Multi-Level Load Balancing Methods for Hierarchical Web Server Clusters

Moon Suk Yeon and Byeong Soo Jeong*
Department of Computer Engineering, Kyung Hee University, Korea; zoom97@khu.ac.kr, jeong@khu.ac.kr

Abstract
With the need for scalability on web servers due to various demands of users and an explosive growth of the Internet users, researchers had addressed how to make well clusters of many web servers in past. In this paper, we propose a hierarchical clustering method of web servers to cover limits of single-level web server with high scalability and availability. Underlying structures of the hierarchical web server clustering are load balancers and real servers at the bottom level. The load balancer is attached on the top level basically and it can be extendable to n-levels of structure. The proposed algorithm dealing with dynamic weights based on agents at the bottom level shows faster response time compared to the existing round-robin and least connection algorithms. It is also possible to apply different algorithms for each level, and we evaluate that the proposed algorithm could reduce the workload imbalance among the real servers.

Keywords: Load Balancing, Web Server Clusters, Weight Algorithm

1. Introduction

The basic condition of a web server is to guarantee satisfactory response time and pause less service even in the condition with lots of requests from users. Also it must provide increasing information continuously and the scalability that copes flexibly even in cases of increasing users is one of the important factors also. Even if it is improved to a server with higher performance, the single-level web server is not enough to satisfy the various factors listed above. Therefore the clustering web servers are being suggested as alternatives.

Generally, there is the load balancer that receives the users’ requests and the real server that actually processes the service in the cluster type web server. In case of the load balancer, it is a part that plays an important role by selecting the optimal real server that will process the service actually by receiving the users’ requests and hand over the request. However in case the users’ requests get flooded, there may be a bottleneck phenomenon on the load balancer or it may cause a result of the whole service being shut down in case it stops with an unexpected reason. Also if the users’ requests are not distributed to several servers that form the cluster, it may bring about the degradation of the whole system. Therefore, in the case of a large scale web server the performance of the system may vary according to how the clustering of many servers is done.

Thus in this thesis, the way to increase the scalability by composing multiple load balancers from the top level into a tree form in order to solve the problem of fatal errors and to improve the efficiency in the large scale web server were studied. Algorithm using the time information and the number of packet processing for the minimization of the load unbalance between the load balancers was applied and the dynamically weighted algorithm using the agent for the minimization of the real servers were proposed.

As a result of the experiment, its scalability is more outstanding than the single web server and the efficiency is better against the cost and the reliability is also high. In other words, by constructing the web server with...
clustering, the improvement of the scalability, fusibility, economic feasibility, etc. that are the requirements of a web server were examined.

2. The Measure to the Hierarchical Clustering Web Servers

2.1 The Problems of the Single-level Clustering Web Servers

Clustering web servers have many advantages compared with single web servers such as scalability, availability and economics. However, single-level clustering web server that consists of a load balancer and a real server have some problems.

First, single-level clustering cannot expect appropriate scalability in a large-scale web server system since the number of real servers is limited by the performance of the load balancer. Second, single-level clustering may have workload imbalance among real servers with a high possibility for inconsistent user access requirements and various patterns and sizes of files. This imbalance problem causes in efficient resource usage. Third, shutting down a load balance incurs to halt the whole system service.

2.2 The Plan on the Measure to the Hierarchical Clustering Web Servers

We propose a hierarchical web server clusters that consist of multiple load balancers at top level as shown Figure 1. The hierarchical web server cluster has multiple load balancers and real servers. This scheme is different from a centralized or distributed cluster. Bottom level nodes play a role as a real server which process services. Rest of nodes plays a role as a load balancer which process load balance.

The hierarchical clustering web server that is proposed can solve the problem of one top level server at a large scale web site not being able to manage many real servers due to the problem of scalability. There are some differences according to the routing methods but in case of the NAT method, up to about 25 real servers can be expanded on one top level server and in case of the IP Tunneling it can be expanded up to about 100 servers. However recent large scale web sites operate over 1000 servers for one domain service, so the scalability cannot be guaranteed with a single-level clustering web server. But the hierarchical clustering web server can be mapped out to the n-level, so the scalability can be increased to the maximum.

2.2.1 Load Balancing Plan of the Top Level

Load balancers at top level take user service requests concurrently. Through packet processing algorithm in Figure 2., the load balancers determine whether they service or discard the packet. The load balancers should be registered in

![Figure 1. Hierarchical clustering web server.](image1)

![Figure 2. Packet processing algorithm for load balancer.](image2)
the same multicast group and synchronized with the same
time in order to get service requests concurrently.

If the SYN packet that notifies the start of the service
request comes in, the time information is applied to the
hash function and the server that will process the service
as formula (1) will be decided.

\[ \text{LB.ID} = \text{System time.second} \times (\text{number of top level}
\text{servers} - \text{number of down servers}) \] (1)

If the server that will serve through formula (1) is
decided, the decided server serves the request of the cli-
ent and renews its hash table. The other servers discard all
packets that they have received.

In case of the FIN packet that notifies the end of the
service request, it checks its hash table and any contents
recorded should be deleted. If the packet that has come
in is neither the SYN packet nor the FIN packet, it is a
general packet that has come in after the SYN packet so
it is transferred to the real server that is processing the
service requests.

However since it only uses the time information as
the hash function, it may cause a load unbalance prob-
lem. In other words, the service requests don’t come in at
every hour in a fixed quantity, but they can be crowded
at a certain time. If the service requests get jammed at a
time when the load balancer of one side is in process,
there may be a load unbalance problem between the load
balancer servers.

In order to solve this problem, each servers check the
packets if the SYN packets arrive by using the time infor-
mation and if it is a packet that needed to be processed
by itself it makes a schedule and increases the number of
packet processing. This number of packet processing is
sent to another server of a same level periodically through
the agent program that moves at the background. The
number of packet processing exchanged with one another
has the sum of the waiting time of the remaining service
requests that are not taken care of at the server queue as
the load information.

As Figure 3 the agent program is installed in the load
balancer and the real server and the agent program in the
real server sends its load information periodically to the
agent in the load balancer. The agent installed in the load
balancer renews the load information table based on the
information received from the real server and if there is
a user’s request the distributed processing module applies
the weight algorithm as Figure 4 and selects the optimal
real server from the load information table and hands
over the service request. In Figure 4, the variable real
server is an array of structure of workload information
having meta data such as sequence id and weights.

3. Experimental Results

3.1 Minimization Test of Load Unbalance
between the Top Level and the Load
Balancer

We set our experimental environment with Linux based
load balancers, real servers and client servers. We vary the
number of load balancers at top level from 1 to 5. The
service processing time was reduced but the number of

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td>To determine whether service or discard the packet</td>
</tr>
<tr>
<td>- Next Level Operation Module</td>
<td>To obtain weights using load information of next level servers</td>
</tr>
<tr>
<td>- Current Level Operation Module</td>
<td>To count the number of processes</td>
</tr>
<tr>
<td>Communication</td>
<td>To communicate with the other servers</td>
</tr>
<tr>
<td>Distribution</td>
<td>To select next level sever having minimum workload</td>
</tr>
</tbody>
</table>

Figure 3. Packet processing algorithm for load balancer.
int get_realserver() {
    int i, rs, flag;
    rs = 0;
    FOR( i=0; i < SERVER_CNT; i++ ) {
        IF ( (RealServer[i].sequence <= RealServer[0].sequence)
            && (RealServer[i].weight < RealServer[i-1].weight) )
            { rs = i; flag = 1; break; }
        ELSE IF ( (RealServer[i].sequence <= RealServer[i+1].sequence)
            && (RealServer[i].weight < RealServer[i].weight) )
            { rs = i; flag = 1; break; }
        flag = 0;
    }
    IF ( flag == 0 ) {
        FOR( i=0; i < SERVER_CNT; i++ )
            RealServer[i].sequence = 0;
        rs = 0;
    }
    RealServer[rs].sequence ++;
    return RealServer[rs].send_sockfd;
}

Figure 4. Weight algorithm for real servers.

Figure 5. Load balancing for top level.

process was increased. Moreover, the proposed packet processing algorithm achieved less work load imbalance among load balancers compared with the algorithm using only time information.

3.2 Minimization Test of Load Unbalance between the Botton Level and the Real Server

For the minimization test of load unbalance between the bottom level and the real server, 3 real servers of RS_1, RS_2, RS_3 and 1 load balancer were set. Figure 6. Static data files of 256K, 16K, 1K were requested in scheduled order and the test was run in intervals of 10 seconds. (a) is the result of the round-robin algorithm and (b) is the result of the weight algorithm using the agent.

The request frequency of the files of various sizes such as 1k, 4k, 16k, 64k, 256k, 1M were adjusted causing the heavy traffic and the light traffic and their performance were compared with the existing algorithm in Figure 7. (Agent 0.5 second intervals).

4. Conclusions

The points to consider when building a large scale web server cluster are first stable and quick service and flexible scalability. In this thesis, in order to solve the problems that the existing general single-level web servers had and to increase the scalability and the availability, the hierarchical web server clusters with multiple load balancers in the top level was suggested. Also in order to distribute the users’ requests effectively to the real server that processes the service, the dynamic weight algorithm using the agent was suggested.

As the result, the problem of scalability was verified by increasing the load balancer in the top level from 1 to 5 and for the load unbalance between the load balancers, the load unbalance and the loss of packets that were the problems of IA-LVS could be solved through the algorithm.
using the time information and the management number of packets. Also by scheduling the dynamic weight algorithm using the agent of the bottom level based the actual load information, the load unbalance between the real servers could be minimized.

5. References