A Comparison of Load Balancing Techniques for Scalable Web Servers

Haakon Bryhni, University of Oslo
Espen Klovning and Øivind Kure, Telenor Research and Development

Abstract

Scalable Web servers can be built using a network of workstations where server capacity can be extended by adding new workstations as the workload increases. The topic of our article is a comparison of different methods to do load-balancing of HTTP traffic for scalable Web servers. We present a classification framework for the different load-balancing methods and compare their performance. In addition, we evaluate in detail one class of methods using a prototype implementation with instruction-level analysis of processing overhead. The comparison is based on a trace-driven simulation of traces from a large ISP (Internet Service Provider) in Norway. The simulation model is used to analyze different load-balancing schemes based on redirection of request in the network and redirection in the mapping between a canonical name (CNAME) and IP address. The latter is vulnerable to spatial and temporal locality, although for the set of traces used, the impact of locality is limited. The best performance is obtained with redirection in the network.

The aim of this article is to compare and evaluate different load-balancing algorithms for scalable Web servers. Our main contributions are a framework to compare the different solutions, a trace-driven simulation study to compare their performance, and a performance evaluation of a prototype implementation.

As the Hypertext Transfer Protocol (HTTP) load increases, there will be a point where either a Web server must be upgraded or the load divided between several servers. By connecting many smaller machines, the price/performance lead of desktop and workstation class machines can be used to build scalable servers at moderate cost. A clustered server may also provide better reliability since appropriate load sharing algorithms can facilitate fault resilience with graceful degradation of performance as machines leave the cluster due to failure or preventive maintenance. A clustered server also makes it possible to add new machines without interrupting service.

The HTTP protocol is stateless, such that a TCP connection must be set up for every object on a Web page. Thus, each request can be routed independently. As shown in Fig. 1, this HTTP property can be used to achieve load sharing in a cluster by directing requests aimed at one logical server to different physical servers with identical content.

Since Web objects are seldom referenced [1], replication is a valid strategy only for the most popular objects. Another alternative is load sharing through file or object placement on different machines. In this case, load sharing is done through the naming structure, and placement changes cannot be made without changing all links pointing to the object. Here we have focused on servers with replicated content. However, content replication and sharing through object placement are orthogonal, and can be combined to achieve further distribution of load.

Most Web links (universal resource locators, URLs) include a canonical name (CNAME) instead of the 4-byte IP address. Through a naming service called Domain Name System (DNS) [2], the CNAME is mapped to the corresponding IP address. With replicated HTTP servers, load sharing requires the ability to map one logical address onto several different physical servers. This mapping can be done at three logical places: at the client, among the servers, or by the network.

We discuss the framework architecture. The performance of a prototype implementation is analyzed. We discuss alternative load-balancing algorithms, and we show simulation results comparing these methods when used to implement a large scalable Web server. We then conclude the article.

The Architecture Framework

In retrieving an object from the Web a CNAME must be mapped to an IP address, a MAC address, and an object locator. Each remapping offers an opportunity to redirect the request to the most appropriate machine from a load balancing viewpoint. Therefore, load balancing architectures are best classified according to where redirection is used; that is, at the client, at the server, or in the network.

Remapping in the Client

The remapping of addresses at the client side can be classified into two groups: those transparent to the client and those that are not transparent. The latter group requires changes to client software. Netscape’s extension to the browser client for load balancing of access to Netscape’s own server is one example [3]. However, the smart client approach increases network traffic by frequent polling, and is vulnerable to delays between polling and sending the request.

The DNS system provides a distributed database for mapping between CNAME and IP addresses. Each naming domain maintains its own local address and hostname information. A domain’s name servers are queried by resolvers in any end system for the mapping between CNAMEs within the domain and IP address. These name servers can transparently return the IP address of the available Web servers in the list in a round robin manner. However, due to the caching strategies with a configurable time to live (TTL) used throughout the Internet in the DNS, performance is influenced by the spatial and temporal distribution of access. Requests from end systems in the same domain will be directed to the same destination since the remote name server will cache the CNAME-to-IP-address mapping. This mapping is reported by the rotating name server at the first request, and is cached by the name server in the client’s local domain. The result may be a skewed load on a server using the rotating name server method if many clients use the same name server.

Remapping in the Server

This method uses a modified Web server as a proxy server for the replicated servers. The proxy will not process the HTTP request, but passes it on to the most suitable machine among the replicated servers. The SWEB project at UCSB [4] is based on HTTP redirection at the server in conjunction with detailed server load estimates. The drawback of this type of solution is that remapping is done at the application level, and the request must traverse the full protocol stack four times before the request can be processed.

Thus, this solution is only interesting when the time to process the request is the limiting factor (e.g., slow disk I/O or compute-intensive queries). In most cases, the proxy server itself will be a bottleneck, and the solution does not scale. As a result, remapping at the server side will not be discussed further.

Remapping in the Network

In this section we describe different architectures based on remapping network elements. In addition, we discuss the effect the HTTP/1.1 protocol may have. These changes could potentially have an effect on network elements that maintain the state of HTTP and IP streams.

Remapping at the network side can be done at two levels: at the network layer itself or between the network and link layers. In this work we call the network element performing the remapping an HTTP scheduler since we primarily focus on solutions for scalable Web servers. Apart from this special mapping, operations in the HTTP scheduler are similar to the normal routing and forwarding operations, as shown in Fig. 2. Normally, remapping will be done close to the servers so that network capacity between the HTTP scheduler and the real servers is high, and server load information is accurate.

In the first alternative, each replicated server has a unique IP address. All IP packets destined for a logical server are inspected, and the destination address is replaced with the address of the replicated server with the lowest load. All users use the same URL to contact the Web server pool (e.g., http://www-cluster.icelcdn.no), mapped by the DNS to the address A.B.C.X. This is the external IP address of the Web server. Each Web server in the pool will have its own IP address in the same subnet. The remapping network element (the HTTP scheduler) must monitor all of the IP traffic and send all incoming IP traffic of a particular HTTP request to the same physical Web server. For the scheme to work transparently, the HTTP scheduler must change the IP address of all IP packets in each direction. In addition, all packets in an

![Figure 1. A scalable Web cluster and clients.](image1)

![Figure 2. Remapping in the network.](image2)
HTTP transaction must be mapped to the same server. Otherwise, a complete multiprocessor with shared state must be implemented. A TCP connection is identified by the tuple \( <\text{TCP}, \text{IP-src_addr src_port}, \text{IP-dst_addr dst_port}> \). A schematic overview of this solution is shown in Figure 3. Handling connections that are terminated incorrectly places specific requirements on the HTTP scheduler. It has to monitor the TCP connection (e.g., the exchange of TCP flags) from setup to teardown. To find out when a connection that is terminated incorrectly can be removed from the connection database, the HTTP scheduler has to mimic the TCP state machine, including the TCP timeout values.

Similar methods have been used with success in other environments such as mobile computing [5]. The IP address replacement requires that access to the Web cluster be done through the HTTP scheduler. Thus, its networking capacity must be significantly higher than each of the Web servers.

In the second alternative, remapping is done between the network and link layers, and all replicated servers have the same IP address as the logical server. Server assignment is done when the IP address is mapped to a link address. The remapping will take different forms depending on network technology. On a legacy LAN or cluster interconnect, each replicated server has a unique link address (MAC address or port number). With an asynchronous transfer mode (ATM) or label-switched system, each server is associated with a set of virtual channels or labels. Regardless, the request is guided to the label/MAC address associated with the least loaded server.

With link-level redirection, the scheduler must hard code the address resolution protocol (ARP) table to avoid issuing ARP requests to the subnet. The existing protocols for dynamic exchange of ARP information cannot be used, since the machines in the cluster have the same IP address. An alternative implementation of the latter architecture is to use layer 2 multicast to all servers in the cluster and use packet filtering to select the packets processed by the individual nodes. This is essentially equivalent to distributing the remapping from a central node to the servers. The advantage is less overhead in the remapping network element at the expense of higher processing cost due to packet filtering and synchronization between the filtering processes.

The advantages of remapping between the network and link layers are that IP packets do not have to be modified. The disadvantage is that all replicated servers must be on the same subnet. The two alternative approaches are identical in terms of necessary state information to be maintained. For each alternative, a mapping between the HTTP session (TCP connection) and server must be maintained. Therefore, the state of each connection must be monitored. In addition, each packet must be inspected during the forwarding operation. In the first alternative, where the IP address is modified, modifying the IP datagram cyclic redundancy check (CRC) checksum field and IP TTL field incur additional processing cost.

Implications for New Versions of the HTTP Protocol

The Internet Engineering Task Force (IETF) has standardized a new protocol (HTTP v1.1) to improve the performance of HTTP transactions. In particular, multiple objects can be transferred per session. The impact of this protocol will depend on the scheduler solution used. The overhead of setting up the necessary information in the router to handle the remapping can be amortized over a larger number of bytes transferred. The actual overhead associated with the forwarding of each datagram will remain unchanged. Memorywise there should be limited impact, since the time integral of the state information should be the same. The load balancing aspect may become more difficult, since each redirection decision will have a larger impact due to varying bandwidth requirements during a longer connection lifetime. In a solution where all the Web servers have the same IP address and the remapping is done at the data link layer, the setup process is the only significant overhead. Thus, the overall overhead of the solution is reduced with the HTTP/1.1 protocol.

Prototype Performance Evaluation

This section evaluates the performance of a prototype implementation of the HTTP scheduler. The prototype was implemented to demonstrate the feasibility of remapping at the IP layer, as well as to provide insight into the processing requirements for the detailed simulation study. Little effort was made to optimize the code path of the remapping algorithm. The prototype was based on the remapping network element scheme with unique server IP addresses, since this solution represents the highest overhead. Thus, the results are conservative estimates for other less costly solutions like remapping at the link layer, where the recalculation of the checksum for the IP header is avoided.

The Method

The measurements were carried out with small probes in the forwarding path of IP. Thus, both the absolute forwarding overhead and the overhead due to HTTP scheduler operation could be found. The actual overhead was measured using both the internal microsecond clock and the Performance Monitoring Counters available for the UltraSparc processor. These counters made it possible to count instructions, cycles, cache misses, branch misses, and so on. The performance evaluation results were obtained using the Solaris 2.5.1 operating system.
Our testbed consisted of Sparc20 clients and servers and a Sun Ultra 1 Creator 3D acting as an HTTP scheduler between two classical IP over ATM networks (i.e., client and server networks), respectively. The NCSA 1.5.1 Web server software was installed in all the Sparc20 Web servers. Each Web server ran in standalone mode with all the Web content residing on local disks.

**HTTP Scheduler Cost**

The HTTP scheduler must parse all packets and decide whether HTTP scheduling is required. If the packet has the HTTP server port number and uses the TCP protocol, the destination address is checked. Unless it is the cluster address (nonexistent IP address), the standard routing path is used. However, if HTTP scheduling is necessary, the scheduler must find a new source or destination IP address for traffic to the client and to the server, respectively (this is the FIND-IP operation in Table 1). Due to the IP address replacement, the TCP checksum must be recalculated (RECALC CRC). Finally, the HTTP scheduler returns to the standard routing code (RETURN).

Table 1 summarizes the cost of the individual scheduler operations as measured in our Solaris/UltraSparc prototype with standard and optimized compile options, respectively.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Instructions: normal/optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIND IP (client→server)</td>
<td>628/180</td>
</tr>
<tr>
<td>RECALC CRC (client→server)</td>
<td>260/83</td>
</tr>
<tr>
<td>RETURN (client→server)</td>
<td>133/-</td>
</tr>
<tr>
<td>Entire routing operation (client→server)</td>
<td>4357/-</td>
</tr>
<tr>
<td>FIND IP (server→client)</td>
<td>650/161</td>
</tr>
<tr>
<td>RECALC CRC (server→client)</td>
<td>250/86</td>
</tr>
<tr>
<td>RETURN (server→client)</td>
<td>123/-</td>
</tr>
<tr>
<td>Entire routing operation (server→client)</td>
<td>4693/-</td>
</tr>
</tbody>
</table>

**Table 1. Scheduling cost.**

**Load Monitoring**

A simple load monitoring scheme with limited accuracy is to use the Internet Control Message Protocol (ICMP) echo request to regularly poll the different servers. An accurate response time of each request can be used as a simple load indication. Regular requests can collide with regular processing activity on the monitored server. Thus, the load between the polling intervals cannot be deduced by this method.

Some of the available load balancing devices use a combination of standard and tailor-made end system monitoring tools to provide more accurate load monitoring. These system tools can provide information to a load monitoring agent in the server. This agent can be polled by the load balancing device or push the information periodically to the device. Load monitoring agents will provide more accurate load information at the cost of imposing requirements on the tools supported by the workstations.

The last method is to monitor the traffic or response time from each server. This must be done in the scheduler. How the load information should be used is more difficult.

**Proposed Scheduling Algorithms**

In the remapping prototype and simulator mentioned in this article, four different load balancing schemes were used. The first is a round-robin approach, denoted RoundRobin, in the following. It distributes requests to the different Web servers in a round-robin manner independent of the load on each Web server. This scheme is very simple to implement but can, of course, overload a Web server if the sequence of requests is nonoptimal.

The second scheme we implemented is denoted Connections. It keeps track of the number of active connections to each server and always directs a new connection to the server with least connections. If two or more servers have the same number of active connections, the load balancer will choose the server with the lowest server identifier (the servers are numbered). This means that servers with low numbers will be preferred to servers with high numbers. A consequence of this policy is that the first server in the order will always be chosen when a new connection arrives in an empty system. If a new connection always arrives in an empty system, the same Web server will be used.

The third scheme we have implemented is the round-trip scheme. It monitors the request/response phase of each connection by monitoring the TCP protocol. For each connection, the elapsed time between forwarding the first byte of the request to the server and the first byte of the response to the client is calculated. The mean elapsed time of all connections during an averaging window is calculated. At the end of the averaging window, the calculated average will be reset. The default averaging window is set to 1 s. If the mean elapsed time of several Web servers is the same in the current averaging window, the Web server with the fewest active connections is chosen. This scheme can be implemented with a limited set of variables for each Web server. Although a sliding window would be preferable, the increase in complexity made us choose the reset approach.

The last scheme is denoted Load Monitor. It uses an averaging window as in the third scheme. However, it keeps track of the amount of transmitted bytes from each Web server since the last averaging reset and uses it to distribute the requests to a designated server. Just like the round-trip scheme, this scheme chooses the server with fewest active connections, in case two or more servers have transmitted the same number of bytes. The drawback of this scheme is the same as that of the round-trip scheme, that is, the discontinuity of the averaging windows and the overhead of registering the information. However, the number of variables for each Web server is small, and can easily be handled by the router.
Performance Evaluation by Simulation

It is shown in Fig. 4 that the server simulation model consists of four components including a Web server model, a Web client model, a DNS model, and a model of the HTTP scheduler responsible for HTTP load balancing. The simulator input is Web server traces. In the simulator, the remapping element and Web servers are connected using a 155 Mb/s ATM LAN. The configurable connection speed to the Web clients is set to 2 Mb/s. This is substantially more than current modem or integrated service digital network (ISDN) speeds, but will be available through asynchronous digital subscriber line (ADSL) or IEEE 802.16 wireless access networks. To calibrate our simulator we did detailed timing studies in our testbed.

The Simulation Method

We used trace-driven simulation since the analyses depend on correlation between the different parameters in the traffic stream. Trace-driven simulation will faithfully reproduce any effects due to a correlation between different input parameters, at the obvious risk that the traces may not be representative. The alternative would be a stochastic model of the HTTP request with appropriate distributions for URLs, time between accesses, sources, and epochs. The WEB-spec benchmark [6] is an example of such models. However, the need to include correlations between the different parameters easily results in a model difficult to specify and validate.

A trace represents a snapshot of the load. It can potentially address the effect correlation between parameters in the traffic characterization can have, and does not require a thorough characterization of the traffic load. However, it has potential shortcomings in the simulation of future servers with substantially higher traffic load. The trace will always reflect the properties of the system it is measured from. The time between requests from the same source is a function of the person sitting at the end system, the network, the state of the network, the state of the server, and the end system. Through replicating the server, the response time for the server will be shortened if the load is kept at the same level. The effect this will have on the access pattern will not be reflected in the results with trace-driven simulation.

An increase in the Web load will occur along different dimensions depending on the number of users accessing the server, the URLs accessed, and the size and complexity of the different pages. Therefore, just how a trace can be extended to represent a higher load level is not straightforward. In [7] we aggregated the load by folding seven days of consecutive Web accesses into a 24-hr trace with good results.

Simulation

The simulation results reported here are based on several 24-hr traces from the proxy server of a large ISP in Norway. In addition, the simulation was run with traces from a Web server at the University of Oslo Department of Informatics. However, since the overall conclusions are the same, for brevity we present only the results based on the trace from the proxy. The full set of results can be found in [7].

The trace discussed here (termed ISP.peak), with a log resolution of 1 s, provides a significant load to the clustered server. This can be seen in Fig. 5, which compares the mean response time for the different schemes as a function of the number of servers. To remove the effect of variations in request size, the response time of serving each individual request in an empty system was subtracted. The results with the adjusted response times are shown in Fig. 5.
Figure 5 illustrates that the round-robin scheme outperforms the other schemes for all traces, followed by the connection policy and round-trip schemes. These results are conclusive across different traces from the ISP and other Web servers we tested.

From this comparison it is clear that the round-robin scheduling strategy gives the lowest average response time, followed by the round-trip strategy for the ISP peak trace. The differences between the scheduling schemes are larger for larger server configurations where the effect of load balancing is more visible.

The fact that all logged requests that arrive at the server within 1 s seem to arrive simultaneously will effect the results by artificially batching the requests together. In an experiment reported in [7], we uniformly distributed the requests over the 1-s arrival epoch. The response time was reduced, but the relative difference between the algorithms remained the same.

Figure 6 illustrates the number of bytes transferred (in percent) from each server in the scalable Web server with 2-, 4-, and 8-server configurations. For most of the scheduling algorithms, each server handles roughly the same number of bytes. However, this is not the case for the connection scheme. The reason for this anomaly is the algorithm itself. Whenever a request arrives to an empty system, the same server is chosen each time. Thus, the noticeable differences in Fig. 6 indicate that the requests arrive in an empty system. The following abbreviations are used in Fig. 6: RORO = round-robin, CONN = connection, XMIT = Xmit-byte, RESP = round-trip, DNS-1h = rotating name-server with TTL = 1 hr and DNS-24h = rotating name-server with TTL = 24 hr.

For traces with a resolution of 1 µs, more requests will arrive at an empty system. Thus, the overall response time will be reduced, as shown in Fig. 7. In this trace, denoted ISP peak accurate, requests are distributed within a period of 1 s, thereby reducing the instantaneous load. The simulation results indicate the ISP load is not enough to load the scalable Web server fully. However, the results in Fig. 7 are similar to those for the ISP peak simulations.

**Load Balancing with a Rotating Name Server.**— Our initial hypothesis was that use of the rotating name server method gives a skewed load compared to use of a remapping element. This hypothesis was based on the fact that caching in the DNS system will remember CNAME-to-IP mappings such that requests coming from the same domain will go to the same server. Using our trace-driven simulator, we can observe this skewed load. However, our analysis has shown that the rotating name-server performs surprisingly well. For example, the rotating name server with TTL = 1 hr gives (on average) connection loads (i.e., percentage of connections handled by each server) ranging from 10.7 to 15.8 percent on a cluster with eight servers. If the TTL value is larger, the load is further skewed. A TTL of 24 hr, for example, gives connection loads ranging from 8.1 to 18.5 percent (on average) for the same trace. In general, a smaller TTL value improves the load balancing. However, there is a lower threshold on the TTL value, since DNS traffic increases with lower TTL values. DNS traffic itself represents network overhead since it does not carry user information.

Another general observation is that the more servers are used in the cluster, the more uneven the load balancing becomes. To summarize, a rotating name server performs well for short TTL values and small clusters.

**Load Balancing in a Remapping Network Element**— The simplest algorithm gives the best results; the round-robin strategy ensures an even connection load (1/4 load share on each server) and an acceptable sharing of the number of bytes transferred. In our simulations, the round-robin policy provides good load sharing combined with the lowest average response time.

The second best results are obtained with the round-trip algorithm. It yields a reasonably even connection load (11.4–13.6 percent for the sample trace) and sharing of the number of bytes transferred comparable to the round-robin approach. However, higher response times are observed with this algorithm than with round-robin.

The connections algorithm outperforms the algorithms
that strive to estimate load based on history (round-trip and Xmitbyte). On average, it gives slightly better response times than the round-robin algorithm, but requires more processing. This is explained by the fact that many of the requests in the simulated traces are of the same kind; thus, it is sufficient to count only connection state information (connections algorithm) or just perform round-robin scheduling. This observation can also lead to the conclusion that slow responsiveness in the round-trip and Xmitbyte algorithms leads to scheduling of request batches with good connection distribution but bad distribution with regard to response time. These results are shown independently and repetitively across different traces. This behavior leads to higher average response times (as shown in Fig. 7) since many requests are sent to the same server before the scheduler reacts to the increased load and changes the server. The synthetic trace (Fig. 8) demonstrates this effect graphically, where we can see that these algorithms do not spread traffic as well as round-robin and connections.

Sensitivity Analysis — The duration of the averaging window used in the Xmitbyte algorithm is significant for performance of this particular policy. As the averaging window \( Z \) decreases, the performance of the algorithm increases and, for small values of \( Z \), approaches the connections scheme. A more detailed description of the sensitivity can be found in [7].

Synthetic Trace
To show the dynamic behavior of the algorithms, we set up a synthetic trace where every fourth access was a request for a large document. The large request is included to interfere with normal scheduling to visualize how the algorithms react when servers are busy serving long-duration transactions. The scheduling of requests in a four-node server is shown in Fig. 8. For the round-robin scheme, the same server is chosen every time, leading to a saw-tooth pattern. Servers that handle large requests are, for all policies indicated by a circle in Fig. 8. We can observe that the connections policy reacts faster to variations in load than the round-trip and Xmitbyte policies.

Conclusion
We discuss various methods to obtain load balancing in a scalable Web cluster. In particular, we discuss the rotating name-server method, currently used in large clustered servers, in comparison with alternative load-balancing methods based on remapping requests and responses in the network.

Our prototype implementations illustrate that the remapping concept is viable. Detailed performance measurements also showed that the overhead of a scheduling entity in a special router is small. In addition, the amount of software necessary to set up a scalable Web server is minimal.

Load balancing has the positive side effect of enabling fault tolerance in a clustered server. If one server is taken down for maintenance or service is stopped after a failure, the load-balancing network element can quickly remove this server from the active server list, and the cluster will be highly available. A cluster based on the rotating name-server solution will not be fully operational before all cached DNS entries are expired (governed by the DNS TTL time).

To summarize, use of a rotating name server yields reasonable load balancing performance, but DNS caching for the most common TTL value (1 hr) introduces skewed load on a clustered server with an average of \( \pm 40 \) percent of the total load. This result is valid across different traces and Web servers with different load levels.

A remapping network element can improve the load balancing in a clustered server. This network element can be provided by router vendors who can easily incorporate the described routing algorithm modifications and use one of the proposed load balancing algorithms. We recommend the round-robin load balancing algorithm for simple implementation and good load sharing, with the round-trip algorithm as a second alternative. If the RoundTrip algorithm is improved with better prediction of the current load, this algorithm has the potential to distribute load better than the round-robin approach, especially in transient high-load conditions.

References

Additional Reading

Biographies
HARALD BRINLY (brinly@actech.no) is CEO of Advanced Communication Technology AS (www.actech.no) and graduated as a Ph.D. from the Department of Informatics, University of Oslo in 1998. He was previously employed at Apple Advanced Technology Group, Cupertino, California, Norwegian Defense Research Establishment, and Norwegian Institute of Technology. His research interests are cluster architectures, high-speed networks and mobile communication systems. He works part-time as an associate professor at the University of Oslo.

ESPER KLOVINGE (esperi@actech.no) received an M.Sc. in computer science from the Norwegian Institute of Technology in 1992. This work was done while he was working as a research scientist (1993-1998) at Telenor R&D, Oslo, Norway. He is now CTO of Advanced Communication Technology AS (www.actech.no). His research interests include all aspects of high-speed networking.

OIVIND KURE (oivind.kure@telenor.com) received his Ph.D. from the University of California, Berkeley. He is currently chief of research at Telenor R&D, Oslo, Norway with an additional position at the Department of Informatics, University of Oslo. His research interests are performance analysis, high-performance servers, multicasting, and network architecture.