Leveraging Web prefetching systems with data deduplication

Pedro Neves, Paulo Ferreira, João Barreto
INESC-ID
Technical University Lisbon
Lisbon, Portugal
[pedro.n.neves, paulo.ferreira, joao.barreto]@ist.utl.pt

***Abstract*— The continued rise in internet users and the ever-greater complexity of Web content eventually led to some degradation in user-perceived latency in the Web. Web caching, prefetching and data deduplication are techniques that were used to try to mitigate this effect. Regardless of the amount of bandwidth available for network traffic, Web prefetching has the potential to consume free bandwidth to its limits. Deduplication explores data redundancies to reduce the amount of data transferred through the network, thereby freeing occupied bandwidth. Therefore, by combining these two techniques, one can expect that it will be possible to significantly reduce the amount of bytes transmitted per each Web request. It is also legitimate to expect that this reduction will correspondingly improve the user-perceived latency in the Web. In the present work, we developed and implemented a system that combines the use of Web prefetching and deduplication techniques. The main objective of that system is to improve user-perceived latency in the Web, relative to current best-of-class systems. The results obtained lead us to conclude that care must be taken when applying the combination of prefetching and deduplication techniques to web navigation. When savings in the amount of transferred bytes are most critical, if a small degradation on latency can be tolerable, then it is a good option to combine both techniques. If however, the most critical for the user is to minimize latency at all costs, and an increase in the amount of bytes transmitted is not an issue, then the best option is to use just prefetching.**

Keywords—prefetching; deduplication; web; latency

I. INTRODUCTION

The generalized dissemination of the Internet and corresponding growth of the World Wide Web have led to a continued rise in the number of connected users and, consequently, to a dramatic increase in global traffic over the years. In parallel, static Web pages, typical of the first years of the internet, have gradually been replaced by increasingly complex pages, filled with rich-media content, with ever-increasing size [1]. As a consequence, at a given point in time the quality of service and, in particular, the user-perceived latency have experienced some degradation. Nevertheless, the improvements in available bandwidth observed in recent years have reduced the impact of network load in user-perceived latency.

Web prefetching attempts to overcome the above-mentioned limitations by proactively fetching resources without waiting for user requests. Preloading web resources into local caches has shown to have an impact in reducing response times, by significantly improving hit rates [2]. Recent client access history is typically used to predict which resources are likely to be requested in the near future. Due to their speculative nature, prefetch predictions are subject to failure, which can lead to cache pollution, bandwidth wasting and overloading of original servers. Therefore, this technique demands careful application – e.g. during user idle times [3] – otherwise it can significantly degrade performance, defeating its original purpose. Web prefetching has been already the subject of some research [2],[4],[5]. However, its use in commercial products is still marginal [6], mainly due to its potential impact on bandwidth.

Deduplication is a technique that reduces overall data footprint through the detection and elimination of redundancies. The specificity of the types of resources transferred on the Web and the increasing dynamicity of Web resources pose additional problems relative to the application of deduplication in other domains. Nevertheless, several attempts made to use data deduplication in the Web have already shown promising results [7],[8],[9],[10]. These works show that data deduplication considerably reduces the overall amount of transmitted bytes and, thereby, it can reduce the user-perceived latency. Data deduplication leverages on the redundancy present in the data transferred on the Web. Some of that redundancy is already eliminated through caching and compression techniques. However, these techniques cannot detect all forms of cross-file and cross-version redundancy [11],[12].

As seen above, independently of the amount of available bandwidth, Web prefetching is able to take bandwidth occupancy to its limits. On the other hand, data deduplication is a technique that allows reducing the volume of transferred data through the network, thereby freeing occupied bandwidth. Therefore, it is feasible to expect that the combined use of

both techniques could potentially bring significant overall improvements to the amount of transmitted bytes per Web request. It is also legitimate to expect that this reduction can have a corresponding effect in the user-perceived latency in the Web. To our knowledge, no previous work has been done which applies these two techniques combined to Web traffic.

In this paper we present a system we designed and implemented that, by combining the use of Web prefetching and data deduplication techniques, aims to improve user-perceived latency in Web navigation, relative to present day standards. We used as starting point an existing Web prefetching simulator framework [33] and extended it, implementing a deduplication technique based on a recent state-of-the-art deduplication system [10].

Based on the obtained results we conclude that the combination of prefetching and deduplication techniques can be a two-edged sword and, therefore, should be applied with care in web navigation. If reducing the total amount of transferred bytes is the most important aspect to consider, and a small impact on the latency can be tolerable by the end user, then it is a good option to combine both techniques. If however, the only concern is to minimize latency at all costs, the amount of bytes transmitted being a secondary issue, then the best option is to use just prefetching.

The remainder of this paper is structured as follows. First we will present the related work regarding web prefetching and deduplication techniques. Then we will describe the architecture of the developed system. Afterwards, we present and discuss the results of our tests. Finally, we present the conclusions to this work.

II. Related Work

A. Web Prefetching

Prefetching is a well-known approach to decrease access times in the memory hierarchy of modern computer architectures [13],[14],[15]). The basic principle behind it is to anticipate future data requests, and getting the necessary data into cache in the background, before an explicit request is made by the CPU. Prefetching has also been proposed as a mechanism to improve user-perceived latency in the Web [16],[17]. Web prefetching's purpose is to preprocess a user's request before it is actually demanded and, in this way, hide the request latency.

The idea seems promising, however prefetching is not straightforward to evaluate and has not yet been widely implemented in commercial systems. It can be found in some browser add-ons [18] and workgroup proxy caches [19], that will prefetch the links of the page the user is currently browsing, or periodically prefetch the pages in a user's bookmarks.

Prefetching is usually transparent to the user: there is no interaction between the user and the prefetching system. Prefetching systems are speculative by nature and therefore there is an intrynsic probability for the predictions to fail. If the prediction is not accurate, cache pollution, bandwidth waste and overload of the original server can occur. Another concern is the need to determine what content may be safely prefetched, as some Web requests can have undesirable side-effects, such as adding items to an online shopping cart. Prefetching must thus be carefully applied: *e.g.*, using idle times in order to avoid performance degradation [20]. Despite the mentioned risks, it has been demonstrated that several prefetching algorithms [21],[22],[23],[24],[25] can considerably reduce the user-perceived latency.

The prefetching predictions can be performed by the server, by the proxy, or by the client itself. Some works suggest performing the predictions at the server [22],[23], argumenting that its predictions can be quite accurate because it is visited by a high number of users. Other studies defend that proxy servers can perform more accurate predictions because their users are much more homogeneous than in an original server, and they can also predict cross-server links, which can reach about 29% of the requests [26]. On the other hand, some authors consider that predictions must be performed by the client browser, because it is better aware of users' preferences [27],[28]. Finally, some studies indicate that the different parts of the web architecture (users, proxies and servers) must collaborate when performing predictions [23].

A generic architecture for prefetching systems is defined: two new elements are added to the generic client-server Web architecture, the prediction and prefetching engines. These can be located in the same or different elements, at any part of Web architecture.

The prediction engine is the part of the prefetching system that has the purpose of guessing which will be the next user's requests. It can be located at any part of the web architecture, whether in the clients [27],[28], in the proxies [24],[29] or in the servers [30],[31],[32],[33]. It can even work in a collaborative way between several elements [23]. The patterns of user's accesses differ depending on the element of the architecture in which the prediction engine is implemented, simply due to the fact that the information each one is able to gather is totally diverse. For example, a predictor located at the web server is not able to gather cross-server transitions and therefore is limited to transitions between pages of the same server. Several algorithms that attempt to predict future accesses were devised [16],[34],[35],[36],[37],[38],[39],[40].

The prediction engine outputs a hint list, which is a set of URIs that are likely to be requested by the user in a near future. The predictor must distinguish between prefetchable and non-prefetchable resources. A web resource is prefetchable if and only if it is cacheable and its retrieval is safe [41].

The prefetching engine's function is to preprocess resource requests that were predicted by the prediction engine. By preprocessing the requests in advance, the perceived latency when the resource is actually requested by the user is reduced. Most proposals for preprocessing have focused mainly on the transference of requested resources in advance by the client [2],[22],[42],[43],[44],[45]. Nevertheless, some works consider the preprocessing of a request by the server [46],[47], whilst others suggest the pre-establishment of connections to the server [48]. Therefore, the prefetching engine can be located at the client, at the proxy, at the server, or in several of these elements.

The ability of web prefetching to reduce user-perceived latency is strongly dependent on where the prefetching engine is located: the closer to the client it is implemented, the higher its impact on latency. Therefore, a prefetching engine located at the client can reduce the whole user-perceived latency. In order to avoid interference between prefetching actions and current user requests, the prefetching engine can consider external factors to decide whether and when to prefetch a resource hinted by the prediction engine. For example, when the prefetching engine is located at the client some commercial products only start the prefetching after the user is idle [6],[49].

Despite having been a subject of research already for some years, the commercial penetration of web prefetching is still poor, mainly due to its potential impact on bandwidth. One of the most representative examples of Web sites that take advantage of the prefetching technique is Google Search: in some situations, a prediction hint pointing to the URL of the first result is included in the response header [50]. In Academia, we highlight a simulation framework developed in the University of Valencia [51]. This Web prefetching framework models the generic Web prefetching architecture, allowing to implement and to check the performance of prefetching algorithms. It is composed of:

- the back end part which has both a surrogate proxy server and the real web server. The web server is external to the simulator environment, which accesses it through the surrogate. The surrogate module is where the prediction engine is implemented.

- the front end has the client component, which represents the user behavior in a prefetch-enabled client browser. The user web access pattern is simulated by feeding the framework with real traces obtained from a Squid proxy. The prefetching engine is implemented in the client component.

B. Data deduplication on the Web

Deduplication attempts to reduce data footprint through detection and elimination of redundancy in the data. Based on these principles it has been applied to the Internet with the purpose of reducing the total transmitted bytes and, consequently, the user-perceived response time. Deduplication leverages on the redundancy present in the data transferred on the web [7],[8],[11].

The major types of deduplication techniques currently used for the Web can be grouped under three main categories:

- Cache-based approaches: in these, the browser keeps whole resources in its cache, indexed by URL. When a URL is requested to the server, the browser checks if the resource is already present in its cache. In case it is, the browser checks a timestamp to see if the resource in cache is up to date. In that case, the browser does not download the content. Otherwise, the whole resource is downloaded; the same also happening if the resource is not present in the cache at all. This approach is currently used on all Internet browsers, and corresponds to the HTTP specification [52]. It has worked well due to its simplicity of use, both on browser and server sides. Nevertheless, it presents some drawbacks, such as: even if the resource hasn't changed, it can happen that the timestamp is modified by the server; in cases like this, when there is a new request for the resource the client may have the same content in its cache, but because of the new timestamp the whole resource will still be downloaded. Furthermore, if client and server have slightly different versions of a given resource, the classic cache approach will not take advantage of the significant redundancy existing between the versions of the resource. The most relevant work in this area is Edge Side Includes (ESI) [53], by Akamai;

- Delta-encoding: in delta-encoding, two files are compared and their differences are computed. The result of this direct comparison – the delta – is thereby obtained. When applied to the Web it means that after a first download of a complete page, in a second request, if there were changes to the page, a delta can be computed between the two versions of that same page. The client then downloads only the delta and reconstructs the new version of the page from the one in its cache and from the delta. One of the issues in this approach is that it demands that the server keeps at all times the latest version of a reference file that was sent to each client, which has a negative impact on disk space usage. Also, with increasing number of reference resources comes an increasing performance overhead: to improve redundancy detection the delta can be encoded from several reference files and this implies that they all need to be analyzed. On the other hand, the redundancy detection algorithm is executed locally on the server with no additional data being transferred between client and server other than the encoded delta and file version metadata. This means that there is no transfer of redundant data. Optimistic Deltas [54] and Cache Based Compaction [55] are examples of relevant work in this area;

- Compare-by-hash: In compare-by-hash both client and server divide resources in data blocks ("chunks"), which are identified by a cryptographic hash and are treated as autonomous data units: they are shared by different resources in cases where the data is redundant. When the client requests a new version of a resource the server determines which chunks correspond to that resource version and sends their hashes to the client. The client then compares the hashes sent by the server with the ones it has stored locally and computes which chunks it still needs to reconstruct the new resource version. Following, it requests the server only these chunks it does not have locally. The server answers the request by sending the new chunks and also the hashes of the redundant chunks. In this way the client is able to properly reconstruct the new resource. One of the problems of compare-by-hash is therefore that it needs an additional roundtrip, since besides the resource request the corresponding hashes must be sent from server to client and vice-versa. Delta-encoding, on the other hand, only sends control data in the response for resource reconstruction. Furthermore, the redundancy detection phase requires exchange of information through the network and so is not a local algorithm. Thus, it suffers double the network latency and packet loss problems and is slower than delta-encoding, which can be an important issue in a real-time system like the Web. In compare-by-hash, finding a compromise regarding chunk size is critical: if it is too small there will be too many hashes to trade between client and server, resulting in a large communication overhead. A large number of chunks also

results in an increased memory use by chunk's hashes. Conversely, the larger the size of the used chunks, the less the possibilities for redundancy detection. Main works in this area are found in refs [7],[9].

A recent state-of-the-art deduplication system is DedupHTTP [10]. It is an end-to-end deduplication system for text resources. Its algorithm combines delta-encoding and compare-by-hash schemes, acting mostly on the server side, with manageable server state and low communication overhead. It presents several advantages over caching, delta-encoding and compare-by-hash, namely:

- it can be deployed in several points of the network architecture;

- supports any Web Browser and Web Server when deployed on proxies;

- it does not enforce synchronization between client and server;

- the redundancy detection algorithm for online deduplication has a low computational overhead;

- the communication protocol has a low overhead, involving only one network roundtrip;

III. Architecture

As seen above, independently of the amount of bandwidth that is available, Web prefetching can end up consuming all free bandwidth. Data deduplication is a technique that can reduce the amount of data transferred in the network, freeing occupied bandwidth. Therefore, the combined use of both techniques is expected to potentially bring considerable improvements to the amount of transmitted bytes per Web request. It is also expectable that this reduction can have a corresponding reduction in the user-perceived latency in Web navigation.

The system we designed and implemented uses both prefetching and deduplication techniques, with the purpose of optimizing user-perceived latency in Web navigation. We used as starting point the Web prefetching simulator framework previously mentioned in section II.A [51], made publicly available by its authors. We then extended this system in order to implement data deduplication. We used a deduplication technique based on a recent state-of-the-art system that applies deduplication to web traffic, dedupHTTP [10]. To this purpose, we introduced additional functional modules in the system that provide deduplication processing capabilities and also the necessary data and communication infrastructure needed to support the new functionalities and allow the new modules to interface with the existing prefetching framework.

IV. Evaluation

A. Metrics

The main objective of the present work was to implement a system that improved user-perceived latency in Web navigation, by combining prefetching and deduplication techniques. In order to evaluate the performance of the system, we use the following metrics:

- *Latency*: we define request latency as the time measured between the start point of a client HTTP GET request for a given resource, until the full reconstruction of that resource on the client module. We report the *latency per resource* value.

- *Bytes saved*: we define the saving in bytes transferred per request as the sum of redundant chunks byte sizes, *i.e.* the sum of the byte sizes of the data chunks that are not sent from surrogate to client, but rather obtained from the client reference resources. We report the *bytes saved* normalized value, relative to operation with prefetch settings identical to the prefetching setting of the measurement (*e.g.*, prefetching turned on and deduplication turned on relative to prefetching turned on and deduplication turned off- PFON_DDPON relative to PFON_DDPOFF).

B. Experiment

The experimental setup used in our tests consisted of two machines:

- one machine performs the role of the web client (Intel Core2Duo@2.66GHz processor, 4GB RAM; OS: Linux Mint 14-32b); it runs the system client module; this machine contains also the traces that simulate the user's web accesses, which are fed to the client module;

- a second machine performs the role of the surrogate (Intel Core i5-2450M@2.5GHz, 6GB RAM; OS: Linux Mint 14-32b); this machine is running both an instance of the simulator's surrogate module and an instance of a HTTP server (Apache); the HTTP server has stored locally all the workloads files;

The experiments were run in a LAN, with a bandwidth of 54Mb/s. For the tests with constrained bandwidth, we simulated the use of Bluetooth, i.e. effective bandwidth of 2.1Mbs. To force the bandwidth to this maximum value, we used the 'tc' utility by adjusting parameters in its configuration file, and we monitored the effective bandwidth value using the 'iperf' utility. The workloads used in the various tests were obtained by downloading the files of a typical news website (www.dn.pt), up to 3 levels depth. In the experiments we request all the workload files, which amounts to approx. 20MB of files transferred between client-server. Client requests are automated using the wget utility. In order to simulate the web navigation by a real user, introducing idle time between user requests to allow prefetching actions, we used the wget switches '-w10' and '--randomwait', which introduces a random delay of [5, 15] seconds between requests. All the tests reported were performed 5 times for each experimental condition set, the results presented correspond to the average of those trials.

C. Results

The first tests we performed regarded the deduplication algorithm: previously reported results [10] showed a dependency of the redundancy detection efficiency on chunk size. Since we implemented a similar deduplication algorithm, we wanted to confirm if this relationship was still true when

using prefetching combined with deduplication. We ran tests with prefetching (PF) and deduplication (DDP) turned on (PFON-DDPON), for several chunk sizes: 32, 64, 128, 256, 512, 1024 and 2048 bytes. The best redundancy detection is obtained for a chunk size of 128 bytes, corresponding to a reduction of almost 34.2% in data transferred. Due to these results, for the remaining tests with deduplication turned on we used a chunk size of 128 bytes.

We then tested the following running conditions: PFOFF-DDPOFF vs PFOFF-DDPON and PFON-DDPOFF vs PFON-DDPON. Results for bytes saved and latency for PFOFF-DDPOFF vs PFOFF-DDPON are shown in Fig. 1 and Fig. 2 respectively; Fig. 3 and Fig. 4 present the results obtained for PFON-DDPOFF vs PFON-DDPON.

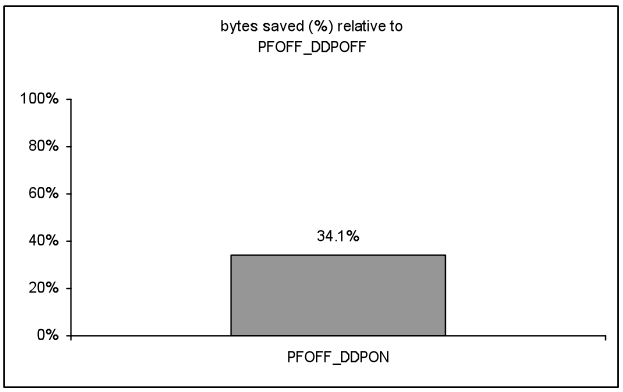

Fig. 1. Graph of bytes saved for PFOFF_DDPON, relative to PFOFF_DDPOFF (Bw=54Mbs).

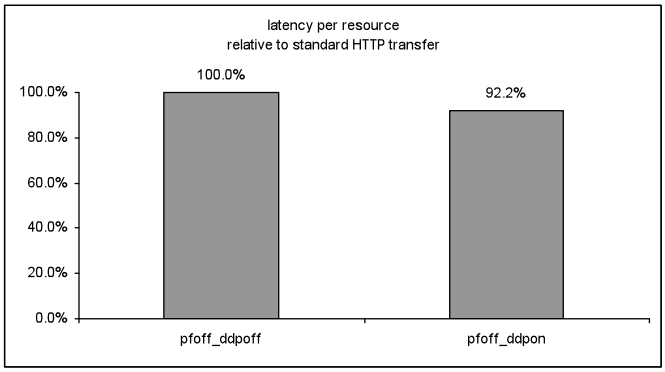

Fig. 2. Graph of latency per resource, relative to standard HTTP transfer: PFOFF_DDPOFF vs PFOFF_DDPON (Bw=54Mbs).

The results for normalized bytes saved in Fig. 1 and Fig. 3 are coherent and show that deduplication allows to obtain approximately 34% in the amount of bytes saved.

The latency results in Fig. 2 show that deduplication allows to reduce the latency in approx. 8% relative to the standard HTTP transfer. However, the comparison in the graph of Fig. 4 shows that, when prefetching is used, the savings in latency are clearly higher when deduplication is turned OFF than when deduplication is used (14.7% and 8.9% saving in latency, respectively). This difference is significant and may be justified by the computational overhead introduced by the deduplication processing.

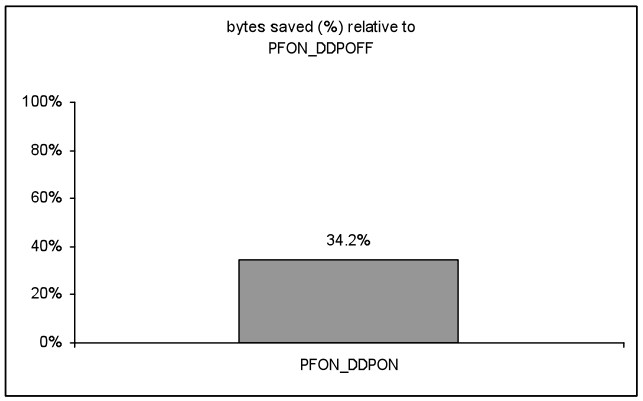

Fig. 3. Graph of bytes saved for PFON_DDPON, relative to PFON_DDPOFF (Bw=54Mbs).

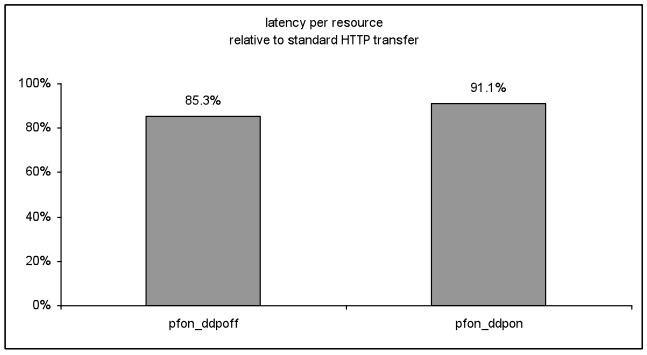

Fig. 4. Graph of latency per resource, relative to standard HTTP transfer: PFON_DDPOFF vs PFON_DDPON (Bw=54Mbs)

These results show that when both prefetching and deduplication are used, deduplication allows to obtain savings in the amount of bytes, but at the cost of having an increased latency relative to the case when only prefetching is turned on.

In order to evaluate the effects of constraining the bandwidth in the achievable redundancy values, we performed comparison tests of PFON_DDPON operation using LAN conditions (54Mbs) vs Bluetooth conditions (2.1Mbs). The results are presented in Fig. 5 and Fig. 6:

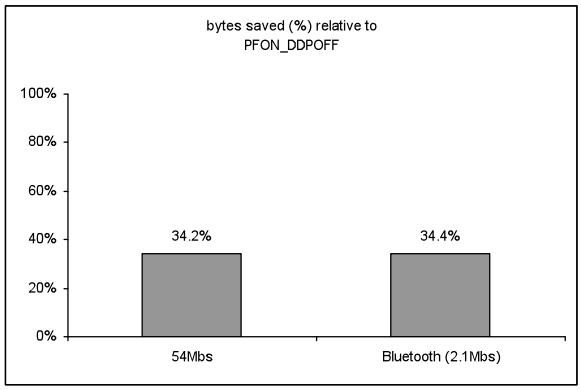

Fig. 5. Graph of bytes saved for PFON_DDPON, relative to PFON_DDPOFF: 54Mbs vs 2.1Mbs.

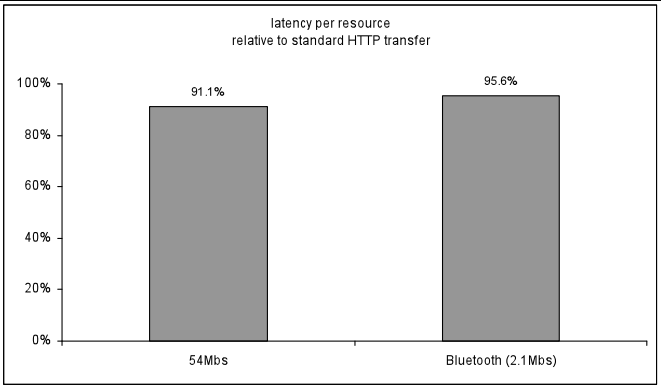

Fig. 6. Graph of latency per resource, relative to standard HTTP transfer: 54Mbs vs 2.1Mbs (PFON_DDPON)..

The values obtained in both cases for bytes saved are coherent with previous measurements. The reduction achieved in latency per resource in the Bluetooth case is half of that obtained for LAN conditions (4.4% and 8.9% reduction in latency, respectively), which can be taken as a consequence of operating under constrained bandwidth. A lower number of prefetching events may eventually occur as a consequence of the reduced bandwidth, since prefetching is highly dependent on idle times between user requests. Since the increased latency means that requests take a longer time to be completed, in consequence idle times may be reduced, thereby reducing the opportunity to start prefetching requests.

V. CONCLUSIONS

In the present work we evaluated the possibility of combining deduplication and prefetching techniques to improve the user-perceived latency in the Web. We developed and implemented a system that combines the use of Web prefetching and deduplication techniques with the main objective of improving user-perceived latency in Web navigation.

Our results show that significant reductions in the volume of bytes transferred relative to a normal HTTP transfer are achievable when the two techniques are used in combination, with savings of approximately 34%. Although the gains in latency were more modest, nevertheless we were able to achieve a reduction of approximately 8% in user-perceived latency. Operation under constrained network conditions impacts the attainable gains. Still, over 4% reduction in the latency was achieved.

The results obtained when using both prefetching and deduplication show high gains in the amount of bytes saved (34.2%). Nevertheless, when both techniques are used simultaneously, the latency reduction (8.9%) is lower than in the case where only prefetching is used (14.7%). This is attributable to the computational overhead introduced by the deduplication algorithm.

These results lead us to conclude that the combination of prefetching and deduplication techniques should be applied carefully to Web navigation. In cases where savings in the amount of bytes transferred are critical (such as pay-per-byte Web access), if the impact of approximately 6% on the latency is considered tolerable, then it is a good option to combine both techniques. If however, the most critical for the user is to minimize latency at all costs, with no concerns regarding the amount of bytes transmitted, then the best option is to use just prefetching.

REFERENCES

[1] http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white paper c11- 520862.html.

[2] Li Fan, Pei Cao, Wei Lin and Quinn Jacobson. Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance. In Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pages 178–187, Atlanta, USA, 1999.

[3] Mark Crovella and Paul Barford. The network effects of prefetching. In Proceedings of the IEEE INFOCOM'98 Conference, San Francisco, USA, 1998.

[4] Azer Bestavros. Using Speculation to Reduce Server Load and Service Time on the WWW. In Proceedings of the 4th ACM International Conference on Information and Knowledge Management, Baltimore, USA, 1995.

[5] Xin Chen and Xiaodong Zhang. Popularity-Based PPM: An Effective Web Prefetching Technique for High Accuracy and Low Storage. In Proceedings of the International Conference on Parallel Processing, Vancouver, Canada, 2002.

[6] Darin Fisher and Gagin Saksena. Link prefetching in Mozilla: A Server driven approach. In Proceedings of the 8th International Workshop on Web Content Caching and Distribution (WCW 2003), New York, USA, 2003.

[7] Neil Spring and David Wetherall, A Protocol-Independent Technique for Eliminating Redundant Network Traffic, In Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, 2000.

[8] Ashok Anand, Chitra Muthukrishnan, Aditya Akella, Ramachandran Ramjee, Redundancy in Network Traffic: Findings and Implications, In Proceedings of SIGMETRICS/Performance'09, Seattle, USA, 2009.

[9] Sean Rhea, Kevin Liang, Eric Brewer, *Value-Based Web Caching*, In Proceedings of the 12th international conference on World Wide Web, Budapest, Hungary, 2003.

[10] Ricardo D. Filipe. Deduplication in HTTP traffic- Redundancy detection and supression on the Web. MSc thesis, Departamento de Engenharia Informática, Instituto Superior Técnico, October 2010.

[11] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta encoding and data compression for http. In Proceedings

of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication, SIGCOMM '97, pages 181–194, New York, NY, USA, 1997. ACM.

[12] T. Kelly and J. Mogul. Aliasing on the world wide web: Prevalence and performance implications, In Proceedings of the 11th international conference on World Wide Web, May 07-11, 2002, Honolulu, Hawaii, USA, 2002.

[13] Alan Jay Smith. Cache memories. ACM Computing Surveys, 14(3):473– 530, September 1982.

[14] Elizabeth Shriver, Christopher Small, and Keith Smith. Why does file system prefetching work?. In Proceedings of the 1999 USENIX Annual Technical Conference, pages 71–83, Monterey, CA, June 1999.

[15] Todd C. Mowry. Tolerating Latency Through Software-Controlled Data Prefetching. PhD thesis, Computer Systems Laboratory, Stanford University, March 1994.

[16] Venkata N. Padmanabhan and Jeffrey C. Mogul. Using predictive prefetching to improve World Wide Web latency. Computer Communication Review, 26(3):22–36, July 1996. In Proceedings of SIGCOMM '96.

[17] Thomas M. Kroeger, Darrell D. E. Long, and Jeffrey C. Mogul. Exploring the bounds of Web latency reduction from caching and prefetching. In Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS '97), December 1997.

[18] http://richardyusan.wordpress.com/fasterfox-mod-by-richardyusan-speed-up/

[19] Rhino Software, Inc. http://www.allegrosurf.com

[20] Mark Crovella and Paul Barford. The network effects of prefetching. In Proceedings of the IEEE INFOCOM'98 Conference, San Francisco, USA, 1998.

[21] Azer Bestavros. Using Speculation to Reduce Server Load and Service Time on the WWW. In Proceedings of the 4th ACM International Conference on Information and Knowledge Management, Baltimore, USA, 1995.

[22] Venkata N. Padmanabhan and Jeffrey C. Mogul. Using Predictive Prefetching to Improve World Wide Web Latency. Computer Communication Review, vol. 26, no. 3, pages 22–36, 1996.

[23] Evangelos Markatos and Catherine Chronaki. A Top-10 Approach to Prefetching on the Web. In Proceedings of the INET' 98, Geneva, Switzerland, 1998.

[24] Li Fan, Pei Cao, Wei Lin and Quinn Jacobson. Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance. In Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pages 178–187, Atlanta, USA, 1999.

[25] Xin Chen and Xiaodong Zhang. Popularity-Based PPM: An Effective Web Prefetching Technique for High Accuracy and Low Storage. In Proceedings of the International Conference on Parallel Processing, Vancouver, Canada, 2002.

[26] Dan Duchamp. Prefetching Hyperlinks. In Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems, Boulder, USA, 1999.

[27] Yuna Kim and Jong Kim. Web Prefetching Using Display-Based Prediction. In Proceedings of the IEEE/WIC International Conference on Web Intelligence, Halifax, Canada, 2003.

[28] Kelvin Lau and Yiu-Kai Ng. A Client-Based Web Prefetching Management System Based on Detection Theory. In Proceedings of the Web Content Caching and Distribution: 9th International Workshop (WCW 2004), pages 129–143, Beijing, China, 2004.

[29] Christos Bouras, Agisilaos Konidaris and Dionysios Kostoulas. Predictive Prefetching on the Web and Its Potential Impact in the Wide Area. World Wide Web, vol. 7, no. 2, pages 143–179, 2004.

[30] Stuart Schechter, Murali Krishnan and Michael D. Smith. Using Path Profiles to Predict HTTP Requests. In Proceedings of the 7th International World Wide Web Conference, Brisbane, Australia, 1998.

[31] Themistoklis Palpanas and Alberto Mendelzon. Web Prefetching Using Partial Match Prediction. In Proceedings of the 4th International Web Caching Workshop, San Diego, USA, 1999.

[32] Heung Ki Lee, Gopinath Vageesan, Ki Hwan Yum and Eun Jung Kim. A PROactive Request Distribution (PRORD) Using Web Log Mining in a Cluster-Based Web Server. In Proceedings of the International Conference on Parallel Processing (ICPP'06), Columbus, USA, 2006.

[33] Josep Domènech, José A. Gil, Julio Sahuquillo and Ana Pont.DDG: An Efficient Prefetching Algorithm for Current Web Generation. In Proceedings of the 1st IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), Boston, USA, 2006.

[34] Azer Bestavros. Using speculation to reduce server load and service time on the WWW. In Proceedings of CIKM'95: The Fourth ACM International Conference on Information and Knowledge Management, Baltimore, MD, November 1995.

[35] Mukund Deshpande and George Karypis. Selective Markov models for predicting Web-page accesses. In Proceedings of the First SIAM International Conference on Data Mining (SDM'2001), Chicago, April 2001.

[36] Themistoklis Palpanas and Alberto Mendelzon. Web Prefetching Using Partial Match Prediction. In Proceedings of the Fourth International Web Caching Workshop (WCW99), San Diego, CA, March 1999. Work in progress.

[37] Li Fan, Quinn Jacobson, Pei Cao, and Wei Lin. Web prefetching between low-bandwidth clients and proxies: Potential and performance. In Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '99), Atlanta, GA, May 1999.

[38] Philip Laird. Discrete sequence prediction and its applications. In Proceedings of the Tenth National Conference on Artificial Intelligence, Menlo Park, CA, 1992. AAAI Press.

[39] Brian D. Davison and Haym Hirsh. Toward an adaptive command line interface. Advances in Human Factors/Ergonomics: Design of Computing Systems: Social and Ergonomic Considerations, pages 505–508, San Francisco, CA, August 1997. Elsevier Science Publishers. In Proceedings of the Seventh International Conference on Human-Computer Interaction.

[40] Stuart Schechter, Murali Krishnan, and Michael D. Smith. Using path profiles to predict HTTP requests. Computer Networks and ISDN Systems, 30:457–467, 1998. In Proceedings of the Seventh International World Wide Web Conference.

[41] Brian Davison, The Design and Evaluation of Web Prefetching and Caching Techniques, PhD Thesis, 2002

[42] Dan Duchamp. Prefetching Hyperlinks. In Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems, Boulder, USA, 1999.

[43] Tamer I. Ibrahim and Cheng Zhong Xu. Neural Nets based Predictive Prefetching to Tolerate WWW Latency. In Proceedings of the 20th IEEE International Conference on Distributed Computing Systems, Taipei, Taiwan, 2000.

[44] Ravi Kokku, Praveen Yalagandula, Arun Venkataramani and Michael Dahlin. NPS: A Non-Interfering Deployable Web Prefetching System. In Proceedings of the USENIX Symposium on Internet Technologies and Systems, Palo Alto, USA, 2003.

[45] Bin Wu and Ajay D. Kshemkalyani. Objective-Optimal Algorithms for Long-term Web Prefetching. IEEE Transactions on Computers, vol. 55, no. 1, pages 2–17, 2006.

[46] Stuart Schechter, Murali Krishnan and Michael D. Smith. Using Path Profiles to Predict HTTP Requests. In Proceedings of the 7th International World Wide Web Conference, Brisbane, Australia, 1998.

[47] Heung Ki Lee, Gopinath Vageesan, Ki Hwan Yum and Eun Jung Kim. A PROactive Request Distribution (PRORD) Using Web Log Mining in a Cluster-Based Web Server. In ICPP(2006) 559-568.

[48] Edith Cohen and Haim Kaplan. Prefetching the means for document transfer: a new approach for reducing Web latency. Computer Networks, vol. 39, no. 4, pages 437–455, 2002.

[49] Google Web Accelerator. http://webaccelerator.google.com/.

[50] Google Search. http://google.com/intl/en/help/features.html.

[51] Josep Domènech, Ana Pont, Julio Sahuquillo and José A. Gil. An Experimental Framework for Testing Web Prefetching Techniques. In

Proceedings of the 30th EUROMICRO Conference 2004, pp. 214–221, Rennes, France, 2004.

[52] R. Fielding, J. Gettys, J. Mogul et al, Hypertext transfer protocol–HTTP/1.1, RFC 2616, pp. 1–114, 1999.

[53] ESI - Edge Side Includes: Overview. http://www.esi.org/overview.html

[54] Gaurav Banga , Fred Douglis , Michael Rabinovich, Optimistic deltas for WWW latency reduction, In Proceedings of the USENIX Annual Technical Conference, p.22-22, January 06-10, 1997, Anaheim, California.

[55] Mun Choon Chan and Thomas Y. C. Woo, Cache-based compaction: A new technique for optimizing web transfer. In Proceedings of IEEE INFOCOM, March 1999.