Reducing Retrieval Latencies in the Web: the Past, the Present, and the Future

Themistoklis Palpanas
Department of Computer Science
University of Toronto
10 King's College Road, Toronto
Ontario, M5S 3G4, CANADA
themis@cs.toronto.edu

Balachander Krishnamurthy
AT&T Labs-Research
180 Park Ave
Florham Park, NJ 07932 USA
bala@research.att.com

Graduate Department of Computer Science
University of Toronto
Technical Report CSRG-378

Abstract

One of the main directions of research in the Web is to reduce the time latencies users experience when navigating through Web sites. The work in this direction originated from relevant research in operating systems (e.g. caching), where the aim is to reduce file system latencies. Caching is already being used in the Web domain. However, recent studies indicate that the benefits from this technique are rather limited. Thus, another method that was first applied in operating systems, prefetching, is now being studied in the Web context. In addition to prefetching, there is a number of other techniques that are being examined in the Web domain and are specifically targeted to this environment. These techniques include compression, and piggybacking of information between the servers and the clients. The main contributions of this survey paper are: A taxonomy of the caching and prefetching techniques, according to the underlying principles, and the algorithms used; a critical survey of the methods that have been proposed so far for dealing with the retrieval latencies problem, and how each one of those fits in the taxonomy; a discussion of the results of the past and ongoing work, and the identification of future research directions.

1 Introduction

Web usage has proliferated during the last years. It is being used for entertaining, commercial, or educational purposes by millions of people around the world, and for some of them is an indispensable tool for their work.

Although Web is a relatively new domain, it has already attracted the attention of the research community. One of the main directions of research is to reduce the time latencies users experience when navigating through Web sites. The work in this direction originated from relevant research in operating systems, where the aim is to reduce file system latencies [3, 17].

Caching is already being used in the Web domain. However, recent studies [22, 12] indicate that the benefits from this technique are rather limited. Thus, another method that was first applied in operating systems, *prefetching*, is now being studied in the Web context. It tries to hide the page retrieval latency, rather than reduce it. When prefetching is employed, Web pages that the user is likely to access soon are transfered before the actual request. If the user does request one of the prefetched pages it will already be in the local cache, reducing the latency to minimum. Otherwise, the transmission of these pages was erroneous, and the bandwidth used was wasted.

In addition to prefetching, there is a number of other techniques that are being examined in the Web domain and are specifically targeted to this environment. These techniques include compression, and piggybacking of information between the servers and the clients. It is evident that these methods can only alleviate the problem, and not solve it since they have a small effect in the overall system, and are usually employed in conjunction with caching or prefetching.

It seems that the solution to the retrieval latencies problem will emerge from the combination of the existing techniques. Nevertheless, more elaborate studies should be conducted in the Web context, in order to identify its special characteristics and requirements.

The main contributions of this paper are:

- A taxonomy of the caching and prefetching techniques, according to the underlying principles, and the algorithms used.
- A critical survey of the methods that have been proposed so far for dealing with the retrieval latencies problem, and how each one of those fits in the taxonomy.
- A discussion of the results of the past and ongoing work, and the identification of future research directions.

1.1 Overview

The rest of this document is organized as follows. The work related to reducing retrieval latencies in both the operating systems and Web domains is discussed in Section 2. The main focus is on caching and prefetching, but other methods are also cited. Section 3 presents a taxonomy of the caching and prefetching techniques studied so far in the research community. Finally, Section 4 explores the lessons learned from the operating systems experience, analyzes the current situation, and proposes future directions of research.

2 Litterature Review

In what follows we present an overview of the cache and prefetching techniques. Work in both the operating systems and Web communities is cited, so as to observe the similarities and differences between the two.

2.1 Operating Systems Context

2.1.1 Caching

Caching has been used successfully for many years, yielding significant improvements in I/O latency [3]. It tries to reduce time latencies induced by the file system by having the requested file pages available in memory.

Apart from the simple scheme, where each client uses a single cache, more elaborate strategies have been proposed to address the problem of scalability. Blaze and Alonso [6] suggest a hierarchical structure of caches, which adapts dynamically to client requests. This policy allows the cooperation of caches in order to service as many requests as possible. The simulations show that 57% to 67% of cache misses can be serviced by neighboring caches, and the server load can be reduced to half. Nevertheless, additional traffic is induced in the network for communication messages, and clients perceive higher latencies for accessing the data. The study does not quantify the latency costs.

Yet, the benefits from exploiting a cache are limited. Ousterhout et al. present [31] a trace-driven simulation which indicates that the relative benefit of caching decreases as cache size increases. The experiments show that beyond a certain point response times cannot be substantially reduced any further, no matter how large the cache is. Moreover, Ousterhout [30] shows that applications are still I/O bound, despite the presence of the cache. It is precisely the requests that cannot be satisfied by the cache (cache misses) that are the bottleneck.

2.1.2 Prefetching

In order to improve performance, the ordinary caching algorithms should be supplemented with other methods, thereby enabling a further increase in cache hits, and thus, a decrease in average waiting time for the user. Prefetching is a concept that helps in that direction. It enables the cache to satisfy more requests by predicting sufficiently ahead of time what these requests will be.

A naive method for doing prefetching is to have the application inform the operating system of its future requirements [34]. When this method works, it is very efficient since the application knows exactly which files it will need and at which point in the future. However, the problems this approach has to overcome are rather significant. First of all, it will only work for applications that embody this technique. So old applications will have to be rewritten, or otherwise they will not benefit at all. Then, the mere task of programming such an application will become extremely cumbersome. The programmer will have to learn and use a set of complex directives to manipulate low level functions with a strong dependence on the physical resources. Furthermore, just the knowledge of future requests is not enough. The prefetching should occur a significant time before a page is actually needed to ensure that the file will be available when requested. This poses an additional difficulty in the programming task.

One way to overcome the aforementioned problems is by instrumenting the compiler to provide all the necessary prefetching information to the operating system [29]. The compiler can add instructions in the code for prefetching pages that will soon be accessed, as well as for evicting pages that will no longer be useful, thus, enabling other applications to use the freed space. This approach was tested against the NAS parallel benchmark suite [2], and resulted in execution speed-ups of twofold and in one case threefold. However, the experiments were performed on a system with multiple disks, which significantly increased the available I/O bandwidth. The effects of varying file system bandwidth were not tested. In addition, the scope of that work is currently limited to numeric applications, so it is expected to be beneficial only in highly restricted settings.

A recent study [17] has tried to explore the effectiveness of combined caching and prefetching techniques using a trace-driven simulation. Assuming a single-process system with multiple disks and full advance knowledge of future requests, combined caching and prefetching algorithms enable some of the existing strategies to reduce I/O latency to zero. Then the application becomes CPU bound. When restricted to one disk though, the stall time (application waiting for file system operations) remains significant.

An additional problem that cannot be tackled by the prefetching methods presented above arises when related file accesses span multiple executables. Typically, each application can only be aware of its own file access patterns. Yet, it is frequently the case that a series of different applications are executed repeatedly (e.g. commands of a shell script or a makefile). In order to capture this type of relations, long-term history information on accesses across applications is exploited. These techniques assume that future request patterns in an operating system will be similar to past ones. As a consequence, in contrast to the previous methods, this one makes predictions without using any application specific knowledge at all. Therefore any application can benefit without having to be rewritten, but predictions are in general fewer or less accurate.

Griffioen and Appleton [15] propose a predictive cache which prefetches file pages based on a *proba-*

bility graph of past accesses. The nodes in the graph represent accessed files, while a directed weighted arc from node A to node B expresses a metric for the probability of requesting file B within w requests after file A. A trace-driven simulation revealed improvements in the time an application has to wait for read operations to complete (*read complete time*) ranging from 22% to 51% compared with the non-prefetching case. Similar results yielded a prototype implementation of the system, tested against a synthetic workload. However, the overall improvement in execution time ranged from 1% to 8%, indicating that only a small fraction of the prefetching benefits is reflected in the total run time. This is due in part to the overhead prefetching imposes on the system. The dominant factor, though, is the ratio of the read complete time to the total execution time.

Curewitz et al. [11] claim that data compression techniques can be successfully used for prefetching. An implementation of this idea, based on *Prediction by Partial Match (PPM)*, is presented by Kroeger and Long [21]. The advantage of this method is that the prediction of future requests depends on the k previous accesses, thus resulting in better predictions. The major drawback is the algorithm's memory requirements. In the trace-driven simulation, cache-hits increase by an average of 25% over a non-prefetching cache. The simulation shows that on average a 4MB predictive cache has a higher hit rate than a 90MB simple *Least Recently Used (LRU)* cache. Results on time latencies are not reported in this paper.

Lei and Duchamp [23] employ *access trees* to make predictions. Access trees record all files accessed during one execution of each program. When an application is re-executed, the current activity is compared against the saved access trees. If a similarity is detected, the files in the access tree are prefetched. The algorithm does not force any prefetches unless a sufficiently similar pattern exists in past access trees. The trace-driven simulation shows a substantial reduction in applications latency, despite the significant CPU overhead. However, the trace used was not based on a real workload.

The problem of predictive caching emerges also in *near-line tertiary storage libraries* [18], although in this case time scales are different. The decision to be made is which large multimedia files should be prefetched from slow tertiary storage to fast disks. A continuous-time Markov-chain model is used to predict what the future accesses will be within time t. A trace-driven simulation of synthetic workloads reveals time latency reductions of up to a factor of two.

2.2 World Wide Web Context

Many of the initial ideas employed on the Web were borrowed from similar work in operating systems. Indeed, it is only recently that the peculiarities of the Web domain started to become apparent [39, 27, 12].

The characteristics of Web resources are explored by Douglis et al. [12]. The trace used in the study showed that 69% of the accesses and 60% of the bytes transfered involved images. The same figures for HTML documents are 20% and 21% respectively. The analysis of the trace indicates that 22% of the resources were accessed more than once, but they accounted for 50% of the requests. From this 50%, 13% were to resources that had been modified since the previous request. Unlike another study [5], which was performed in the educational environment, this trace suggests that the smaller and most frequently referenced resources tend to change more often. It was also observed that pages in the educational domain are modified noticeably less often than pages in other domains. Another collection of traces (three form academic environments, two scientific, and one commercial) [1] shows that 10% of the files accessed accounted for 90% of the requests and bytes transferred. Furthermore, 15%-40% of the files and bytes accessed were accessed only once. The above results manifest the diversity and changing nature of the Web.

2.2.1 Caching

Williams et al. [39] investigate the operation of proxy caches, i.e., caches "near" the clients. Their study shows that *SIZE*, a policy where the largest document is removed first, always performs better than *LRU* (and at within 10% of the optimal) in terms of document cache hits (as opposed to byte cache hits). When a single large document is evicted from the cache, space is freed for many small ones. Moreover, the majority of the requests are for small documents. Nevertheless, *SIZE* proves to be the worst policy when byte cache hits are measured since large documents tend not to be cached.

In accordance with the aforementioned results, Lorenzetti et al. [24] propose a new replacement policy, *Lowest Relative Value (LRV)*. It takes into account the size of the document, the number of past references, and the time of the last reference. Such a cost model enables the algorithm to maximize both the document and byte cache hits. The implementation of the algorithm, however, requires some parameters that may only be determined through experiments, and could vary with time. Trace-driven simulations show that *LRV* performs as well as, or better than the best policy considered so far. The simulations do not account for time latencies.

Two more algorithms based on a cost model for replacing pages are presented by Cao and Irani [7] and by Tewari et al. [37]. The first algorithm is the *GreedyDual-Size*, which incorporates locality with various cost metrics (e.g., page size, page retrieval latency, etc.). The second one, the *Resource-Based Caching (RBC)* algorithm, is designed to support not only static data types (i.e., text and images), but also continuous media data (i.e., streaming audio and video). Simulations of these algorithms (synthetic workload for the second one) depict that they perform better than other schemes. However, a different cost function is used for every performance metric. Thus, no single instance of these algorithms is universally better than all the others.

A cache which resides on the server, *Main Memory Web Cache*, is proposed by Markatos [27]. This kind of cache can substantially alleviate the server load by having the majority of the requests serviced from the memory instead of the file system. A trace-driven simulation consisting of five diverse traces shows that even a small amount of main memory (512KB) is enough for the cache to experience a document hit rate of more than 60%. The algorithm used implements an *LRU* removal policy, but has also a limit for the maximum allowed document size that can be cached. The limit is dynamically adapted so as to maximize the document cache hit rate. These traces demonstrate that a significant percentage of the requests involve a small number of documents, and even a small cache can significantly reduce the server load. However, no figures are provided for the modification frequency of the pages. Note that a high frequency would result in poorer performance.

The use of cooperating caches in the Web context is aiming at improving the performance of this mechanism in terms of scalability, robustness, and efficiency. A recent study [25] investigates the scheme where many caches in a flat structure cooperate. The client sends a request by selecting at random one of the available caches. If the cache does not store the particular document, it multicasts the request to the other participating caches. This technique provides caches with a means for sharing their contents. Yet, the communication protocol limits the scalability of the system. The aforementioned problem is not present in the Harvest cache [9], which employs a tree of caches. Clients make requests only to leaf nodes, which in the case of a cache miss forward the request to the sibling and parent caches in the hierarchy. Successive misses will propagate the request up in the tree, until the original server is reached. Simulations indicate that the additional latencies for servicing a request under both schemes are insignificant. However, no results are available for the network overhead, and for the relative benefits of employing a cooperative cache scheme in terms of cache hits and the associated latencies. The scalability issue is addressed by Fan et al. [13], who propose a new protocol for cache sharing. Under this protocol, each proxy maintains a space-efficient summary of the contents of the other participating proxies. Therefore, the CPU overhead and the number of inter-cache messages are significantly reduced compared to the current approach.

Additional, more subtle parameters of the proxy cache operation are addressed in other studies. Krishnamurthy and Wills [19] explore the cache coherency problem, i.e., assuring the validity of cached resources, and present a new technique, the *Piggyback Cache Validation (PCV)*. The cache, instead of sending explicit validation requests to the server to find out the expiration times of the cached documents one at a time, piggybacks a list of documents from that server whenever a communication occurs. A trace-driven simulation for different cache sizes shows that among other policies currently employed, *PCV* results in fewer requests sent to the server, while maintaining a close-to-strong coherent cache. Furthermore, it yields the lowest costs in terms of response time, bandwidth used, and number of validation messages. By extending this technique, the server could piggyback resource invalidations back to the cache, which in turn could free some space sooner. This idea, *Piggyback Server Invalidation (PSI)*, is explored in another study [20]. The hybrid policy of *PCV* and *PSI* techniques improves significantly the performance of the currently applied algorithms. Though, the computation overhead induced on the server has not been sufficiently studied yet.

The work of Shim et al. [36] suggests a unified algorithm for cache replacement and consistency. The algorithm uses a cost function in order to decide which documents to replace in the cache. The factors determining the cost of a document are the size, the retrieval latency, the frequency of accesses, and the frequency of validation checks for that document. Trace-driven simulations [35, 36] indicate that the new algorithm achieves a better hit ratio than LRU, while improving the cache consistency level over the current approach (i.e., for documents whose modified status is unknown, validate with the server upon request). However, no results are reported for the byte hit ratio. Furthermore, although the algorithm needs to keep track of several parameters, its complexity is not analyzed.

The benefits from *delta encoding* (transmitting only the difference between two versions of the same document) and *data compression* for the transfer of Web documents are investigated by Mogul et al. [28]. The trace-driven simulation uses an unlimited-size cache and a filtered trace (only status-200 responses, with no images). It demonstrates that the response body-bytes saved are at least 98.5% for half of the delta-eligible responses, or 30% for all responses. Latency is reduced by 12%. However, the latency benefits tend to decrease as the bandwidth increases. Apart from the computation overhead of this method, there is a space overhead, since the server needs to keep several old versions for each file. The study suggests, though, that these overheads are minimal.

2.2.2 Prefetching

Bestavros proposes two server-initiated protocols [5]. The first one is a hierarchical data dissemination mechanism that reduces network traffic by propagating Web documents from servers to server proxies that are closer to clients. The second protocol, a *speculative service*, sends to the client —along with the requested document— a number of other documents that the server predicts will be requested in the near future. The predictions are produced by the closure of the matrix whose *(i,j)* element represents the probability that document j will be requested within a certain time window after the request of document i. The trace-driven simulation reveals that a slight increase in network traffic yields significant reductions on server load, service time, and client cache miss rate. These results however, were obtained by allowing a document to predict the request of its images, thus distorting the figures.

Previous work on operating systems [15] has been the basis of Padmanabhan and Mogul's research [32] on prediction of future requests. A trace-driven simulation along with a linear model for the network is used to test the algorithm. The results indicate that the benefits from prefetching are significant, even for slow (i.e., modem) connections. Prefetching can reduce the average latency by up to 30% with a 25% increase in network traffic. In contrast, a doubling of the bandwidth reduces latency only by 20%.

The use of partial match prediction, a technique taken from the data compression literature, for prefetching in the Web, is explored by Palpanas [33]. This study indicates that a considerable fraction of the user's requests can be predicted with high accuracy (e.g., predicts 18%-23% of the requests with 90%-80% accuracy), therefore, the additional network traffic is kept low. Furthermore, the simulations show significant benefits for the cache in terms of document hits. When prefetching is applied the cache experiences up to 6.5 times more hits.

A variation of the same technique is evaluated by Jacobson and Cao [16] for prefetching between proxies and low-bandwidth clients. This work shows that prefetching can reduce latency by nearly 10%, though, a significant part of this reduction is attributed to the caching effect of the prefetch buffer.

A cooperation of the client and the server is proposed by Wills and Sommers [40]. The trace-driven simulation suggests that depending on the client's browsing habits, client information can assist the server's suggestions for prefetching. The server uses the prediction model presented by Bestavros [5].

A recent study attempts to determine bounds in the performance of proxy caching and prefetching [22]. The authors performed a trace-driven simulation with infinite-size cache and complete knowledge of future requests. The external latency between the proxy cache and the server accounts for 77% of the total latency. Caching achieved a reduction of 26%, prefetching 57%, and the combination of both 60%. Although these figures are quite different for the second trace used, they exhibit the same trends. The weak association of proxies and latency reduction is confirmed by a real-case study of cache performance [26]. The simulation also shows that the available bandwidth for prefetching has little effect on latency reduction for which the dominant factor is how far in advance of the actual need the data can be prefetched. The study, however, does not comment on the number of single accesses to servers that inherently cannot be prefetched.

Crovella and Barford [10] discuss the effects of prefetching on the network. A trace driven simulation indicates that straightforward approaches to prefetching increase the burstiness of traffic. Instead, the authors propose a transport rate control mechanism, which is intended to reduce the bursting effects by using as little bandwidth as possible while ensuring that the documents are delivered on time. The simulation denotes that rate-controlled prefetching significantly improves network performance compared not only with the straightforward approach, but also with the non-prefetching case.

3 Taxonomy

Several ideas have been proposed so far for caching and prefetching policies, both in the operating systems and Web context. In the following paragraphs we present a classification of the techniques discussed so far according to various criteria. Such a classification helps to find the similarities among the different approaches, contrast the suggested solutions, and identify new directions.

Cache Replacement Policy: This is the area where much of the research interest has focused on. As new pages come into the cache, the replacement policy determines which pages to evict in order to make space for the incoming ones. In operating systems, the prevalent policy is the *Least Recently Used*, which evicts the page requested the least recently. In the Web domain, the special characteristics of the medium have led to the investigation of several new policies, aiming to maximize different performance metrics. The additional parameters taken into account are the variance in page sizes, the diverse retrieval costs of the documents from the sources, and the different requirements of each data type. Some of these policies employ *cost functions* based on the aforementioned factors.

Cache Location: The physical location of the cache is of critical importance for its efficiency and functionality. There are four basic configurations, each one favoring a different set of clients and files.

1. *In the client.* Only the specific client is benefiting. These benefits though, are rather restricted since the resources allocated for such a cache are usually limited.
2. *In the server.* In this case the cache enables some hot files to be serviced faster from memory rather than from disk. In addition, the server saves in terms of time and resources used for accessing the disk.
3. *Near the clients.* There is a common cache for a large community of users. The allocated resources permit the cache to store a significant number of files originating from many sources, and the users to profit from the access patterns of their neighbors.
4. *Near the servers.* The cache is storing frequently accessed files pertaining to a small set of servers. Such caches act as a mechanism to disseminate popular information towards the clients.

Cache Architecture: A cache can be *isolated*, communicating solely with its clients and the servers, or *cooperative*. In the second case, a set of caches are cooperating in order to improve their performance. A cache miss in one cache may be serviced from a nearby cache that stores the requested file, transparently to the user. The set of cooperating caches may be organized in a *flat* or *hierarchical* structure.

Cache Behaviour: A *passive* cache reacts only in response to user requests. A file is loaded and stored in the cache only if some client wants to access it. In contrast, *active* caches employ an intelligent way of determining future user requests. Thus, they prefetch files which are likely to be accessed in the near future, and that are not in the cache, resulting in improved performance.

Prefetching Algorithm: This is a prediction algorithm, speculating the next user requests. It is based on information about the past access patterns. Depending on the origin of this information, prefetching algorithms can be classified in three categories.

1. *Client-based.* The algorithm uses information residing in the client side in order to make predictions. This information may pertain to a single user, or to a community of users. The algorithm is suited to the particular clients.
2. *Server-based.* Only information in the server side is used to drive the algorithm. The predictions are based on the reference patterns of many users, and they are targeted exclusively to the files stored in the participating servers.
3. *Combined client and server.* Information from both sides is combined in the prefetching algorithm. The predictions may be tailored for each particular user, while maintaining the specific knowledge for the individual file that only the server can hold.

Note, that the implementation of the ideas discussed above is not always simple and straight-forward. Some of them involve extensive modifications in the current infrastructure. However, solutions requiring radical changes in the existing protocols are extremely unlikely to be adopted in the real world.

4 Conclusions and Future Work

Although caching and prefetching have been shown to improve performance in both the operating systems context and the Web domain, the later has special characteristics that impose additional restrictions. The above studies suggest that existing techniques can be tailored to better suit the Web domain. Table 1 summarizes the similarities and differences between operating systems and the Web.

Caches in the Web context suffer from the frequent changes of documents and user interests, thus, they perform significantly worse than caches in operating systems. Experiments show that the hit rates of caches in operating systems are twice as much as the hit rates experienced by caches in the Web. Cost-based replacement policies seem to increase hit rates. Though, the problem of selecting the cost function has not been sufficiently explored yet. Another way of improving the performance is by employing cooperative architectures for caches. This strategy will result in more cache hits and less server workload since more documents will be serviced by the network of caches and not by the original server. Nevertheless, a rigorous study should be made in order to quantify the potential benefits and costs of this approach.

Studies suggest that dynamic documents constitute an increasing percentage of contents on the Web. These documents cannot be cached by conventional proxies. Yet, Cao et al. [8] recently proposed a caching scheme that allows proxy servers to cache such documents and service request to them. This

<table>
<tr><th></th><th>Operating Systems</th><th>World Wide Web</th></tr>
<tr><td rowspan="2">passive cache</td><td colspan="2">• Relative benefit of cache decreases as cache size increases.
• Latency time bounded by cache misses.</td></tr>
<tr><td>• Page size fixed.
• Files and access patterns do not tend to change often.
• Potentials for increased cache performance and reduced server load from cooperative caches</td><td>• Page size varies and affects the policies used.
• Changes in documents and user requests are frequent.
• Cache benefits increase as bandwidth decreases(?)
• Cache in the client side, or in the server side or both(?)
• Costs and benefits of cooperative caches(?)</td></tr>
<tr><td>active cache</td><td>• Can potentially reduce I/O latency to zero.
• System and/or application generated predictions.
• Fairly limited number of possible accesses; scripts guarantee an access sequence.
• Achieves same performance as a substantially larger passive cache.</td><td>• A first study indicates a bound in the reduction (though, the assumptions differ).
• Server generated predictions. Client side or combination of both(?)
• Much more uncertainty in predictions.
• Significantly improves the performance of a passive cache.
• Eligible to delta encoding and compression.
• Harder to simulate. Model for the Web(?)</td></tr>
</table>

Table 1: Comparison of operating systems and the Web. Question marks identify issues that have not been sufficiently researched yet.

is accomplished by coupling the document with a small executable which can perform the necessary processing on behalf of the server.

Delta encoding and data compression can be useful since they can accommodate the fast changing nature of the Web. This is so because document changes are frequent but not radical. The benefits from this methods are apparent when the server's computation power is high, but tend to decrease as bandwidth increases. In the latter case prefetching can be more beneficial.

Unlike the operating systems environment, the Web presents much uncertainty, and exhibits a quickly changing behaviour. Thus, prefetching can only be advantageous when clients navigate in a Web site with a purpose, i.e., not aimlessly hopping from one page to another, and when the overall access patterns in servers do not vary extremely. Communication between clients and servers may help in that direction since clients can observe the navigation patterns of a single user across Web sites, whereas servers can combine the access patterns of many users to a single site. It is evident that different combinations of clients and servers would lead to substantially diverse results, therefore specialization of the above techniques is required. Nevertheless, combined client- and server-based prefetching algorithms seem to be the most promising, and should by further investigated.

Different methodologies for making predictions may be investigated. The links inside a web page give a good indication on what the next access may be. The same information can be exploited in the other way around, by dynamically changing the structure of a page (or even the structure of a whole site) according to what users are most interested in. Prefetching can also be employed for improving the characteristics of the network traffic. Correct predictions of the future traffic enable a better utilization and management of the available resources. Therefore, a tight cooperation between the prefetcher and the network layer seems promising. A prefetching engine that is informed about the network traffic can adjust its operation in order to suit the network capabilities at each point in time.

It is the combined use of caching and prefetching techniques that yields the best results, but it has not been sufficiently explored yet. The simulations discussed so far give an indication as to what is beneficial and what is not. However, it is difficult to quantify in absolute terms the acquired results, since the measurement environment is too complex and noisy. Therefore, a complete model of the Web should be defined. Such a model should take into account the different user behaviors and the diverse requirements of dissimilar organization servers. The network, connection, and server-load models should also become more elaborate in order to capture the interdependencies of all the aforementioned factors. A first step towards this direction is a study by Barford and Crovella [4], which proposes a methodology for generating realistic Web workload.

Orthogonal to the ideas discussed above is the need for a formal framework under which applications will operate on the Web. Franklin and Zdonik [14] propose a Dissemination-Based Information System (DBIS), which is an integration of data delivery mechanisms and information broker hierarchies. The DBIS can serve as the basis for constructing distributed information systems in the Web. Such systems could use the DBIS in order to manipulate and broadcast information. They could enrich the information as it is passed from node to node, filter it, or perform any other necessary operation, according to some policies. WebOS [38] is another proposal in the same direction. The study argues for the usefulness of an operating system for the Web, providing a common set of operating system services upon which applications can be developed. An example of such an application is a facility for the dynamic replication of overloaded services in response to client demands.

Evidently, the existence of some framework for the support of applications on the Web would enhance and improve the services provided to the users. All the aforementioned methods and techniques could be incorporated in this framework. In addition, the ability they would obtain to dynamically respond to the changing nature of the Web would be a step towards a self-moderating, self-configurable system.

References

[1] Martin F. Arlitt and Carey L. Williamson. Internet Web Servers: Workload Characterization and Performance Implications. *IEEE/ACM Transactions on Networking*, 5(5):631–645, 1997.

[2] D. Bailey, J. Barton, T. Lasinski, and H. Simon. The NAS Parallel Benchmarks. Technical Report RNR-91-002, NASA Ames Research Centre, 1991.

[3] Mary Baker, Satoshi Asami, Etienne Deprit, John Ousterhout, and Margo Seltzer. Non-Volatile Memory for Fast, Reliable File Systems. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 10–22, September 1992.

[4] Paul Barford and Mark Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. Technical Report BU-CS-97-006, Computer Science Department, Boston University, 1997.

[5] Azer Bestavros. Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time in Distributed Information Systems. In *International Conference on Data Engineering*, pages 180–189, New Orleans, LO, February 1996.

[6] Matthew Blaze and Rafael Alonso. Dynamic Hierarchical Caching in Large-Scale Distributed File Systems. In *USENIX Annual Technical Conference*, 1997.

[7] Pei Cao and Sandy Irani. Cost-Aware WWW Proxy Caching Algorithms. In *USENIX Symposium on Internet Technology and Systems*, pages 193–206, 1997.

[8] Pei Cao, Jin Zhang, and Kevin Beach. Active Cache: Caching Dynamic Contents on the Web. In *International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 373–388, 1998.

[9] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrel. A Hierarchical Internet Object Cache. In *Winter USENIX Conference*, 1996.

[10] Mark Crovella and Paul Barford. The Network Effects of Prefetching. In *IEEE Infocom*, San Francisco, CA, USA, 1998.

[11] Kenneth M. Curewitz, P. Krishnan, and Jeffrey Scott Vitter. Practical Prefetching via Data Compression. In *ACM SIGMOD International Conference*, pages 257–266, Washington, DC, USA, June 1993.

[12] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey C. Mogul. Rate of Change and other Metrics: a Live Study of the World Wide Web. In *USENIX Symposium on Internet Technology and Systems*, pages 147–158, Berkeley, CA, USA, December 1997.

[13] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. In *ACM SIGCOMM Conference*, 1998 (to appear).

[14] Michael Franklin and Stanley Zdonik. A Framework for Scalable Dissemination-Based Systems. In *ACM OOPSLA Conference*, 1997.

[15] James Griffioen and Randy Appleton. The Design, Implementation, and Evaluation of a Predictive Caching File System. Technical Report CS-264-96, Department of Computer Science, University of Kentucky, June 1996.

[16] Quinn Jacobson and Pei Cao. Potential and Limits of Web Prefetching Between Low-Bandwidth Clients and Proxies. In *Web Caching Workshop*, June 1998.

[17] Tracy Kimbrel, Andrew Tomkins, R. Hugo Patterson, Brian Bershad, Pei Cao, Edward W. Felten, Garth A. Gibson, Anna R. Karlin, and Kai Li. A Trace-Driven Comparison of Algorithms for Parallel Prefetching and Caching. In *USENIX Association Symposium on Operating Systems Design and Implementation*, pages 19–34, October 1996.

[18] Achim Kraiss and Gerhard Weikum. Vertical Data Migration in Large Near-Line Document Archives Based on Markov-Chain Predictions. In *VLDB International Conference*, pages 246–255, Athens, Greece, August 1997.

[19] Balachander Krishnamurthy and Craig E. Wills. Study of Piggyback Cache Validation for Proxy Caches in the World Wide Web. In *USENIX Symposium on Internet Technology and Systems*, pages 1–12, Berkeley, CA, USA, December 1997.

[20] Balachander Krishnamurthy and Craig E. Wills. Piggyback Server Invalidation for Proxy Cache Coherency. In *International World Wide Web Conference*, 1998.

[21] Thomas M. Kroeger and Darell D. E. Long. Predicting File System Actions from Prior Events. In *Winter USENIX Conference*, 1996.

[22] Thomas M. Kroeger, Darrell D. E. Long, and Jeffrey C. Mogul. Exploring the Bounds of Web Latency Reduction from Caching and Prefetching. In *USENIX Symposium on Internet Technologies and Systems*, pages 319–328, San Diego, CA, USA, January 1997.

[23] Hui Lei and Dan Duchamp. An Analytical Approach to File Prefetching. In *USENIX Annual Technical Conference*, pages 275–288, Berkeley, CA, USA, January 1997.

[24] Paolo Lorenzetti, Luigi Rizzo, and Lorenzo Vicisano. Replacement Policies for a Proxy Cache. http://www.iet.unipi.it/ ˜luigi/ caching.ps.gz, 1997.

[25] Radhika Malpani, Jacob Lorch, and David Berger. Making World Wide Web Caching Servers Cooperate. In *International World Wide Web Conference*, 1995.

[26] Carlos Maltzahn, Kathy J. Richardson, and Dirk Grunwald. Performance Issues of Enterprise Level Web Proxies. In *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 13–23, New York, NY, USA, June 1997.

[27] Evangelos P. Markatos. Main Memory Caching of Web Documents. In *International World Wide Web Conference*, Paris, France, May 1996.

[28] Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential Benefits of Delta Encoding and Data Compression for HTTP. In *ACM SIGCOMM Conference*, pages 181–194, Cannes, France, September 1997.

[29] Todd C. Mowry, Angela K. Demke, and Orran Krieger. Automatic Compiler-Inserted I/O Prefetching for Out-of-Core Applications. In *USENIX Association Symposium on Operating Systems Design and Implementation*, pages 3–17, October 1996.

[30] John K. Ousterhout. Why Aren't Operating Systems Getting Faster as Fast as Hardware? In *Summer USENIX Conference*, pages 247–256, 1990.

[31] John K. Ousterhout, Hervé Da Costa, David Harrison, John A. Kunze, Mike Kupfer, and James G. Thompson. A Trace-Driven Analysis of the UNIX 4.2 BSD File System. In *Symposium on Operating Systems*, pages 15–24, Orcas Island, WA, USA, December 1985.

[32] Venkata N. Padmanabhan and Jeffrey C. Mogul. Using Predictive Prefetching to Improve World Wide Web Latency. *ACM SIGCOMM Computer Communication Review*, 27(3):22–36, 1996.

[33] Themistoklis Palpanas. Web Prefetching Using Partial Match Prediction. Technical Report CSRG-376, Department of Computer Science, University of Toronto, March 1998.

[34] R. Hugo Patterson, Garth A. Gibson, and M. Satyanarayanan. A Status Report on Research in Transparent Informed Prefetching. *SIGOPS Operating Systems Review*, 27(2):21–34, 1993.

[35] Peter Scheuermann, Junho Shim, and Radek Vingralek. A Case for Delay-Conscious Caching of Web Documents. In *International World Wide Web Conference*, Santa Clara, CA, USA, April 1997.

[36] Junho Shim, Peter Scheuermann, and Radek Vingralek. A Unified Algorithm for Cache Replacement and Consistency in Web Proxy Servers. In *International Workshop on the Web and Databases*, Valencia, Spain, March 1998.

[37] Renu Tewari, Harrick M. Vin, Asit Dan, and Dinkar Sitaram. Resource-Based Caching for Web Servers. In *SPIE/ACM Conference on Multimedia Computing and Networking*, 1998.

[38] Amin Vahdat, Eshwar Belani, Paul Eastham, Chad Yoshikawa, Thomas Anderson, David Culler, and Michael Dahlin. WebOS: Operating System Services for Wide Area Applications. In *Seventh Symposium on High Performance Distributed Computing*, 1998.

[39] Stephen Williams, Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, and Edward A. Fox. Removal Policies in Network Caches for World-Wide Web Documents. In *ACM SIGCOMM Conference*, pages 293–305, New York, NY, USA, August 1996.

[40] Craig E. Wills and Joel Sommers. Prefetching on the Web Through Merger of Client and Server Profiles. http://www.cs.wpi.edu/ ˜cew/ papers/ webprofile.ps, 1997.