Enhancing the Web's Infrastructure — From Caching to Replication

Michael Baentsch, Lothar Baum, Georg Molter, Steffen Rothkugel, Peter Sturm
Systemsoftware Research Group, CS Department
University of Kaiserslautern
D-67653 Kaiserslautern, Germany
{baentsch, lbaum, molter, sroth, sturm}@informatik.uni-kl.de

One of the most obvious challenges facing the Internet can be observed every day when surfing the Web. As people start their working day, the application-level performance of the Web decreases drastically as the infrastructure of the WWW had not been designed for the huge amount of users. Caching is a standard solution for this kind of problem, which is why the current situation of caching in the WWW forms the initial part of this paper. As several issues are shown to remain, the more advanced concept of replication and naming is presented afterwards. It is shown that the latter not only helps to reduce the overall bandwidth as well as user-perceived latency, but that a more fault-tolerant and more evenly balanced system results. The presented system, CgR/WLIS, *is shown to be realizable in a transparent and backward compatible manner as compared to the installed base of WWW software.*

TODAY'S WEB

A few years ago, the notion of an *Information Superhighway* has been envisioned. Today, it is beginning to become a reality—including traffic jams as on conventional (motor) highways. The fortunate message is that anyone having access to the Internet may potentially profit from the vast amount of information available. Due to the fact that most members of the still growing Internet community surf the Web at daytime, we are faced with the problem of network congestion at those times. Low bandwidth, especially for intercontinental links, is the immediate consequence. The user itself experiences high latencies. The problems become even more serious when looking at the *hot pages of the day*. One major difficulty arises due to the most common case that all requests of all users have to be served by a single machine. In this single-server approach, the Web server becomes a bottleneck. Even worse, if this machine fails, the access path to the information is lost. There is a single point of failure. Therefore, solutions for a couple of interrelated problems have to be come up with in order to preserve the usability of the World Wide Web:

- Decrease document retrieval latency
  While this is the most beneficial goal from the users point of view, it is the most complex to achieve.
- Reduce the amount of data transferred
  This is the primary goal for anyone having to pay for network usage.
- Increase document availability
  According to the problems of the single-server approach already discussed, we have to look for strategies to distribute documents among several servers.
- Retain transparency
  Requests for a document have to be handled fully transparent without the user being aware of the existence of an underlying infrastructure. Thus, the distribution of documents has to be hidden from the Web user.
- Balance the bandwidth usage
  Relinquish bandwidth at peak-times by shifting network access to periods of low traffic. The users surfing at daytime immediately benefit from the higher bandwidth.
- Backward compatibility
  A rather pragmatic issue to be followed is to remain compatible with the existing Web. Currently installed software must continue to function properly.

A very popular and widely accepted approach to address at least some of the problems stated above is the usage of caching proxies. The next chapter looks at what caching can accomplish, which issues remain to be solved and in which way. Afterwards, a discussion of another more current solution developed at our research group is introduced.

CACHING

The rationale behind caching is the idea to bring the data as close to its consumer as possible in order to minimize access times. Primarily, this technique has been applied to memory hierarchies employed in nearly every computer. While the processor's registers are located at the top of the memory hierarchy, the main memory typically can be found several levels below, followed by the disks. The closer one gets to the actual job of processing the data, the smaller the available memory becomes but the faster it can be accessed. The advantages of caching rely on the principle of reference locality — data that has just been used is assumed to be accessed again in the near future. The first time the data is requested, it has to be loaded from the memory level where it is stored, for example from a file residing on disk. According to the notion of reference locality, the data may be cached. From now on, further requests can be satisfied directly from the cache. Thus, performance gains can be obtained up from the second request for the same data. Because of space constraints, the hard job in any caching system is to decide *which* data to cache. A severe problem known as *cache coherency* arises if the data is changed on the originating level without recognition by the cache. Caching strategies have to tackle this issue as well.

When considering caching in the WWW, the notion of a memory hierarchy could be extended when interpreting Web servers as another, external level. As a first approach of caching, a client-local disk- and/or in-memory-cache has been established in Web browsers. However, caching only makes sense when the same document is requested more than once. Thus, the profit of caching can be increased noticeably when the cache is shared among several users. The idea of caching proxies arose [15]. A proxy acts as a mediator between the user's machines and the outside world. From the user's point of view, the proxy acts like a Web server. Each request is sent to and answered by the proxy, which in turn may forward the request to the originating server, acting itself as a client from the server's point of view. Therefore, the proxy has an ideal position to include another cache. It is shared among several users serving all their requests, so the probability of a document to be accessed more than once increases. As a side effect, this machine can act as a firewall, permitting local machines to access the outside world in a controllable manner.

Another special aspect of caching within the Web is the divergence in profit that can be obtained by different pages. In a classical cache, items can be treated equally, making it easy for a *replacement policy* to select an item to be overwritten within the cache. There are several additional properties to be considered when caching is applied to the Web. The first one is the size of documents to be cached. In case of a disk cache, there is an upper limit defined for the size of the caching area. Now assume a full cache and a request for a big document we want to store in the cache. The burden placed on the replacement policy drastically increases. Is it more sensible to replace a single big document than several smaller ones? There are many factors to be considered besides the size of the document. Among them are the pattern of document access and the time required to reload the documents. This loading time in turn depends on the origin of the document. Data that has to be transferred over international links typically needs longer to be retrieved than information from servers in the same country. Because bandwidth varies through the course of a day, an additional indeterminism and difficulty is introduced.

Furthermore, the cache coherency problem already stated above arises. With respect to the Web, this problem is often referred to as *document staleness*. Cached documents may change on the originating server. Since caching proxies do not know about changes, any further request satisfied from the cache would deliver out-of-date information. Unfortunately, the expiration date potentially included in each document transferred through the Hypertext Transfer Protocol (HTTP, [2], [7], [11]), is still used by too few Web servers. As a consequence, caching proxies have to come up with heuristics to determine document staleness. One possible heuristic is known as *time-to-live* (TTL). The rationale behind this approach is the experience that documents that have not been modified recently will probably not be changed in the near future. In contrast, recently changed documents should not be kept too long in the cache without checking their consistency. Based on the date of the last modification included in every reply from a Web server, a timing-window called TTL is associated with each document as it is put into the cache. At the time a document is requested, its TTL is checked. An access inside the timing window is served directly from the cache, quietly assuming the document to be consistent. After TTL has expired, a conditional reload is performed. The originating server will answer either with the new document or a special reply indicating unchanged data.

Advantages

For quantifying the profit drawn from caching, the underlying metrics have to be redefined, taking the size of the document into account. The simple measurement of a cache hit where each item cached is considered to obtain the same profit is no longer sufficient. A more sensible approach taking the size of the cached document into account is referred to as the byte hit rate.

Despite all problems of caching within the Web, there are numerous advantages remaining that can be observed. Based on the log files created by specially instrumented proxies of our University as well as of our research group, we performed various traces. The questions to be answered by our investigations included:

- Does reference locality exist with respect to the Web?
- How does the available application-level bandwidth vary on the course of a day?
- Which hit rates can theoretically be achieved?
- What profit is obtained by using currently installed caches?

Reference locality in the Web

Based on the logs of April 96, we summed the number of references of each requested document. A total of 220.000 documents distributed over 11.000 servers have been accessed. As shown in figure 1, only a very small subset of pages incorporates a high number of references while most documents are accessed relatively seldom.

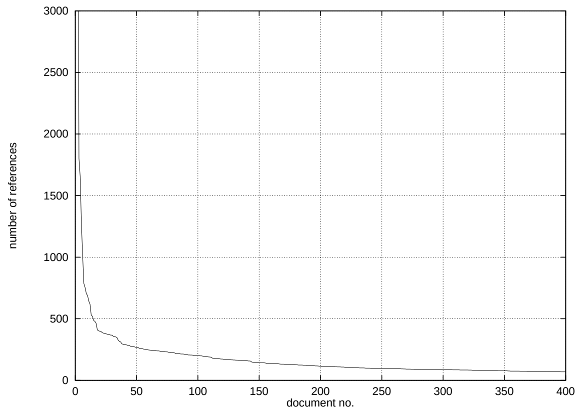

Fig. 1: Reference locality of Web pages

Averaged over 8 months, we found that 14% of all cached pages are responsible for 42% of the cached data while retaining only 7% of the overall disk space. So we can conclude that reference locality does indeed exist with respect to the Web and therefore caching really makes sense.

Variation of the available bandwidth
Exploiting the logs of three months, we calculated the bandwidth from the document sizes and the duration of the transfers. The results are presented in figure 2.

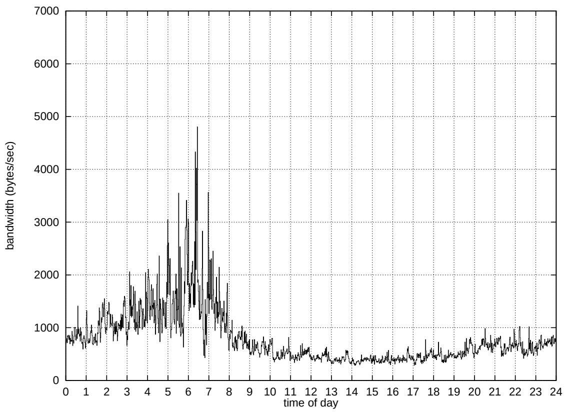

Fig. 2: Available bandwidth

It should be noted that the variation in bandwidth of international links is much more severe than that of national ones. When using transatlantic links from Germany to the US, bandwidth varies between 100 bytes/sec around noon and 5.000 bytes/sec at about 6 a.m., a difference of a factor of 50. Regarding only nationwide transfers, the measurements only vary by a factor of 7 between 1.000 bytes/sec and 7.000 bytes/sec. In contrast, the graph's behaviour remains nearly the same when comparing national and international accesses. The early morning hours appear to be best suited for surfing the Web.

Theoretically achievable hit rates
A challenging point for investigations is to determine which hit rates are theoretically achievable. Based on our log files we performed a simulation using a virtual cache with unlimited size. The logs of April 96 have been applied to warm up the cache, so all documents requested in this month are ready to be delivered directly from the cache. Now we traced the requests of the following month, serving the documents from the cache whenever possible and counting the hits. Any newly accessed document has been added to the virtual cache. Document staleness has been ignored for this investigation. The results are as follows:

Cache hit rate:	56,5%
Byte hit rate:	40,6%
Transferred data:	3.650.950.731 bytes

The term *transferred data* indicates the amount of data transmitted between the cache and the outside Web servers. An amazingly high cache hit rate of 56,5% could have been achieved, while the byte hit rate of 40,6% is significantly lower. From this fact we can conclude that small documents more likely contribute to the cache hits.

As already pointed out, any document stored in the cache has been delivered directly, resulting in 28,2% of stale data to be presented to the user.

Profit of real caching
The logs of the CERN httpd proxy server that was used during the period surveyed have been applied to evaluate the profit actually achieved. There were 500 MB of disk space allocated as caching area. According to the caching policy applied by CERN httpd, on the course of a day *all* requested pages are cached, regardless of the disk space needed. Through nighttime, documents get deleted until a low watermark cache size has been reached. As long as possible, expired pages are selected for deletion. The results as obtained from evaluating the logs are listed below:

Cache hit rate:	21,3%
Byte hit rate:	16,6%
Transferred data:	4.992.987.253 bytes

Note that the stale rate cannot be reported. It is not possible to determine from the logs whether a document served from the cache was stale or not.

Comparing the results with the optimal cache discussed above, we must conclude that real caching is far from being optimal. Only 40% of the theoretically possible byte hit rate has been achieved. Though the advantages gained from caching are clearly recognizable, caching is not able to tackle all the goals for a better Web already mentioned above.

Problems caching does not solve
Referring back to the goals listed in the second section, the first issue is to decrease document retrieval latency. Caching addresses only parts of the problem. The *first* time a document is requested, it has to be retrieved from the originating server. Only beginning with the *second* access the cache may help, thus reducing the total amount of data transferred, which is the second goal mentioned above. From a theoretical point of view it would therefore be fortunate if the first user to request the document could also benefit from a globally knowledgeable cache as 69.9% (according to our measurements) of the pages have been retrieved only once.

Increasing document availability — goal number three — is *not* addressed by pure caching. Even when a document is cached, checking whether it is up to date remains impossible when the originating server is down. The only way out is to deliver the document anyway, despite its possible staleness. In the case the document is not in the cache, it cannot be delivered at all.

When using caching proxies, the user only has to register the proxy at his Web browser. Afterwards, documents are either delivered directly from the caching proxy or they are retrieved from the originating server. In either case, the decision made by the proxy is fully hidden from the user. From this point of view, caching proxies fulfill the fourth goal of transparent access. However, because the proxy uses heuristics to determine document staleness, the pages delivered may be out-of-date. The user would obtain a different document when retrieving it directly from the originating server. Therefore, no full transparency can be accomplished by only using caching proxies.

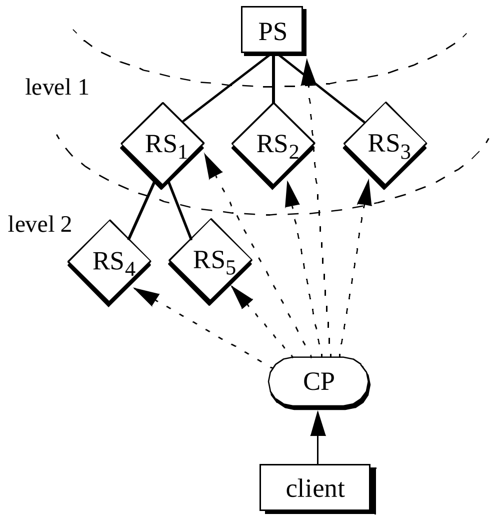

Fig. 3: Overview of the CgR scheme. One Primary Server (PS) forms the root of a logical tree of Replicated Servers (RS), which are serving (part of) the PS' replicated namespace. Via a Client-side Proxy (CP), the client can access either RS or PS resp.

In achieving the fifth goal of balancing the available bandwidth usage, caching is not appropriate. All document loading and staleness checks are made in the critical path exactly at the time of the request. Therefore, network load follows the user's temporal preferences of surfing the Web, namely during daytime.

In order to address the remaining problems, we have to look for a more advanced solution, taking our last goal into account — backward compatibility. Our proposal taking all the goals into account will be explained in the next section.

A PROMISING SOLUTION: REPLICATION

The basic idea to address the problems caching does not solve is *replication*. It has already been pointed out that one major goal to achieve is the increased document availability. This is the primary aim of replication. Additionally, we will see how bandwidth balancing and backward compatibility may seamlessly be integrated in a replication scheme. This section will focus on the concepts of replication with a special look at the Web. Following, the realization and a prototype implementation will be described.

Mechanisms

Today, documents accessible in the Web typically are stored at a single server. Whenever the server is down, the one and only path to the original data is lost. In order to increase the document availability, alternative access paths have to be provided. This is not a particularly new problem. Consider FTP sites, which are usually mirrored to increase their availability *and* to balance the load between multiple FTP servers. The user has to be aware of their existence in order to benefit from selecting a close one from the network's point of view. As a result, the network load is reduced, because shorter and more local links are used — another advantage gained from replication.

The down side of replication is the need to maintain consistency between the data stored on the original server and all its mirrors. Fortunately, we may benefit from two properties typical for the Web. First, by far the most requests for a document are read-only accesses, which are not critical for consistency maintenance. The second property to be mentioned is that changes to documents in most cases occur at the originating site by the document's creator. Therefore, besides a propagation of such changes, we only have to come up with a solution for the uncommon case of write-accesses by normal users.

A more challenging aspect is how to introduce the additional servers into today's Web. The chief requirement to make replication really useful is to achieve *replication transparency*. Particularly, users should *not* be aware of the existence of multiple servers. From the user's point of view, a service like the request for a document is presented to one *logical* server. The logical server acts as a representation of a group of physical servers. In particular, a request should not be directed to any of the physical servers. In case of its failure, the request could still be fulfilled by a different server.

Naming

Currently, a document available in the Web is represented by its URL (Uniform Resource Locator). Because URLs literally contain the name of the server, we actually run into the problem of transparently directing a request to a specific physical server in a logical group of replicated servers. Certain solutions based on DNS lookup exist, providing different server IP-addresses if asked for a name resolution [6]. However, this strategy only allows for whole-site replication. As has been determined by several groups, generally only a small set of documents a server offers are very popular (see [13] e.g.). Therefore, a more sensible approach would allow the replication of only those documents.

Caching goes Replication

Combining the paradigms of caching and replication promises a synergetic solution addressing all the five goals for a better Web. The concept we have put into practice is called *Caching goes Replication* (CgR). The basic idea of our approach is that of an active caching scheme, where servers can decide which documents should be cached and where this should be done.

For the current Web scenario, this implies transforming previously only passively caching servers into *Replicated Servers* (RS, see figure 3). These RS will then actively duplicate parts of the original *Primary Server*'s (PS) URL namespace, i.e. a subset of the data provided by them. The selection of which caching servers to convert to RS can either be done manually or automatically based on appropriate heuristics. While the latter is aimed at lowering the overall latency of all data accesses, a manual selection is especially sensible if it is known, which data is important to the community of users accessing the replicate server. Besides the conversion of existing caches to RS, new dedicated RS might be set up as well. Creating a hierarchical structure of RS and PS helps improving scalability: the PS will only have to know the set of its direct replicates (RS_1-RS_3 in figure 3). These first level replicates will in turn themselves have RS (e.g. RS_4 and RS_5) for which they act as the PS, creating a logical tree of Replicated Servers.

One might wonder how an active propagation of data to-be-replicated fits into the client/server architecture of the Web. However, this does not impose any technical problem: servers can initiate a propagation by sending a normal HTTP GET request bearing a particular notification. CgR enhanced RS will interpret this notification as a command to request the data to-be-replicated themselves. A more serious question to be addressed is the one of clients selecting RS, PS, and normal WWW servers, resp. At the first glance, this might induce modifications to the clients making them aware of their choices. By choosing a CgR enhanced *Client-side Proxy* (CP), all CgR specific actions can transparently be performed without modifying the clients or their interface to the Web. Fortunately, virtually every Web browser offers the possibility to redirect requests to a proxy server. This CP will direct WWW requests no longer only to the conventional (primary) servers, but also to available RS if possible. Such proxies can either be located near the client as the name CP suggests, or they can be set up by information providers who want to mask the application-level replicated nature of their server network.

Web Location and Information Service

The concept of CP just offers the basic mechanisms for clients to address a group of servers. What is still needed is a way to propagate the information about which RP exist and what replicates they hold. For this purpose, we developed the *Web Location and Information Service* (WLIS, pronounced „Willis“). The primary purpose of this application-level name service is to keep track of which URL namespaces are replicated and which servers belong to logical groups of PS and RS. A natural place to implement the WLIS service is the CP (as assumed below), but it can also be included in the PS or can be offered by separate WLIS servers. The currently realized former approach permits the inclusion of replication information in form of hyperlinks to RS into the user-delivered document. This gives the end-user the choice of transparently using CgR/ WLIS or of directly accessing a specific site, e.g. for reasons of data freshness. The approach of separate WLIS servers in contrast helps to reduce the load of document serving machines.

As a necessity for good scalability, WLIS information has to be collected and propagated automatically. Figure 4 gives an idea of how the distributed WLIS database is being set up. Assuming that no WLIS information is available, the CP will forward any requests directly to the appropriate PS (step 1). The PS itself knows about its first-level RS (RS_1-RS_3) and will include this information in a special HTTP header field of its answer. This initial WLIS information thus reaches the CP piggybacked with the conventional answer, from which it will be removed and used to build a list of RS associated to the respective PS. For subsequent requests to the respective URL namespace, the CP will address either the PS or one of the three newly „learned“ RS_1-RS_3. At this point, heuristics for estimating available bandwidth and thus for choosing an „optimal“ server are applied. If the selection is made to query one of the RS (e.g. RS_1 in figure 4, step 2), this one will in turn answer with the requested document and a list of its own replicates (RS_4 and RS_5 in this example). The CP will add this information to its database, and as these steps are performed over time, the CP will continuously learn about the whole server group. A third request might e.g. be directed to RS_5 (step 3). Manual insertion of WLIS information can further speed up the learning process especially in the case of already explicitly established WWW mirror sites [1].

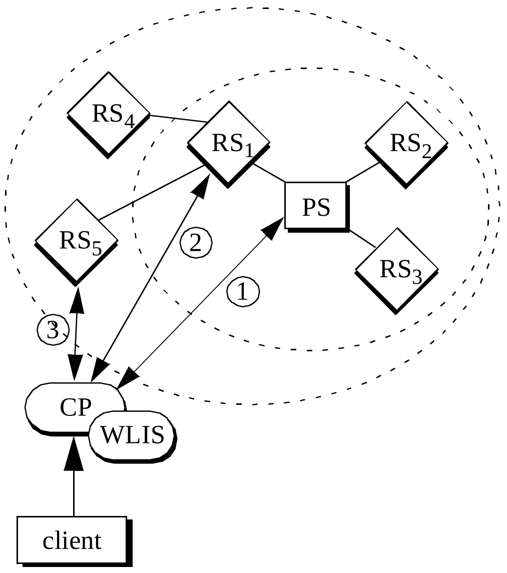

Fig. 4: Propagation of WLIS information. Over steps 1-3, the CP „learns“ about all RS.

Results

The combination of CgR and WLIS as it is currently implemented has been evaluated in our research group and has proven to be a sensible solution. Sets of RS for several different PS have been set up and are in daily use. As an additional service, our CP also offer replication on demand, allowing users to explicitly request an URL namespace subtree to be replicated. However, due to our limited possibilities to install CgR enhanced PS and RS in a wider extent, simulations are used to estimate the benefits of adding replication to the conventional caching mechanisms. The following graphs show the results of a simulation based on the WWW requests originating from our university in May 1996. For the tests, a single RS was addressed with certain quotas of its overall cache space assigned to exclusively holding replicates. Caches and replicate quotas were previously warmed with data of the preceding month, i.e., they were filled with the requests of April 1996. LRU is used as the replacement policy for the conventional cache. On the arrival of new documents, this strategy will replace documents that were least recently used (LRU). LRU is deployed in most existing caching proxies.

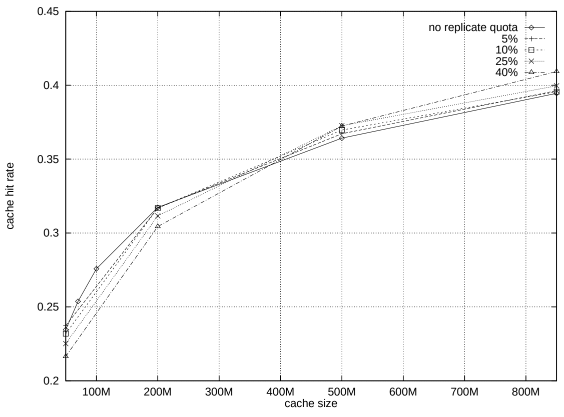

Fig. 5: Cache hit rate for a CgR enhanced Replicated Server (RS) with different quotas of its cache assigned to hold replicates.

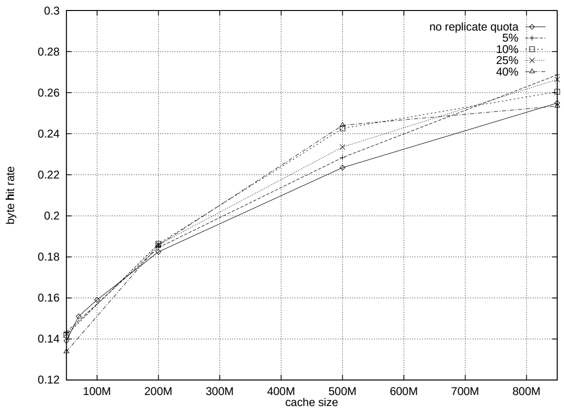

Fig. 6: Byte hit rate for a CgR enhanced Replicated Server (RS) with different quotas of its cache assigned to hold replicates.

Figure 5 shows the cache hit rate, figure 6 the byte hit rate to be achieved at different cache sizes and varying replicate quotas. For small cache sizes, reserving a considerable amount of cache space exclusively for replicates decreases overall performance. As long as the requests do not show a high reference locality that can be served from the replicates, this is an expected effect. Beyond a break-even point (which is about 350MByte for our tests), however, replication improves the cache hit rate by about 4% at 850MByte cache size. A more obvious effect can be observed on the byte hit rate. For a 500MByte cache, a 40% replicate quota can improve the byte hit rate by 9.2%. A reason for the bad performance of the 10% and 40% replicate quotas at higher cache sizes could not yet be identified - this is subject to further investigations.

The user-perceivable average transmission times for requested documents are shown in figure 7. The bad performance of the replication-based approach for small cache sizes relates to the lower cache hit and byte hit rates as to be seen in figure 5 and figure 6. With increasing cache sizes, active replication can reduce transmission times by about 1.5%.

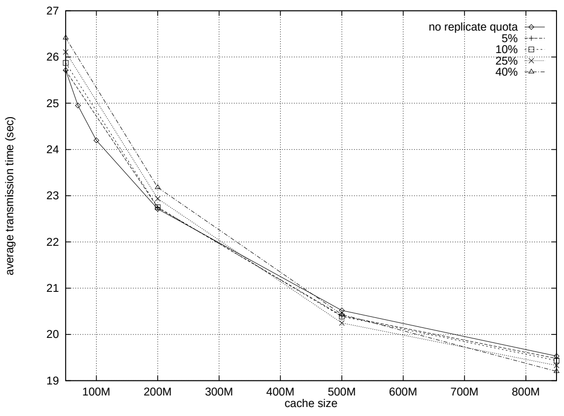

Fig. 7: Average transmission time for documents requested at a CgR enhanced Replicated Server (RS).

Another advantage of the replication concept is the more evenly distributed network bandwidth usage. Namespace replicates will be updated during low-traffic hours and the bandwidth requirements in usual peak-traffic hours are reduced. The reduction in the amount of data transmitted during normal requests in figure 8 validates this assumption. With only one 500MByte RS the data to be transmitted can be reduced by 3.8%. Additionally, the traffic is shifted from inside the network closer to the client, relieving the core network. The unexpected behaviour of the 10% and 40% replicate quotas at higher cache sizes relates to the corresponding lower byte hit rates as seen in figure 6.

As measurable improvements can already be achieved by using only one RS, applying CgR in a broader context promises even more significant benefits. With lots of sites accessing RS instead of the originating PS, e.g., the 130MBytes of transmitted data saved with a single RS (referring to figure 8 at a cache size of 500MByte) are multiplied. This fact considerably reduces the HTTP traffic on the connections from RS to PS, which will often be on network backbones. The traffic caused by keeping replicates consistent will be placed in times of low background traffic, balancing the bandwidth usage. Besides the impact of CgR when applied in large extend, setting up replicated servers for dedicated user groups with distinct reference locality will further improve hit rates and thus the benefits of replication.

Other important advantages of CgR are not directly expressed by the figures above. Among them are the possibility of realizing HTTP traffic shaping and the introduction of failure-tolerant groups of Web servers. Especially the latter aspect can greatly improve service availability, as current Web servers are single points of failure. Another point is, that active replication can much better achieve document freshness. In contrast to the caching proxy approach, a definite maximum staleness of documents can be guaranteed.

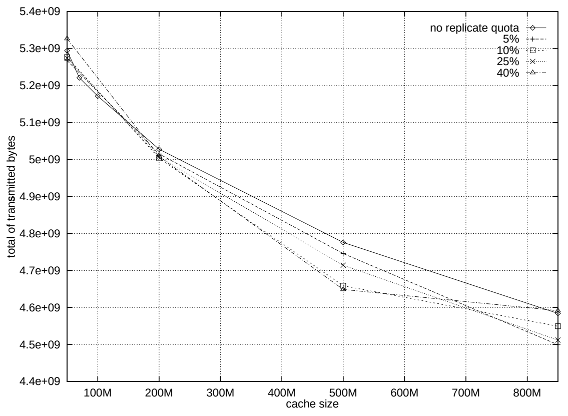

Fig. 8: Total amount of transmitted data from all servers when accessing a CgR enhanced Replicated Server (RS) with different quotas of its cache assigned to hold replicates.

Apart from these improvements, CgR and WLIS have been designed to encapsulate all necessary modifications in CP and the servers (RS and PS). Neither browsers nor the HTTP protocol have to be modified, so CgR is compatible with the existing Web infrastructure.

RELATED WORK

The general advantages of caching in the World-Wide Web have already been pointed out in section 2. For more detailed discussions of advantages and drawbacks of caching proxies, the interested reader may be referred to [3], [5], [6], [13] or [16]. As far as server-driven approaches are concerned, several other works deserve particular attention. In the area of more conceptually oriented papers, [4] presents a list of basic advantages of server initiated cache invalidation. Partially building on this work, [8] shows in a more thorough way the advantages of replication on the Web including detailed simulations of their concepts. However, as in [4], neither the naming problem has been tackled, nor an actually installable piece of software, i.e., a WWW server or proxy resulted from their work. As far as proxying software is concerned, Harvest [9], [14] and Squid [12] deserve particular attention. Though originally intended to provide efficient caching infrastructures, both systems gradually incorporate elements being of advantage to server initiated replication strategies. Particularly their ability to be hierarchically structured prove to be of high value in this respect.

FURTHER WORK

With respect to the future, several paths to further work are immediately visible. From a performance tuning point of view, the most essential information yet to be gathered is a list of data to be replicated to obtain the largest benefit as well in terms of overall network traffic reduction as well as with respect to user-perceptible latency reduction. In the realm of the former, the actual algorithm and timetable for replicating modified data from primary to replica servers need to be established so as not to interfere with peak times of network traffic generated otherwise. In the opposite direction, the question remains to be solved, how the selection of appropriate RS has to be decided, given the solution of the more imminent problem, namely the definition of the term *appropriate*. As probably no clear, firm, and decisive answers applicable to any combination of PS and RS can be given in any of the areas outlined above, new heuristics have to be developed. Since they initially will be based on the simulations presented above, further explanations and models for the behavior observed during our tests have to be developed. Additionally, more traces will have to be collected to permit more substantial statements on the WWW data retrieval behavior of heterogeneous user communities.

A completely different area of further work concerns the inclusion of client-initiated data uploads to the servers. Such write operations tend to be much more involved in an unreliable environment as set up by the Internet than in more easily controllable local area networks where most work on this topic has been performed so far. However, as multicasting on wide-area networks becomes more commonplace than it is today, techniques like those presented in [10] might be considered in the future.

The introduction of an automated document retrieval automaton into a RS fetching data on behalf of a person wishing to be able to easily and quickly access certain sets of documents sometime later proved to be very attractive. In this area, further work can be started investigating in how far a combination of CgR/WLIS can be used to effectively —though certainly not reliably— shape traffic at application level, e.g. by re-routing requests as suggested by usage patterns observed earlier. In this respect, such a system might even become an initial, distributed platform for experiments with qualities of service. The latter can certainly not be guaranteed as the Internet is an inherently unreliable system, but in expectation of respective facilities in the next generation IP protocol [17], it might provide first experiences with user-perceptible end-to-end guarantees.

CONCLUSION

Caching and replication have shown their advantages in many areas of computing. This paper intended to revisit and extend on more basic research in this field with a distinct focus on its usefulness at the application level, particularly the World-Wide Web. Results of simulations have been presented supporting the conceptual advantages of caching and replication. In order to fully reap the benefits of replication particularly in the WWW, an application-level naming service has been introduced. This Web Location and Information Service originally has been intended to provide for a transparent introduction of fault-tolerance and load-balancing in a replication enhanced Web. During the work carried out, it grew into a full-fledged naming service of potential impact on other protocols and applications within the Internet permitting the latter to scale and thus, grow in a more healthy manner than today. The concepts and initial measurements presented in this paper have been shown to be extensible in various directions. One of the most interesting ones is the creation of a first notion of quality of service over the inherently unreliable Internet.

REFERENCES

1. M. Baentsch, G. Molter, P. Sturm: *Introducing Application-level Replication and Naming into today’s Web*; Computer Networks and ISDN Systems, Vol. 28, No. 6, April 1995.

2. T. Berners-Lee (editor): *The Hypertext Transfer Protocol*, RFC 1945, 1996.

3. M. Abrams, C.R. Standridge, G. Abdulla, S. Williams, E.A. Fox: *Caching Proxies: Limitations and Potentials*; Proc. 4th International Conference on the World-Wide Web, Boston, Dec. 1994.

4. A. Bestavros: *Demand-based Document Dissemination for the World-Wide Web*; Technical Report TR-95-003, Boston University, Comp. Sci. Dept., Feb. 1995.

5. J.-C. Bolot, P. Hoschka: *Performance engineering of the World-Wide Web: Application to dimensioning and cache design*; Computer Networks and ISDN Systems, Vol. 28, No. 6; Proc. 5th International Conference on the World-Wide Web, Paris, May 1996.

6. K.C. Claffy: *Web traffic characterization: an assessment of the impact of caching documents from NCSA’s Web server*; Proc. 2nd International Conference on the World-Wide Web, Chicago, 1994.

7. R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, T. Berners-Lee: *Hypertext Transfer Protocol -- HTTP/1.1*; Internet Draft, Aug. 1996, available at: http://www.w3.org/pub/WWW/Protocols/HTTP/1.1/draft-ietf-http-v11-spec-07.txt.

8. J.S. Gwertzman, M. Seltzer: *The Case for Geographical Push-Caching*; Proc. 5th Workshop on Hot Topics in Operating Systems, Orcas Island, Wa, May 1995.

9. The Harvest Information Discovery and Access System; Online documents at http://harvest.cs.colorado.edu and at http://harvest.cs.colorado.edu/harvest/Cached-1.4-announce.txt.

10. J. Knight, S. Guest: *Using Multicast Communications to Distribute Code and Data in Wide Area Networks*; Software-Practice and Experience, Vol.25, No.5, May 1995.

11. World-Wide Web Consortium: *Hypertext Transfer Protocol - Next Generation*; Online reference at http://www.w3.org/pub/WWW/Protocols/HTTP-NG/Overview.html.

12. D. Wessels: *Squid Internet Object Cache*; Homepage of the Squid project, http://www.nlanr.net/Squid/, 1996.

13. A. Bestavros, R.L. Carter, M.E. Crovella, C.R. Cunha, A. Heddaya, S.A. Mirdad: *Application-level Document Caching in the Internet*; Proc. 2nd Workshop on Services in Distributed and Networked Environments, SDNE‘95, Whistler, Canada, June 1995.

14. A. Chankhunthod, P.B. Danzig, C. Neerdaels, M.F. Schwartz, K.J. Worrell: *A Hierarchical Internet Object Cache*; Proc. 1996 USENIX Winter Technical Conference, San Diego, Jan. 1996.

15. A. Luotonen, K. Altis: *WWW Proxies*; Computer Networks and ISDN Systems, Vol. 27, No. 2; Proc. 1st International Conference on the World-Wide Web, May 1994.

16. N.G. Smith: *The UK national Web cache - The state of the art*; Computer Networks and ISDN Systems, Vol. 28, No. 7; Proc. 5th International Conference on the World-Wide Web, Paris, May 1996.

17. S.O. Bradner, A. Mankin (eds.): *IPng - Internet Protocol Next Generation*; Addison-Wesley, Sep. 1995.