Server-Based Dynamic Server Selection Algorithms

Yingfei Dong[1], Zhi-Li Zhang[1], and Y. Thomas Hou[2]

[1] Dept. of Computer Science and Engineering, University of Minnesota,Minneapolis, MN 55455
{dong,zhang}@cs.umn.edu,
Tel:612-626-7526, FAX: 612-625-0572.
[2] Fujitsu Labs of America,595 Lawrence Expressway, Sunnyvale, CA 94085
thou@fla.fujitsu.com

Abstract. Server selection is an important problem in replicated server systems distributed over the Internet. In this paper, we study two server selection algorithms under a server-based framework we have developed. These algorithms utilize server load and network performance information collected through a shared passive measurement mechanism to determine the appropriate server for processing a client request. The performance of these algorithms is studied using simulations. Comparison with two naive server selection algorithms is also made. The initial simulation results show that our dynamic server selection algorithms have superior performance over the two naive ones, and as a result, demonstrate the importance of dynamic server selection mechanisms in a replicated server system.

Keywords: Server replication, server-based dynamic server selection, QoS, passive measurement

1 Problem Formulation and Related Work

Server replication (or mirroring) is a common technique that has been used to provide scalable distributed service over the Internet. If done appropriately, server replication can avoid server overload and significantly reduce client access latency. In addition to the issues such as where to place replicated servers in the global Internet and how clients can locate a server with the desired service, an important and challenging problem in server replication is how to select a server to process a client request so as to provide the "best service" for the client. This problem, referred to as the *server selection* problem, is the focus of our paper.

In addressing the server selection problem, two procedures are typically involved. First, statistics about server/network performance need to be collected. Based on the collected statistics, a server selection algorithm then selects the "best" server among a pool of eligible servers for processing a client request. Depending on where the statistics are collected and the server selection decision is made, existing server selection approaches [5,7,8,11,14,16,19,23] can be classified as either *client-based* or *server-based*.

I. Chong (Ed.): ICOIN 2002, LNCS 2344, pp. 575–584, 2002.

Under the client-based approach, statistics about the network and server performance is typically collected using the *active probing* method [7,8,11,16,19]: a client [8] or its "proxy" (e.g., the service resolver in [11]) sends probe packets to measure the network performance.[1] As an exception, Seshan *et al.*[23] employs a novel *passive measurement* method where clients share their network performance discovery through performance monitoring at the client side. The major drawback of the client based server selection approach is that it is not *transparent* to clients: either clients or their proxies (e.g., service resolvers) need to know the name/location of all the servers providing a given service. Furthermore, the client-based approach requires modification of client browser software and installation of network measurement tools at every client side, and in some cases, it may rely on the deployment of an Internet-wide measurement infrastructure (e.g., modification of DNS to incorporate service resolvers). In the case where active probing measurement techniques are used, the extra traffic introduced by probe packets can lead to network bandwidth wastage or even congestion. In the long term the client-based approach may have its appeal, especially when an Internet-wide measurement infrastructure [15,21,22] is in place. However, in the immediate future, server selection mechanisms using the client-based approach are unlikely to be widely available to many clients to take advantage of replicated server systems.

In contrast, the server-based approach relies on the cooperation of servers providing a given service to determine the "best" server for a client and inform the client the location of the server via, say, HTTP redirect mechanism. An example of the server-based approach is HARVEST system [5] which uses a simple network metric, *hop count*, as the criterion to select the "best" server for clients.

In [14], we propose and develop a server-based measurement and server selection framework which employs passive network measurement techniques and performance information sharing among servers to dynamically select the "optimal" server for processing client requests. The proposed framework contains three major components: (1) a server-based passive network measurement mechanism based on tcpdump [18] and a prototype of which called *Woodpecker* is described in [10], (2) a metrics exchanging protocol for sharing server load and network performance information among the servers, which will be described in a future paper, and (3) a dynamic server selection mechanism for selecting an "optimal" server to process a given client request, which is the focus of this paper.

In this paper we describe two server-based *dynamic* server selection algorithms. These two algorithms utilize both the server load and network performance information collected by each server and shared among them. Based on these performance statistics, a server decides whether itself or another server would be the appropriate server to process a client request it receives. In the latter case, the client request is redirected to the selected server (see Fig. 1).

[1] In the case of [11] servers also collaborate with service resolvers by "pushing" their performance statistics to service resolvers periodically.

The performance of these algorithms is studied using simulations. Comparison with two naive server selection algorithms is also made. The initial simulation results show that our dynamic server selection algorithms have superior performance over the two naive ones, and as a result, demonstrate the importance of dynamic server selection mechanisms in a replicated server system.

Our server-based approach to the problem of server selection has the following salient features. First, it is client-transparent. Second, it does not require any modification to any client software or relies on the availability of an Internet-wide measurement infrastructure. Therefore it is readily deployable. Third, by using shared passive network performance measurement, we avoid the wastage of network resources of active probing measurement techniques. The network bandwidth overhead incurred by metrics exchanges among the servers is relatively low, as the number of servers in a replicated system is generally small, in particular, when compared to the number of clients (or the sub-networks). Fourth, by taking both the server load and network congestion status of paths between servers and clients, we can employ relatively sophisticated dynamic server selection algorithms to choose the "optimal" server to service client requests without incurring too much latency and overhead. Our approach is particularly amenable to an enterprise-based replicated server system, where a number of servers connected through an enterprise network provide certain services over the Internet to "regular" clients who need access to the services frequently.

The remainder of this paper is organized as follows. In Section 2, the two dynamic server selection algorithms are presented. The performance results are shown in Section 3. In Section 4, we conclude the paper and discuss the future work.

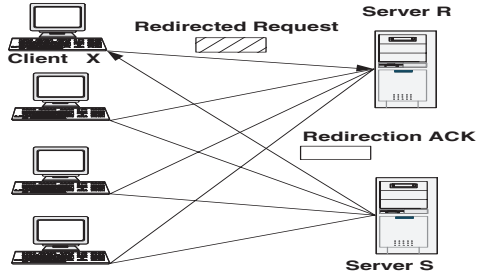


Fig. 1. An example system

2 Server-Based Dynamic Server Selection Algorithms

In this section, we present two dynamic server selection algorithms. Before describing our algorithms, we define two major performance metrics used in our server selection algorithms: *server loads* and *fast paths*, which indicate the status of servers and different network paths from servers to clients.

```
1. if (a client X is a regular client)
2.  if (server load is light) and (the path from S to X is not slow)
3.    accept the request
4.  else
5.    if (all other servers are heavily loaded)
6.      accept the request
7.    else
8.      randomly redirect the request to R other than S
9. else
10. if (server load is light) or (all other servers are heavily loaded)
11.   accept the request and generate metrics about the client
12. else
13.   redirect to a light-loaded server
```

Fig. 2. Random Redirection Server Selection Algorithm

2.1 Performance Metrics

Server Load: For given server S, its load, denoted by L_S, is defined as the ratio of T_{total} to R_{max}. Here, T_{total} is the time period required to serve all requests in the waiting queue on S, and is related to the capability of S. It is computed based on the current and historical information of S including the processing delay on S and the transferring delay from S to a client. R_{max} is the maximum response delay that a typical client can accept for a request of average size, such as 10 KBytes [6]. L_S is called *light* if $L_S \leq 80\%$; otherwise, it is called *heavily loaded*.

Fast Path: As shown in Figure 1, if the transferring delay of a reply from a server S to a client X is k times longer than the transferring delay from another server R to X plus the cost of redirecting the request from S to R, the path from S to X is called *slow*, and the path from R to X is called $fast$. Here, k is an experimental parameter, such as 2 or 4. The transferring delay depends on the size of the reply and the bottleneck bandwidth of the path. For simplicity, the bottleneck bandwidth is approximated through the TCP-friendly formula [17,24]. The redirection cost is the delay of an acknowledgment from S to X plus the delay of the redirected request from X to R. Both transferring delays are estimated through the RTTs of the paths which are obtained through the passive measurement on servers.

2.2 Server Selection Algorithms

Our two simple server selection algorithms are given here. When a client X sends a request to server S, there are two situations in which this request may be redirected: (1) server S is overloaded, or (2) the path from S to X is *slow*. We give two algorithms to choose another server R which may provide better service under both situations. The first algorithm is called *Random Redirection (RR)* which randomly redirects a request to another server R other than S. The

```
1. if (a client X is a regular client)
2.  if (server load is light) and (the path from S to X is not slow)
3.    accept the request
4.  else
5.    computing equivalent classes
6.    if (no better server)
7.      accept the request
8.    else
9.      redirect the request to the ''best'' server R
10. else
11. if (server load is light) or (all other servers are heavily loaded)
12.   accept the request and generate metrics about the client
13. else
14.   redirect to a light-loaded server
```

Fig. 3. Best-guess Redirection Server Selection Algorithm

second algorithm examines all servers and selects the "best" one as R based on the current metrics on S. Because the current metrics on server S may not be accurate due to the delay of metrics exchange among servers, the second method is called *Best-guess Redirection (BR)*. RR is the simplest method which doesn't required too much calculation. BR, however, is one of the most complicated methods, in which server must know the status of all servers and related network paths to clients and compare all the possible choices.

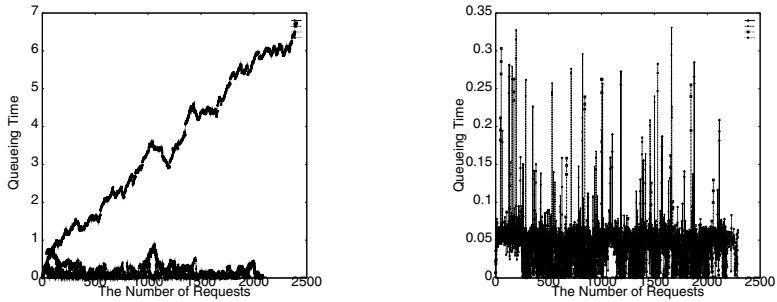
Figure 2 and Figure 3 gives the basic frame of algorithms RR and BR. When a server S receives a request from a client X, S checks whether accepting this request into its waiting queue or redirecting this request to another server R. As shown from line 1 to 3, if X is a regular client, and S's load is light and the path from S to X is not slow, S accepts this request. Otherwise, in RR method in Figure 2 from line 5 to 8, S randomly redirect the request to another server R other than S; or, in BR method in Figure 3 from line 5 to 9, S computes equivalent classes to find whether a better server exists. *The equivalent class* of server S respective to a specific request is the set of servers whose current response time for this specific request is within a threshold compared with that of the server S. If all the servers are in the same equivalent class (i.e., there is no better server), S has to accept the request. Otherwise, S redirects the request to R which could provide "best" service.

Lines 10 to 14 shows the case that there is no information available about this client. If S's load is light, it accepts this request. Otherwise, it redirects this request to a light loaded server. In order to avoid the request oscillating among the servers, S has to accept this request if all other servers are heavily-loaded.

3 Simulation

3.1 System Models

In our simulation, a system consists of clients, servers and network paths. Figure 1 shows an example system with 2 servers, 4 clients and 8 network paths.



(a) Fixed-Group Algorithm.

(b) Random-Redirection Algorithm.

Fig. 4. Queueing delays of all 4 servers under FG and RR algorithms

Client Model: A client generates requests and receives replies from servers. For generating requests, the size of a request is Pareto-distributed with an average request size of 10 KBytes [6] and a shape parameter 1.66 [9]; the inter-arrival of requests is exponential distributed. In addition, we assume a new request can be generated even though previous requests have not been served. For receiving a reply, if the reply is a redirection acknowledgment, the client resends the request to the redirected server; if the reply is data, the client records the response time of the request.

Server Model: When a server receives a request, it first checks whether to accept this request or to redirect it to another server using the server selection algorithms presented above. If a redirecting decision is made, the server sends a redirecting acknowledgment back to the client; otherwise, the server accepts this request into a FIFO queue which holds all waiting requests when the server is serving a previous request.

A server always processes the request at the head of the waiting queue if the queue is not empty. The cost of processing a request on a server consists of its queuing cost, its operating system cost for starting and ending a service, and its storage accessing cost. Queuing cost is the time period between the request's arriving and leaving the queue. Operating system cost is counted using an average value (e.g., 0.1ms). Storage accessing cost are computed through a storage model [13] with the parameters, such as an average seek time (e.g., 10ms), an average rotation delay (e.g., 1ms), a controller overhead (e.g., 1ms), and the transferring cost from the storage to the memory with a 20 Mbytes/second I/O bus. The effect of disk cache and memory cache will be considered in our future study.

Network Model: Each client has a network path to each server, while the bottleneck bandwidth of the path can be approximated using the formula in [17,24] through the Maximum Transfer Unit (MTU), the Round-Trip Time (RTT) and the loss rate of the path.

$$Bandwidth = \frac{1.3 \cdot MTU}{RTT \cdot \sqrt{LossRate}}$$

From the results in [4,6,9,17,24], the typical values of loss rates, MTUs, and RTTs of network paths are: a loss rate is between 0.01% and 5%, a MTU is between 576 bytes to 1500 bytes, and a RTT is between 20 ms and 500 ms. In our simulation, the loss rate, MTU and RTT of a path are generated within the above ranges to simulate the dynamic status of the network paths.

Table 1. Response times among all clients

Method	AVG	MAX	MIN	STD	# Requests Served
FS	4.048	14.16	0.007	3.953	6955
FG	1.189	6.932	0.009	1.851	7949
RR	0.142	3.489	0.003	0.196	8341
BR	0.140	2.647	0.003	0.210	8343

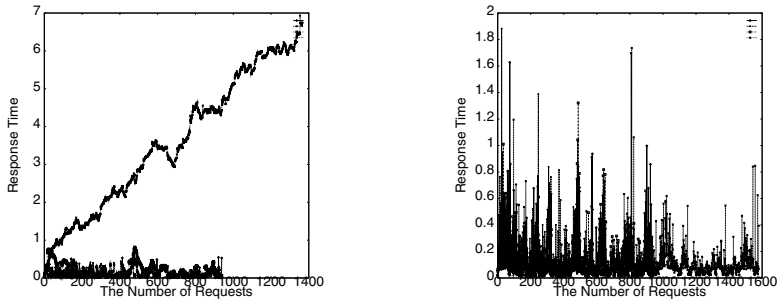
3.2 Initial Simulation Results

In this section, we present our initial simulation results. Besides the proposed two server selection algorithms, RR and BR, two other static server assignment approaches are also tested in our simulation to investigate the effect of the server selection algorithms. The first one is called *Fixed-Server (FS)* assignment method: a client keeps requesting services from a fixed server which is initially randomly chosen by the client, and a server has a fixed number of clients during a simulation. This is the most naive situation. The second method is called *Fixed-Group (FG)* assignment approach: all clients are divided into equal size groups, and each group of clients only contacts with a single default server. In this method, each server has the same number of clients during a simulation. FS and FG are similar to the methods used in existing systems.

Our initial simulation results are given as follows. In this group of simulations, 16 clients require services from 4 servers during a period of 1 minute. In Table 1, the average response time of all clients under four approaches are given, as are the maximum, minimum and standard deviation of the response time. The total number of requests served are listed in the last column of Table 1. It is easy to see that RR and BR served more requests than FS and FG.

Table 2 shows the queuing times on the simulated servers. Similarly, the average value, the maximum, the minimum and the standard deviation are given.

Choosing the average results of BR as the base 1.00, Table 3 clearly shows that RR and BR are significant better than FS and FG both in the average response time among clients and the average queuing time among servers. An interesting finding is that RR is only slightly worse than BR. BR requires much more computing cost than RR, but RR did more redirections than BR. In our simulation, 51664 times of redirections took place in the RR approach, while



(a) Fixed-Group Algorithm.

(b) Random-Redirection Algorithm.

Fig. 5. Response times of the first 4 clients under FG and RR algorithms

Table 2. Queuing delays among all servers

Method	AVG	MAX	MIN	STD
FS	4.028	14.17	0.0	3.956
FG	1.166	6.749	0.0	1.850
RR	0.049	0.342	0.0	0.028
BR	0.044	0.327	0.0	0.028

47686 times of redirections happened in the BR approach. A large scale simulation will show even greater differences between them.

Table 3. Summary of simulation results

	FS	FG	RR	BR
Avg Response Time over all clients	28.91	8.49	1.01	1.00
Avg Queuing Time over all servers	85.70	24.80	1.04	1.00

The dynamic methods, such as RR and BR, distributed the requests more evenly among servers than the static methods such as FS and FG. The FG method is better than the FS method in static methods while the RR method is slightly worse than the BR method in dynamic methods. Therefore, we choose the FG and RR methods as representatives to compare static methods with dynamic methods. Figure 4 shows the queuing times of requests on four server in the FG and RR methods. The x axis is the number of requests which have been served on servers, and the y axis is the queuing time. Figure 4(a) shows that one waiting queue keeps building up in the FG method; however, Figure 4(b) shows that all the queuing times of requests are distributed in a small range in the RR method. Figure 5 shows response times of 4 clients under the FG and RR

methods. The x axis is the number of requests which clients have received their replies, and the y axis is the response time. In Figure 5(a), the response time of one client keeps increasing in the FG method; on the other hand, Figure 5(b) clearly shows that, the response time of all client stay within a small range in the RR method.

4 Concluding Remarks

In this paper, we study two server selection algorithms under a server-based framework that we have developed in [14]. Both algorithms employ server load and network performance metrics collected through a shared passive measurement mechanism to determine the appropriate server for a client request. Simulations show that both dynamic algorithms are significantly better than static methods in terms of response times and queuing delays. Work is under way to implement this mechanism and investigate its performance over the Internet.

References

1. H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm, and R. H. Katz, "Analyzing stability in wide-area network performance," *in Proc. ACM SIGMETRICS'97,* June 1997.
2. H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm, and R. H. Katz, "TCP behavior of a busy Internet server: analysis and improvements," *in Proc. IEEE INFOCOM'98,* San Francisco, CA, March 1998.
3. P. Barford and M. E. Crovella, "Generating representative web workloads for network and server performance evaluation," *Technical Report BU-CS-97-006,* Dec. 1997.
4. J. Bolot, "Characterizing end-to-end packet delay and loss in the I Internet," *Journal of High Speed Networks,* vol.2, no.3, pp. 305–323, Dec. 1993.
5. C. M. Bowman, *et. al.,* "Harvest: a scalable, customizable, discovery and access system," *Technical Report CU-CS-732-94,* Dept. of Computer Science, University of Colorado at Boulder, 1994.
6. N. Cardwell, S. Savage, and T. Anderson "Modeling the performance of short TCP connections," Oct. 1998.
7. R. L. Carter and M. E. Crovella, "Measuring bottleneck link speed in packet-switched networks," *Performance Evaluation,* vol. 27 & 28, 1996.
8. R. L. Carter and M. E. Crovella, "Dynamic server selection using bandwidth probing in wide-area networks," *in Proc. IEEE INFOCOM'97,* Kobe, Japan.
9. C. R. Cunha, A. Bestavros, and M. E. Crovella, "Characteristics of WWW client-based traces," *in Proc. ACM SIGMETRICS'96.*
10. Y. Dong, Y. T. Hou, and Z.-L. Zhang, "A novel server-based measurement infrastructure for Enterprise Networks ", *Technique Report TR 98-031,* Dept. of Computer Science, University of Minnesota, 1998.
11. Z. Fei, S. Bhattacharjee, E. W. Zegura, and M. Ammar, "A novel server selection technique for improving the response time of a replicated service," *in Proc. IEEE INFOCOM'98,* San Francisco, CA, 1998.

12. J. Guyton and M. Schwartz, "Locating nearby copies of replicated Internet services," *in Proc. ACM SIGCOMM'95,* Cambridge, MA, 1995.
13. J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach,* Second Edition, Morgan Kaufmann Publishers, Inc., 1996.
14. Y. T. Hou, Y. Dong, and Z.-L. Zhang, "Network performance measurement and analysis - part 1: a server-based measurement infrastructure", *Technical Memorandum FLA-NCRTM98-01,* Fujitsu Laboratories of America, CA, July, 1998.
15. Internet Distance Maps Project, `http://idmaps.eecs.umich.edu/.`
16. V. Jacobson, `pathchar` - A Tool to Infer Characteristics of Internet Paths, `http://ee.lbl.gov/nrg-talks.html`, April 1997.
17. J. Mahdavi and Sally Floyd, "TCP-Friendly Unicast Rate-Based Flow Control", The TCP-Friendly Website.
18. S. McCanne, C. Leres, and V. Jacobson, `tcpdump` , Lawrence Berkeley National Laboratory.
19. K. Moore, J. Cox, and S. Green, "SONAR - A Network Proximity Service," *Internet-Draft,* `http://www.netlib.org/utk/projects/sonar/,` Aug. 1996.
20. A. Myers, P. Dinda, and H. Zhang, "Performance characteristics of mirror server on the Internet," *In Proc. IEEE INFOCOM'99,* March 1999.
21. V. Paxson, "Measurements and analysis of end-to-end Internet dynamics," *Ph.D. Dissertation,* U.C. Berkeley, 1997.
22. V. Paxson, J. Mahdavi, A. Adams, and M. Mathis, "An architecture for large-scale Internet measurement," *IEEE Commun. Magazine,* pp. 48–54, Aug. 1998.
23. S. Seshan, M. Stemm, and R. Katz, "SPAND: shared passive network performance discovery," *in Proc. USENIX'97,* 1997.
24. M. Yajnik, J. Kurose, and D. Towsley "Packet loss correlation in the MBone multicast network," *Technical Report UMASS CMPSCI 96-32,* Dept. of Computer Science, Univ. of Mass. at Amherst, 1996.