Web Caching

Antonios Danalis
University of Delaware, USA

INTRODUCTION

The popularity of the World Wide Web has led to an exponential increase of the traffic generated by its users for over a decade. Such a growth, over such a long period of time, would have saturated both the content providers and the network links had Web caching not been efficiently deployed. Web caching can improve the overall performance of the World Wide Web in several ways, depending on the decisions made regarding the deployment of the corresponding caches. By placing caches in strategic positions, the core network traffic can be reduced, the load of a content provider can be scaled down, and the quality of service, as the users perceive it, can be improved. In this article we present an overview of the major design and implementation challenges in Web caching, as well as their solutions.

BACKGROUND

A Web cache can be placed at different positions on the Web and yield different benefits. Namely, it can be placed next to a content provider, in strategic positions inside the network, at the boundaries of local area networks, or inside the host that runs the requesting browser.

Most users tend to access particular pages quite often. An example could be the "home page" that has been set on their browser, or Web sites that contain documentation they are interested in. Furthermore, during browsing, many users visit the same pages multiple times in short periods of time, by using the history of their browsers. To exploit these access patterns, most browsers keep local copies of the frequently or recently accessed pages. This way, the user is served in near zero time, and the network traffic is reduced. Although this technique can have a positive effect on the browsing experience of a single user, it has a minimal effect on the network traffic. This is mostly due to the small disk space used by this type of caching and the lack of sharing between different user caches.

To yield significant results, dedicated computer systems, called proxies, are installed at the edges of local or wide area networks. These systems can achieve significant reduction in the network traffic and improvement in the user perceived quality of service, by filtering and serving the Web requests generated inside the entire network they serve. If a user has defined in the browser's settings a particular proxy to be used, every time he or she requests a Web page the browser will send this request to the proxy. If the proxy happens to have the page, the user will be served promptly without the original content provider being contacted. If the proxy cannot serve the request, it will fetch the appropriate Web objects (such as the text documents, images, applets) from the original server, and possibly keep a copy for later use. Transparent proxies are a special kind of proxy that end users do not explicitly specify in their browser's settings. Rather, the gateway of the network identifies Web requests and forwards them to the proxy. This model is more efficient, and easier to administrate, since all the users of the local network will be using the proxy, and they need not know about possible changes concerning the proxy.

To improve the performance of a content provider, a cache can be deployed on the server side. These caches, also called server accelerators, or reverse proxies, are usually deployed in front of clusters of Web servers (Challenger, Iyengar & Dantzig, 1999) and cache the most popular documents in their main memory. Since a few highly popular documents are in most cases responsible for a large percentage of requests, by using a server-side cache, a Web server can decrease its disk I/O load (Abrams et al., 1995; Markatos, 1996; Tatarinov, Rousskov & Soloviev, 1997) and significantly improve the request serving quality of service. Nevertheless, a reverse proxy benefits only a particular content provider and does not reduce the network traffic.

In addition to client and server-side caching, there exists the approach of adaptive caching (Michel et al., 1998). In this scheme caches can exist at different points inside the network, and be configured to cooperate with one another, in order to improve the overall performance and balance the load. In this model, different caches, potentially belonging to different organizations, are organized into dynamic groups, which each one can join or leave depending on content demand. The communication needed for a group to form and for documents to be located is done through the Cache Group Management Protocol (CGMP) and the Content Routing Protocol (CRP), respectively. Adaptive caching can deal very well with request surges, occurring when some documents become highly popular in a short period of time (usually articles in online newspapers), but assumes that cooperation across administrative boundaries is not an issue.

CRITICAL ISSUES OF WEB CACHING

To improve their effectiveness, caches can be combined in groups to serve requests cooperatively. Such groups may be organized in meshes, as in the case of adaptive caching, or in hierarchical formations, as in the case of the Harvest Cache project (Chankhunthod et al., 1996). In hierarchical designs, caches are usually organized in tree-like structures, and are configured such that child nodes can query their parents and their siblings, but never their children. Although grouping several caches can improve the scalability of the caching system, control messages between a large number of nodes could saturate the parent nodes and the network links (Baentsch, 1997). To make the communication between the caches efficient, several special purpose protocols have been introduced.

The most popular inter-cache communication protocols are ICP, CRP, CARP, cache digest, and WCCP (Melve, 1999). The Internet Cache Protocol (ICP) was developed for the communication of the caches in the Harvest project and was refined within Squid (SQUID). It is a lightweight message format, used by the caches for issuing queries and replies, to exchange information among one another regarding the optimal location from where a certain object can be retrieved. The large number of messages exchanged in ICP, when the number of caches is high, has been shown to impede scalability (Baentsch, 1997; Fan et al., 2000). To avoid similar effects, the Content Routing Protocol (CRP) uses multicast to query cache meshes. CRP exploits the overlapping that exists in cache group formations to propagate queries or popular objects between the groups. To further improve on the performance of the inter-cache communication protocol, cache digests can be used (Fan et al., 2000; Rousskov & Wessels, 1998). Cache digests compact (using a lossy algorithm) the information about the contents of a cache, and make it available to its neighbors. By checking the digests of its neighbors, a cache can identify, with some uncertainty, which neighbors are likely to have a given document. In addition to these protocols, there exist two proprietary ones, CARP and WCCP, designed by Microsoft and Cisco respectively. Unlike the protocols mentioned before, the Cache Array Routing Protocol (CARP) implements routing in a deterministic way. In particular, it uses hashing to identify which member of the proxy array has the requested document. This way it avoids the scalability problems that appear in ICP due to the large number of control messages. The Web Cache Communication Protocol (WCCP) is designed to support transparent caching. The idea is that a router (such as Cisco Cache Engine) that can recognize Web traffic will intercept user requests and redirect them to a cache.

The protocol used for the communication between the Web clients, proxies, and servers is the Hyper-Text Transfer Protocol (HTTP). HTTP runs on top of TCP/IP, which is the most common protocol used in the Internet for reliable transfers, flow control and congestion avoidance. To initialize a connection, TCP sends a special packet (SYN) from the client to the server, and the server responds with an acknowledgment. After initializing the connection, in order to request a page, the client has to send an additional message to the server, and wait for the reply. In the first version of HTTP, this four-step procedure had to take place for all individual objects of a page, such as images, or applets, separately. To reduce the number of round-trips needed, persistent connections, that is, connections that remain open after the retrieval of a file, were proposed (Caceres et al., 1998; Heidemann, Obraczka & Touch, 1997; Mogul, 1995). Persistent connections allow a client to fetch all the components of a Web page by facing the connection startup latency only once. In addition to the multiple steps handshake, slow-start is another feature of TCP that has negative effects in the context of Web transfers. As the name implies, slow starts demand that newly created TCP connections transmit data very slowly and they increase the throughput as they transmit more and more. In the general case, where flow control and congestion avoidance is necessary, this technique is very efficient. In the case of Web transfers though, where the majority is short-lived since most documents are a few kilobytes long (Arlitt & Williamson, 1996; Cunha, Bestavros & Crovella, 1995; Shriver et al., 2001), slow start translates to slow transfer. In order to deal with these issues, HTTP/1.1 (Fielding et al., 1997) was introduced. This new version, among other improvements, supports persistent connections and has been shown to dramatically outperform its predecessor (Nielsen et al., 1997).

In heavily loaded Web caching systems, disk I/O can become a significant bottleneck (Markatos et al., 1999; Mogul, 1999; Rousskov & Soloviev, 1998), because disks are significantly slower than the main memory, and traditional file systems are not optimized to handle Web traffic. To reduce this overhead, Gabber and Shriver (2000) proposed the use of a special purpose file system and Markatos et al. (1999) and Maltzahn, Richardson and Grunwald (1999) proposed the use of special purpose storing policies that reduce the overhead of file creation, deletion, and access. Such policies take into account Web access patterns when they make decisions regarding file placement. For example, in most cases, the HTML text of a Web page and the embedded objects (such as images) of the page will be requested one after the other. By placing these objects in files located close to each other on the disk (Maltzahn, Richardson & Grunwald, 1999), or in neighboring locations within the same file (Markatos et al., 1999), both reading and writing will access almost sequential disk blocks, yielding a significant performance improvement. Additionally, a specialized file system (Gabber & Shriver, 2000) could avoid keeping metadata to improve performance, or could use in-memory data structures to quickly identify cached objects without performing any disk I/O (Tomlinson, Major & Lee, 1999).