Implementing Distributed Server Groups for the World Wide Web

Michael Garland, Sebastian Grassia, Robert Monroe, Siddhartha Puri

25 January 1995
CMU-CS-95-114

Carnegie Mellon

Implementing Distributed Server Groups for the World Wide Web

Michael Garland, Sebastian Grassia, Robert Monroe, Siddhartha Puri

25 January 1995
CMU-CS-95-114

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890

Abstract

The World Wide Web (WWW) has recently become a very popular facility for the dissemination of information. As a result of this popularity, it is experiencing rapidly increasing traffic load. Single machine servers cannot keep pace with the ever greater load being placed upon them. To alleviate this problem, we have implemented a distributed Web server group. The server group can effectively balance request load amongst its members (within about 10% of optimal), and client response time is no worse than in the single server case. Client response time was not improved because the measured client traffic consumed all available network throughput. The distributed operation of the server groups is completely transparent to standard Web clients.

This research is sponsored in part by the Wright Laboratory, Aeronautical SystemsCenter, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant F33615-93-1-1330. The US Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation thereon. Support also came from the Air Force Materiel Command under contract number F19628-93-C-0171. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Wright Laboratory or the United States Government.

Keywords: World Wide Web, distributed computing, hypertext transfer protocol (HTTP), load balancing, server groups, distributed information systems.

1. Introduction

As the exponential growth of the World Wide Web continues with no near-term end in sight, the load placed on popular Web servers is often too great for a single server machine to handle. When the volume of requests to a server becomes prohibitively high the server tends to drop requests and eventually crashes. The service provider needs more computing power to meet the demands of the clients requesting the service. One solution is to simply buy a bigger machine that can handle the large load of service requests. This solution is, however, unacceptable; it provides only temporary relief from server overload. If server usage continues to grow rapidly, even the more powerful server will eventually be overwhelmed. This problem is likely to become more significant over the next few years as the number of commercial service providers and the size of the data objects being exchanged are likely to increase dramatically.

A fairly obvious solution to this problem is to implement a distributed Web server which will spread incoming request load among several machines. There is, however, a significant problem with this approach. The HyperText Transfer Protocol (HTTP/1.0) used by Web clients and servers identifies a resource on the Web using a valid hostname. Consequently, a distributed server needs to present a single hostname to maintain compatibility with existing Web clients. We can alleviate this problem by extending the concept of distributed process groups to the Web server level. A distributed server group exports a single logical name and address to the outside world. From the viewpoint of a client sending an HTTP request, the server group is a single server that will handle the HTTP request in the same way that any other Web server would. Once the server group receives a request it transparently allocates it to an appropriate member of the group for processing.

By supporting distributed Web server groups, we can provide a mechanism for smoothly scaling the server capacity of Web service providers. The use of server groups can reduce the latency of servicing a request provided there is sufficient available network bandwidth to accommodate higher request throughput. Our Web server group implementation can effectively balance request load, and is transparent to Web clients which support the HTTP/1.0 protocol.

2. The World Wide Web

2.1. The HTTP Protocol

The fundamental mechanism underlying the Web is the HTTP protocol. The protocol is stateless, object-oriented, and textual. A complete description of the current protocol can be found on-line at [CERN]. For our purposes, we are only interested in the basic structure of the protocol.

HTTP is an RPC-like protocol; clients make requests of servers and receive responses. The general format of a request is:

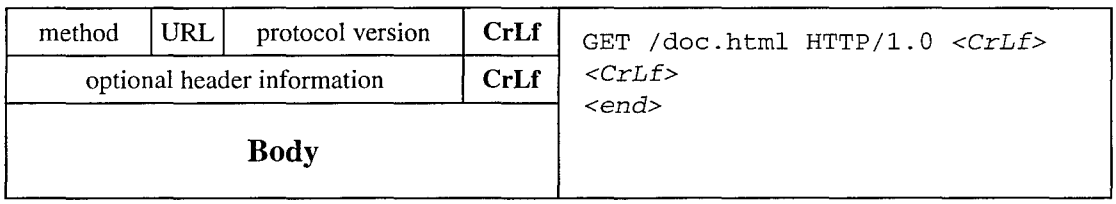

Figure 1: HTTP request format.

```
GET /doc.html HTTP/1.0 <CrLf>
<CrLf>
<end>
```

Figure 2: Example HTTP request.

This figure needs some explanation. A *URL* is simply a name for an object; typically, the hierarchical naming scheme of the Unix file system is used. The *method* given in the request is a method to be invoked on the object named by the given URL. The **CrLf** tag represent the pair of characters carriage return and line feed. The *optional header information* is a series of MIME headers which can be used to specify a variety of things including client name, user name,

encoding type, date, authorization information, etc. The *body* of the request is interpreted by the method which is invoked; it is in essence the argument list for the method.

Once the server receives a request from a client, it processes the request and returns a response. A reply from the server has the following structure:

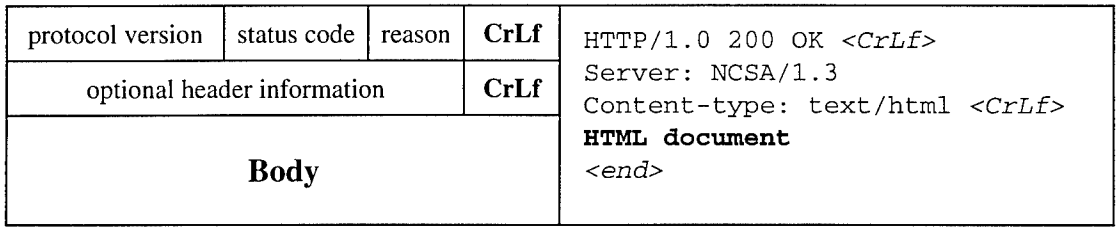

Figure 3: HTTP response format.

```
HTTP/1.0 200 OK <CrLf>
Server: NCSA/1.3
Content-type: text/html <CrLf>
HTML document
<end>
```

Figure 4: Example HTTP response.

The *status code* indicates the result of the request; the *reason* field provides a human-readable textual explanation of the status code. The rest of the fields of the response fulfill similar roles to their counterparts in the request.

In current patterns of Web use, the `GET` method is by far the most frequently used. In fact, Web sessions typically consist exclusively of `GET`s. This reflects the fact that the Web is currently used primarily as a means of fetching files, albeit in many formats, including hypertext.

2.2. Web Servers and Clients

Servers and clients use the HTTP protocol described above to exchange information. The general model of the Web currently in use is this: servers export a hierarchical file space to clients. This file space is populated by "documents" of various types. The primary function of Web clients is to acquire documents from this file space in response to user actions. The client and server must also negotiate a document format which is acceptable to them both. Web servers and clients communicate with each other over TCP/IP streams. The general model of an HTTP transaction is:

- The client opens a TCP/IP connection to the server,
- The client sends its request over this stream,
- The server sends its response back,
- And the connection is closed.

In the case of the most widely used Web server (NCSA's `httpd`), the server forks a new process for each connection. That process receives the request from the client, processes it, sends back a response, and exits.

3. A Distributed Web Server Group

Since it's inception, the Web has experienced very rapid growth in usage and amount of data being provided. The growth of the user community has generated increasing demand for data, and the raw amount of data has also increased due to the proliferation of large multimedia resources. The result has been continually increasing stress on Web servers.

A natural solution is to group several server machines into a single server group. The server group would provide a single logical service but would physically share incoming requests amongst its member machines. The need for this is indeed real; NCSA has implemented a similar scheme for their server after a single server machine was no longer able to cope with the heavy load [KBM94].

In order for a Web server group to be useful it must operate transparently within the existing Web. In other words, it

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.