Prospects of caching in a distributed digital library

Hollmann, Jochen; Ardö, Anders; Stenström, Per

2003

Prospects of Caching in a Distributed Digital Library

Technical Report 03-04

Jochen Hollmann[†], Anders Ardö[‡], Per Stenström[†]

[†]CSE, Chalmers, 412 96 Göteborg, Sweden
{*joho,pers*}*@ce.chalmers.se*

[‡]Lund University, 221 00 Lund, Sweden
anders@it.lth.se

Abstract

Many independent publishers are today offering digital libraries with fulltext archives. In an attempt to provide a single user-interface to a large set of archives, DTVs-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 delays caused by the network and queuing delays in servers make the user-perceived interactive performance poor.

In this paper, we study the prospects of caching articles at the client level as well as intermediate points as manifested by gateways that implement the interfaces to the many fulltext archives. A central research question is what the nature of the locality is in the user accesses to such a digital library. Based on access logs to drive simulations, we find that client side caching can result in a 20% hitrate. However, at the gateway level, where multiple users may access the same article, the temporal locality is poor and caching is not relevant. We have also studied whether 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 found that spatial locality is quite poor, too. Finally we find that the reason for the cache behaviour is the extremly low accesses frequency observed.

1 Introduction

As computer networks in the form of the World Wide Web went mainstream, at least in the research community, many publishers of research related literature have started to provide their content over the Internet as Digital Libraries. However, while all publishers still run their own web interface and servers as a gateway to give the user access to their literature, there have been efforts to integrate the archives of different publishers into one system. One such initiative is the DADS system [1, 10].

Figure 1, taken from our previous paper [6], shows the schematic architecture of such a distributed digital library (DL) system. On the right side we have the fulltext servers of the publishers containing the text archives, often taking the form of a large number of PDF files. Those servers are located at remote places around the world, which is the reason why such a system will always be distributed. On the left side we have the digital library users, that send queries to access research articles.

A gateway constitutes a confluence point for multiple users to get access to the multiple fulltext servers. The gateway has access to an index containing meta data and references to all fulltexts of all accessible publishers. Meta data includes for example the author, title, publication date and place, keywords and the like. Thus, the speed by which articles can be located is greatly enhanced as the gateway has the meta data necessary to locate the fulltext server of each and every article.

Unfortunately, owing to the geographically distributed system architecture, there is a quite significant network latency between the gateway and the fulltext servers. In a typical query, the user will perceive a latency time that consists of the waiting time for setting up the network connection to the fulltext server and the time to fetch the article in the fulltext server's memory and transferring it to the user including queuing delays in the network and at the servers. Additionally, there are also waiting times associated with starting up a viewer and rendering the document on the users local desktop. These client-local latencies will be disregarded in the rest of the paper.

Of course, if all fulltext archives would be replicated across the gateways as local archives, one could potentially get rid of a significant portion of the network latency. However, as the amount of storage needed to store all articles is increasing, such an approach scales not well and will lead to prohibitive costs for storage.

Caching is founded on the principle of *reference locality*. Reference locality either takes the form of temporal or spatial. Whereas temporal locality states that an access unit is reused, spatial locality means that nearby access units are likely to be accessed. In a DL system, an access unit can be an article,

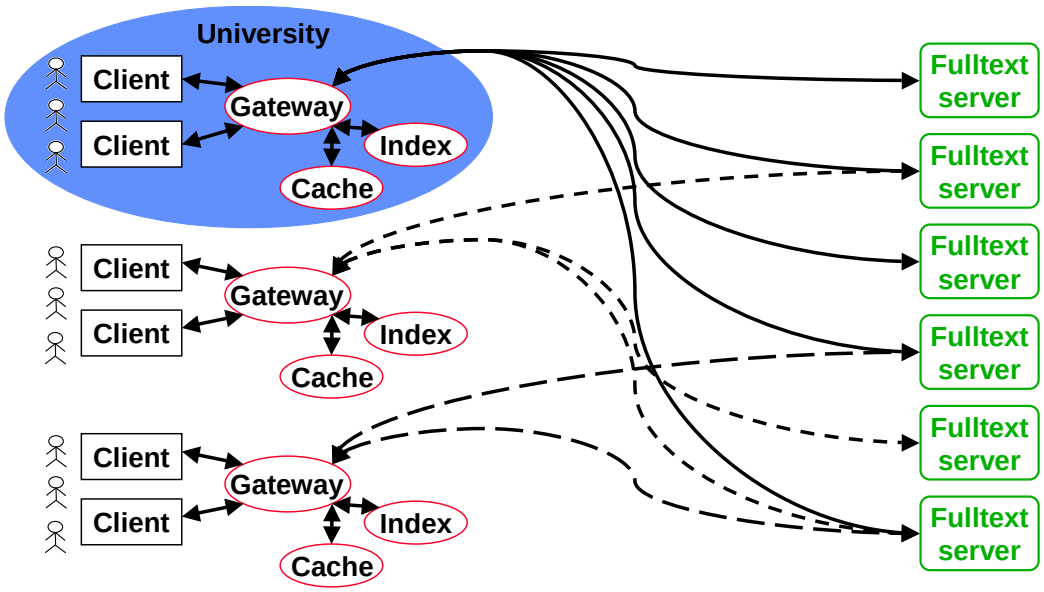

Figure 1: Distributed Digital Library Architecture

an issue, a volume, or a journal.

Consistency, the handling of changing data in a caching system, is an important issue in areas like processor caches or web proxy caching. But published research articles will never change. Hence this simplifies caching, because objects to be replaced can always be removed without further handling and can stay in the cache indefinitely, without getting stale.

In this paper, we study for the first time the limitations of caching DL access units in a distributed digital library. We consider caching at the level of gateways and clients. Whereas caching at clients only can exploit locality based on one user, caching at gateways can exploit locality among multiple users connected to the same gateway.

We have collected a log of user accesses spanning two years. From this log, we can deduce time of the access request and the DL unit accesses at that time. By feeding the access log through a simulation model of a system organization according to Figure 1, we have been able to study the effectiveness of caching. With the model, we have been able to experiment with the size of the cache and the replacement policy. We experiment with various history based algorithms published in the literature [8, 7, 2, 11]. To establish an upper boundary on the effectiveness of caching, we have studied the hitrate for a fully associative cache that implements an optimal replacement algorithm. This algorithm always selects as a target the article in the cache that will be accessed furthest away in the future.

The contributions of this paper are the following main observations. We

find that there exists some locality at the client level and a hitrate of 20% can be achieved with a simple replacement algorithm. Whereas the optimal algorithm was able to identify that there is a fair amount of reuse also at the gateway level (10%), most of the published history-based replacement algorithms like LRU, LRU-K [8], MFU [11] and LFU [11] with its variant LFU-AGE, did not manage to capture this. In fact, the best of them did only marginally better than a replacement algorithm that randomly selects a target. We also studied to what extent spatial locality in the access stream can be exploited to improve performance of caching in DL systems. For example, if several article accesses are collocated in the same issue, volume or journal, it makes sense to cache a larger granularity such as issue, volume or journal. Our findings are quite negative; while some spatial locality was observed, it seems not easy to exploit it.

Looking for the fundamental reasons for the poor caching behaviour at the gateways we found that most DL units are simply not reaccessed within a reasonable time frame. In fact we found indicies, that our observation time of two years were too short to capture the low frequency accesses.

As for the rest of the paper, in Section 2 we introduce caching strategies used for our studies and put them also in the context of other caching areas, in particular web caching. In Section 3 we present our analysis method and show our analysis results in Section 4. Finally we summarize and conclude with obtained results.

2 Caching strategies

2.1 Reference locality

For caches to be effective, the reference stream, in our case user accesses to fulltext servers, must exhibit locality either in the temporal or in the spatial domain. In the following, we put these concepts in perspective of accesses to a Digital Library system.

- *temporal locality* is based on the observation that in many systems re-accessed objects are accessed shortly after they have been accessed. As an example the content of a variable within a computer program may be accessed very often within a subroutine, hence keeping it in the cache during this time makes sense. For digital libraries, this would mean that for example a document is read often within a short period of time and then it will not be accessed for a long period.

- *spatial locality* is based on the observation that accesses are not spread out evenly but instead the probability to access objects closed by is much higher. An example where this property is used would be paging as done by an operating system. Since most programs keep variable locations within a limited region, paging in a page does not only serve a single variable accessed, but all close by variables, too. 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 those objects can potentially serve upcoming accesses to other documents within the same object.

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

Of course caches only improve the system if objects are accessed multiple times. In the following, we will refer to objects that are only accessed once as *one-timers*. The opposite, objects, which are accessed more than once, are called *multi-timers*. While it is pointless to cache one-timers, they can in fact displace objects that will be accessed in the near future.

Re-accesses to the same access object have their origin 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 reuses the document. We will call this type of re-access *reuse*. Another possibility is that different clients access the same object. Since we assume that client machines are independent and used by different users, re-accesses of this type must be caused by *popularity* within a user group.

In the following, we will consider user accesses to articles, issues, volumes, and journals to study how significant spatial locality is. While the sizes of these ojectss may vary, our baseline assumption is that all articles have the same size. Hence, we will estimate cache sizes based on a unity size of an article.

The key question raised in this research is whether there is enough exploitable locality, temporal or spatial, to make caching feasible in a digital library system. To study temporal locality, we next review some published history-based replacement algorithms that will be used in the experiments later in the paper.

2.3 Replacement strategies

The design space of profitable caching strategies is between two only 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 in 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, when they experience their last hit. This can reduce the computation time of the replacement algorithm.

In case of caching on the article level, where always only one article is removed to make space for another one, OPT must be the optimal strategy. 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. Now assume that there is no space left in the cache and a not yet present object B is accessed right now. We know that the following access to B will take place at t_B and $t_B < t_A$. If we do not replace A by B, we will have a miss now, at t_B and a hit at t_A. If we replace A by B we will have a miss now, a hit at t_B and a miss at t_A. Hence this switch makes things not worse. But the second approach has the advantage, that there may be hits to B in the time between t_B and t_A, which can improve the hitrate.

On the issue, volume and journal level, OPT applies the same scheme. But since those objects differ considerably in size, this strategy will only be a lower bound for the optimal strategy. We may lose performance when moving in an object replacing two others having only one hit. In this case we have traded two hits against one, which is sub optimal. Also, in case an object is larger than the cache size, it will not be cached at all.

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 randomly selected objects. Strategies which do better, should be considered; strategies which do worse, should not be applied.

The well know strategy least recently used (LRU) will be used in most of our experiments as a simple strategy based on temporal locality. We have implemented it as follows. On a miss, we check the available space in the cache. If the object does not fit in, we search for the object with the oldest access time stamp and remove it. Then we start over. Once space is available, we move the object into the cache. LRU is widely used and often represents the baseline to compare improved caching strategies.

LRU considers only the last access. O'Neil et al. [8] 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 with the (k-

1)-age and so on. O'Neil et al. [9] did also prove that LRU-K is optimal on the available information. It's 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 a second chance, by being moved to the unprotected area. This strategy is advantageous, if many objects are accessed only once, because those objects will never make it to the protected segment, which will be reserved for objects with more accesses. Arlitt found, that SLRU works best for web caching, if 60% of the cache is protected area [2].

The Most Frequently Used (MFU) and Least Frequently Used (LFU) algorithms, both mentioned by Vakali[11], represent two more history based approaches. MFU removes those objects, which have a high access frequency. All objects above a certain MFU threshold are removed from the cache. Hence it 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 we implemented it, reduces the access counts in regular intervals by 10% to give older accesses less weight, which had no accesses recently. Vakali [11] gives a comprehensive comparison and extension of the above strategies and variants of them in the context of web proxy caching.

One difference between web caching and for example a processor cache is, that object size varies a lot. A single access can transfer a few bytes or up to hundreds of megabytes. Therefore for the performance of web caching there are two measures: The *hitrate*, which is the relative number of hits to objects of the total number of object accesses and the *byte hitrate* which is the number of bytes served from the cache compared to the total number of bytes requested. Williams et al. [12] give an overview of basic removal algorithms in this context.

In order to increase the hitrate, it is useful to favor small objects, because then more objects fit into the cache having a similar effect as using a larger cache. For the byte hitrate it is better to keep large objects, because a re-transfer would reduce the byte hitrate considerably.

Efficient algorithms for web proxy caches like GreedyDual-Size (GDS) proposed by Cao and Irani [4] or LRU-SP [5] by Cheng and Kambayashi are optimized for both hitrate and byte hitrate and are therefore less relevant for us. Busari et al. [3] have studied the effects of hierarchical web proxy

caching. Their approach comes closest to our studies.

3 Analysis Method

We now move on to the experimental approach used to derive our results.

3.1 Logging Infrastructure

Our analysis is based on log files from DTVs Article Database Service (DADS) [1, 10], a digital library for journal and conference articles developed at the Technical Knowledge Center (DTV) of the Technical University of Denmark. DADS is implemented according to Figure 1. The gateway is implemented using a HTTP server running scripts, which communicates with an index database performing the searches on the users demand. DADS index database contains around 20 million entries for articles from all major scientific publishers.

Once the user finds an article, it is fetched from the fulltext server, if the paper is available online. The user environment is a standard web browser, which interacts with the gateway HTTP server. We had no influence on the web browsers, hence we do not know if and what kind of caching is implemented at this level. During the time we recorded the log files, DADS itself did not have any cache implemented.

We have captured all incoming HTTP requests recorded from a gateway. The log files contain, for incoming fulltext accesses, the client's IP-address, a time stamp, the article number, the status code as well as more data, which is not of interest in our study.

The log files gathered are from the gateway used primarily by the Technical University of Denmark including DTV. For our analysis, we have removed all requests from domains outside the university. We base our studies on the time frame from March 2000 to January 2002, totaling 700 days.

For this study we are only interested in article access patterns. In particular we are interested which articles the users try to access as well as when this happens. 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 download from the fulltext archive. Because fulltext is only available from a limited number of database records (approximately 20%), we have decided to use accesses to the above described delivery page instead of the actual fulltext accesses. This is reasonable, because the user has to pass this decision page for every single real fulltext access and does not know in advance, if the fulltext will be accessi-

ble electronically or not. This way we have access traces available 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 [6].

The gathered log files do not contain any information about the issue, volume or journal where the accessed articles were published. Therefore we have taken this data from the index database available within the DADS system.

Combining the information from the log files with the index database allows us to create a suitable trace. Such a trace contains a time stamp for any accesses as well as a unique identifier for the article, issue, volume and journal, the size in articles for each unit.

3.2 Simulation

In order to study the user access patterns and the effects of different caching strategies we use a trace driven simulation model based on the traces described above.

Our simulator takes a trace as input and reads through it, line by line, executing the caching strategies. The cache is implemented as a set of access unit IDs. The cache also counts how many articles it contains at every moment. Besides this, for every unit accessed, there is the possibility to store additional data, like the last access time, 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 like future accesses are recorded on a per unit base, so that the OPT strategies have this data available later on. The simulator has also 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.

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 bring in a new large unit. This is done recursively by calculating new targets to be removed, until the cache can take in the new object. An exception is the case where the particular unit to be cached does not fit into the cache at all. In this case no replacement happens and the unit is not brought into the cache.

The output of a simulation run will be a hitrate, computed over the total time of the trace.

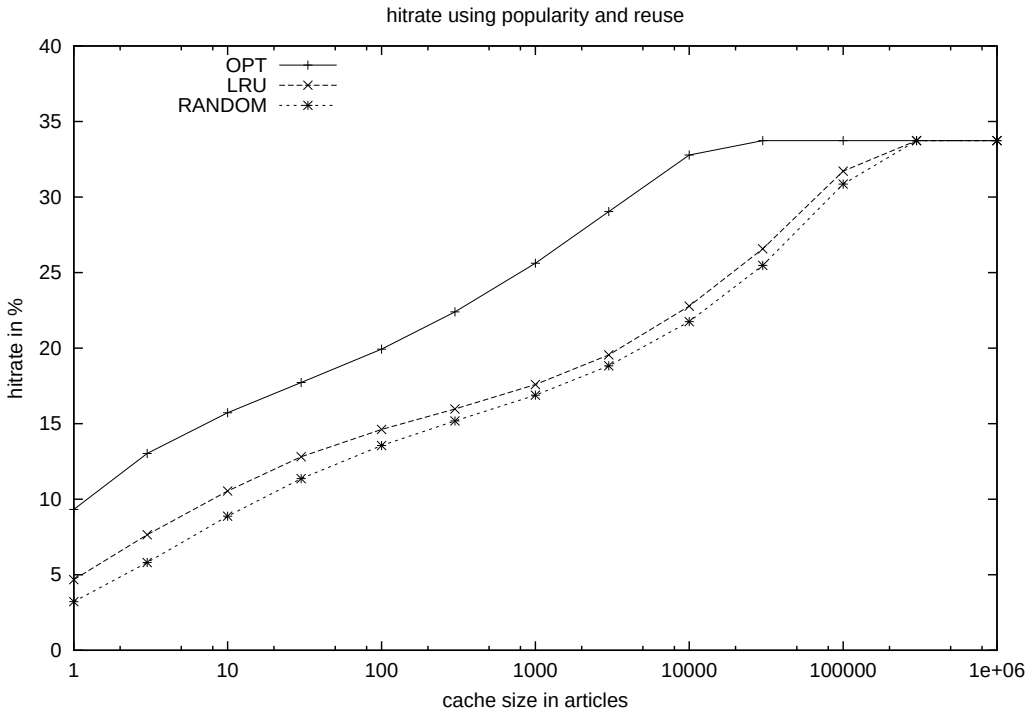

Figure 2: Hitrate achieved at a central proxy cache running OPT, LRU and RANDOM taking advantage of popularity and reuse, because no (perfect) client caching was simulated

4 Analysis Results and Discussion

4.1 Reuse versus Popularity

In a first experiment we study the effects of client side caching versus central proxy caching. Ideally we would like to have a trace of a system where no caching is implemented at all. But in practice this is not the case. Most users have already a local client cache integrated into their web browser, which we assume to keep recently accessed web documents in order to reduce latency. Those caches influence cache hits due to reuse, but not due to popularity.

Figure 2 shows the simulation results for implementing a central proxy cache, taking advantage of both reuse and popularity. On the x-axes, using a logarithmic scale, we have varied the cache size from one article to a million articles, which represent 5% of the total archive size. The curves show the hitrate for OPT, LRU and RANDOM. We can see that OPT does a lot better than both LRU and RANDOM, with LRU being only slightly better than RANDOM. We can also observe that a cache size of a little bit above

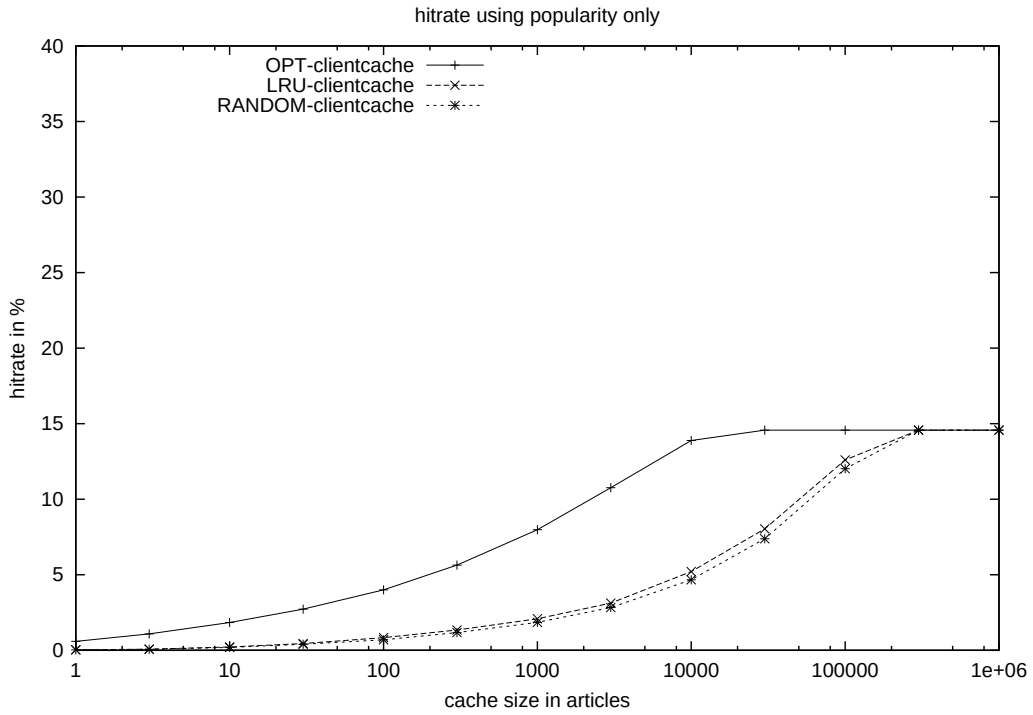

Figure 3: Hitrate achieved at a central proxy cache running OPT, LRU and RANDOM with perfect client caching taking only advantage of popularity

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. Note that this point depends heavily on the size 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 to 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 we did an experiment, where we assumed perfect client caches. 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 document.

The results are shown in Figure 3. While the shape of the curve has not changed for cache sizes above 10000 articles, we have lost more than half of the maximal achievable hitrate due to the filtering of accesses by the perfect client caches. Also, the results of LRU and RANDOM got closer to each other. We have done this experiment with other history based strategies like

LRU-K, MFU etc, but all those results were worse than RANDOM. This may be a hint that temporal locality present is mainly caused by reuse.

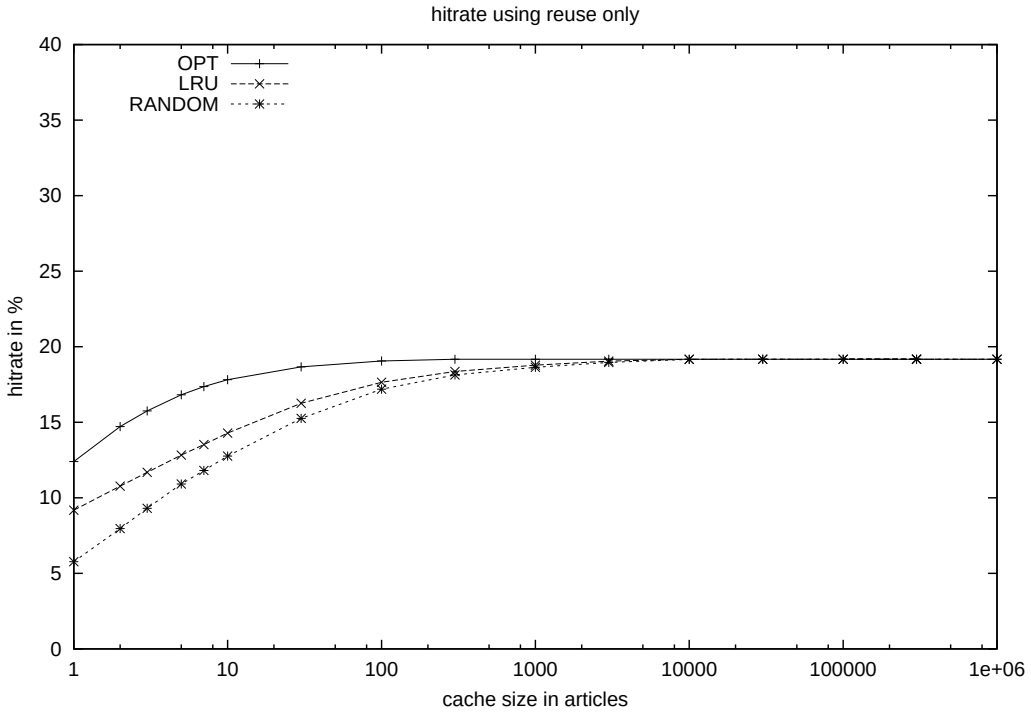

Figure 4: Hitrate average at a client side cache running OPT, LRU and RANDOM taking advantage of reuse only

To answer this question we have done a third experiment, simulating client caches. We used the full trace and split it into multiple traces, one per client machine. Those sub-traces were then used to drive the simulation model starting for each sub-trace with an empty cache. The hitrate observed is the average hitrate achievable and is shown in Figure 4. Note that the cache sizes naturally are much smaller per client, even if the sum of all cache sizes may be larger than in the previous case.

This time we do observe more temporal locality, because LRU behaves a lot better than RANDOM. In practice those results may be even better, because our traces are, at least partly, filtered by browser caches in place. From the observations we can say that most of the temporal locality is actually due to reuse and not due to popularity.

We also can observe that the cache size for each client to achieve a considerable hit-rate can be small when compared to a central approach. Since client side disk space is cheap, ideally a digital library application would keep

every accessed fulltext article on the client side. This would require to have a cache size of 10000 articles (collected over almost 2 years), which would be approximately 10GB, for the machines accessing many articles. But for most machines this would only require 100 articles, approximately 100MB, easily implementable with todays 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 classes with different strategies. If a regular expression, matching all fulltext archives, would prevent applying a strategy with expiry, such a perfect cache would be implemented.

4.2 Caching based on popularity

For the remaining part of this paper we will only consider the trace, which was filtered by a perfect client cache, removing all occurrences of reuse. We have already seen in Figure 3, that temporal locality gives not much advantage over a random caching scheme centrally. Hence we have tried to explore spatial locality instead changing the objects cached to larger entities.

Figure 5 shows the theoretical possibilities of employing spatial locality, comparing the OPT strategy on the article, issue, volume and journal level. Note that this study does not consider the bandwidth used and latency experienced for fetching whole issues, volumes or journals.

We can observe that the best strategy seems to be based on journals. Hence we show in Figure 6 a comparison on different strategies running on the journal level. To see if we can really employ spatial locality or whether the higher hitrates achievable are only based on the fact, that more articles are present in the cache we have implemented a strategy called *RANFETCH*. RANFETCH brings in the average amount of articles of a journal, but evenly spread out 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 should be 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 not a good choice, however, because to achieve reasonable hitrates – say 40% – one needs a cache comparable to more than 10% of the archive. In a previous paper [6] we reported that prefetching can be used to achieve at least comparable performance. In an upcoming paper we will verify the effectiveness of prefetching using a prototype.

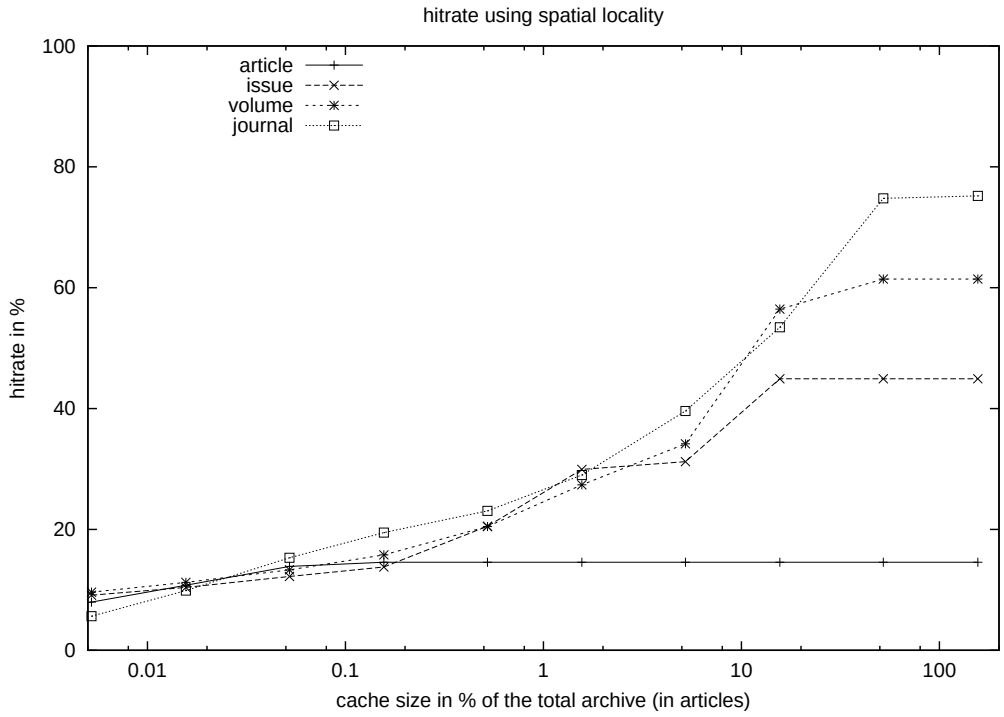

Figure 5: OPT strategy caching articles, issues, volumes or journals

4.3 Revisiting the caching problem or the problem of observability

We have seen that central caching does not work well, especially on the article level. Because we have tried out many different, poorly performing caching strategies, we are in the last part of this paper looking for the reasons. Doing simple counting of the accesses, again assuming a perfect client cache, we found the numbers shown in Figure 7

As we can see the vast majority of articles accessed are one-timers. Those articles should not be cached at all, since they will never be re-accessed again. The group of articles which had 5 and more accesses is also extremely small. Hence this group will not achieve a substantial part of the hitrate either. Most of the hitrate is actually achieved by the second and third accesses to an article. This basically means that already, on the first accesses, the caching algorithm has to decide whether the article is going to be a one-timer or a multi-timer. But on 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 8 shows the empirical, cumulative distribution of time between accesses. Since

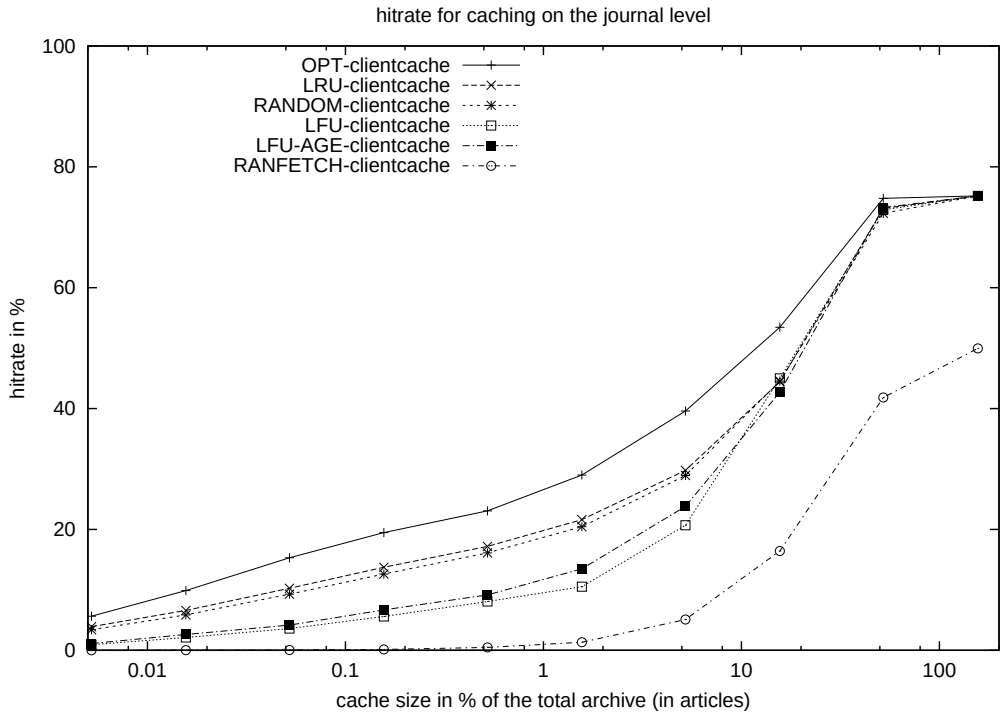

Figure 6: Comparison between the OPT, LRU, RANDOM, LFU, LFU-AGE and RANFETCH replacement strategy on the journal level

our trace is about 700 days long and typically more than 40% of the accesses have more than 100 days between two accesses, we can question if all accesses, which we called one-timers, are really one-times. Maybe our observation time of nearly 2 years is simply too short to observe enough re-accesses.

To investigate this deeper we have modified our OPT strategy to have a time horizon. 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. This means that in the beginning of the trace OPT will, in our case, have 700 days, while as we proceed the amount of trace available would decrease all the time.

For the last experiment we want to define a *time horizon*, the time the OPT strategy is allowed to look into the future. Re-accesses not occurring 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 chances, so the potential problem will remain that the time horizon is larger than the remaining trace.

Figure 9 shows the results of running the OPT strategy with different time horizons using different cache sizes. The topmost line characterizes

	articles	% of total	% of MT
one-timers	187456	84.97%	
two accesses	24287	11.01%	73.24%
three accesses	5637	2.56%	17.00%
four accesses	1761	0.80%	5.31%
five and more	1475	0.67%	4.45%
multi-timers (MT)	33160	15.03%	100.00%
total	220616	100.00%	

Figure 7: Distribution of accesses

the maximum hitrate achievable with any cache size. Lines below show the limitation by cache size above a certain time horizon.

We can observe, that it is very important to have a long time horizon to achieve a considerable hitrate. But this basically means that the length of the trace has an influence on the hitrate, supporting the hypothesis that our trace may be too short to observe enough multi-timers.

But even if we assume having a trace of many decades, which is not yet feasible, and if we find re-accesses patterns usable for caching, those 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 over the years.

5 Conclusions and alternatives

We have investigated the possibilities to improve the user-perceived latency to access fulltext archives in digital libraries using caching. We found that caching at the client level faces even in the case of the OPT strategy a limited hitrate. But it is both feasible to be implemented with small cache sizes and delivers a hitrate 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, we have found that there is spatial locality when caching issues, volumes or journals but too little to be employed with reasonable cache sizes. Acceptable hitrates can only be achieved if the cache size is at least 10% of the total archive. We have already found hints that prefetching achieving hit-rates above 50% is a good alternative [6] and will show that those results can be verified in practice in an upcoming paper.

We found two reasons for the bad central caching behavior: It is extremely hard to separate one-timers from multi-timers, which leads to cache pollution.

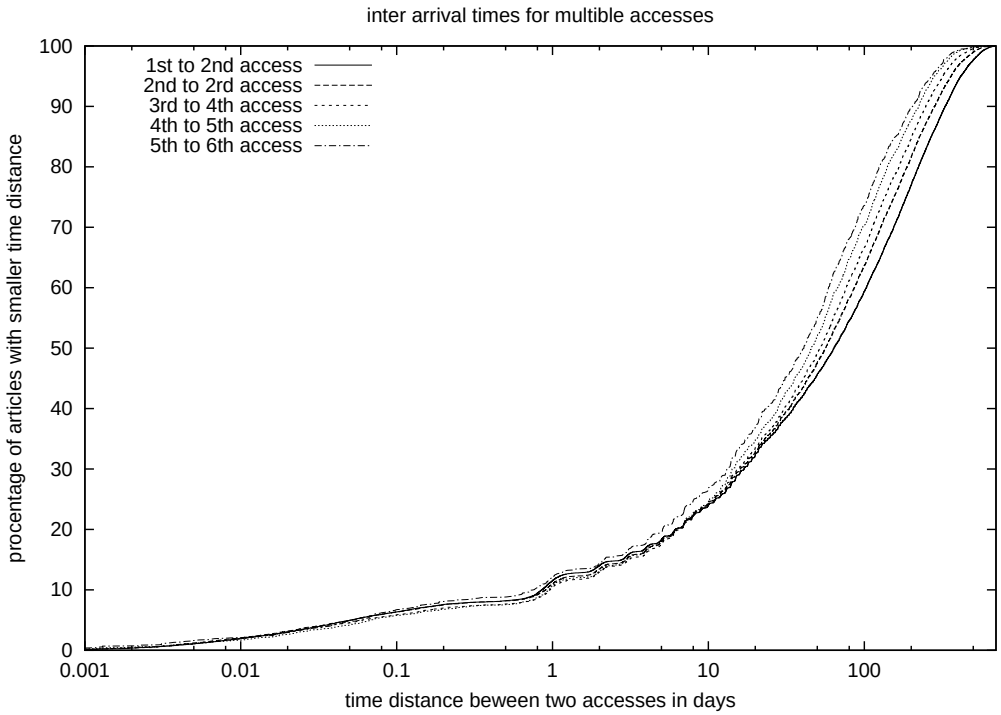

Figure 8: Interarrival times between accesses $(n-1)$ and n, $n \in \{2,3,4,5,6\}$

And re-accesses are occurring with very low frequencies, so that articles have to have been kept for "years" in the cache. Apart from the technical issues, there will also by copyright issues involved.

When building a digital library our recommendation is to use client caching in combination with prefetching at the gateway level, since prefetching is cutting latencies for both one- and multi-timers independent of access patterns. Using different strategies at different hierarchy levels, as suggested by us, is in a way similar to the finding of Busari et al. [3], who suggested using a size-base caching strategy on one and a history based strategy on the other level.

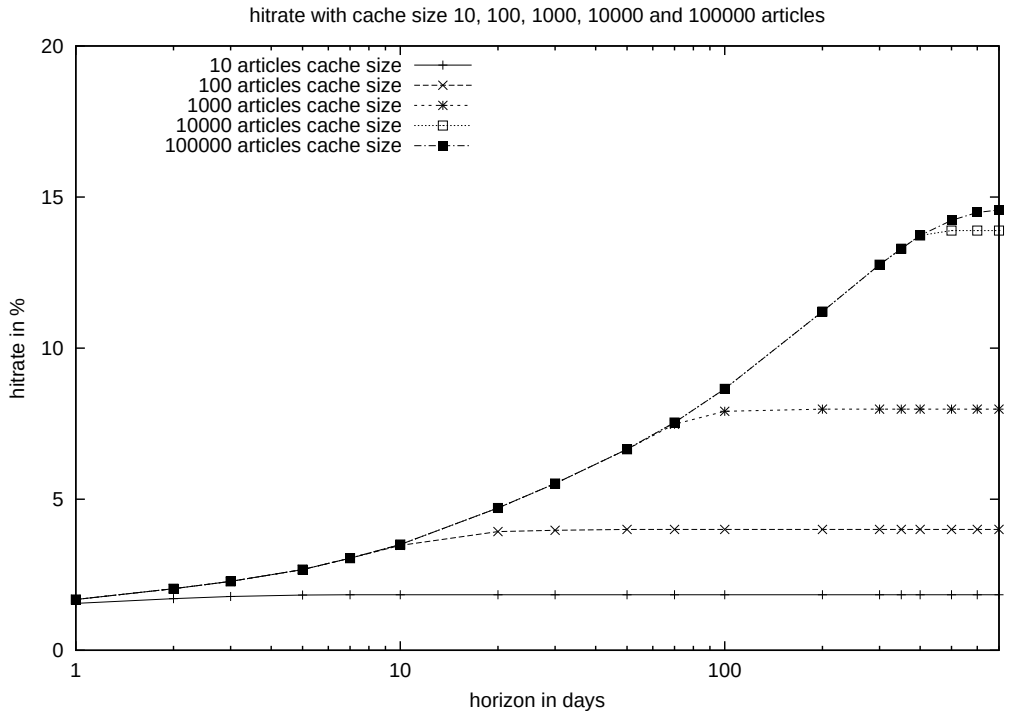

Figure 9: Maximal achievable hitrate for OPT strategy for time horizons

References

[1] A. Ardö, F. Falcoz, T. Nielsen, and S. B. Shanawa. Integrating article databases and full text archives into a digital journal collection. In C. Nikolaou and C. Stephanidis, editors, *Proceedings of the Second European Conference on Research and Advanced Technology for Digital Libraries, ECDL'98*, volume 1513 of *Lecture Notes in Computer Science*, pages 641–642. Springer-Verlag, sep 1998.

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

[3] M. Busari and 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*, pages 379–388. IEEE Comput. Soc, Aug 2001.

[4] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 193–206. USENIX Assoc, Dec 1997.

[5] K. Cheng and Y. Kambayashi. Lru-sp: a size-adjusted and popularity-aware lru replacement algorithm for web caching. In *The 24th Annual Interna-*

tional Computer Software and Applications Conference, pages 48–53. IEEE Computer Society, Oct 2000.

[6] J. Hollmann, A. Ardö, and 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*, pages 221–228, Fort Lauderdale, FL, USA, April 2002. IEEE.

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

[8] E. J. O'Neil, P. E. O'Neil, and 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*, volume 22, pages 297–306. ACM, 1993.

[9] E. J. O'Neil, P. E. O'Neil, and G. Weikum. An optimality proof of the lru-k page replacement algorithm. *Journal of the ACM - Association for Computing Machinery*, 46(1):92–112, 1999.

[10] M. Sandfær, A. Ardö, F. Falcoz, and S. Shanawa. The architecture of DADS - a large digital library of scientific journals. In *Online Information 99, Proceedings*, pages 217–223, dec 1999.

[11] A. Vakali. Proxy cache replacement algorithms: A history-based approach. *World Wide Web*, 4(4):277–297, 2001.

[12] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and 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*, pages 293–305. ACM Press, 1996.