A Scalable Load-Sharing Architecture for Distributed Applications

Mohamed-Vall M. Salem
Département d'Informatique, Université de Montréal, CP 6128, Succ. Centre-Ville, Montréal, QC, H3C 3J7, Canada,
salem@iro.umontreal.ca

Johnny W. Wong
Computer Science Department, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
jwwong@bcr.uwaterloo.ca

Gregor v. Bochmann
School of Information Technology and Engineering (*SITE*), University of Ottawa, P.O. Box 450, Stn A, Ottawa, Ontario, K1N 6N5,Canada,
bochmann@site.uottawa.ca

Abstract. In recent years, we have seen a significant growth in the development and deployment of distributed applications over the Internet. For large-scale deployment, the system architecture should be able to scale to many users. A common technique for scalable design is replicated servers. The management of replicated architectures introduces new challenges. One very important challenge is how can a client locate the appropriate replica of a server without being aware of the specific details of replica organization, and how can this process scale to a large number of users. In this paper, we investigate the delegation of server selection functionality to an independent brokerage service. A "broker" is used to distribute load to replicated servers. Server selection is "session" based, and intermediate network entities are not required in load balancing activities. Several algorithms for server selection are developed and their performance, under the proposed architecture, is evaluated by simulation.

1. Introduction

In recent years, we have seen a significant growth in the development and deployment of distributed applications over the Internet. These include electronic commerce applications that support the dissemination of information regarding a company's products, and the sale of goods and services. For such applications, the user typically accesses a company's web site using a web browser; he or she may subsequently submit a request to purchase selected items. From the user's perspective, an important parameter for the success of electronic commerce applications is system response time. For large-scale deployment, the server must be able to scale to many users. The need for scalable server design has been recognized since the early stage of web development. Techniques employed include the use of faster and/or multiple processors, caching [1,2] and server replication [3-7]. This paper is focused on the use of replicated server to improve scalability.

Several replicated server architectures have been investigated. In [3,4], modified domain name servers are used to distribute incoming client requests to different servers. Other approaches use intermediate network entities to assist with load balancing. For example, in IBM's NetDispatcher [5], a dispatcher intercepts TCP/IP packets from clients, and forwards those belonging to the same connection to a selected server. As another example, Cisco's LocalDirector [6] performs full TCP/IP address translation to relate the communication between a client and a selected server on a per connection basis. An anycast service [7] has also been proposed for load balancing in our context. This service allows one to locate servers across the Internet. Servers are grouped under an anycast domain name (ADN). When a client attempts to resolve a given ADN address, it resolves to an IP address of one of the servers in the group. A valuable survey of different cluster based scaling techniques is described in [8].

In our investigation, we consider the use of a "broker" to distribute load to replicated servers. The salient features of our architecture are: (i) server selection is "session" based where the broker assigns a client to a specific server for a given duration of time (called the quantum), and

(ii) intermediate network entities (e.g., TCP connection routers or protocol translators) are not required in load balancing activities. Furthermore, the broker is only involved in the initial server selection, and the user interacts directly with the assigned server during the quantum. Our architecture also allows the broker to gather information about server status and client requirements, and use such information for load balancing purposes. An important aspect of our architecture is the algorithm used to select servers in order to optimize the response time. Several algorithms will be developed and evaluated in this paper. These algorithms are different from those reported in the literature [9, 12, 4, and 5] because server selection is done per individual session and not per TCP connection.

The organization of this paper is as follows. Section 2 describes our architecture, its functional components and their interaction. The server selection algorithms under consideration are described in Section 3. The performance of these algorithms is evaluated by simulation. Our simulation model is described in Section 4, and the results are discussed in Section 5. Finally, Section 6 contains a summary of our findings.

2. Proposed Architecture

Our objective is to design an architecture that has the following properties:

- Scalable to a large number of clients.
- Support for the provision of quality of service.
- Support for feedback mechanisms by which up-to-date performance information on system components is available.

2.1 Basic Architecture

Our basic architecture is depicted in Figure 1. In this architecture, scalability is achieved by server replication. As mentioned previously, server selection is done by a "broker", and is based on the notion of a session. The broker assigns clients to servers using a server selection algorithm. Each time a client is assigned to a server, this client may interact with the server for the duration of time equal to the quantum size.

Suppose a client would like to access a given web site, say store1.com (see Figure 1). The protocol between the client and broker is as follows. If the IP address corresponding to store1.com is in the client's cache, then the request is sent directly to that IP address. Otherwise a server selection request is sent to store1.com. This URL is mapped by the DNS to the IP address of the broker. The broker upon receiving the client's request selects a server and returns the IP address of this server, together with the quantum size, to the client. The client then sends its request to this IP address and keeps the IP address in its cache. The cache entry will be deleted when the quantum expires. Note that the above protocol needs to be implemented at the client and we assume that this is possible.

The quantum size should not be too short because this would tend to increase the frequency of server assignment by the broker. The broker may then become the system bottleneck. In addition, the broker has the capability to monitor the performance of the different servers. In our architecture, performance data can be collected by monitoring agents at the servers, and sent to the broker. These data may be used for load balancing purposes. The protocol between the broker and server is quite straightforward. It simply involves the periodic transmission of performance data from server to broker. We assume that it is possible to instrument the server for data collection purposes, and to implement the broker-server protocol at the server.

Our architecture allows for a flexible organization of resources used by web sites. The broker could be at the server site under the same authority as the replicated servers. This is applicable, for example, to sites with heavy load and high degree of replication. Different sites may also share the same broker. In this case, the broker could be an independent brokerage service that manages the assignment of servers for affiliated sites.

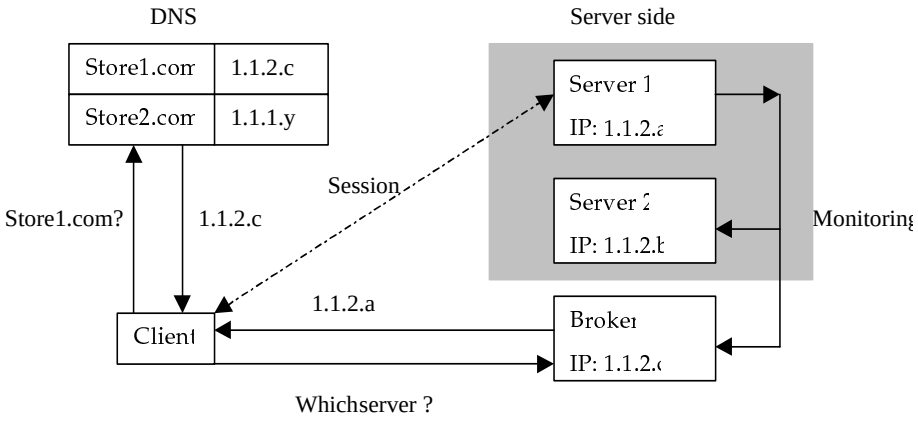

Figure 1: Basic Architecture

2.2 Extensions

Our architecture is quite flexible. For example, the client-broker protocol can be extended to include quality of service negotiation, which can be based on client classification, client's performance requirement, or server availability. Additional scalability can be provided by the use of multiple brokers. These issues will be addressed in future papers.

3. Server Selection Algorithm

We distinguish between "static" algorithms and "dynamic" algorithms depending on whether data on run-time server behavior are used in server selection decisions or not.

3.1 Static Algorithms

For static algorithms, data on run-time server behavior are not used. The two algorithms under consideration are similar to those reported in [5]. Suppose there are N servers. These algorithms can be described as follows:

(A) **Round Robin (RR)** – servers are selected in cyclic order, i.e., server 1, server 2, …, server N, repeatedly.

(B) **Weighted Round Robin (WRR)** – cyclic ordering is also used, but some servers are selected more frequently than the others are within a cycle. Consider, for example, the case of two servers. If server 1 should handle twice the load as server 2, then server selection is made using the following ordering: server 1, server 1, server 2, server 1, server 1, server 2, and so on.

3.2 Dynamic Algorithms

For dynamic algorithms, data on the servers' run-time behavior are used in server selection decisions. As mentioned previously, such data are collected by the servers and periodically transmitted to the broker. Two dynamic algorithms are considered in our investigation. These algorithms require the availability of measurement data for the following parameters:

r_i – *Mean response time of server i,*

s_i - *Mean service time at server i*

u_i – *Utilization of server i, defined as the fractions of time the server was busy.*

h_i – *Mean think time as seen by server (mean time between successive requests over the same session).*

λ_i – *Mean request arrival rate per session at server i, and is given by* $\lambda_i = 1/(r_i + h_i)$

Λ_i – *Mean request arrival rate at server i*

for $i = 1, 2, \ldots, N$.

It should be noted that from the perspective of the server, each request is for an individual object, such as a static file, an image, a dynamic request, etc. As a result, the think time, as seen by the server, is measured from the completion of one request to the reception of the next request from the same client. The details of our algorithms are as follows:

(C) Least Active Sessions (LAS)

Based on our architecture, the broker is able to determine how many sessions have been assigned to a given server at any point in time. This can be done, for example, by keeping track of when the quanta allocated to the different sessions expire. This information, however, may not be very useful for load balancing purposes because some of the sessions may not be active during a given measurement interval. By an active session, we mean the client is submitting requests to the assigned server. The number of active sessions at the various servers can be estimated using the measurement data. The number of active sessions at server i (denoted by NS_i) is given by [11]:

$$NS_i = (r_i + h_i) * \Lambda_i \qquad (1)$$

Suppose it is required that the mean response time does not exceed $r_{\max}$, otherwise the users may find the performance unacceptable. Assuming that $r_{\max}$ occurs when the system is closed to saturation (i.e. when the arrival rate Λ_i is equal to $\Lambda_{\max}$, the maximum arrival rate that the server can handle and which is estimated by $1/s_i$), a good estimate for the maximum number of active sessions to be supported by server i is:

$$NS_{i,\max} = (r_{\max} + h_i) * \Lambda_{\max} \qquad (2)$$

We now describe the LAS algorithm. Let L_i be the "session saturation" of server i ($i = 1, 2,.., N$) defined by the equation $L_i = NS_i / NS_{i,\max}$. The algorithm proceeds as follows:

- Initially, L_i is set to zero for each server i.
- Each time new measurement data are available, NS_i and $NS_{i,\max}$ are computed using Equations (1) and (2) for each server i, and L_i is updated according to the above formula.
- Each time a server selection is requested by a client, server j is selected where $L_j = \min_i \{L_j\}$ (i.e. the server with the lowest session saturation). L_i is incremented by 1.

(D) Least Utilization (LU)

This algorithm uses the values u_i of the utilization of servers as observed periodically. Instead of L_i, we use the utilization level u_i of the servers to select the best server. The LU algorithm can be described as follows:

- Initially, u_i is set to zero for each server *i*.
- Each time new measurement data are available, the current value of u_i is determined.
- Each time a server selection is requested by a client, server *j* is selected where $u_j = \min_i \{u_j\}$, and u_i is incremented by $\lambda_i * s_i$ which is an estimate of the additional utilization incurred by the new session.

4. Simulation Model

Our simulation model consists of a single broker, *N* servers and *M* concurrent clients. For each client, a page request is submitted at the end of a "user think time"[1]. Each page request corresponds to one or more "object requests" to be submitted to the server. We assume that at the client, object requests are submitted sequentially as required in HTTP 1.0. This is modeled as follows. When a response is received for an object request, the next object request is submitted after a processing at the client. When all the objects have been received, the page request is satisfied, and the client starts the next user think time. We further assume that objects are not cached and network delays are assumed to be negligible. Our modeling of the client behavior can easily be extended to represent the pipelining of object requests as allowed in HTTP 1.1. As in [12], two heavy tailed distributions, namely Pareto and Weibull, are used to model the user think time, the number of objects per page request, the processing time between page requests. The probability density function of the Pareto distribution is given by $f(x) = \alpha k^{\alpha} x^{-\alpha-1}$ where $\alpha, k > 0$ and $x \geq k$. The probability density function of the Weibull distribution is given by $f(x) = \beta/\alpha^{\beta})x^{\beta-1} e^{-(x/\alpha)^{\beta}}$ where $\alpha, \beta > 0$. The size of each object request is modeled by a Bounded Pareto distribution given by $f(x) = \frac{\alpha k^{\alpha}}{1-(k/p)^{\alpha}} x^{-\alpha-1}$ where $\alpha, k, p > 0$ and $k \leq x \leq p$. The values of the parameters of these distributions used in our simulation experiments are the same as those reported in [12]. These parameter values are summarized in Table 1.

Each server is modeled by a "capacity" parameter as described in [4], which is the time required to process one byte of data. For example, if the capacity of a server is 10^{-6} sec/byte, and the average size of an object is 10,000 bytes, then the server can process on average 100 objects per second. For the dynamic algorithms, the required parameters are measured during the simulation. These parameters, as defined in Section 3.2, are mean response time, utilization, mean think time, and arrival rate, at each of the servers. The measurements are taken over regular intervals of duration D. In our simulation experiments; D is set to 10 seconds.

Parameter	Description
User think time	Pareto (α=1.5, k=3)
Embedded objects per page	Pareto (α=2.43, k=2.3)
Object processing time	Weibull (α=0.146, β=0.382)
Object size	Bounded Pareto($\alpha = 1.25, k = 1800, p = 10^8$)

[1] Note that the user think time is different from the think time as seen by the server discussed in section 3, which includes user think time as well as processing time between object requests.

5. Simulation Results

In our experiments, the following two configuration are used:

- Homogeneous: four servers, each has capacity of 10^{-6} sec/byte
- Heterogeneous: five servers: two with capacity 10^{-6} sec/byte and three with capacity $1.5x10^{-6}$ sec/byte

Note that the total number of objects processed per second is the same for each of the two configurations. Two levels of load are simulated:

- Heavy: 2000 concurrent clients, which yield average utilization of approximately 95% among the servers
- Moderate: 1000 concurrent clients, which yield average utilization of approximately 63% among the servers

The performance measures of interest are:

- U(x) – Prob [utilization $\leq$ x]
- R(x) – Prob [response time $\leq$ x]

U(x) is a measure that indicates how balanced the load is. It can be interpreted as follows. During the simulation, the utilization is obtained for each interval of duration D. U(x) is the percentage of intervals where the measured utilization is less than or equal to x. In our simulation experiments, D is set to 10 seconds.

5.1 Homogeneous configuration

Consider first the homogenous server configuration. We show in Figure 2, the results for U(x) and R(x) for RR under heavy load. The quantum (or session length) is chosen to be 3 minutes. The corresponding results for WRR, LAS and LU are similar. We observe that the utilization and response time characteristics of individual servers are identical. The same observation is made for the case of moderate load (results not shown). We conclude that with homogeneous server, a simple algorithm such as Round Robin is very effective in load balancing.

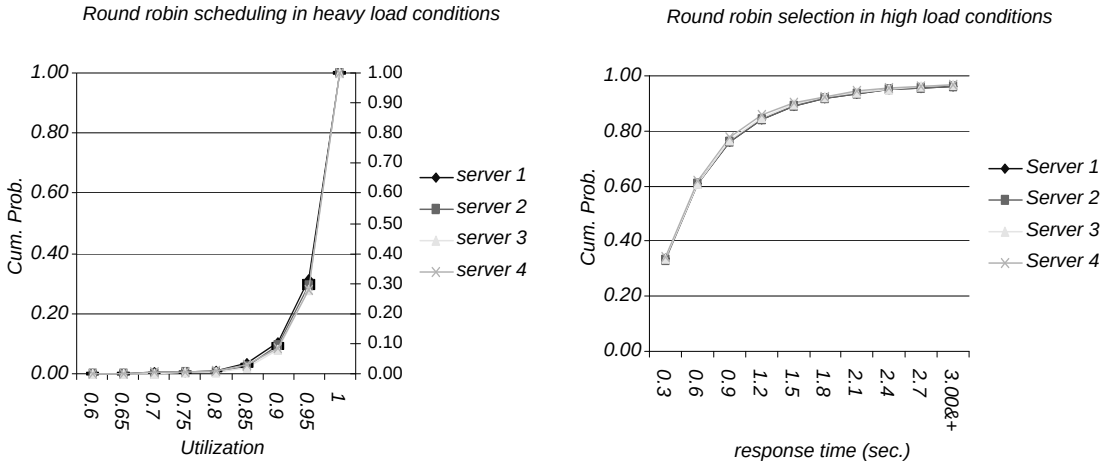

Figure 2: Performance achieved by Round-Robin selection in a cluster of homogeneous servers

5.2 *Heterogeneous configuration*

We next consider the heterogeneous server configuration. The results for U(x) and R(x) for RR under heavy load are shown in Figure 3. We observe that RR yields significant load imbalance among the servers. More specifically, for a fast server, the value of U(x) = Prob [utilization ≤ x] is 81% for x = 0.9, while a slow server is almost always saturated. The load imbalance is also reflected in the response time results. For example, at a fast server, 95% of the requests have a response time less than 1.5 seconds, while only 67% of the requests receive the same performance at a slow server. Similar behaviors are observed for the case of moderate load, although the load imbalance and its effect on response time are less noticeable. The above results can be explained as follows. RR does not distinguish between servers of different capacities, and treats all the servers the same as far as load distribution is concerned. This tends to under utilize the fast servers and overload the slow servers. We thus conclude that RR is not a suitable algorithm for a heterogeneous server configuration.

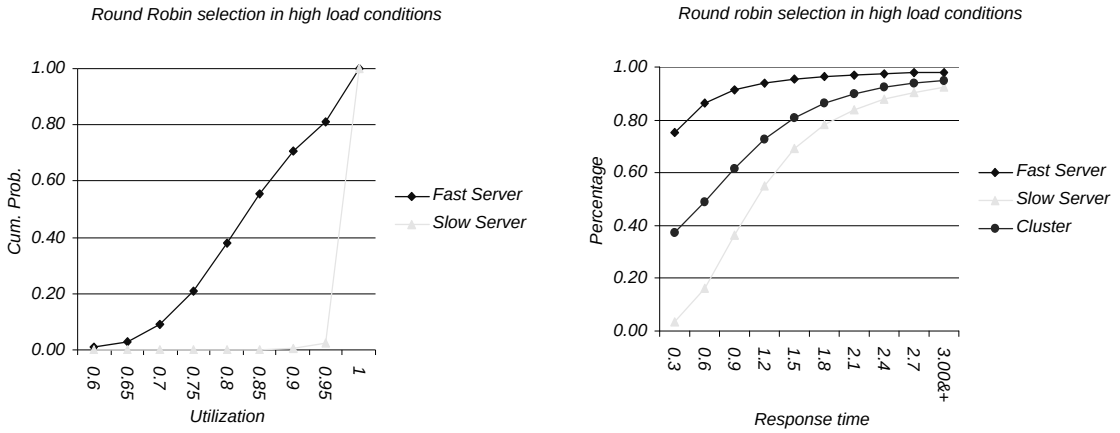

Figure 3: Performance achieved by Round-Robin selection in a cluster of heterogeneous servers

In Figure 4, the results for U(x) and R(x) for WRR under heavy load are shown. We observe that the load imbalance as exhibited by RR has been eliminated. Similar behaviors are also observed for the LAS and LU algorithms (results not shown). We next compare the performance of WRR, LAS, and LU.

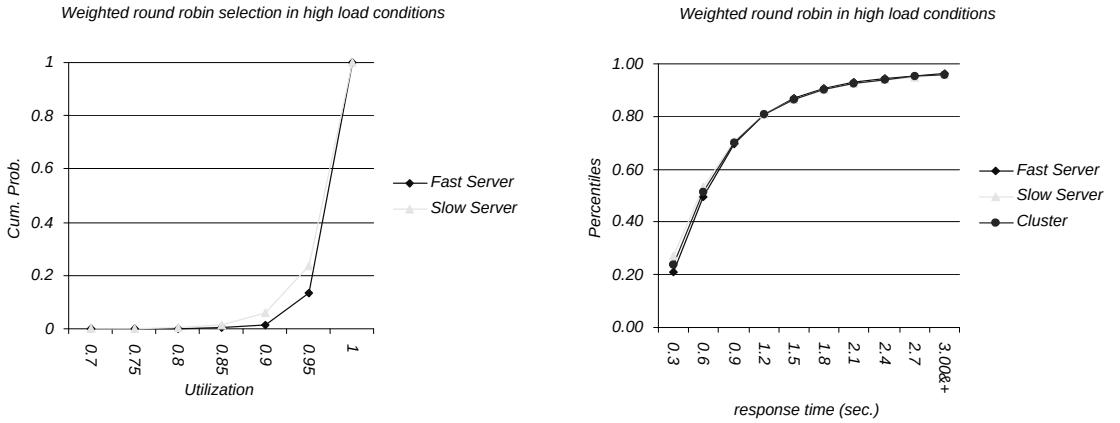

Figure 4: Performance achieved by Weighted Round-Robin selection in a cluster of heterogeneous servers

In Figure 5, the average values of U(x) over the five servers are shown. We observe that LAS has the best performance, followed by LU, and then by WRR. Specifically, for LAS, the value of U(x) = Prob [utilization $\leq$ x] is 20% for x = 0.9, while those for LU and WRR are 12% and 4%, respectively. The corresponding U(x) values for x = 0.95 are 38%, 32% and 20% for LAS, LU and WRR. Finally, the results for R(x) for the three algorithms are shown in Figure 9. LAS and LU have similar performance, and both of them are slightly better than WRR.

WRR has the advantage of simplicity, assuming that the weights can be determined efficiently. It should be the preferred algorithm unless the available capacities at the various servers change frequently, which requires an adaptive mechanism to adjust the weights. Changes in available capacity may be due to factors such as temporary effects of other jobs running on the server. The dynamic algorithms LAS and LU have the ability to adapt to changes to available capacities, and would be good alternatives under such environments. Finally, the results of the simulations did not show any significant influence of the size of the quantum on the performance of the selection algorithms.

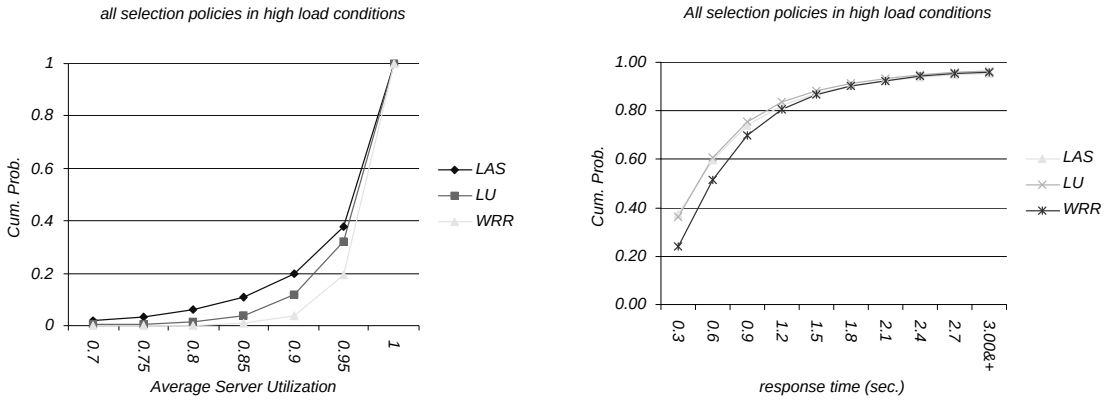

Figure 5: Performances achieved by different selection policies for a cluster of heterogeneous servers in high load conditions.

6. Conclusions

In this paper, we introduced a scalable architecture for distributed applications. This architecture is based on the notion of a "broker". Clients are assigned to a server during a session. Provisions are made for a server to transmit measurement data to the broker. Such data are useful for server selection purposes. Within our architecture, two existing algorithms (RR and WRR) and two new algorithms (LAS and LU) have been evaluated by simulation. Results show that a simple algorithm like WRR is very effective in balancing the load among the servers. In cases where the available capacities at the various servers change frequently, an adaptive mechanism is desirable, and LAS and LU would be good alternatives. The results of the simulations did not show any significant influence of the size of the quantum on the performance of the selection algorithms.

Our architecture is practical and can be readily implemented. The implementation, however, requires modification of client software. Our architecture can also be extended to include service negotiation and multiple brokers.

7. Acknowledgement

This work was supported by the Canadian Institute for Telecommunications Research under the Networks of Centres of Excellence Program of the Government of Canada, and by the IBM Toronto Laboratory Centre for Advanced Studies.

8. References

[1] P. Danzing, E. Hall, and M. Schwartz, "A case for caching file objects inside internetworks", in Proceeding of SIGCOMM 93, pp. 239-248, 1993.

[2] Katz, E. D., Butler, M., and McGrath, R.A.:"A Scalable HTTP Server: The NCSA J. Pitkow and M. Recker, "A simple and yet robust caching algorithm based on dynamic access patterns", in Proceeding of ACM Multimedia 94, pp. 15-23, 1994.

[3] Katz, E. D., Butler, M., and McGrath, R.A.:"A Scalable HTTP Server: The NCSA Prototype". Computer Networks and ISDN systems, First International Conference on the World Wide Web, Elsevier Science BV

[4] V. Cardellini, M. Colajanni, P. S. Yu, ``DNS dispatching algorithms with state estimators for scalable Web-server clusters", World Wide Web Journal, Baltzer Science, Vol. 2, No. 3, pp. 101-113, Aug. 1999.

[5] D. Dias, W. Kish, R. Mukherjee, and R. Tewari: "A Scalable and Highly Available Web Server", Proc. Of the 41st IEEE Computer Society International Conference, February 1996.

[6] Cisco Local Director, http://www.cisco.com/warp/public/cc/pd/cxsr/400/index.shtml

[7] S. Bhattacharjee, M. H. Ammar, E.W. Zegura, V. Shah, Z. Fei: "Application Layer Anycasting". In Proceedings of INFOCOM 97, 1997.

[8] T. Schroeder , S. Goddar, B. Ramamurthy, "Scalable Web Server Clustering Technologies", IEEE Network, May-June 2000, pp. 38-45.

[9] Z. Fei , S. Bhattacharjee, E.W. Zegura, M. H. Ammar: "A novel Server Selection Techniques for Improving the Response Time of a Replicated Service". In Proceedings of INFOCOM 98, 1998

[10] M. Colajanni, P.S. Yu, D.M. Dias, ``Analysis of task assignment policies in scalable distributed Web-server systems", IEEE Trans. on Parallel and Distributed Systems, Vol. 9, No. 6, June 1998.

[11] Raj Jain: "The Art of Computer Systems Performance Analysis", John Wiley & Sons, Inc, 1991

[12] Paul Barford and Marck Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation", in Proceeding of the 1998 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp. 151-160, July 1998.