2001년 한국정보처리학회 춘계 학술발표논문집 제8권 제1호

Mechanism for Efficient Use of Server's Resource on the Web

Yoon-Jung Rhee, Nam-Sup Park, Tai-Yoon Kim
Dept. of Computer Science & Engineering, Korea University

e-mail: genuine@netlab.korea.ac.kr

Abstract.

HTTP/1.1 standard reduces latencies and overhead from closing and re-establishing connections by supporting persistent connections as a default, which encourage multiple transfers of objects over one connection. HTTP/1.1, however, does not define explicitly connection-closing time but specifies a certain fixed holding time model. This model may induce wasting server's resource when server maintains conn ection with the idle-state client that requests no data for a certain time. This paper proposes the mechanism of a heuristic connection management supported by the client-side under persistent HTTP, in addition to HTTP/1.1's fixed holding time model on server-side. The client exploits the tag information within transferred HTML page so that decides connection-closing time. As a result, the mechanism allows server to use server's resource more efficiently without server's efforts.

1. Introduction

TCP connections are established with a 3-way handshake; and typically several additional round trip times (RTT) are needed for TCP to achieve appropriate transmission speed. Each connection establishment induces user-perceived latency and processing overhead. Opening a single connection per request through connection setup and slow-start costs causes problems of performance and latency. Thus, persistent connections were proposed [3,4] and are new a default with the HTTP/1.1 standard [5]. HTTP/1.1 reduces latencies and overhead from closing and re-establishing connections by supporting persistent connections as a default, which encourage multiple transfers of objects over one connection.

HTTP/1.1, however, must decide when to terminate inactive persistent connections. HTTP/1.1 specifies that connections should remain open until explicitly closed, by either party. That is to say HTTP/1.1 does not define explicitly when to terminate TCP connection. Current implementation of HTTP/1.1 uses a certain fixed holding-time model. This model may induce wasting server's resource. Current latency problems are caused by not only networks problem but also servers overloads having limited resource. This paper proposes the mechanism of a heuristic connection management on the client-side under persistent HTTP, in addition to HTTP/1.1's fixed holding time model on server-side. The client exploits the tag information in transferred HTML page so that decides connection-closing time.

In Section 2 we discuss the related works involved in implementation of persistent connection of HTTP/1.1. Section 3 contains our proposal of connection management. We finish with a conclusions and future works in Section 4.

2. Related Research.

2.1 Issues of Persistent Connection

HTTP/1.1 does not specify explicit connection-closing time but provides only one example for a policy, suggesting using a timeout value beyond which an inactive connection should be closed [5]. A connection kept open until the next HTTP request reduces latency and TCP connection.

An open TCP connection with an idle-state client that requests no data consumes a servers resource, a socket and buffer space memory. The minimum size for a socket buffer must exceed the size of the largest TCP packet and many implementations pre-allocate buffers when establishing connections establishment overhead. The number of available sockets is also limited. Many BSD-based operational systems have small default or maximum values for the number of simultaneously-open connections (a typical value of 256) but newer systems are shipped with higher maximum values.

Researches indicate that with current implementations, large numbers of (even idle) connections can have a detrimental impact on servers throughput [6].

The issues of connection management is to strike a good balance between benefit and cost of maintaining open connections and to enforce some quality of service and fairness issues.

2.2 Policies of Connection Management

The current version 1.3 of the Apache HTTP Server [7] uses a fixed holding-time for all connections (the default is set to 15 seconds), and a limit on the maximum allowed number of requests per connection (at most 100). The Apache implementation is a quick answer to the emerging need for connection management. The wide applicability and potential benefit of good connection-management makes it deserving further study.

Persistent connection management is performed at the HTTP-application layer. Current implementations of Web servers use a holding-time model rather than a typical caching model.

Using holding-times, a server sets a holding time for connection when it is established or when a request arrives. While the holding-time lasts, the connection is available for transporting and servicing incoming HTTP requests. The server resets the holding-time when a new request arrives and closes the connections when the holding-time expires.

In a caching model there is a fixed limit on the number of simultaneously-open connections. Connections remains open cached until terminated by client or evicted to accommodate a new connection request.

A holding-time policy is more efficient to deploy due to architectural constraints whereas a cache-replacement policy more naturally adapts to varying server load.

Policies in the two models are closely related when server load is predictable [1]; a holding-time policy assigning the same value to all current connections is analogous to the cache-replacement policy LRU (evict the connection that was Least Recently Used). In fact, under reasonable assumptions the holding-time value can be adjusted through time as to emulate LRU under a fixed cache size (and hence adapt to varying server load) [1]. Heuristics to adjust the holding-time parameter were recently proposed and evaluated on server logs [2].

Heuristics to adjust the holding-time parameter on server-side were recently proposed and evaluated on server logs [1,2]. Demanding additional processing for searching heuristic parameters for every connection, if it was used on popular busy servers, it may have an inferior effect on server performance, so consequently deteriorate overall latencies.

A problem in the effectiveness of connection-management policies in both models is the ability to distinguish connections that are more likely to be active sooner. LRU exploits the strong presence of reference locality but does not use further information to distinguish between connections.

3. Connection Management on Client-side

3.1 Prototype of Proposal Mechanism

We define finishing time of transmission for HTML page and all embedded file in it as connection-closing time. For this definition to be implemented, we present a mechanism, which both client and server are able to close the TCP connection.

We present simple algorithm for implementing prototype for our proposal. Used methods are limited to *GET* message for file request and *CLOSE* message for closing connection.

Client starts to establish connection with a

corresponding server by users ask, requesting *GET* message for first HTML file to the server. After receiving HTML document file, client parses tag attribute information (e.g. img, applet, embed) in it about embedded objects (e.g. image files, java applet class files, sound files) and request corresponding object files to the server through *GET* message. When the last embedded file arrives, client sends *CLOSE* message to the server for closing current connection and terminates connection. Server also closes connection when server finishes the connection.

Server establishes socket and watches incoming connection request. After Received connection request, server establishes connection with the client and sends repeatedly the file corresponding requested file name. Server send *CLOSE* message to current client for closing connection by the fixed holding-time model that maintains connection for a certain time and close the connection with the client. After then, server releases resources, socket and socket buffer memory having been assigned to the client. Therefore, the next clients requesting connections to the server are able to receive faster and more fairness service.

4. Experimental Results

To validate the proposal we implemented prototype clients and servers. The clients and the servers are implemented in the JAVA programming language.

- Two different kinds of prototype clients:
  - One for our client-side heuristic connection management (CHCM) policy
  - The other for holding-time policy and http/1.0
- Two kinds of prototype servers:
  - One for http/1.0
  - The other for holding-time policy and CHCM policy

Each client was run on 300Mhz Pentium II PC with 32MB of physical memory running the windows95. Each server was run on two different environments: one is 600Mhz Pentium III with 32MB to make easy Web server environment, and the others 386 PC with 8M bytes memory with to make relatively very busy Web server, both of them running Linux 2.0.32 with the kernel compiled by increasing the socket listen queue to 256 and increasing the MAXUSERs kernel parameter to 256. All of the servers are implemented by thread-based and event-driven feature and collect CPU and physical memory statistics.

For the busy Web server environment, we assume a process-per-request model, with pools of processes. Our mechanism of the servers limits resource usage of processes by limiting concurrency. This is achieved by imposing an upper bound on the number of processes in the pool [19]. If all processes are busy, additional incoming transactions of new clients are delayed (in the OS) until a process becomes available. We measured network retrieval times, not including the time it took to render images on the display.

In our experiments, we measured the time required to load a document and all of its embedded images on the clients. We created documents with different numbers of embedded images, and with images of 45K bytes sizes. We did these measurements for both easy server environment and busy case accessed via a 1.544Mbit/sec T1 link. Also, we measured the HTTP throughput and server CPU utilization of the servers.

In the busy server case, we measured CPU Utilization percentages and latencies according as clients requesting access to servers increase when the size of processes pool was limited to 20 and the number of embedded images in HTML documents was 10.

Figure 1 shows the CPU Utilizations of holding-time policy, CHCM policy and http/1.0. CPU Utilization of CHCM became lower than that of holding-time from 20 that the number of clients is the same size of processes pool.

Figure 2 shows the load time of holding-time policy, CHCM policy and http/1.0. Latency of holding-time is slightly lower than CHCM up to 20 clients, but not noticeable. From 20 clients, Latency of CHCM become lower than holding-time.

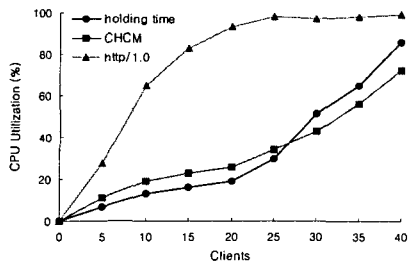

Figure 1. CPU Utilizations for Busy Web server environment according as clients

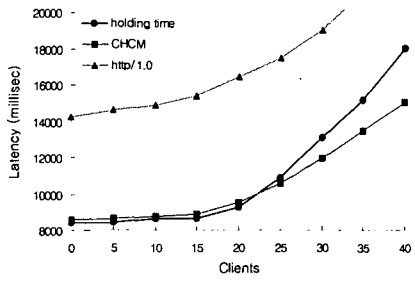

Figure 2. Latencies for Busy Web server environment according as clients

CHCM policy provides significant improvements for busy server transactions. In this case, CHCM policy reduces latency by about 5% to 20% and CPU Utilization by about 5% to 15%; this implies that though, in holding-time policy, if accesses from clients to a server increase explosively and then concurrent connections of the server with clients are occupied, the access requests of new clients to the server should wait until the holding-time of any idle connections occupied becomes expired and therefore the part of resources such as memory and socket buffers available. In CHCM policy, however, they can receive service without waiting for idle connections with connected clients being expired. And advantage of CHCM is no additional overloading that is imposed to server and can degrade performance of server if the connection same mechanism runs on server-side, because it runs on client-side.

5. Conclusions and Future Works

We proposed the mechanism of a heuristic connection management supported by the client-side in addition to fixed holding-time model on server-side, under persistent HTTP. For the mechanism of heuristics, we defined finishing time of transmission for HTML page and all embedded file in it as connection-closing time on client-side. The client exploits the tag information in transferred HTML page so that decides connection-closing time. As the Processing for parsing of tag information in HTML file occurs on client-side, the mechanism allows server to use server's resource more efficiently without decreasing performance of server-side and give services to clients more fairly. Therefore our mechanism supports a good balance between benefit and cost of maintaining open connections and to enforce some quality of service and fairness issues. As future works, we will add pipelining, part of http/1.1 features, to our CHCM mechanism for better implementation results.

References

1. E. Cohen and H. Kaplan: Exploiting regularities in Web traffic patterns for cache replacement, in: Proc. 31st Annu. ACM Symp. On Theory of Computing, ACM, 1999.
2. M. Elaud, C.J. Sreenan, P. Ramanathan and P. Agrawal: Use of server load to dynamically select connection-closing time for HTTP/1.1 servers, Submitted for publication, March 1999.
3. J.C. Mogul: The case for persistent-connection HTTP, Comp. Commun. Rev. 25 (4) (1995) 299-313.
4. V.N. Padmanabhan and I.C. Mogul: improving HTTP latency, Comput. Networks ISDN Syst. 28(1/2) (1995) 25-35.
5. T. Berners-Lee, R. Fieding, J. Gettys, J.C. Mogul, H. Frystyk, L. Masinter, and P. Leach: Hypertext Transfer Protocol HTTP/1.1 RFC2616 Jun 1999.
6. L.A. Belady: A study of replacement s for virtual storage computers, IBM Systems Journal 5 (1996) 78-101.
7. Apache HTTP server project.