Cache Management for Web–Powered Databases

Dimitrios Katsaros
Aristotle University of Thessaloniki, Greece

Yannis Manolopoulos
Aristotle University of Thessaloniki, Greece

INTRODUCTION

In recent years, the World Wide Web, or simply the Web (Berners-Lee, Caililiau, Luotonen, Nielsen, & Secret, 1994), has become the primary means for information dissemination. It is a hypertext-based application and uses the hypertext transfer protocol (HTTP) for file transfers.

During its first years, the Web consisted of static hypertext markup language (HTML) pages stored on the file systems of the connected machines. When new needs arose, e.g., database access, it was realized that we could not afford in terms of storage to replicate the data we want to publish in the Web server's disk in the form of HTML pages. So, instead of static pages, an application program should run on the Web server to receive requests from clients, retrieve the relevant data from the source, and then pack the information into HTML or extensible markup language (XML) format. Even the emerged "semistructured" XML databases that store data directly into the XML format need an application program that will connect to the database management system (DBMS) and retrieve the XML file or fragment. Thus, a new architecture was born: in the traditional couple of a Web client and a Web server, a third part is added, which is the application program that runs on the Web server and serves data from an underlying repository that, in most cases, is a database. This architecture is referred to as *Web-powered database* and is depicted in Figure 1. In this scheme, there are three tiers: the database back-end, the Web/application server, and the Web client.

BACKGROUND

Due to the existence of *temporal locality* in Web request streams, we can exploit the technique of *caching*, that is, temporal storage of data closer to the consumer. Caching can save resources, i.e., network bandwidth, because fewer packets travel in the network, and time, because we have faster response times. Caching can be implemented at various points along the path of the flow of data from the repository to the final consumer. So, we may have caching at the DBMS, at the Web server's memory or disk, at various points in the network (i.e., proxy caches), or at the consumer's endpoint. Web proxies may cooperate so as to have several proxies to serve each other's misses. All the caches present at various points comprise a *memory hierarchy*. The most important part of a cache is the mechanism that determines which data will be accommodated in the cache space and is referred to as the cache *admission/replacement policy*.

Requests for "first-time accessed" data cannot benefit from caching. In these cases, due to the existence of *spatial locality* in request streams, we can exploit the technique of preloading or *prefetching*, which acts complementary to caching. Prefetching may increase the amount of traveling data, but on the other hand, it can significantly reduce the latency associated with every request.

The role of a cache is to store temporally a set of objects that will most probably be requested by its clients. A cache replacement policy assigns a value to every cached object, called *utility value (UV)*, and evicts from cache the object with the least utility value. The aim of the replacement policy is to improve the cache's effectiveness by optimizing two performance measures: the *hit ratio* and the *cost–savings ratio (CSR)*. The former measure is defined as

$$HR = \sum h_i / \sum r_i$$

and the latter is defined as

$$CSR = \sum c_i * h_i / \sum ci * r_i$$

where h_i is the number of references to object i satisfied by the cache out of the r_i total references to i, and c_i is the cost of fetching object i in cache. The cost can be defined either as the object's size s_i or as the downloading latency c_i. In the former case, the CSR coincides with the byte–hit ratio (BHR); in the latter case, the CSR coincides with the delay–savings ratio (DSR).

Figure 1. Architecture of a typical Web-powered database.

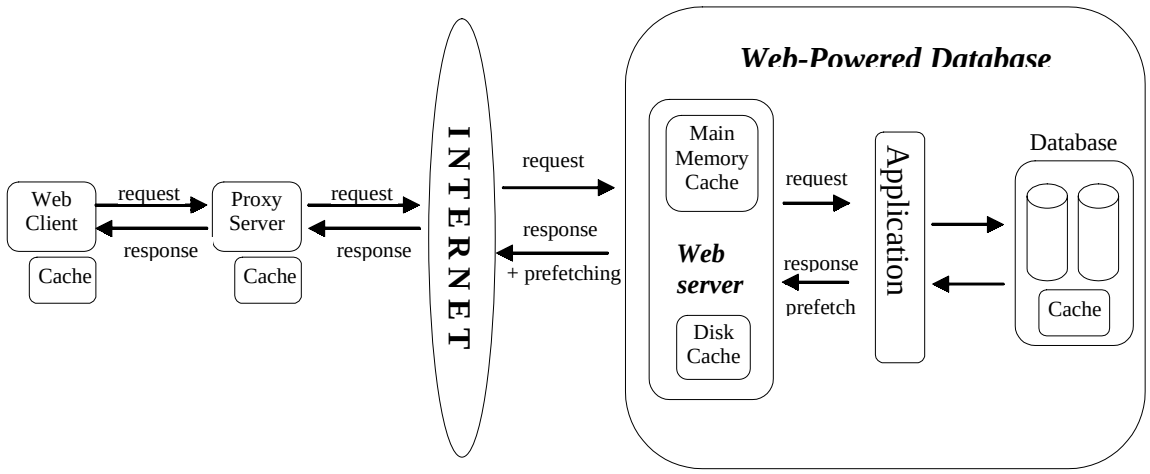

Table 1. A list of factors differentiating Web caching from traditional caching

1. Variable Object Size
The Web object's size varies considerably. It ranges from a few bytes (e.g., small HTML or text files) to several megabytes (e.g., large multimedia files). In contrary, the objects that move through the levels of the caching hierarchy in operating systems or database systems have fixed size, which is equal to the size of a disk block.

2. Variable Fetching Cost
The cost (time penalty) for retrieving a Web object varies significantly. Different objects may have different fetching costs, because they differ in size or in their distance from the client (in terms of network hops). Moreover, the same file may have different fetching costs at different time instances, depending on the server (e.g., heavy load) and network conditions (e.g., congestion).

3. The Depth of the Web Caching Hierarchy
Because caching can happen nearly anywhere, including on the server, on the user's machine, at Internet service protocols (ISPs), at telecommunication companies, at the peering points of national networks, etc., the depth of this hierarchy is significantly larger than the respective depth in computer systems. The large depth of this hierarchy significantly affects the characteristics of the request stream.

4. The Origin of the Web Request Streams
The requests seen by the Web caches, especially by the proxies and the reverse proxies, are not generated by a few programmed processes like the request streams encountered in traditional computer systems. They mainly originate from large human populations with diverse and varying interests.

CACHING IN WEB-POWERED DATABASES

Web cache replacement (Katsaros & Manolopoulos, 2002) is one of the most important areas of Web caching for several reasons. First, studies have shown that the cache HR and BHR grow in a *log-like fashion* as a function of cache size (Breslau, Cao, Fan, Phillips, & Shenker, 1999). Thus, a better algorithm that increases HR by only several percentage points would be equivalent to a several-fold increase in cache size. Second, the growth rate of Web content is much higher than the rate with which memory sizes for Web caches are likely to grow. Finally, the benefit of even a slight improvement in cache performance may have an appreciable effect on network traffic, especially when such gains are compounded through a hierarchy of

caches. There are several factors that distinguish Web caching from caching in traditional computer architectures (see Table 1).

The majority of the replacement policies focus on the first two factors. The main drawback in the design of these policies is that they do not achieve a balance between HR and CSR. Some of them are called *recency-based policies* and favor the HR, e.g., the Greedy Dual-Size (Cao & Irani, 1997), the Size-Adjusted LRU (Aggrawal, Wolf, & Yu, 1999), whereas some others are called *frequency-based policies* and favor the CSR (BHR or DSR), e.g., LRV (Rizzo & Vicisano, 2000), LFU-DA (Diley & Arlitt, 1999), and LNC-R-W3 (Shim, Scheuermann, & Vingralek, 1999). Notable exceptions are the LUV (Bahn, Koh, Noh, & Min, 2002), GD* (Jin & Bestavros, 2001), and CRF (Katsaros & Manolopoulos, 2004), which try to combine recency and frequency. The drawback of LUV and GD* though, is the existence of some manually tunable parameters. CRF is a self-tunable policy that gracefully combines recency and frequency and addresses all the particularities of the Web caching environment.

PREFETCHING IN WEB-POWERED DATABASES

Prefetching is employed in order to cure caching's shortcomings. An effective and efficient prefetching scheme should maximize the number of cache hits due to its action and at the same time minimize the incurred cost due to the prefetched objects. This cost may represent cache space, network bandwidth, server overloading, etc. Although the implementation mechanism of prefetching is very important, the core issue in employing a prefetching scheme is the deduction of future object requests.

In general, there exist two alternatives for the deduction of the future client requests. The first is termed *informed prefetching*, and the second is termed *predictive prefetching*. In Table 2, we explain why the former is not appropriate for the Web environment.

A predictive prefetching algorithm is based on the notion of Markov predictors. Let $T_r = \langle tr_1, \ldots, tr_k \rangle$ be a sequence of consecutive requests (called a transaction) and $S = \langle d_1, \ldots, d_n \rangle$, $n \leq k$ be a sequence of accesses, which is a subsequence of T_r. Given a collection of nTr transactions, where the sequence S appears fr(S) times, then the appearance probability *P(S)* of S is equal to *P(S)* = fr(S)/nTr.

If $S = \langle d_1, \ldots, d_n \rangle$ is a sequence of accesses, then the conditional probability that the next accesses will be to $d_{n+1}, \ldots, d_{n+m}$ is $P(d_{n+1}, \ldots, d_{n+m} | d_1, \ldots, d_n)$, and it equals the following:

$$P(d_{n+1},\ldots,d_{n+m} \mid d_1,\ldots,d_n) = : \frac{P(d_1,\ldots,d_n,d_{n+1},\ldots,d_{n+m})}{P(d_1,\ldots,d_n)}$$

Given a collection of transactions, rules of the form

$$d_1, \ldots, d_n \Rightarrow d_{n+1}, \ldots, d_{n+m} \quad (1)$$

can be derived, where $P(d_{n+1}, \ldots, d_{n+m} | d_1, \ldots, d_n)$ is not less than a user-defined value T_c. $P(d_{n+1}, \ldots, d_{n+m} | d_1, \ldots, d_n)$ is the confidence of the rule. The left part of the rule is called the *head* and has size equal to *n*, whereas the right part

Table 2. Informed and predictive prefetching

Informed Prefetching Can be applied in cases where the client program knows exactly the resources the client is going to request and reveals them into the cache (Patterson, Gibson, Ginting, Stodolsky, & Zelenka, 1995). Informed prefetching is actually a scheduling policy subject to a set of constraints regarding cache space, timeliness of prefetching, and available bandwidth. It requires the communication bandwidth between the applications and the cache to be stable. Thus, it can be implemented only in cases where the cache is embedded into the application, e.g., databases and operating systems.
Predictive Prefetching How will future references be deduced if not revealed by the client program? The only possibility is to take advantage of the spatial locality present in Web request streams. Spatial locality captures the coreference of some resources. The remaining issue is to "quantify" spatial locality, that is, to discover the dependencies between references for different data. Such dependencies can be discovered from past requests and can be used for making predictions about future requests.

is called the *body* and has size equal to *m*. The dependency of forthcoming accesses on past accesses defines a *Markov chain*.

Definition of a *n - m* Markov Predictor

An n - m Markov predictor calculates conditional probabilities $P(d_{n+1}, \ldots, d_{n+m}|d_1, \ldots, d_n)$ between document accesses and discovers rules of the form (1), which are used to predict future accesses.

A predictive prefetching algorithm is defined as a collection of n - 1, n - 2, …, n - m Markov predictors. Existing predictive Web prefetching mechanisms can be categorized into three families.

The algorithms of the first family use the concept of the *dependency graph* (DG) (Padmanabhan & Mogul, 1996; Jiang & Kleinrock, 1998). The DG has a node for each object that has ever been accessed and an arc from node X to node Y, if and only if Y was accessed within w accesses after X and the same client did both accesses. DG maintains the number of accesses to each node X and the number of transitions from X to Y. It calculates conditional probabilities $P(d_i|d_j)$ for all d_i, d_j belonging to a transaction. Therefore, DG discovers rules of the form $d_i \Rightarrow d_j$, or equivalently, it uses a 1-1 Markov predictor.

The algorithms belonging to the second category are based on the notion of a *k-order PPM predictor* (Fan, Cao, Lin, & Jacobson, 1999). A k-order PPM predictor maintains j - 1 Markov predictors, for all $1 \leq j \leq k$ (k is a user-specified constant). It employs Markov predictors that have the constraint that the preceding j requests must be consecutive in the request stream. These Markov predictors calculate conditional probabilities of the form $P(d_{n+1}|d_n), \ldots, P(d_{n+1}|d_{n-k+1}, \ldots, d_n)$ and determine the corresponding rules, which have head sizes equal to 1,2, …, k.

A generalization of the above two families was presented in Nanopoulos, Katsaros, and Manolopoulos (2003). That scheme uses *n - m* Markov predictors that calculate conditional probabilities of the form $P(d_{n+1}, \ldots, d_{n+m}|d_1, \ldots, d_n)$, though the document accesses need not be consecutive. Moreover, the maximum head size *n* and the maximum body size *m* are not set by the user but are adaptively estimated by the data.

FUTURE TRENDS

We all know that the Internet, and consequently the Web, faces performance problems. Performance problems arise in any of the following three general areas:

1. Web server processing delays
2. Internet delays
3. "Last-mile" delays (delays between the subscriber and the Internet, e.g., due to a slow dial-up modem connection)

The idea in alleviating these problems is to make content delivery from origin servers more "distributed," moving some of their content to the "edge" of the Internet. Based upon this idea, the content delivery networks (CDN) emerged recently. CDN are designed to take advantage of the geographic locations of end users. Rather than serving content from the origin Web site, the content distribution model makes copies of "key" content on multiple content delivery servers sites distributed through the Internet, close to the users requesting that content.

A CDN, running on thousands of servers distributed across the Internet, contracts with content providers to deliver their content. A CDN addresses efficiently the aforementioned performance problems:

1. With the "hottest" content "outsourced," the load on the origin server is reduced.
2. The connection from a local content delivery server is shorter than between the origin Web server and the user, thus reducing latency.
3. Because many users share the CDN servers, this service greatly increases the hit ratio.

The deployment of large-scale CDNs brings some new challenges to caching. One very important issue is that of object placement, that is, the selection of the edge servers where an object should be stored (Tang & Chanson, 2002). This problem could also be seen in combination with that of object replacement (Korupolu & Dahlin, 2002). Another very important issue is related to maintaining the coherency of the cached objects, especially in the case that these objects are dynamically generated (Tang & Chanson, 2004).

Prefetching is also challenging in the context of CDN because of the huge number of clients served. To deal with this problem, the cache satellite distribution systems (Armon & Levy, 2004), CSDS for short, have emerged as a technology for feeding the caches with the information clients are expected to request, ahead of time. In such a system, the participating caches periodically report to a central station about requests received from their clients. The central station selects a collection of Web documents that are "pushed" via a satellite broadcast to the participating caches, so that upon a future local request for the documents, they will already reside in the local cache and will not need to be fetched from the terrestrial network.

CONCLUSION

Web-powered databases have emerged to support numerous Web applications, like e-commerce systems, but due to the large number of visitors they attract, they face serious performance problems. To reduce or even alleviate such problems, the techniques of caching and prefetching have been employed. The different physical characteristics of the Web objects, the existence of hyperlinks, and the nature of the Web caching hierarchy call for different caching and prefetching solutions than those investigated in traditional operating and database systems. The present article recorded the most critical factors that affect the design of caching and prefetching policies for Web-powered databases and surveyed the major families and their representatives for these policies.

REFERENCES

Aggrawal, C., Wolf, J., & Yu, P. S. (1999). Caching on the World Wide Web. *IEEE Transactions on Knowledge and Data Engineering, 11*(1), 94–107.

Armon, A., & Levy, H. (2004). Cache satellite distribution systems: Modeling, analysis, and efficient operation. *IEEE Journal on Selected Areas in Communications, 22*(2), 218–228.

Bahn, H., Koh, K., Noh, S. H., & Min, S. L. (2002). Efficient replacement of nonuniform objects in Web caches. *IEEE Computer, 35*(6), 65–73.

Berners-Lee, T., Caililiau, R., Luotonen, A., Nielsen, H. F., & Secret, A. (1994). The World Wide Web. *Communications of the ACM, 37*(8), 76–82.

Breslau, L., Cao, P., Fan, L., Phillips, G., & Shenker, S. (1999). Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of the IEEE Conference on Computer Communications* (INFOCOM) (pp. 126–134). Washington: IEEE Press.

Cao, P., & Irani, S. (1997). Cost-aware WWW proxy caching algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems* (USITS) (pp. 193–206). USENIX Press.

Dilley, J., & Arlitt, M. (1999). Improving proxy cache performance: Analysis of three replacement policies. *IEEE Internet Computing, 3*(6), 44–55.

Fan, L., Cao, P., Lin, W., & Jacobson, Q. (1999). Web prefetching between low-bandwidth clients and proxies: Potential and performance. In *Proceedings of ACM International Conference on Measurement and Modeling of Computer Systems* (SIGMETRICS) (pp. 178–187). ACM Press.

Jiang, Z., & Kleinrock, L. (1998). An adaptive network prefetch scheme. *IEEE Journal on Selected Areas in Communications, 16*(3), 358–368.

Jin, S., & Bestavros, A. (2001). GreedyDual* Web caching algorithm: Exploiting the two sources of temporal locality in Web request streams. *Computer Communications, 24*(2), 174–183.

Katsaros, D., & Manolopoulos, Y. (2002). Cache management for Web-powered databases. In D. Taniar & W. Rahayou (Eds.), *Web-powered databases* (pp. 201–242). Hershey, PA: Idea Group Publishing.

Katsaros, D., & Manolopoulos, Y. (2004). Caching in Web memory hierarchies. In *Proceedings of the ACM Symposium on Applied Computing* (SAC) (pp. 1109–1113). ACM Press.

Korupolu, M. R., & Dahlin, M. (2002). Coordinated placement and replacement for large-scale distributed caches. *IEEE Transactions on Knowledge and Data Engineering, 14*(6), 1317–1329.

Nanopoulos, A., Katsaros, D., & Manolopoulos, Y. (2003). A data mining algorithm for generalized Web prefetching. *IEEE Transactions on Knowledge and Data Engineering, 15*(5), 1155–1169.

Padmanabhan, P., & Mogul, J. (1996). Using predictive prefetching to improve World Wide Web latency. *ACM SIGCOMM Computer Communication Review, 26*(3), 22–36.

Patterson, H. R., Gibson, G. A., Ginting, E., Stodolsky, D., & Zelenka, J. (1995). Informed prefetching and caching. In *Proceedings of the ACM Symposium on Operating System Principles* (SOSP) (pp. 79–95). ACM Press.

Rizzo, L., & Vicisano, L. (2000). Replacement policies for a proxy cache. *IEEE/ACM Transactions on Networking, 8*(2), 158–170.

Shim, J., Scheuermann, P., & Vingralek, R. (1999). Proxy cache algorithms: Design, implementation and performance. *IEEE Transactions on Knowledge and Data Engineering, 11*(4), 549–562.

Tang, X., & Chanson, S. T. (2002). Coordinated en-route Web caching. *IEEE Transactions on Computers, 51*(6), 595–607.

Tang, X., & Chanson, S. T. (2004). The minimal cost distribution tree problem for recursive expiration-based consistency management. *IEEE Transactions on Parallel and Distributed Systems, 15*(3), 214–227.

KEY TERMS

Caching proxy: A caching proxy or proxy server or proxy is a server that acts as an intermediary between a client and a content server. It intercepts the requests of the client and checks whether it can serve the client from its own cache, and if not, it forwards the requests to the content server.

Capacity miss: This miss occurs because the cache cannot accommodate all requested objects.

CDN (content distribution network or content delivery network): This is a network of cache servers owned by the same Internet Service Provider that delivers content to users on behalf of content providers. CDN servers are typically shared, delivering content belonging to multiple Web sites, though all servers may not be used for all sites.

Compulsory miss: A compulsory or cold-start miss occurs in the first access to a Web object.

CPN (capacity provisioning network): This is a network of cache servers owned, operated, and coordinated through capacity trading by different Internet Service Providers. Unlike CDN, with the purpose of replicating content from specifically contracted content providers, CPN's goal is to cache whatever content users access from around the world of content servers. Qualitatively, a CDN services the supply side of content distribution; a CPN services the demand side.

Flash crowd problem: It occurs when the request load overwhelms some aspect of a Web site's infrastructure, such as the front-end Web server, network equipment or bandwidth, or the back-end transaction-processing infrastructure. The resulting overload can crash a site or cause unusually high response times.

Spatial locality: Describes the effect that when an object is referenced, its "nearby" objects will tend to be referenced soon. The notion of "nearness" can include the minimum number of hyperlinks that link the two objects, or it can describe semantic nearness.

Temporal locality: Describes the effect that an object, which has just been referenced, has a high (or increased) probability of being referenced again in the near future. Formally: let $p_i(k)$ denote the probability that, following a reference to object i, the next reference to object i occurs within k references. We say that a particular object shows temporal locality of reference if there exists a $k > 0$ such that $p_i(k) > 1 - (1 - 1/n)^k$.