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 degrade user-perceived latency in the Web. Web caching, prefetching and data deduplication are techniques that are 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. To the best of our knowledge no previous work has been done which applies these two techniques combined to Web traffic. The motivation of this work is to ask if, by combining these two techniques, it is possible to significantly reduce the amount of bytes transmitted per user request, thereby improving 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. By testing our system using real-world user navigation traces, we show that deduplication is able to substantially reduce the network costs of prefetching (up to 34% savings in transferred bytes), without degrading the latency gains due to prefetching. By adjusting deduplication parameters it is possible to improve the latency relative to when using only prefetching.**

Keywords—prefetching; deduplication; web; latency

I. Introduction

The generalized dissemination of the Internet and corresponding growth of the WWW have led to a continued rise in the number of connected users. In parallel, Web pages became increasingly complex [1]. As a consequence, the quality of service and, in particular, the user-perceived latency have experienced some degradation.

Web prefetching attempts to overcome the above-mentioned limitations by proactively fetching resources without waiting for user requests. Due to their speculative nature, prefetch predictions are subject to failure, which can lead to bandwidth wasting. Therefore, this technique demands careful application otherwise it can significantly degrade performance, defeating its original purpose.

Deduplication is a technique that reduces overall data footprint through the detection and elimination of redundancies. Several attempts made to use data deduplication in the Web have already shown that it can considerably reduce the overall amount of transmitted bytes and, thereby, the user-perceived latency [4],[5],[6].

Independently of the amount of available bandwidth, Web prefetching is able to take bandwidth occupancy to its limits. On the other hand, data deduplication allows reducing the volume of transferred data, thereby freeing occupied bandwidth. Therefore, it is feasible to expect that the combined use of both techniques could potentially bring significant 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 at improving user-perceived latency in Web navigation. We used as starting point an existing Web prefetching simulator framework [20] and extended it, implementing a deduplication technique based on a recent state-of-the-art deduplication system [7].

Based on the obtained results we conclude that the combination of prefetching and deduplication techniques has the potential to significantly reduce the network costs of prefetching. This is achievable without compromising the latency reduction due to prefetching. By tuning the chunk size value it is possible to adjust the system behavior according to the user's needs, either maximizing savings in bytes transferred or minimizing the latency.

The remainder of this paper is structured as follows. Section II presents the related work regarding web prefetching and deduplication techniques. Section III describes the architecture of the developed system. Section IV presents and discusses the results of our tests. Finally, Section V presents the conclusions to this work.

This work was supported by national funds through FCT – Fundação para a Ciência e a Tecnologia, under projects PEst-OE/EEI/LA0021/2013, PTDC/EIA-EIA/122785/2010 and PTDC/EIA-EIA/113993/2009.

II. Related Work

A. Web Prefetching

Prefetching has been proposed as a mechanism to improve user-perceived latency in the Web [9],[10]. The purpose of Web prefetching is to preprocess each user's request before it is actually demanded and, in this way, hide the request latency. Prefetching is usually transparent to the user: there is no interaction between the user and the prefetching system.

Prefetching systems are speculative by nature, 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. Prefetching must thus be carefully applied: *e.g.*, using idle times in order to avoid performance degradation [11]. Despite these risks, it has been demonstrated that several prefetching algorithms [12],[13] can considerably reduce the user-perceived latency.

Prefetching systems are implemented by adding two new elements 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 has the purpose of guessing which will be the next user requests. It can be located at any part of the web architecture, whether in the clients [14],[15], in the proxies [2],[16] or in the servers [17],[18]. It can even work in a collaborative way between several elements [13]. 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 prefetching engine's function is to preprocess resource requests that were predicted by the prediction engine, thereby reducing the user-perceived latency when the resource is actually requested. The ability to reduce the 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 [2],[9],[19]. In order to avoid interference between prefetching actions and current user requests, the prefetching engine usually only starts the prefetching after the user is idle [3].

Web prefetching has been the subject of academic research already for some years [2],[9],[10],[11],[13],[20]. However, its commercial penetration is still poor, mainly due to the potential impact on bandwidth. Nevertheless, one notable example is the Google Search Web site, in which the top hit for a search query is prefetched in the background while the search results are presented to the user [21].

B. Data deduplication on the Web

Deduplication attempts to reduce data footprint through detection and elimination of redundant 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 [4],[5],[8].

Deduplication techniques currently used for the Web can be grouped under three main categories:

- Classic Caching: 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 and if its timestamp 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 used on all Internet browsers. It has worked well due to its simplicity, both on browser and server sides. Nevertheless, it has some drawbacks, namely it is an all-or-nothing approach: it detects the presence of redundant resources, but it is not able to detect partial redundancy between versions of the same resource.

- Delta-encoding (DE): in DE, two files are compared and their differences – the delta – are computed. This 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 the 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 [22],[23]. Issues in this approach are that it demands that the server keeps at all times the latest version of a reference file that was sent to each client. Also, with increasing number of reference resources comes an increasing performance overhead. 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.

- Compare-by-hash (CBH): In CBH 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 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. It then 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 [4],[6]. CBH therefore needs an additional roundtrip relative to DE, since besides the resource request the corresponding hashes must be sent from server to client and vice-versa. In CBH, 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. Conversely, the larger the size of the used chunks, the less the possibilities for redundancy detection.

III. Architecture

The system we designed and implemented uses both prefetching and deduplication techniques. We used as starting point a Web prefetching simulator framework [20], made publicly available by its authors. This framework is composed of: 1) a back end part which has both a surrogate proxy server, where the prediction engine is implemented, and a real web server, external to the simulator environment; 2) the front end, which has the client component, simulates the user behavior in a prefetch-enabled client browser. The prefetching engine is implemented in the client component.

We used a deduplication technique based on a recent state-of-the-art system that applies deduplication to web traffic, DedupHTTP [7]. It is an end-to-end deduplication system for text resources. Its algorithm combines DE and CBH schemes, acting mostly on the server side, with manageable server state and low communication overhead. It can be deployed in several points of the network architecture and it does not enforce synchronization between client and server.

To develop and implement the final system, we extended the existing prefetching framework in order to implement data deduplication. 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. The final system has two main components: a client module, which implements the prefetching engine and client deduplication functions; a surrogate proxy module, containing the prediction engine and which implements server-side deduplication functions. Detailed information on the final system architecture can be found in Ref. [24].

IV. Evaluation

A. Experiment

The experimental setup used in our tests consisted of two machines: one performing the role of the web client (Intel Core2Duo@2.66GHz processor, 4GB RAM); another acting as the surrogate (Intel Core i5-2450M@2.5GHz, 6GB RAM), which also runs an instance of a HTTP server (Apache).

The experiments were run in a LAN (Bw: 54Mbs). For the tests with constrained bandwidth, we simulated the use of Bluetooth (Bw: 2.1Mbs). The workloads used 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.

To evaluate system performance the metrics used were the *latency per resource* and the amount of *bytes saved*. All the tests reported were performed 5 times for each experimental condition set, the results presented correspond to the average of those trials.

B. Results

We ran tests with both prefetching and deduplication turned on. Since previously reported results for the deduplication algorithm have shown a dependency of the redundancy detection efficiency on chunk size [7], we performed the tests for several average chunk sizes: 32, 64, 128, 256, 512, 1024 and 2048 bytes. All these tests were performed in LAN conditions (Bw: 54Mbs). Results are presented in Table 1.

Chunk size (bytes)	Bytes Saved relative to prefetching without deduplication (%)	StdDev (bytes saved)	Latency per resource relative to standard HTTP transfer (%)	StdDev (latency)
32	30.3%	1.9%	96.3%	3.5%
64	33.1%	2.0%	93.3%	3.4%
128	34.2%	2.1%	91.1%	3.5%
256	33.8%	2.1%	89.9%	3.2%
512	32.8%	1.9%	85.4%	2.7%
1024	31.5%	2.1%	84.0%	2.5%
2048	28.8%	2.0%	83.0%	2.2%
bluetooth (128)	34.4%	2.0%	95.6%	3.2%
prefetching without dedup	---	---	85.3%	1.9%

Table 1- Bytes saved and latency vs chunk size, using both prefetching and deduplication (values for prefetching without deduplication correspond to the baseline case; Bw=54Mbs).

For chunk sizes below 128 bytes, we have lower values for bytes saved and also the latency is worse than for the case where only prefetching is used. Clearly, in these cases there is no advantage in combining both techniques, relative to using just prefetching.

The best value for bytes saved is obtained for a chunk size of 128 bytes, corresponding to a reduction of 34.2% in data transferred. Nevertheless, the latency reduction is lower in this case (8.9%) than when only prefetching is used (14.7%). This difference is significant and may be justified by the computational overhead introduced by the deduplication processing.

In the case of 2048-byte chunks, we obtained lower bytes savings (28.8%) than with 128-byte chunks (34.2%). However, in this case the highest latency reduction value was obtained (17%), a value that is higher than the one measured when only prefetching is used (14.7%).

These results show that when both prefetching and deduplication are used, the adjustment of chunk size allows distinguishing between several limit cases:

- if the main goal is to maximize savings in the amount of bytes transferred, a chunk size of 128 bytes should be used;
- if the user's concern is minimizing latency, chunks of 2048 bytes or higher size should be selected;
- for chunks of size lower than 128 bytes there is no advantage in combining both techniques, relative to using just prefetching.

In order to evaluate the effects of operation under constrained bandwidth, we performed tests using Bluetooth conditions (Bw: 2.1Mbs), with 128 bytes chunk size. The results are shown in Table 1. Comparing with the results for chunk size of 128 bytes in LAN conditions, it can be seen that the bytes savings obtained are similar in both cases (LAN=34.2%, Bluetooth=34.4%). However, the latency reduction achieved in Bluetooth conditions 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 developed and implemented a system that for the first time combines the use of Web prefetching and deduplication techniques, with the objective of improving user-perceived latency in the Web.

The results lead us to conclude that the combination of prefetching and deduplication techniques significantly reduces the network usage costs due to prefetching, without significantly degrading the latency. Careful selection of the chunk size parameter allows tuning the system performance taking into account the user's needs.

When savings in the amount of bytes transferred are critical (such as pay-per-byte Web access), if an impact of approximately 6% on the latency is considered tolerable, then a chunk size of 128 bytes is the best choice. 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 chunk size should be adjusted to 2048 bytes or higher.

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] 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.

[4] 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.

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

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

[7] Ricardo Filipe and João Barreto. Towards full on-line deduplication of the Web. In Proceedings of INFORUM 2010, Braga, Portugal, September 2010

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

[24] P. Neves, *Data Deduplication in Web Prefetching Systems*, MSc Thesis, Technical University of Lisbon, 2013.