Scalable protocols for the Internet to Reduce Service Time and Server Load

P. Krishna Reddy and Masaru Kitsuregawa

Institute of Industrial Science
The University of Tokyo
7-22-1, Roppongi, Minato-ku
Tokyo 106, Japan
{reddy, kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract. In this paper, we have proposed scalable protocols for the Internet to reduce service time and server load. In this approach, we assume that a client cooperates by acting as a server to the cached web pages. To reduce the service time, the server forwards the incoming requests to other clients which have cached the web pages. As a result, the service time scales well as server load increases. Also, with these protocols, the traffic is evenly distributed among the network. This approach is a generalization of caching technology. However, for implementation, the proposed approach requires support from the Internet user, system development, and governing communities. If adopted, we believe that the proposed approach provides a scalable solution to the problems of server overloading and long service times in the Internet.

1 Introduction

Currently, the World Wide Web (or Web) is experiencing exponential growth. In coming ten years, billions of users will employ powerful computing and display devices and try to pull multi-media information, which will put huge demand on the corresponding servers and network links [1]. As a result, load on the popular servers could increase in an unbounded manner. Also, the increasing use of the Web results in increased network bandwidth usage, thus straining the capacity of the networks on which it runs. As a result, it leads to more and more servers becoming "hot-spot" sites where the high frequency of requests makes servicing difficult. The best example is the case of many multi-media information servers on the Internet, which are unreachable as soon as they become popular. The main reason for this is current protocols for accessing distributed information systems do not scale, partly due to the inability of servers to cope with the increasing volume of client requests. Therefore the distributed large-scale, dynamic nature of the Internet speaks to the need for open, flexible, and scalable solutions.

In this paper we propose a scalable approach to cope with the increasing server load. The basic principle derives from client-peer architecture [2], where each participating computing node is capable of both client and server roles. It can be observed that the client-peer architecture is scalable because distributed server can run on all nodes, and therefore server resources could scale with the systems as a whole. That is each computer (new) shares the load and accesses resources from other servers. In this paper we have tried to extend same idea to Internet. In case of the Internet, it can be noticed that a client (user) caches a large number of web pages and this information is not used once client scans these pages. In the proposed approach, for the sake of improved performance, a client cooperates by acting as a server to web pages cached by it. When a server serves information to a client, the necessary information (i.e., details of web pages served and client) is recorded in the cached page informer (CPI) of the corresponding server. With the help of CPI, the server redirects incoming requests to the client which has recently copied corresponding web page. In this way, the server load could be reduced by distributing it among other clients.

Since the WWW suffers with the problems of high latency, network congestion, and server overload, considerable effort has been spent investigating different methods to improve performance. The fundamental issues that have been considered include cache topology, cache replacement policy, cache consistency, whether caching is server- or client initiated, and cache-ability of different objects. There are two basic approaches to caching that have been explored: client side and server side solutions. In the server side solutions, servers shed load by duplicating their documents at caching servers spread throughout WWW [3, 6]. Client side solutions usually use some sort of caching proxy [7] that fields requests from one or more clients and caches objects on the client's behalf. However, proxy caches are not always efficient. First, caching only works with statically and infrequently changing documents. Dynamically created documents, which are becoming more popular with commercial content providers, currently can not be cached. Also, the effectiveness of client-based caching for WWW is limited. In [4] it was concluded that proxy-caching is ultimately limited by the low level of sharing of remote documents amongst clients of the same site. These findings agrees with Glassman's predictions[10] and was further confirmed for general proxy caching by Abrams et al [8].

In [5], the reasons for limited effectiveness of WWW client-based caching are given. The access patterns in a WWW exhibit three locality of reference properties: temporal, geographical and spatial. Temporal locality of reference implies that recently accessed objects are likely to be accessed again in the future. Geographical locality of reference implies that an object accessed by a client is likely to be accessed again in the future by "nearby" clients. The property is similar to the processor locality of reference exhibited in parallel applications. Spatial locality of reference implies that an object "neighboring" a recently accessed object is likely to be accessed in the future. If client-based caching is done on a per-session basis (i.e., the cache is cleared at the start of each client session), then the only locality of reference property that could be exploited is the temporal locality of reference. The results of client based caching study [3] suggest that for a single client, the temporal locality of reference is quite limited, especially for remote documents.

In [9] a protocol that allows multiple caching proxies to cooperate and share their caches, thus increasing their robustness and scalability. Our approach differs from this as in our approach the clients or proxies do not communicate with each other. Our approach is a generalization of "caching" technology to increase Internet performance. Using the proposed approach, both temporal and geographical locality of references could be exploited to reduce server load and service time.

The rest of the paper is organized as follows. In the next section we explain the architectural framework and present server and client protocols. In section 3, we discuss the design issues and advantages. In the last section we provide conclusions.

2 Architectural framework and protocols

In this paper we refer user's machines that access the Internet as clients and computers that provide information as servers. The URL (Uniform resource locator) uniquely identifies a Web page. A web page may be an HTML text file, image file, sound file or video file or combination of all these.

Figure 1 depicts the proposed architecture. In this architecture, each server maintains the CPI database. The received URL requests are stored in the server queue. These URLs are served as per the server protocols, which will be explained later. When a server serves a page, the information such as URL of the page, client address, page size is stored in the CPI database. To reduce the load, the server redirects the URL requests to the corresponding clients which cached these pages. Therefore, from the server, a page is either directly transferred to the target client or its URL request is forwarded to other client which has cached the corresponding page, which would in turn transferred to the requested client.

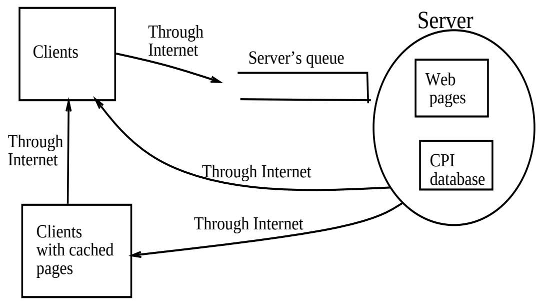

Figure 1: Depiction of architecture

2.1 Simple protocols

We first present client and server protocols under simplified assumptions. Next, we explain drawbacks of these protocols and present final protocols. We assume that each server maintains a CPI database. Also, url_i denotes the URL of the i'th page and $page\_size_i$ denotes the size of the page pointed by url_i.

We make following assumptions for simple protocols.

– All the requests sent by a server reaches to client.
– All pages received by a client are cached by the client. For this purpose we assume that unlimited memory is available at the client.
– A client is up around the clock.

Simple client protocol

1. To access a web page, a client sends url_k to the corresponding server.
2. On receiving a web page, a client caches it.
3. When a client receives a request from the corresponding server to serve the cached page, it sends the page to the client specified by the server.

Simple server protocol

All the client's requests are stored in the server's queue. Initially, CPI contains no information.

1. For each URL request url_k in the server queue, CPI database is checked. If $url_k \notin CPI$ the page is sent directly to the requested client after entering the information $< url_k, client\_id,\ page\_size_k, creation\_time_k >$ in CPI. (In this *client_id*, indicates the address of the client which has requested the web page url_k. The variable $creation\_time_k$ specifies the instant when the web page url_k was created at the server. This variable is used to determine whether the cached version has become obsolete or not.)
2. Otherwise, the request is forwarded to the client which has cached the page url_k after entering the information $< url_k, client\_id,\ page\_size_k, creation\_time_k >$ in the CPI.

2.2 Limitations of simple protocols

We now explain the limitations of simple protocols and propose improvements to overcome them.

– **Server performance**
In simple protocol, a server always redirects the URL requests to those clients which have cached these pages. When the performance is OK, following proposed protocols indeed deteriorates performance by increasing service time. In fact, the proposed solution is effective in situations when the server's performance is unacceptable. We define variable 'threshold service time (TST)' to separate server's mode of operation. The variable TST denotes maximum mean service time at which the server's performance is acceptable. If the service time exceeds TST, we say the server performance is slow and unacceptable.
– **Page size**
The simple protocols do not take the page size into consideration. It is obvious that the request redirected by a server travels first to the client which has cached it and then to the target client. Therefore it travels more distance

than the distance from the server to the client. This results in an overhead if the server depends on the clients to serve even small pages. However, if page is large enough, the overhead caused by longer distance could be nullified by utilizing the server time to process other requests. Obviously it would be efficient if small pages are directly served by a server. To facilitate this, we define a variable *threshold page size* (TPS). If the page size exceeds TPS, the server tries to request the clients to serve the corresponding page.

– **Rejections**
  When a server redirects request to other clients that has cached a web page, the request might not reach the client. Even though it reaches, the client might not oblige the request. Because the redirected request could not have reached the client due to transmission failures or the client's computer might be down. The simple protocol does not consider the possibility of such rejections. We incorporate the handling of rejections in the improved protocol as follows.

  We assume a page could be cached either by a client or by a proxy server/ISP. This information is sent by a client machine when it sends URL request to the server. We assume that a client operates at least for some duration which is termed as a threshold operating time (TOT). Otherwise, if a client (proxy server or ISP) which operates around the clock caches pages, it has to follow cache management mechanisms to accommodate new pages by deleting old pages[11]. We term the duration of cached page as an 'age' of the page. We define the term 'threshold age (TA)', which specifies the maximum duration the client keeps in a cache. If age exceeds TA, the client deletes the page. The variables TOT and TA could be fixed in an adaptive manner.

  If a redirected request could not reach the client which has cached the web page, corresponding error message is received by the server which has redirected the request. In this situation the server has to handle the rejected requests in a prioritized manner and therefore should be served by the server itself. So, in the improved protocol we maintain two queues for a server. One is 'normal queue' in which all the client's URL requests are stored. And the other is 'priority queue' where those requests which could not be served by the clients (cached) are stored.

– **CPI entry for rejected requests**
  The simple protocols assume that the clients always cache web pages sent by the server with out fail. Since the client could reject the request, this assumption is invalid unless the target client confirms the receipt. So, we incorporate boolean variable $confirm$ in the entry of CPI. Initially, the confirm is *no*, when the client acknowledges the receipt of the page the confirm is set to *yes*.

– **Caching by multiple clients**
  It can be observed that if multiple clients request a page within a short duration, it would be cached at multiple clients. In this case when a server receives a request from a client, it has to select appropriate client among multiple clients to forward a request. We assume that given two server addresses,

distance between them can be calculated through appropriate protocol. The distance may be a function of bandwidth of the weakest link, number of hubs, transmission time between the two and so on. By knowing distance, when multiple clients cache a web page, the server forwards the incoming URL request to the client which is nearer to the target client.

2.3 Protocols

We now present final protocols by incorporating the improvements discussed in the preceding section.

Client protocol

1. When a client sends the URL request, it sends the identifier of the machine which would cache that page. (It may be either identifier of the client or proxy server/ISP.)
2. When a client receives a request from the corresponding server to serve the cached page, it sends the page to the client specified by the server. When a client receives a web page it caches that page and sends the acknowledgment to the original server.
3. When a client receives a request from the corresponding server to serve the cached page, it sends the page to the client specified by the server.
4. For a page, if the age exceeds TA (or TOT) it is deleted from the client's cache.

Server protocol It is to be noted that we maintain two queues for a server : priority queue and normal queue (see section 2.2).

1. If the request is from a priority queue, send the page to the requested client after step 6.
2. If the request is from a normal queue and $mean\ service\ time \leq TST$ send the page to the requested client after step 6.
3. If the request is from a normal queue and $mean\ service\ time > TST$ the following actions are performed.
   (a) For each url_k in the input queue if $page\_size_k \leq TPS$ send the page to the requested client.
   (b) For each url_k in the queue, if $page\_size_k > TPS$,
      i. If $url_k \in CPI$ and $confirm = yes$ and $age(url_k) \leq TA$ (or TOT) and cached page is not obsolete, the request is redirected to the corresponding client that has cached url_k after performing actions in step 6. (If multiple clients cache url_k the request is redirected to the client that is nearer to the requested client)
      ii. Otherwise, the page is directly sent to target client after step 6.
4. For any url_k, if $age(url_k) \geq TOT$ ($age(url_k) \geq TA$) the entry is deleted from CPI.
5. If a server receives acknowledgment from the requested client, it updates corresponding CPI entry field as "$confirm = yes$".

6. Suppose the request is for url_k. If $page\_size(url_k) \geq TPS$, then $< url_k, client\_id, page\_size(url_k), creation\_time_k, confirm = no >$ is entered in the CPI database.

3 Design issues and advantages

3.1 Design issues

In the proposed approach, we assume that a client serves the pages it has cached. This requires inclusion of serving capability to the Internet browsers. If some other server (proxy/ service provider) provides caching facility for a client, this should be indicated in the client's request so that the Server could redirect requests to proxy server. This requires modification of browser protocols and data request format.

Also, the proposed approach requires cooperation from web community (user, system development and governing communities) for its realization. Already, proxy servers and mirrors are employed to reduce the server load. The proposed solution is a more generalized notion of a proxy server technology, providing opportunity to every client to cooperate in reducing the server load.

3.2 Advantages

- The implementation of proposed protocols is simple and modular. It is simple because implementation does not lead to major changes in the existing architecture. Also, it is modular because the server can implement the proposed protocols even if a single client cooperates in serving the cached pages. The server protocols could be extended gradually to other clients.
- It is a fully scalable solution. As the server load increases, the number of clients that cache the data increases. Therefore, increasing number of requests are forwarded to clients. With respect to server's point of view this substantially reduces its load. Also, with respect to client's point of view the load imposed on it is very very less. As a consequence, the service time and server load could be considerably reduced without burdening the clients.
- The proposed solution balances the traffic over the entire network. In this approach the clients exchange pages. As clients span over wide area, the corresponding load is also distributed among the network, which increases the link usage.
- If every client cooperates, more load could be served with less powerful architecture. So large amounts of load can be served with less expensive server architectures.

4 Conclusions

In this paper we have proposed scalable server and client protocols to reduce service time and server load in the Internet environment. In this approach, we have assumed that a client cooperates by serving pages cached by it to other clients as directed by original server. With this approach the service time can scale well with the increasing server load. Also, this approach distributes the load evenly among the network. However, the proposed approach requires modification of protocol format, inclusion of a server functionality to existing Internet browsers and support from Internet community for its implementation. Currently, caching technology is employed to reduce the service time and server load. The proposed solution is a generalization of caching technology. As a part of future work, we will evaluate the performance through simulation experiments and then conduct the experiments on the Internet.

Acknowledgments
This work is supported by "Research for the future" (in Japanese Mirai Kaitaku) under the program of Japan Society for the Promotion of Science, Japan.

References

1. Phil Bernstein et. al., The asilomar report on database research, ACM SIGMOD RECORD 27(4), 1998.
2. Thomas E.Anderson, Michael Dahlin, jaenna M.Neefe, David A Patterson, drew S.Roselli, and Randolph Wang. Server-less network file systems. ACM Transactions on Computer Systems, 14(1):41-79, February 1996.
3. Azer Bestavros, Using speculation to reduce server load and service time on the WWW, *proc. of ACM Fourth International Conference On Information and Knowledge Management (CKIM'95)*, 1995, 403-410.
4. Azer Bestavros, Robert Carter, Mark Crovella, Abdelsalam Heddaya, and Sulaiman Mirdad. application level document caching in the internet. In IEEE SDNE'96: The second International Workshop on Services in Distributed and Networked Environments, June 1995.
5. Azer Bestavros and Carlos Cunha, Server initiated document dissemination for the WWW. IEEE Data Engineering Bulletin, 19(3):3-11, September 1996.
6. T.T.Kwan, R.E.McGrath, and D.A.Reed. NCSA's world wide web server : design and performance. IEEE Computer, 28(11):68-74, November 1995.
7. A.Luotonen and K.Altis. Worl-Wide Web proxies. Computer Networks and ISDN Systems, 27(2), 1994. ¡URL: http://www1.cern.ch/ PapersWWW94/luotonen.ps¿
8. Marc Abrams, Charles R.Atandridge, Ghaleb Abdulla, Stephen Williams, and Edward A. Fox. Caching proxies: limitations and potentials. In proceedings of the Fourth International Conference on the WWW, Boston, MA, December 1995.
9. Radhika Malpani, Jacob Lorch and David Berger, Making world wide web caching servers cooperate. In 4th International world wide web conference, pages 107-117, Boston, Dec. 1995.
10. Steven Glassman. Acaching relay for the worl wide web. In proceedings of the first International Conference on the WWW, 1994.
11. S.Williams, M.Abrams, C.R.Standridge, G.Abdulla, E.A.Fox. Removal policies in network caches for Worl-Wide Web documents. ACM SIGCOMM, 1996, pp. 293-305.