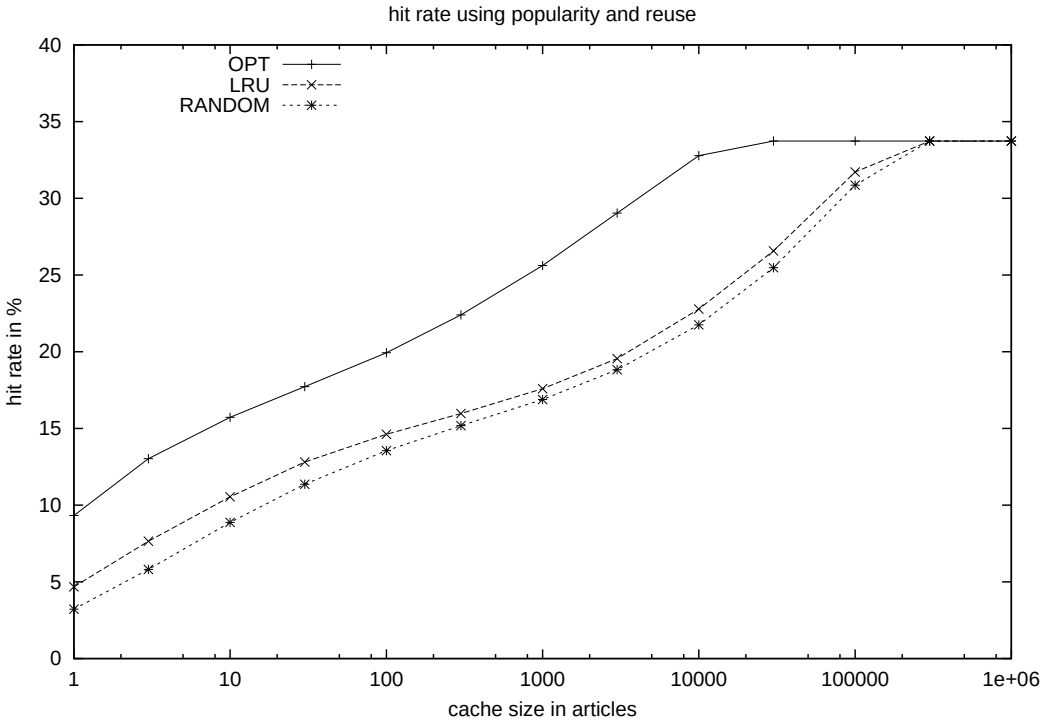

Fig. 4. Hit rate achieved at a central proxy cache running OPT, LRU and RANDOM taking advantage of popularity and reuse, because no (perfect) client caching was simulated.

considering that browser caches have filtered the access stream already. Note that this point depends heavily on the length of the trace, because over time a larger variety of articles is accessed and needs to fit into the cache. Since the archive contains approximately 20 000 000 articles, this point corresponds with about 1% of the total archive. OPT achieves the same performance with 0.1% of the total archive.

To separate the caching effects of reuse versus popularity, a perfect client cache is assumed next. Such a perfect cache would serve all re-accesses to a particular article from a particular client. This can easily be simulated by considering only the first access from a particular client to a particular article, which not even browser caching would be able to affect.

The results are shown in Figure 5. While the shape of the curve has not changed for cache sizes above 10 000 articles, more than half of the maximum achievable hit rate is lost due to the filtering of accesses by the perfect client caches. Also, the results of LRU and RANDOM got closer to each other. Other history-based strategies like LRU-K, MFU etc. have been tested, but their results were worse than RANDOM, which tend to indicate that the temporal locality is mainly caused by reuse.

To verify whether most of the temporal locality stems from reuse, and not popularity, a third experiment explores client cache behavior. The full trace is

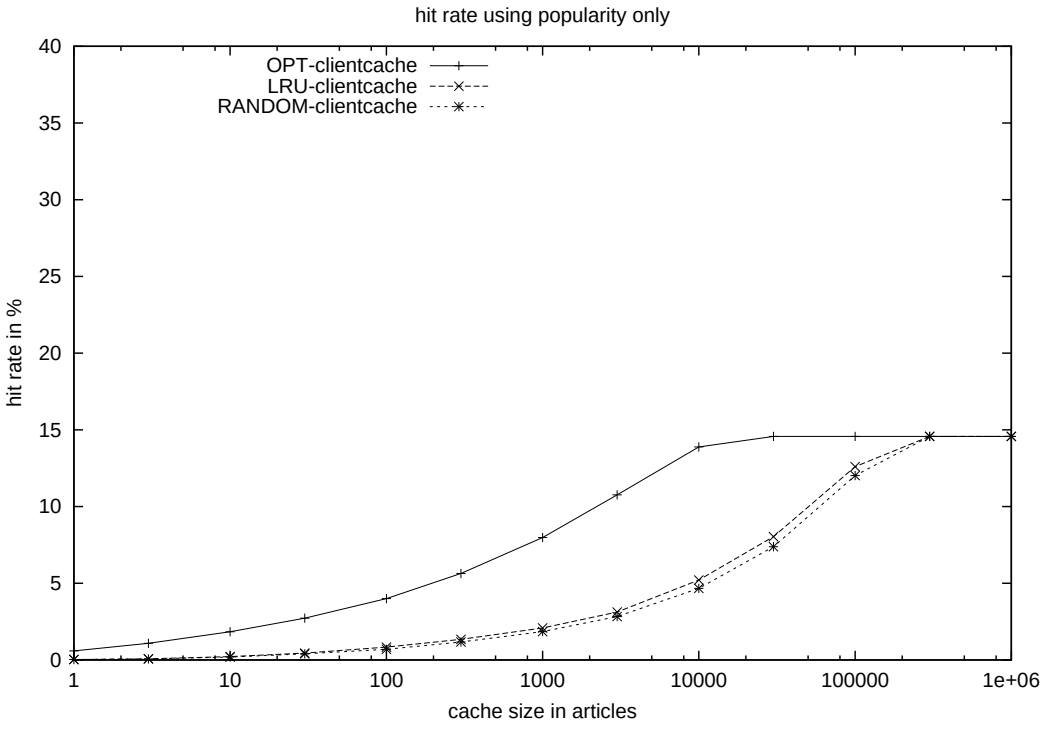

Fig. 5. Hit rate achieved at a central proxy cache running OPT, LRU and RANDOM with perfect client caching taking only advantage of popularity.

used and split it into multiple traces, one per client machine. These sub-traces were then used to drive the simulation model starting with an empty cache for each sub-trace. The hit rate observed is the average hit rate achievable, shown in Figure 6. Note that the browser may strip off a considerable amount of hits.

Now LRU behaves a lot better than RANDOM, which is a strong indication for temporal locality. In the case of web browser caching the results may be even better, because then no browser cache filtering is in place. The observations show that temporal locality is mostly due to reuse and not due to popularity.

Note also that the cache size for each client to achieve a considerable hit rate can be small when compared to a central approach, though the sum of all cache sizes may be larger than in the previous case. Since client-side disk space is cheap, an ideal digital library application would keep every accessed fulltext article on the client-side. This would require a cache size of 10 000 articles (collected over almost 2 years), which would be approximately 10GB, for machines accessing many articles. But for most machines this would only require 100 articles, approximately 100MB, easily implementable with today's client disks.

To implement this, web browsers would only have to support different caching schemes for different URLs. Regular expressions could be used to define caching

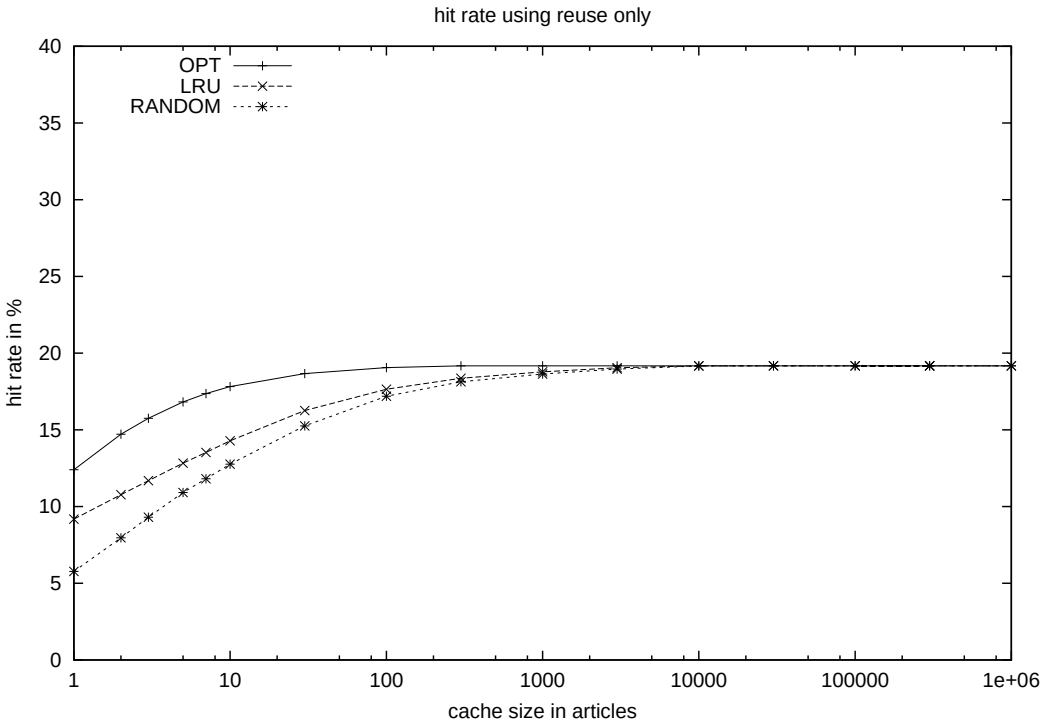

Fig. 6. Hit rate average at a client-side cache running OPT, LRU and RANDOM taking advantage of reuse only.

classes with different strategies. If a regular expression matching all fulltext archives would prevent applying a strategy with expiration, such a perfect cache would be implemented.

4.2 Caching based on popularity

From now on, perfect client caches are assumed, removing all occurrences of reuse. As already seen in Figure 5, exploiting temporal locality centrally does not give much advantage over a random caching scheme, hence spatial locality is investigated next. Note that the partitioning problem itself is not studied and that this study relies on the partitioning of the publishing process given as issues, volumes and journals.

Figure 7 shows the benefits of exploiting spatial locality, comparing the OPT strategy at the article, issue, volume, and journal level, which could capture locality within issues or journals as well as within subject areas.

Concluding from Figure 7, the best strategy seems to be based on journals as access units. Figure 8 shows a comparison of different strategies at the journal level. To separate the effects of exploiting spatial locality from higher hit rates achievable by the presence of more articles in the cache, a strategy called *RANFETCH* was implemented. RANFETCH brings in the average number

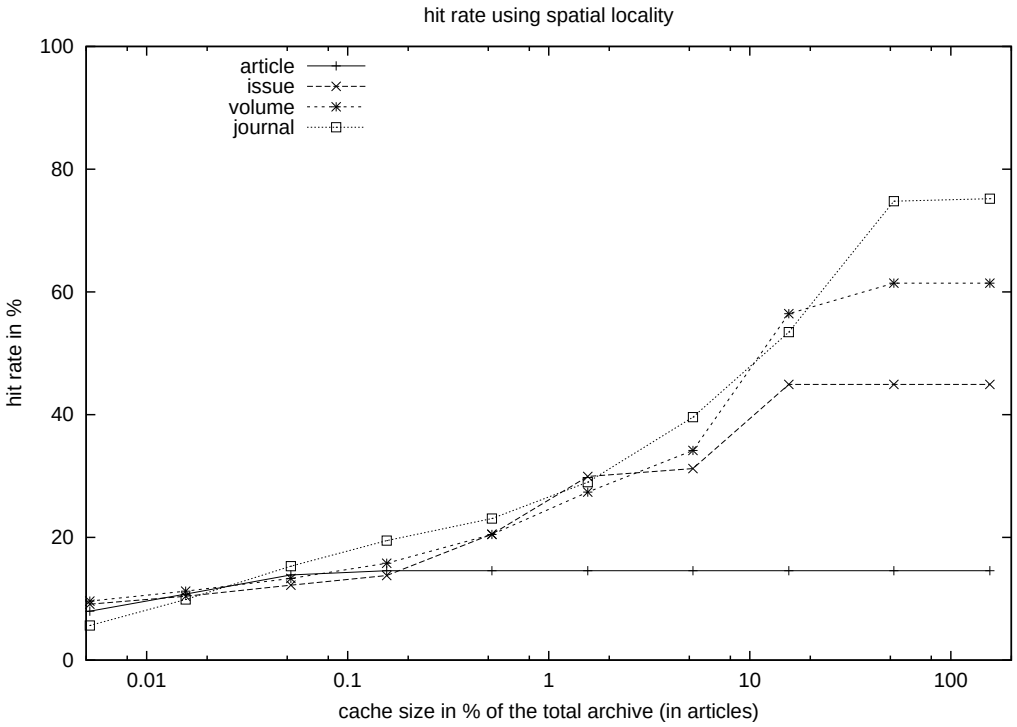

Fig. 7. OPT strategy caching articles, issues, volumes or journals.

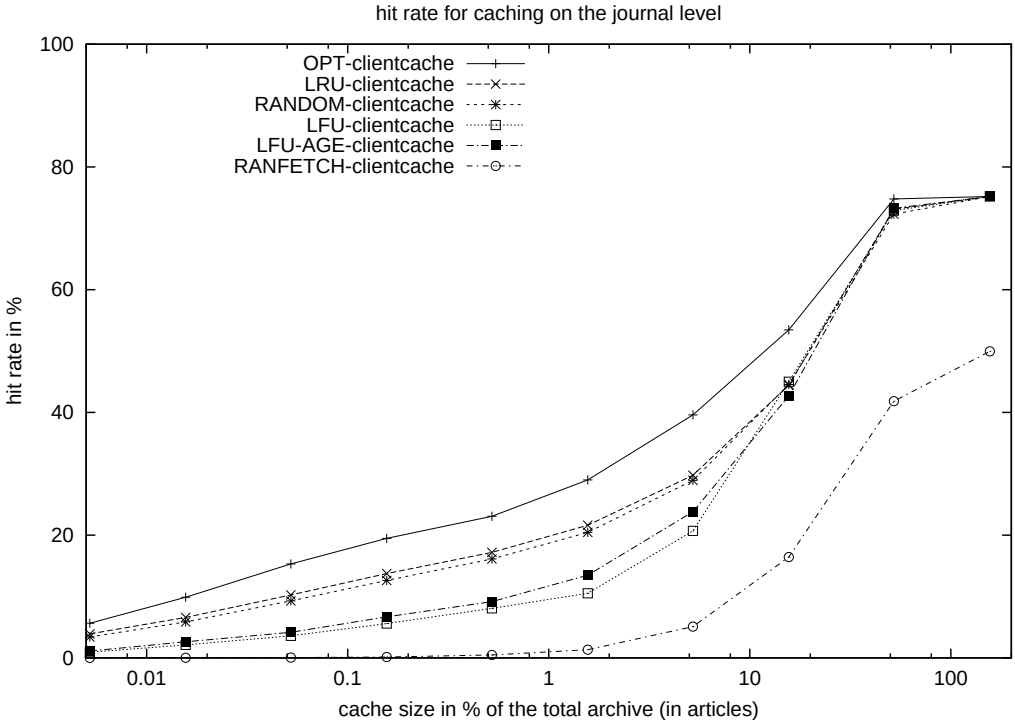

Fig. 8. Comparison between the OPT, LRU, RANDOM, LFU, LFU-AGE and RANFETCH replacement strategy on the journal level.

of articles of a journal, but spread out evenly over the whole article archive, if a randomly selected chunk is not yet present in the cache. If the accesses between articles from the same journal are independent, then RANFETCH should perform similarly to the other strategies.

Since this is not the case, spatial locality is present and could be used for caching strategies. In practice, it is hard to make this efficient, however, because to achieve reasonable hit rates – say 40% – one needs a cache size comparable to more than 10% of the archive. In previous papers we claimed [1] that prefetching can be used to achieve at least comparable performance and verified this by using a prototype [21].

4.3 Reasons for limited cache performance

Central caching is not effective, especially at the article level. Since many different caching strategies performed quite poor, this last part of the paper focuses on the reasons. Doing simple counting of the accesses, again assuming a perfect client cache as filter, the following numbers shown in Table 1 can be found.

A vast majority of articles accessed are one-timers. These articles should not

Table 1
Distribution of Accesses:

	articles	% of total	% of Multi-timers
one-timers	187 456	84.97%	
two accesses	24 287	11.01%	73.24%
three accesses	5 637	2.56%	17.00%
four accesses	1 761	0.80%	5.31%
five and more	1 475	0.67%	4.45%
multi-timers (MT)	33 160	15.03%	100.00%
total	220 616	100.00%	

be cached at all, since they will never be accessed again. The group of articles which had 5 or more accesses is quite small, too. Hence this group will not achieve a substantial part of the hit rate either. Most of the hit rate is actually achieved by the second and third accesses to an article. This basically means that already at the first access the caching algorithm has to decide whether the article is going to be a one-timer or a multi-timer. But at the first access there is no history information available, hence all history-based strategies are likely to fail.

Additionally, the time between accesses tends to be very long. Figure 9 shows

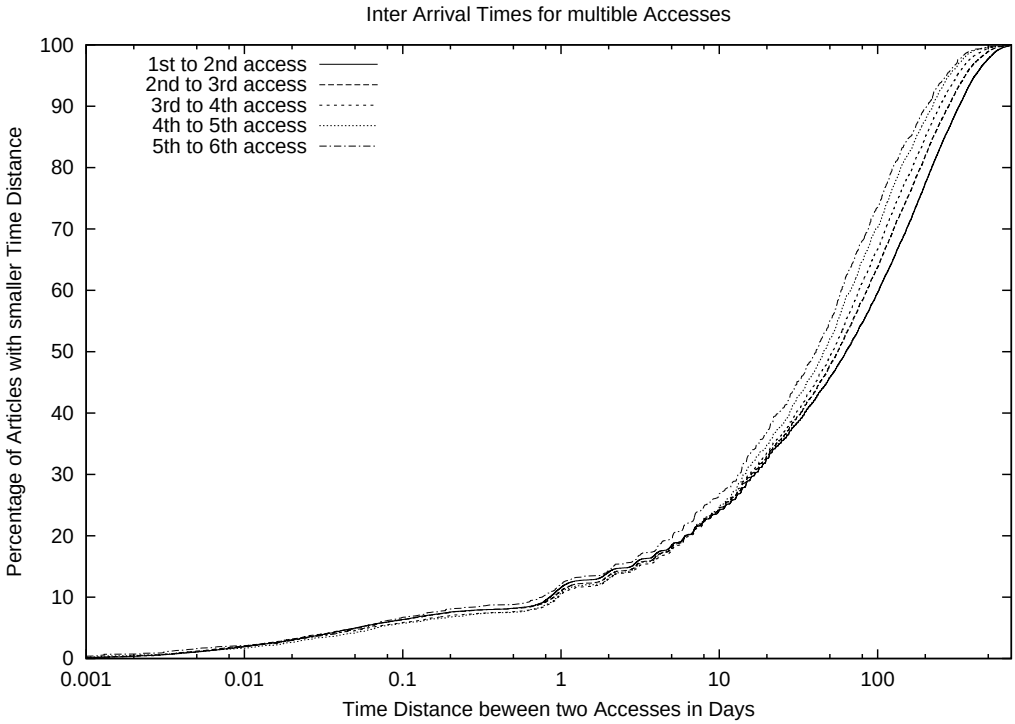

Fig. 9. Inter Arrival Times between Accesses $(n-1)$ and n, $n \in \{2,3,4,5,6\}$.

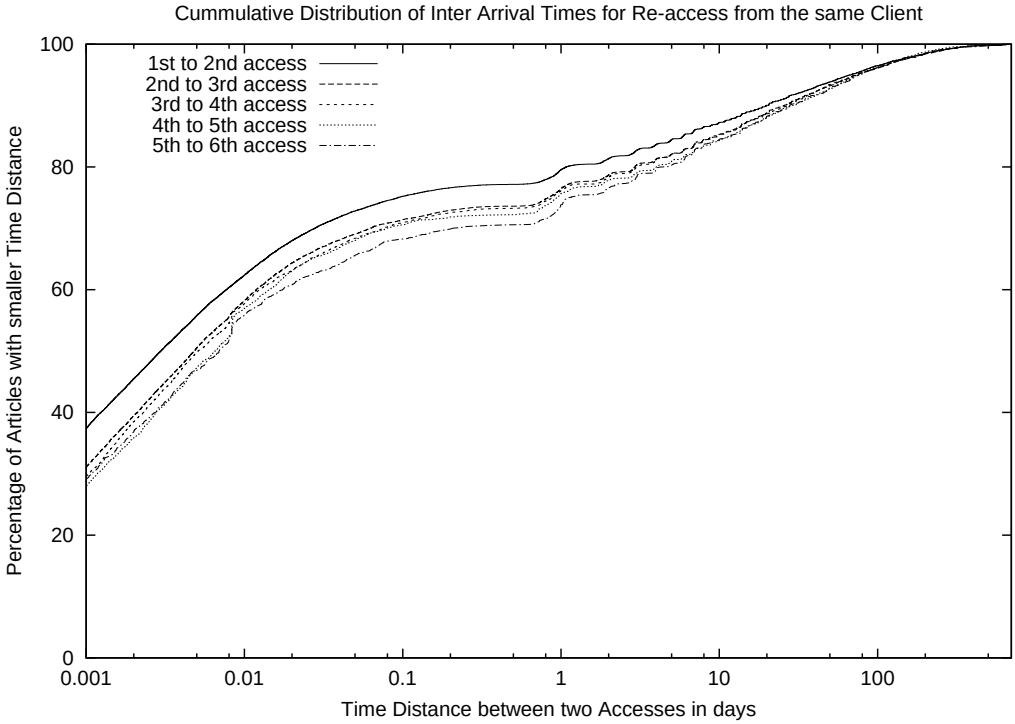

Fig. 10. Inter Arrival Times between Accesses for client side reaccess.

the empirical, cumulative distribution of time between accesses due to *popularity*, while Figure 10 shows a similar distribution for re-accesses from the same clients, matching to Figure 6. Both figures, 9 and 10, present waves around a time distance of one day — those are the day and night patterns of the users living in the same time zone. But the likelihood of re-accesses within a certain time frame is fundamentally different. Re-accesses due to *reuse* occur with a likelihood of around 70% within a day, while re-accesses due to *popularity* occur only with a likelihood of around 10% in the same time frame. This is a verification for the very long time distances of re-accesses in the *popularity* case.

Since the trace is about 700 days long and typically more than 40% of the accesses have more than 100 days between two accesses, one can question if all accesses, which are defined as one-timers in the trace, are really one-timers. Maybe the observation time of almost 2 years is simply too short to observe all re-accesses.

To investigate this further a modified OPT strategy with a time horizon is used. The standard OPT strategy is allowed to look as far in the future as the trace permits, i.e. for the full duration of it. As the simulator proceeds through the trace, the amount of time into the future that is available is decreasing.

For the last experiment a *time horizon* is defined; the time the OPT strategy is allowed to look into the future. Re-accesses that do not occur within this

time are not considered, and hence the corresponding articles will be removed from the cache, even if they would be accessed further into the future again. There are no other changes, so the potential problem will remain that the time horizon is larger than the remaining trace.

Figure 11 shows the results of running the OPT strategy with different time horizons based on articles as access units using different cache sizes. The topmost line characterizes the maximum hit rate achievable with any cache size. Lines below show the limitation by cache size above a certain time horizon.

The main observation is the importance of having a long time horizon in order to achieve a considerable hit rate. But this basically means that the length of the trace has an influence on the hit rate, supporting the hypothesis that the trace may be too short to observe a sufficient number of multi-timers.

But even if a trace of many decades is assumed, which is not yet feasible, and if re-access patterns usable for caching could be found, these patterns would most likely span years. And it is questionable if someone would like to keep articles in a cache over a couple of server and disk generations.

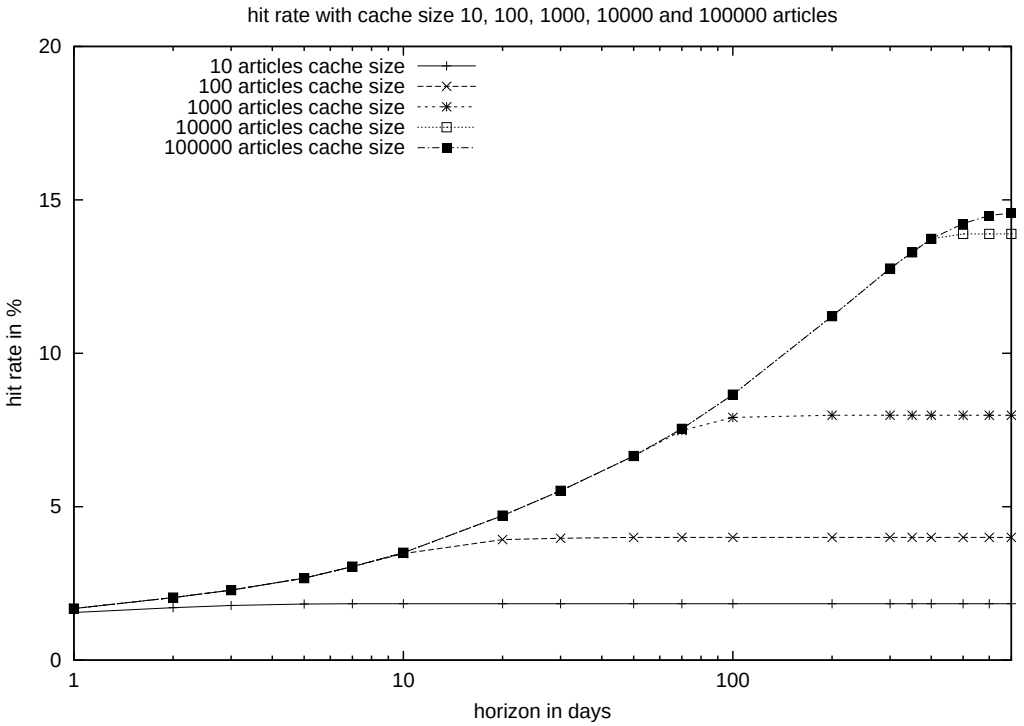

Fig. 11. Maximal achievable hit rate for OPT strategy for time horizons.

5 Concluding Remarks

Improving user-perceived latency through caching occurring when accessing fulltext archives in digital libraries have been investigated. Caching at the client level faces even in the case of the OPT strategy a limited hit rate, but it is both feasible to be implemented with small cache sizes and delivers a hit rate which is acceptably close to optimal on average. Improving web browsers to allow specific URL patterns to be cached forever would allow implementing perfect client caches with small changes to the web browsers.

As for central caching based on popularity, there is spatial locality when caching issues, volumes or journals, but too little to be employed with reasonable cache sizes. Acceptable hit rates around 50% can only be achieved if the cache size is at least 10% of the total archive. As a comparison, prefetching achieves the similar hit-rates [1,21] with much less resource usage. However the results may improve when the central cache is located closer to the servers as this might increase the user groups enough to include people with interest in the same articles.

Two reasons are found for the limited central caching behavior due to popularity. First, it is extremely hard to separate out one-timers from multi-timers, leading to cache pollution. Second, re-accesses are occurring with long time gaps so that articles have to have been kept for "years" in the cache. Despite this, caching of articles in a digital library system can be exploited with some success.

6 Acknowledgment

We are grateful to Prof. Dr. Ulrich Rüde and Prof Dr. Graham Horton from the Department of System Simulation at the Friedrich-Alexander-Universität, Erlangen - Nürnberg, Germany for providing us with infrastructure and a magnitude of insights how to gather and extract relevant data. Dr. Kerstin Hollmann and Fredrik Warg unselfishly took over the job of proof-reading.

This research is supported by the NORDUnet2 initiative.

References

[1] J. Hollmann, A. Ardö, P. Stenström, Empirical observations regarding predictability in user access-behavior in a distributed digital library system, in: Proceedings of the 16th International Parallel and Distributed Processing Symposium, IEEE, Fort Lauderdale, FL, USA, 2002, pp. 221–228.
URL http://ieeexplore.ieee.org/iel5/7926/21854/01016636.pdf

[2] A. Ardö, F. Falcoz, T. Nielsen, S. B. Shanawa, Integrating article databases and full text archives into a digital journal collection, in: C. Nikolaou, C. Stephanidis (Eds.), Proceedings of the Second European Conference on Research and Advanced Technology for Digital Libraries, ECDL'98, Vol. 1513 of Lecture Notes in Computer Science, Springer-Verlag, 1998, pp. 641–642.
URL
http://link.springer.de/link/service/series/0558/bibs/1513/15130641.htm

[3] M. Sandfær, A. Ardö, F. Falcoz, S. Shanawa, The architecture of DADS - a large digital library of scientific journals, in: Online Information 99, Proceedings, 1999, pp. 217–223.

[4] E. G. C. Jr., P. Denning, Operating Systems Theory, Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

[5] E. J. O'Neil, P. E. O'Neil, G. Weikum, The LRU-K page replacement algorithm for database disk buffering, in: Proceedings of SIGMOD '93. 1993 ACM SIGMOD. International Conference on Management of Data, Vol. 22, ACM, 1993, pp. 297–306.
URL http://doi.acm.org/10.1145/170035.170081

[6] E. J. O'Neil, P. E. O'Neil, G. Weikum, An optimality proof of the lru-k page replacement algorithm, Journal of the ACM - Association for Computing Machinery 46 (1) (1999) 92–112.
URL http://doi.acm.org/10.1145/300515.300518

[7] R. Karedla, J. S. Love, B. G. Wherry, Caching strategies to improve disk system performance, IEEE Computer 27 (3) (1994) 38–46.

[8] M. Arlitt, R. Friedrich, T. Jin, Performance evaluation of web proxy cache replacement policies, Performance Evaluation 39 (1-4) (2000) 149–164.

[9] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, Web caching and zipf-like distributions: Evidence and implications, in: Proceedings of the Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE, 1999, pp. 126–134.

[10] A. J. Smith, Cache memories, ACM Computing Surveys (CSUR) 14 (3) (1982) 473–1530.
URL http://doi.acm.org/10.1145/98457.98523

[11] P. J. Denning, Virtual memory, ACM Computing Surveys (CSUR) 2 (3) (1970) 153–189.
URL http://doi.acm.org/10.1145/356571.356573

[12] P. J. Denning, Virtual memory, ACM Computing Surveys (CSUR) 28 (1) (1996) 213–216.
URL http://doi.acm.org/10.1145/234313.234403

[13] W. Effelsberg, T. Haerder, Principles of database buffer management, ACM Transactions on Database Systems (TODS) 9 (4) (1984) 560–595.
URL http://portal.acm.org/citation.cfm?doid=1994.2022

[14] J. T. Robinson, M. V. Devarakonda, Data cache management using frequency-based replacement, Performance Evaluation Review 18 (1) (1990) 134–142.
URL http://doi.acm.org/10.1145/98457.98523

[15] S. Podlipnig, L. Böszörmenyi, A survey of web cache replacement strategies, ACM Computing Surveys (CSUR) 35 (4) (2003) 374–398.
URL http://doi.acm.org/10.1145/954339.954341

[16] P. Cao, S. Irani, Cost-aware www proxy caching algorithms, in: Proceedings of the USENIX Symposium on Internet Technologies and Systems, USENIX Assoc, 1997, pp. 193–206.
URL http://citeseer.nj.nec.com/cao97costaware.html

[17] K. Cheng, Y. Kambayashi, Lru-sp: a size-adjusted and popularity-aware lru replacement algorithm for web caching, in: The 24th Annual International Computer Software and Applications Conference, IEEE Computer Society, 2000, pp. 48–53.

[18] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, E. A. Fox, Removal policies in network caches for world wide web documents, in: Proceedings of the ACM SIGCOMM '96 Conference. Applications, Technologies, Architectures, and Protocols for Computer Communications, ACM Press, 1996, pp. 293–305.
URL http://ei.cs.vt.edu/ succeed/96sigcomm/

[19] M. Busari, C. Williamson, Simulation evaluation of a heterogeneous web proxy caching hierarchy, in: Proceedings of the Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, IEEE Comput. Soc, 2001, pp. 379–388.

[20] I. N. nad B Shneiderman, Flowchart techniques for structured programming, SIGPLAN Notice 8 (8) (1973) 12–16.
URL http://doi.acm.org/10.1145/953349.953350

[21] J. Hollmann, A. Ardö, P. Stenström, An evaluation of document prefetching in a distributed digital library, in: T. Koch, I. T. Solvberg (Eds.), 7th European Conference on Research and Advanced Technology for Digital Libraries, Lecture Notes in Computer Science, Springer, 2003, pp. 276–287.
URL
http://www.springeronline.com/sgw/cda/frontpage/0,10735,5-164-22-7042117-0,00.html

JOCHEN HOLLMANN is a Ph.D. candidate in Computer Engineering at Chalmers University of Technology. His main research interest is data placement and organization in distributed systems based on user behavior. He received a Licentiate of Engineering from Chalmers University of Technology, Sweden and a Masters degree in Computer Science (Diplom) from the University of Erlangen, Germany.

ANDERS ARDÖ is Associate Professor at department of Information Technology, Lund University, where he is managing the Knowledge Discovery and Digital Library Research Group (KnowLib).

His main areas of research interest are digital library development; parallel/distributed computing; focused Web crawling; advanced methods for information discovery and retrieval; and automated subject classification/text mining. He has been head of Lund University Library, development department (NetLab) and Head of research and Principal Scientist at the Technical Knowledge Center and Library of Denmark (DTV). He served as Panel Chair of the European Conference on Research and Advanced Technology for Digital Libraries , ECDL 2003.

PER STENSTRÖM is a Professor of Computer Engineering at Chalmers University of Technology and Deputy Dean of the IT University of Göteborg.

His research interests are devoted to design principles for high-performance computer systems. He is an author of two textbooks and a hundred research publications. He has served on more than 30 program committees of major conferences in the computer architecture field and is an editor of the Journal of Parallel and Distributed Computing. He served as General as well as Program Chair of the ACM/IEEE Int. Symposium on Computer Architecture in 2001 and 2004, resp. He is a IEEE Fellow and a member of the ACM.