

# Remaining Capacity Based Load Balancing Architecture for Heterogeneous Web Server System

**Tsang-Long Pao**

Dept. Computer Science and Engineering  
Tatung University  
Taipei, ROC

**Jian-Bo Chen**

Dept. Computer Science and Engineering  
Tatung University  
Taipei, ROC

***Abstract** – Load balancing system can improve the performance of a popular Web server. Most of the load balancing systems use homogeneous Web servers as their backend servers. In this situation, the dispatching algorithm can obtain load sharing among the Web servers quite easily. However, when the Web server system contains servers with different capacities, we must use factors such as the remaining capacity of these Web servers as the criteria for decision making. In our proposed system, we use the remaining capacity as the parameter of the dispatching algorithm and compare the performance with systems using round robin and least connection algorithms. The hit rate, utilization, and drop rate are compared. We also derived a formula to calculate how many servers are needed when the system is overloaded and what kind of capacity the server is required.*

**Keywords:** load balance, heterogeneous web server cluster, round robin, least connection, remaining capacity.

## 1 Introduction

With the explosive growth of the World Wide Web, the traffic of popular Web sites is far beyond the capacity of a single server. Usually, the distributed or parallel architecture with a virtual single interface is implemented. These sites have a competitive motivation to offer better service to their clients at lower cost. The load balanced Web server architecture [1,2], can provide high performance services for large number of clients. Although load balanced servers consist of one or more servers, they act as a single unit. Load balanced servers offer the advantages of user transparency that allows the clients to work with multiple servers without any specific configuration.

However, most of the load balanced algorithms are focused on the homogeneous Web servers, that is, all of the Web servers are of the same specification. These Web servers have the same processing powers, memory installation, network speed, I/O speed, etc. In the homogeneous Web server system, the load balancer can redirect the client request to the most appropriate server based on many well-known criteria, such as round robin

or least connection. In a server-state based algorithm, each Web server must report its loading information to the load balancer. The load balancer then selects the best server to serve the request. But in the heterogeneous Web server system, the load balancer should redirect the client request to the best Web server not only by considering the loading information but also by the capacity of the server. In other words, we can use the remaining capacity of the Web server to decide which server is the most appropriate.

We compare the remaining capacity load balancing algorithm to the round robin and least connection algorithms in the heterogeneous Web server system. We explore the hit rate and utilization as well as the drop rate for the servers with different capacity. Furthermore, we will also discuss the capacity and the number of servers to add into this load balancing system when it is over loaded [3,4].

This paper is organized as follows. In Section 2, we will address some related researches about load balancing architecture and solution. The proposed model will be discussed in Section 3. We will present the simulations and performance analysis in Section 4. Finally, conclusions are given in Section 5.

## 2 Related Work

In this section, we will discuss some backgrounds and related researches about load balancing architectures and solutions.

### 2.1 Cluster Approach

The cluster server system consists of many independent servers that work together. The workload is evenly dispatched to independent servers by using some proper dispatching algorithms. The advantage is that adding more servers into the cluster is easily. From the client point of view, the servers are acted as a single unit. But the cluster system is hard to setup and maintain. It must use some specific hardware and software to group servers into a single unit.

## 2.2 Server Switch Approach

During the last few years, an active commercial market for server switching products has emerged [5]. Many of these products are Ethernet switches supplemented with built-in processing power to examine the incoming packet, manage service traffic intelligently, and assign client request to server based on request content, client session, and/or server status. Server switches and their request routing policies play a key role in managing content and server resources for scalable Internet services [6,7]. Commercial server switches are available from Netel (Alteon), Cisco (Arrowpoint), Extreme, F5 Networks, and other companies.

## 2.3 DNS-based Approach

In the distributed Web server architectures that use request routing mechanisms on the cluster side, there is no action needed in the client side. Architecture transparency is typically obtained through a single virtual interface to the outside world, at least at the URL level. The authoritative DNS server for the distributed Web server translates the symbolic site name (URL) to the IP address of one of the Web server in the system. This process allows the cluster DNS to implement policies to select the most appropriate server and distribute client requests. The DNS, however, has a limited control on the request reaching the Web cluster [8,9]. Furthermore, there will be many intermediate name servers between the client and the cluster DNS which may cache the logical name to IP address mapping. Moreover, the client will also cache some address resolution. Consequently, it is possible that the server load become unbalance.

## 2.4 Dispatcher-based Approach

To centralize request scheduling and fully control the client-request routing, a dispatcher-based approach is proposed. In this approach, a network component of the Web server system acts as a dispatcher. Request routing among servers is transparent. Unlike DNS-based architectures, which deal with addresses at the URL level, the dispatcher has a single, virtual IP address. Dispatcher-based architectures typically use simple algorithms to select the Web server to handle incoming requests. Simple algorithms help to minimize request processing.

A well-known dispatching algorithm is round robin. With the round robin algorithm, new connections are issued to each server in turn; that is, the first connection is redirected to the first Web server, the next connection is redirected to the second Web server, and so on. When all the Web servers in this system have their share, the redirection process starts over again. Another well-known dispatching algorithm is least connection. With the least connections algorithm, the number of connections currently opens on each Web server is measured in real

time. The server with the fewest current connections is considered to be the best choice for the next client connection request.

# 3 Proposed System Architecture and Algorithm

## 3.1 System Architecture

Consider a load balancing system with  $N$  Web servers and one load balancer as shown in Fig. 1. When a client wants to access a Web page, it will issue a DNS query to find the IP address of the Web server. The DNS server will reply with the IP address of the load balancer instead of the one of the Web servers that provides the service. The client then issues an HTTP request to the load balancer. When the load balancer receives the request, a redirection page will be sent back to the client. The redirection page contains the IP address or the domain name of one of the Web servers which is the best one to serve this request. The client then issues the HTTP request again to the real Web server. The Web server will then serve the client requests and transfer documents directly to the client [10].

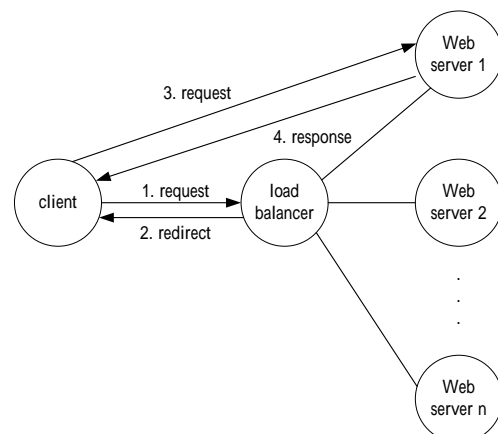


Figure 1. Load balancing system architecture

Each Web server has a status report module similar to the one presented in [8]. This module calculates the current utilization and connections of the Web server and periodically reports this information to the load collection module. The load collection module in the load balancer collects the status information of all the Web servers and passes them to the load balancing module. The load balancing module uses these status information to calculate the remaining capacities of all the servers. The server with maximum remaining capacity is chosen as the best server. When the client request arrived, a dynamic Web page will be replied to the client which redirect the client to the most appropriate Web server [11]. The architecture of the proposed system is shown in Fig. 2.

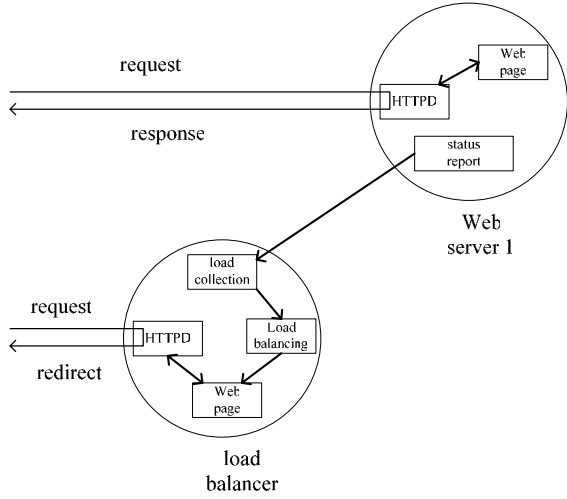


Figure 2. Architecture of the proposed system

### 3.2 Load Balancing Algorithm

In our proposed model, there are  $N$  heterogeneous servers  $\{S_1, S_2, S_3 \dots S_N\}$  in the system. The status report module in each server periodically reports its current load information to the load collection module. The load balancing algorithm calculates the remaining capacities of the  $N$  servers as  $\{R_1, R_2, R_3 \dots R_N\}$ . And then selects the best server  $i$ , where  $\max(R_1, R_2, R_3 \dots R_N) = R_i$ . If a request task  $T$  comes to the load balancer, the address of Web server  $S_i$  with the maximum remaining capacity  $R_i$  will be sent back to the client. We assume that the maximum capacity needed by the request  $T$  is  $T_k$ . If the maximum remaining capacity  $R_i$  is less than the request  $T_k$ , it means that all the Web servers do not have enough remaining capacity to serve the request task  $T_k$ . The load balancer drops this request task  $T_k$  and return a server busy page to the client. When the Web servers finish their previously assigned requests, they will then have capacity to serve future requests.

Assume that the CPU clock of the servers are  $\{C_1, C_2, C_3 \dots C_N\}$ . The status report module of server  $i$  reports the current load information parameters as  $\{c_i, m_i, n_i, t_i\}$ , where  $c_i$  is CPU idle percentage,  $m_i$  is available memory,  $n_i$  is current connection number, and  $t_i$  is network transmission. In the UNIX-based Web server, we can get these four parameters by using the commands shown in Fig. 3. We define the four membership functions as in Fig. 4. Then the remaining capacity of server  $S_i$  is obtained by Eq. (1).

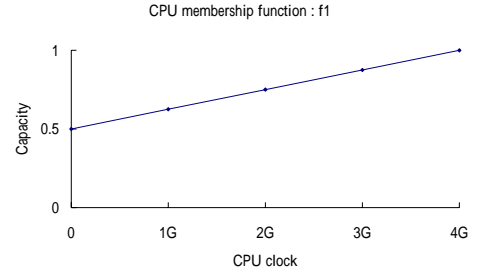
$$R_i = \alpha * f_1(C_i) * c_i + \beta * f_2(m_i) + \gamma * f_3(n_i) + \delta * f_4(t_i) \quad (1)$$

where  $\alpha, \beta, \gamma, \delta$  are weights and  $\alpha + \beta + \gamma + \delta = 1$ . Practically, we find that the available memory and current connection have the most significant impact to server loading. In the simulation presented in Section 4, we use

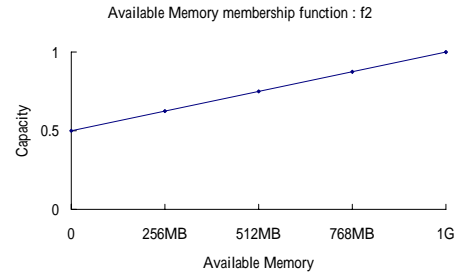
$\alpha=0.1, \beta=0.4, \gamma=0.4, \delta=0.1$  as weights.

```
$idle=$(vmstat | sed -n 3p | cut -b 74-76);
$mem=$(vmstat | sed -n 3p | cut -b 13-19);
$conn=$(netstat -tn | wc -l | cut -b 1-5);
$trans=$(netstat -i | sed -n 3p | cut -b 45-54);
```

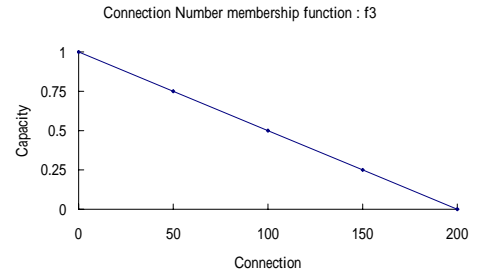
Figure 3. Server load parameters extraction



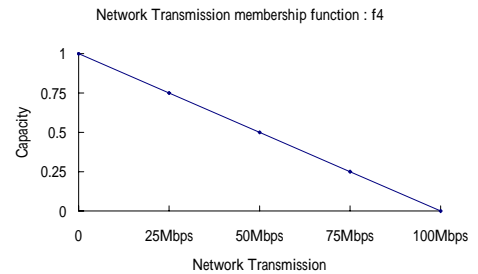
(a) CPU



(b) Available Memory



(c) Connection



(d) Network Transmission

Figure 4. Membership functions of the four parameters

The pseudo codes for the proposed algorithm are as follows:

- Report phase (done periodically):
  1. calculate the four parameters  $\{c_i, m_i, n_i, t_i\}$ ;
  2. report  $(c_i, m_i, n_i, t_i)$  to the load balancer;
- Decision phase (at each client request)
  1.  $R_i = *f_1(C_i) + *f_2(m_i) + *f_3(n_i) + *f_4(t_i)$ , for each Web server;
  2.  $R_i = \max(R_1, R_2, R_3 \dots R_n)$ ;
  3. if  $(\text{Capacity}(R_i)) > \text{Require}(T_k)$
  4. send  $(T_k, S_i)$ ;
  5. else
  6. drop  $(T_k)$ ;

## 4 Simulations and Performance Analysis

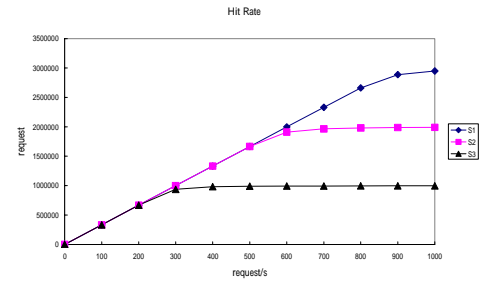
To evaluate the performance of the proposed approach, we performed simulation using round robin (RR) and least connection (LC) load balancing algorithm as well as our proposed remaining capacity (RC) algorithm. The performance evaluation includes connection hit rate, server utilization, drop rate, and system scalability.

### 4.1 Connection Hit Rate and Server Utilization

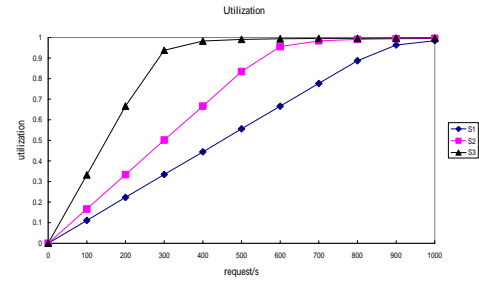
In our testing environment, we use three Web servers, denoted as  $S_1, S_2, S_3$ . The capacities of these three Web server are  $S_1 > S_2 > S_3$ . We generate the client requests from 100 requests per second to 1000 requests per second. Figure 5 shows the results for the RR algorithm. Figure 5(a) shows the hit rate for the three heterogeneous Web servers. According to our environment, the  $S_3$  has lowest capacity. When the arrival rate of client requests is about 300 requests per second, the  $S_3$  server does not have enough remaining capacity. If the arrival rate increases, the  $S_3$  server can only serve the same amount of client requests and begin to drop the requests. If we continuously increase the arrival rate to about 600 requests per second, the second server  $S_2$  will use up all its capacity. If the arrival rate goes up to about 900 requests per second, the first server  $S_1$  also runs out of its capacity. The utilizations of the three servers are shown in Fig. 5(b). At 300 requests per second, the utilization of  $S_3$  rises to about 95%. If the arrival rate increases, the server  $S_3$  will begin to drop requests.

Figure 6 shows the results of the simulation of the LC algorithm. In the LC algorithm, the load balancer will select the server with the least connection. Since we use heterogeneous servers; the server with least connection may not be able to serve extra request because it does not

have enough remaining capacity. In this situation, if a server with least connection but with 95% loading, the load balancer will not redirect client request to this server. Instead, it redirects the request to the next least loaded server. The hit rate of LC algorithm is similar to the RR algorithm. But in this case, the  $S_2$  will reach 95% loading at 500 requests per second and  $S_1$  at 700 requests per second. It seems that the RR algorithm is better than the LC algorithm. But when we consider the drop rate, we can find that the RR algorithm drops more requests than LC algorithm.



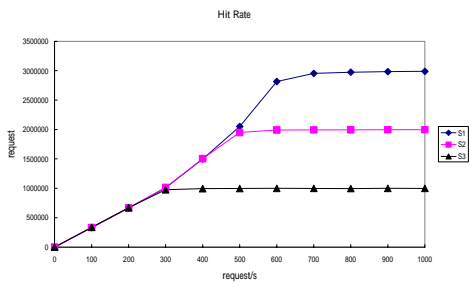
(a) Hit rate



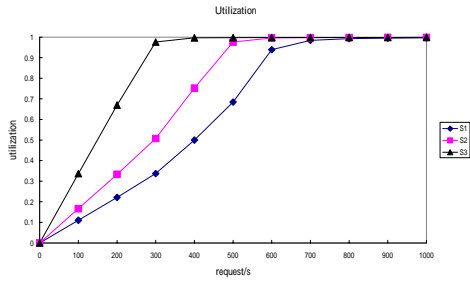
(b) Utilization

Figure 5. RR hit rate and utilization

With our proposed RC algorithm, all the three servers will reach 95% loading in 700 requests per second. This means that the three heterogeneous servers will with 95% loading at the specific arrival rate. In other word, the system can serve as much requests as possible with minimum drop rate. When the arrival rate is less then 300, the least capable server  $S_3$  will serve less connection and its utilization will be lower. In this situation, the average response time will be shorter than the RR and LC algorithm.



(a) Hit rate



(b) Utilization

Figure 6. LC hit rate and utilization

system must have as little drop rate as possible. Now we compare the drop rate of these three load balancing algorithms. As shown in Fig. 8, the RR algorithm begins to drop client request at 300 requests per second, but in the LC and RC algorithm, they begin to drop requests at 600 and 700 requests per second, respectively. Because the LC algorithm considers only the connection number, it redirects the request to the Web server according to the current connection. But with RC algorithm, the remaining capacity will be considered, so the request will be redirected to the server with largest remaining capacity. Thus the drop rate is less than that of LC algorithm.

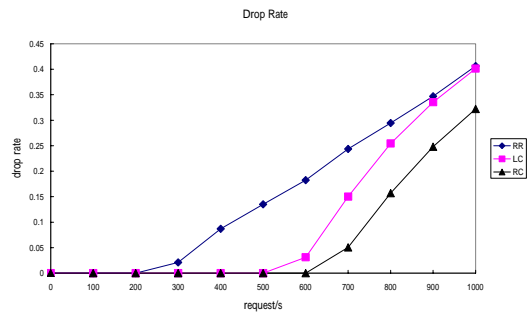
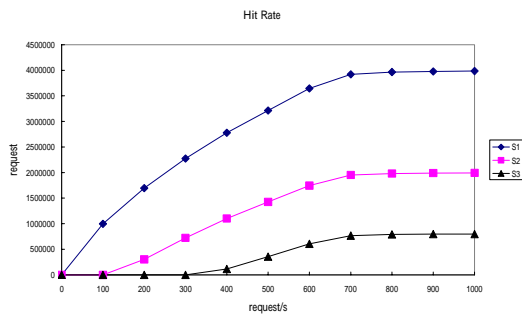
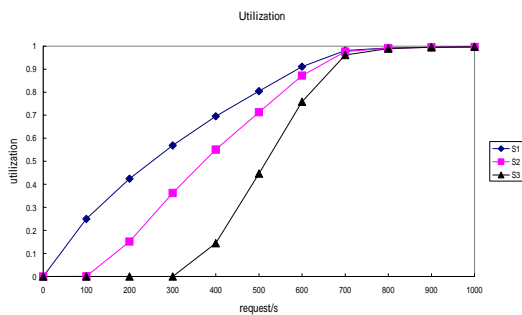


Figure 8. Drop rate of the three algorithms



(a) Hit rate



(b) Utilization

Figure 7. RC hit rate and utilization

### 4.3 Scalability of the Load Balancing System

In section 4.1, we can find that the three Web servers can not handle the arrival rate at 1000 requests per second. How many servers do we need if we want to handle 1000 requests per second, and the drop rate will be less than say, 5%? The scalability test results are shown in Figures 9 to 11. Figure 9 shows the result of using RR algorithm. When the total number of servers is three, the drop rate is 40%. The upper line shows that when we add the least capable server, say  $S_3$ , the drop rate decrease slowly. When we add  $S_1$  or  $S_2$ , the overall drop rate will be 5% at a total seven servers. It means that no matter  $S_1$  or  $S_2$  we add, they will have the same drop rate when the total number of servers are larger then seven.

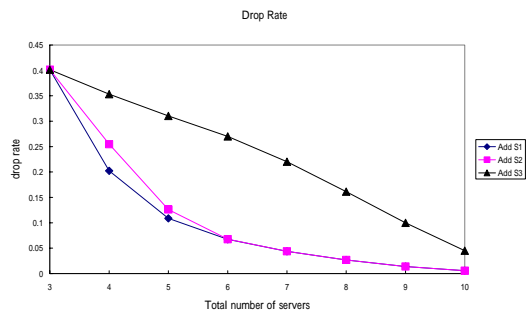


Figure 9. Scalability with RR algorithm

### 4.2 Drop Rate Comparison

When the server reaches its service capacity but the load balancer still redirects the client request to that server, the request will be dropped. An optimal load balancing

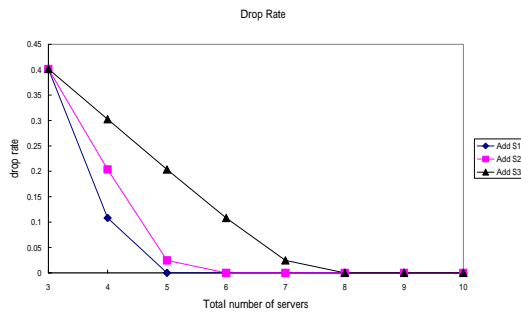


Figure 10. Scalability with LC algorithm

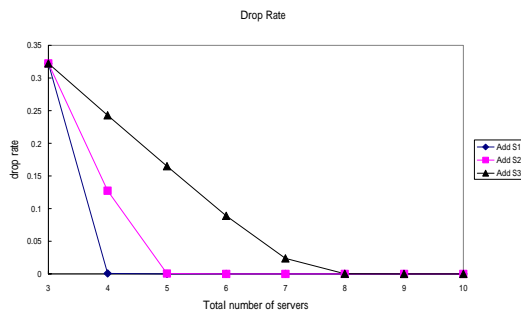


Figure 11. Scalability with RC algorithm

If the LC algorithm is used, we need to add four  $S_3$ , or two  $S_2$  or  $S_1$  to lower down the drop rate to less than 5%. With the proposed RC algorithm, we need only add one  $S_1$  server to handle the requests. Figures 10 and 11 show the results.

## 5 Conclusion

In this paper, the remaining capacity of each server of the heterogeneous Web server cluster is used as the decision making criteria for the load balancer. It is not appropriate to evenly distribute the load to the servers with different capacity as in the least connection or round robin algorithm. The least capable server must serve fewer requests because it does not have enough processing power. Otherwise, the response time will increase rapidly and the drop rate will increase too. In addition, we try to scale the load balancing system so that the drop rates will be less than 5%. From the simulation results, we can further decide the number of required servers under a certain criteria and the tolerable drop rate.

## 6 Acknowledgements

A part of this research was supported by the National Science Council, Republic of China, under contract NSC-94-2745-E-036-003

## 7 References

- [1] D. Mosedale, W.Foss, and R. McCool, "Lessons Learned Administering Netscape's Internet Site", *IEEE Internet Computing*, Vol. 1, No. 2, Mar-Apr. 1997.
- [2] Eric Dean Katz, Michelle Butler, and Robert McGrath, "A Scalable HTTP Server: The NCSA Protocol", *Proc. First International Conference on the World-Wide Web*, 1994.
- [3] Xin Liu, Andrew A. Chien, "Traffic-based Load Balance for Scalable Network Emulation", *Proc. ACM/IEEE Supercomputing 2003*, Nov. 2003.
- [4] Schroeder, T., Goddard, S., and Ramamurthy, B., "Scalable Web server clustering technologies", *IEEE Network*, vol. 14, no 3, May-June 2000.
- [5] Jeffery S. Chase, "Server Switching: Yesterday and Tomorrow", *Proc. Second IEEE Workshop on Internet Applications (WIAPP)*, pp. 23-24 July 2001.
- [6] Netel Networks, *Alteon Link Optimizer Application Guide*, Release 1.0, 2002
- [7] Tsang-Long Pao, Jian-Bo Chen, and I-Ching Cheng, "An Analysis of Server Load Balance Algorithms for Server Switching", *Proc. Ming-Chung University International Academic Conference*, Mar, 2004.
- [8] Chatterjee, D.; Tari, Z.; Zomaya, A., "A Task-Based Adaptive TTL Approach for Web Server Load Balancing", *Proc. 10th IEEE Symposium on Computers and Communications (ISCC)*, pp. 27-30 June 2005.
- [9] V. Cardellini, M. Colajanni, and P. Yu. "DNS dispatching algorithms with state estimators for scalable web-server clusters", *World Wide Web Journal*, Baltzer Science, 2(2):101 – 113, 1999.
- [10] Yeung, K.H., Suen, K.W., and Wong, K.Y., "Least load dispatching algorithm for parallel Web server nodes", *Proc. IEE Communications*, vol. 149, no. 4, Aug. 2002.
- [11] Cardellini, V., Colajanni, M., and Yu, P.S., "Request redirection algorithms for distributed Web systems", *IEEE Trans. Parallel and Distributed Systems*, vol 14, no 4, Apr. 2003.