2001년 한국정보처리학회 추계 학술발표논문집 제8권 제2호

하이퍼미디어 데이터를 위한 차별화 된 서비스 연구

이윤정, 김태윤

고려대학교, 컴퓨터학과

Differentiated Service for Hypermedia data on the Web

Yoon-Jung Rhee, and Tai-Yun Kim

Dept. of Computer Science & Engineering, Korea University
{genuine, tykim}@netlab.korea.ac.kr

Abstract.
Most implementations of HTTP servers do not distinguish among requests for hypermedia data from different clients. Commercialization of Web site is becoming increasingly common. Therefore providing quality of service with members paying to the site is often an important issue for the hosts. For some uses, such as web prefetching or multiple priority schemes, different levels of service are desirable. We propose server-side TCP connection management mechanisms to provide two different levels of Web service, high and regular levels by setting different timeout for inactive connection. Therefore this mechanism can effectively provide different service classes even in the absence of operating system and network support.

1 Introduction

The explosive growth of the WWW and the ever-increasing resource demands on the servers [3,8,9]. It is impossible for Web server with limited resources to process all of requests from clients by high quality of service when requests increase rapidly. And Commercialization of Web site is becoming increasingly common. Naturally, the membership users with paying fee to the site expect higher quality of service than non-members.

However, most Web servers today do not provide support for differentiated quality of service. To do so would require the incoming requests be classified into different categories and different levels of service be applied to each category. Apache [7], one of the most used Web servers, handles incoming requests in a first-come first-served manner. All the requests correctly received are eventually handled, regardless of the type of requests and the load on the system. Thus, it is an interesting and important issue that the Web servers provide different levels of quality of service to members and non-members [3].

HTTP/1.1 standard [5] 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 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 [1,2]. HTTP/1.1 must decide when to terminate inactive persistent connections. 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 network's problem but also server's overloads having limited resource. A connection kept open until the next HTTP request reduces latency and TCP connection.

To provide differentiated service would require that the incoming requests be classified into different classes and different levels of service be applied to each class. In this paper, we propose TCP connection management mechanisms to provide different levels of quality of service.

2 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 [1,2,5,6]. A connection kept open until the next HTTP request reduces latency and TCP connection. In this chapter, we introduce issues of persistent connection for connection management.

An open TCP connection with an idle-state client that requests no data consumes a server's resource, a socket and buffer space memory. 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) Researches indicate that with current implementations, large numbers of (even idle) connections can have a detrimental impact on server's throughput [1,2].

In a holding-time model, while the time lasts, the connection is available for transporting and serving incoming HTTP requests. The server resets the holding-time when a new request arrives and closes the connections when the holding-time expires [1,2]. The current version 1.3 of the Apache HTTP Server 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). Using holding-times, a server sets a holding time for connection when it is established or when a request arrives.

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 [5,6].

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. 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 [1,2].

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.

3 Connection Management for Differentiated Service

Providing differentiated services would require that the incoming requests be classified into different classes and different levels of service be applied to each class. In this chapter, we present our differentiated service mechanism, which manage TCP connection by means of offer different levels.

3.1 Differentiated Service Model

Our mechanism uses different Holding-time changed from the fixed Holding-time model to provide differentiated service for each class clients.

We classify clients (users) into upper class for membership users and default class for non-membership users, and provide default holding time with default class and additional time to default holding time with upper class. The upper class users can reduce processing overheads and network latencies caused by re-establishing TCP connection when holding time expires, comparing with default class users.

3.2 Prototype System

In this section, we present operational scenarios of proposed prototype system. Then, we shows its' operational time line. Client starts to establish connection with pertinent server by user's ask, requests HTML file. Server establish socket and watches incoming connection request. After Received connection request, server establishes TCP connection with the client. Then, it analyses user's class, assigns proper holding-time to the client, response to requested file, and starts the holding-time. Server must keep TCP connection during the holding-time. If it receives new requests from the client restarts the holding-time. But, if no request after the holding-time, the server closes TCP connection with the client and releases resources, socket and socket buffer memory having been assigned to the client. Figure 1 shows operational time line of our prototype mechanism.

3.3 Class Broker

In this section, we suggest Class Broker that determines proper holding-time to each client during establishing TCP connection.

The Class Broker can be implemented inside the Web server or on membership management DB server. The Followings are roles of Class Broker.

- Manages class table by membership
- Determines holding time of new client when the client requests access to the Web server

3.3 Server

In this section, we describe Web server of our proposal mechanism. We present simple algorithm for implementing prototype for our proposal. Figure 2 is proposed Web server prototype Algorithm.

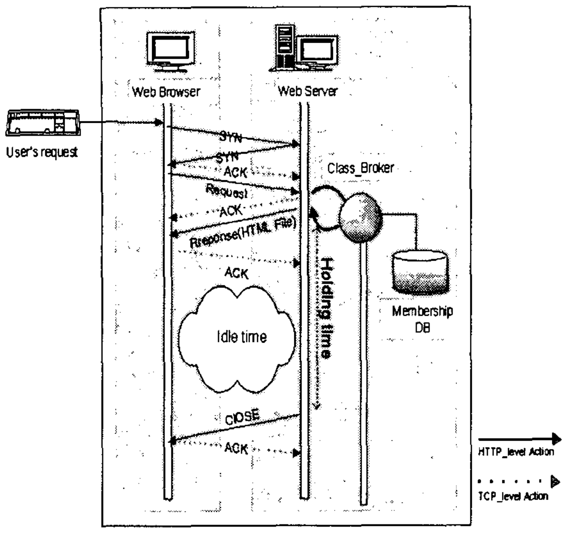

Fig. 1. Time line of proposed mechanism

```
Server()
{
  establish server.socket
  while()
  {
    if accept "SYN" from client
    {
      open client.socket
      Holding_Time = Class_Broker(client)
      time = Holding_Time
      while (time)
      {
        read stream
        if method in stream is "GET"
        {
          get filename from stream
          if file exist then response requested file
          else response "file not found"
        }
        else if method in stream is "CLOSE"
                  then close client socket and break
        time = Holding_Time
      }
      close client socket
    }
  }
}

Class_Broker(client)
{
  if client is in UpperClass
      then assign (Default_Time + 'α ') to client
  else assign Default_Time to client
}
```

Fig. 2. Web Server prototype Algorithm

Used methods are limited to GET message for file request and CLOSE message for closing connection.

The followings are main operation of proposed Web server.

- Calls the Class Broker when access request from new clients
- Sets holding time returned by Class Broker to the client.
- Terminates TCP connection with the client when holding time expires.

We expect that this mechanism supports balanced service reducing latencies and overhead by supporting persistent connections and server's overloading by connection management. The prototype of this mechanism is presently under development with java based. Current implementation focuses on compare three models, fixed holding time model, upper class and default class of our mechanism.

4 conclusions and future work

We propose server-side TCP connection management mechanisms to provide two different levels of Web service, high and regular levels by setting different timeout for inactive connection. It is impossible for Web server with limited resources to process all of requests from clients by high quality of service when requests increase rapidly. It is an important issue that the Web servers provide different levels of quality of service to members and non-members. We present the mechanism that sets different holding time to each client (user) by classes and provide fast access of hypermedia. This mechanism can effectively provide different service classes even in the absence of operating system and network support. As future works, we will analyze and evaluate performance of the mechanism.

Reference

1. Y-J. Rhee, N-S. Park, T-Y. Kim: Heuristic Connection Management for Improving Server-side Performance On the Web, in Proc Workshop on OHS6 (Texas, May 30, 2000), LNCS-1903
2. E. Cohen, H. Kaplan, J. Oldham. Managing TCP Connections under persistent HTTP. 1999 Elservier Science
3. J. Almedia, M. Dabu, A. Manikntty, P. Cao: Providing differentiated levels of service in web content hosting, in Proc. 1998 Workshop on Internet Server Performance Madison (Wisconsin, June 23, 1998)
4. L. Eggert, J. Heidemann: Application-Level Differentiated Services for Web Servers, in World Wide Web Journal, vol 3, 1999

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. http://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf
6. 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.
7. D. Robinnson, Apache Group: APACHE – An HTTP Server, Reference Manual, 1995. http://www.apache.org
8. J. Mogul: Network Behavior of a Busy Web Server and its Clients, in Research Report 95/5, DEC Western Research Laboratory, October 1995
9. J. Mogul: Operating System Support for Busy Internet Servers, in Proc. Of the Fith Workshop on Hot Topics in Operating System, May 1995