Improving the QoS of Web Services via Client-Based Load Distribution

Alan Massaru Nakai[1], **Edmundo Madeira**[1], **and Luiz E. Buzato**[1]

[1]Institute of Computing – University of Campinas (Unicamp)
Campinas, Brazil

{amnakai,edmundo,buzato}@ic.unicamp.br

Abstract. *The replication of a web service over geographically distributed locations can improve the QoS perceived by its clients. An important issue in such a deployment is the efficiency of the policy applied to distribute client requests among the replicas. In this paper, we propose a new approach for client-based load distribution that adaptively changes the fraction of load each client submits to each service replica to try to minimize overall response times. Our results show that the proposed strategy can achieve better response times than algorithms that eagerly try to choose the best replica for each client.*

1. Introduction

The replication of a web service over geographically distributed locations can improve the QoS perceived by its clients. An important issue in such a deployment is the efficiency of the policy applied to distribute client requests among the replicas.

Most of the load distribution solutions proposed so far are based on mechanisms that redirect client requests via DNS [Colajanni et al. 1998, Barroso et al. 2003, Pan et al. 2003, Yokota et al. 2004, Moon and Kim 2005, Su et al. 2006, Nakai et al. 2009a] or proxy web servers [Cardellini et al. 2003, Chatterjee et al. 2005, Ranjan and Knightly 2008, Pathan and Buyya 2009]. Client-based solutions have been neglected because of their lack of transparency to the clients (a.k.a. web browsers). However, the web is becoming the universal support for applications that support previously unforeseen usage scenarios. For example, it is now common to find smart clients that can generate interactions in a programmatic way (e.g. SOA). In scenarios like this, where transparency is not an obstacle, client-based load distribution becomes an interesting alternative.

A main advantage of this kind of solution is the fact that the client can monitor end-to-end network latencies (response times) in a better way than the DNS and/or web servers. In some cases, a mechanism for client-side server selection may be the only option available. For example, a client may want to distribute their requests among a set of equivalent web services that do not belong to the same provider. In this case, server-side solutions are not applicable.

The client-side server selection techniques found in the literature can be divided into two groups: those that equitably distribute the requests among the distributed web services, and those that try to choose the best replica to which a client should send all requests. In the latter case, the focus has been the criteria used for decision making (e.g. latency, bandwidth, and best response time), and the way the criteria are applied (e.g. using historical data and/or dynamic probing of servers).

Although previous work has studied the efficiency of the server selection mechanisms for a single client, they do not have assessed the effect of these mechanisms for the whole system. In this work, we have simulated scenarios where several clients, generating different workloads, access replicas of a web service distributed world wide. In these simulations, we assessed server selection policies that are representative examples of the two groups of solutions. Our experiments indicate that the two types of solutions led the system to load-unbalanced states and, consequently, to the worsening of the response times perceived by the clients.

With respect to this problem, we propose a new approach for client-based server selection that adaptively assigns different selection probabilities to each server regarding network latencies and end-to-end response times. Our results show that the proposed strategy can achieve better response times than algorithms that eagerly try to choose the best replica for each client.

The main contributions of this paper are: (i) the evaluation of client-based server selection policies in scenarios where several clients use the same policy; and (ii) the proposal of a new solution that outperforms existing ones by dynamically adapting the fraction of load each client submits to each server.

The remaining of the text is organized as follows. Section 2 discusses related work. Section 3 presents our policy. Section 4 describes our simulations and Section 5 shows the results. Finally, in Section 6 we present our final remarks and future work.

2. Related Work

Dikes et al. [Dykes et al. 2000] present a comparison of 6 client-side server selection policies for the retrieval of objects via the Internet: **Random**, **Latency**, **BW**, **Probe**, **ProbeBW**, and **ProbeBW2**. **Random** selects a server randomly (equitably distributes the load). **Latency** and **BW** select the server that offers the best historical network latency and bandwidth, respectively. **Probe** probes the network latency of all servers (via ping) and selects the first to reply. **ProbeBW** considers only n servers with the fastest historical bandwidths and applies Probe to them. **ProbeBW2** probes the network latency of all servers and applies BW to the first n servers to reply. In summary, the results showed that the policies based on dynamic network probes presented best service response times. A similar result was presented by Conti, Gregori, and Panzieri in [Conti et al. 2001].

In the papers [Conti et al. 2005a] and [Conti et al. 2005b], Conti, Gregori and Lapenna compare a probe-based policy to a parallel solution. In the parallel solution, the total amount of data to be retrieved is divided in equal fixed-size blocks. Then, one block of data is requested to each service replica. The authors argue that this strategy can be useful when: (i) the objects to be downloaded are large; (ii) the client operations are read-only; (iii) the client wants better throughput; and (iv) the performance of the service replicas is not the bottleneck of the system.

Mendonça et al. [Mendonça et al. 2008] compared 5 client-side server selection policies for SOAP web services: **Random**, **HTTPPing**, **Parallel Invocation**, **Best Last**, and **Best Median**. Random and HTTPing are equivalent to Random and Probe from [Dykes et al. 2000]. In Parallel Invocation, the client requests all servers in parallel and waits for the fastest response. Best Last selects the server with the best recent response time. Best Median selects the server with the best median response time. According to the authors, Best Last and Best Median are the best choice in most of cases.

The main difference between our work and the previous ones is that we are concerned with the effect of the server selection policy in scenarios where several clients use the same policy. We show that representative examples of the policies mentioned before present better or worse performances depending on the system condition. Therefore, we propose a new adaptive solution that outperforms the others by producing best response times.

3. Adaptive Server Selection

The server selection problem can be defined as follows. Let P_i be the probability of a client to send a request to the server i, with $1 \leq i \leq n$ and $\sum_{i=1}^{n} P_i = 1$. The problem is to find P_i that offers the best QoS. As we can see, all solutions mentioned before belong to one of the two types:

1. $P_i = 1/n$: Equitably distributes the load among n servers;
2. $\begin{cases} P_i = 1, \text{ if } i = k \\ P_i = 0, \text{ if } i \neq k \end{cases}$: Where k is selected according to some policy criterion (latency, bandwidth, response times, etc.).

Examples of server selection policies of the first type are: Round Robin, Random, and Parallel Invocation. The positive points of this type of policies are: (i) they are simple and do not require the client to gather server information; and (ii) they distribute client requests among all servers, thus reducing the possibility of server overload. Figure 1 (i) shows a scenario that is favorable to these policies. In this scenario, the equitable distribution helps not to overload any server. A negative point is that, since these policies do not consider any server information (e.g. latency, response times), they send requests to "worst" servers with the same probability as they send them to the "best" servers. Figure 1 (ii) illustrates an unfavorable scenario, where servers 3 and 4 do not have enough capacity to serve the incoming load and, thus, become overloaded.

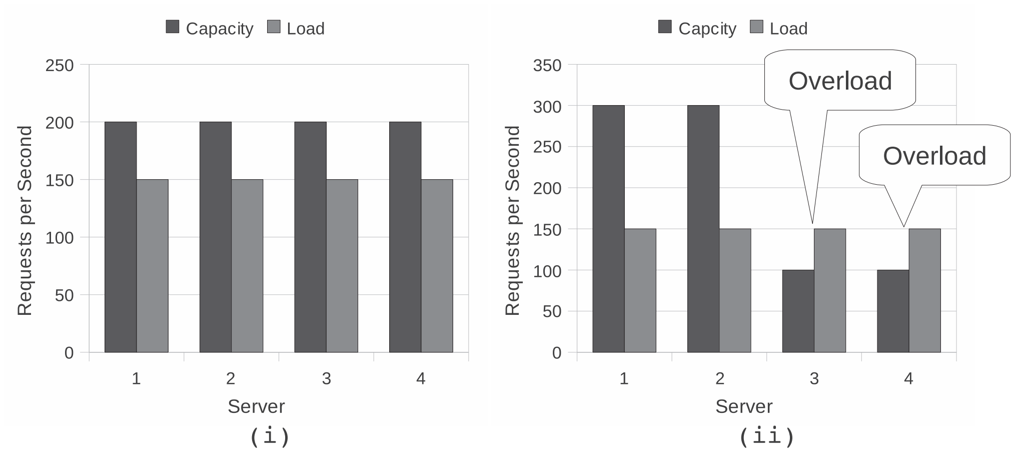

Figure 1. Equitably distribution server selection. (i) Favorable scenario. (ii) Unfavourable scenario.

The second type includes those server selection policies that use some criterion to select the best server. The advantage of this kind of solution is that it allows the client to perceive changes in the responsiveness of the selected server and react to these changes. These polices can offer good performance if the aggregated workload assigned to the "best server" does not exceed its capacity. Otherwise, the clients may start chasing for the best server and this leads another server to become overloaded. This situation is illustrated in Figure 2. In a time $t1$ (Figure 2(i)) , clients select server 1, that becomes overloaded. Once the clients perceive the changes in the responsiveness of server 1, they select server 2 (Figure 2(ii)), and then, it also becomes overloaded.

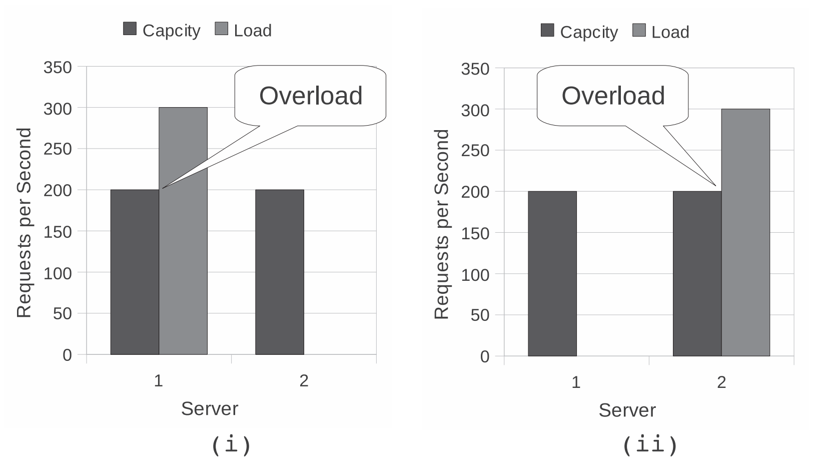

Figure 2. Best server. (i) t1: best server overloaded. (ii) t2: Chase for the best server effect.

Considering the advantages and problems of these two types of solutions, we propose a new approach that dynamically adjusts Pi in order to minimize response times by avoiding the overloading of the servers. Our hypothesis is simple: if clients cooperatively balance the load of the system by distributing requests in a smart way, all of them can benefit from that.

In our solution, as well as in the Probe policy ([Dykes et al. 2000]), the clients probe the network latencies of the servers in a predefined time interval. Each client maintains a list of servers that is sorted according to their network latencies. Here, we assume that, in normal conditions, the response times of nearby servers tend to be better than the response times of farther ones. The main idea of our solution is to find the highest selection probability the client can assign to each server, giving higher priorities to the nearest ones, without overload them. In order to accomplish this objective, we propose the heuristic described in the following.

The pseudo-code in Figure 3 describes our heuristic. At the beginning of its execution, a client sets $Pi = 1/n$, as in the Random policy. It sends requests to the servers according to P_i and maintains a sliding-window mean of the response times of all servers (MRT) updated (lines 11 to 17). Another process monitors MRT (lines 20 to 22). Whenever the mean of server i (MRT_i) does not exceeds MRT_j, $i+1 \leq j \leq n$, the probability to select the server i is increased in a predefined amount INC (lines 27 to 30). In this case, INC is subtracted from P_j in a direct proportion to MRT_j. If $MRT_i > MRT_j$ or if a request to server i fails, then the client assumes that server i is overloaded and performs a contention action. It decreases P_i in a predefined amount DEC, which is added to P_j in an inverse proportion to MRT_j (lines 32 to 35). DEC should be very much higher than

```
Definitions:
S: set of servers, ordered by network latencies
Si: ith server
Pi: probability of selecting the server i
MRTi: mean response time of the server i
INC: probability increment
DEC: probability decrement
t_UPDATE: time between probability updates

#CLIENT
On each request:
  SELECT server Sk, according to Pi (1 <= i <= n)
  status <- SEND request to Sk
IF status == SUCCESS
  Update MRTk
ELSE
  Alarm[k] <- TRUE

#MONITOR
Continuously:
  IF MRTi (1 <= i <= n-1) > MRTj (i+1 <= j <= n)
    Alarm[i] <- TRUE

#PROBABILITY UPDATE
On each t_UPDATE seconds:
  FOR i<-1:(n-1)
    IF alarm[i] == FALSE
      Pi <- Pi + INC
      Subtract INC from Pj (i+1 <= j <= n)
                   in direct proportion to MRTj
    ELSE
      Pi <- Pi - DEC
      Add DEC to Pj (i+1 <= j <= n)
              in inverse proportion to MRTj
      Alarm[i] <- FALSE
```

Figure 3. Heuristic for adaptive server selection probabilities.

INC, in order to grant that, once a problem in the responsiveness of server i is detected, the aggregated load sent to server i is reduced to less than its capacity.

It is important to note that, in our approach, clients cooperate to maintain the load balancing of the system in a very indirect way. There are no interactions among clients or between clients and servers to exchange load and/or state information. The only information each client has, as in the other solutions, is the response times produced by the servers. Using this information, a client can avoid overloading the "best" servers by willingly sending fractions of its requests to other servers. This happens in isolation from the other clients. However, since all clients perform in the same way, the overall system load stands balanced. Considering that the clients send as much requests as possible to better servers and the severs are not overloaded, the system can provides very good response times.

4. Methodology

In order to evaluate our solution, we have implemented a simulator using the CSIM for Java[1], a discrete event simulator framework. In the following, we describe how we have simulated the web services and their clients, the topology of the simulations, and the configuration parameters.

4.1. Load Generation and Web Servers

We used the PackMime Internet traffic model [Cao et al. 2004, pac] to generate HTTP traffic in our simulations. PackMime has been obtained from a large-scale empirical study of real web traffic and has been implemented in the *ns-2*[2], a well known network simulator. In order to use the model in our simulations, we have implemented a Java version of the PackMime.

PackMime allows the generation of both HTTP/1.0 and HTTP/1.1 traffic. The intensity of the traffic is controlled by the rate parameter, which is the average number of new connections started per second. The implementation provides a set of random variable generators that drive the traffic generation. Each random variable follows a specific distribution. The distribution families and the parameters used in PackMime are described in [Cao et al. 2004]. In our simulations, we used the following random variables:

- **PackMimeHTTPFlowArrive**: interarrival time between consecutive connections;
- **PackMimeHTTPNumberPages**: number of pages requested in the same connection (if using HTTP/1.1);
- **PackMimeHTTPObjsPerPage**: number of objects embedded in a page;
- **PackMimeHTTPFileSize**: sizes of files (pages and objects);
- **PackMimeHTTPTimeBtwnPages**: gap between page requests;
- **PackMimeHTTPTimeBtwnObjs**: gap between object requests;

We assumed that each geographically distributed replica of the web server is composed of a cluster of servers. Each server is simulated as a queueing system with fixed service time of 10 ms. Thus, a cluster with k servers provides a capacity of k*100 requests/second. The request interarrival time distribution is defined by the PackMime model [Cao et al. 2004, pac]. Figure 4 illustrates the workload generating 210 connections per second. Note that, since each connection generates more than one HTTP request, the number of requests per second reach peaks of 380. It is important to note that, although the service time is fixed, the response time perceived by the clients also includes the server queue time. Therefore, different loads do affect the entire response time.

In order to divide the load between different clients, we adopted a solution similar to the one proposed by [Colajanni and Yu 2002]. In this paper, the authors propose to divide clients (in this case, end customers) into domains according to the Zipf's distribution,

[1]http://www.mesquite.com/
[2]http://www.isi.edu/nsnam/ns/

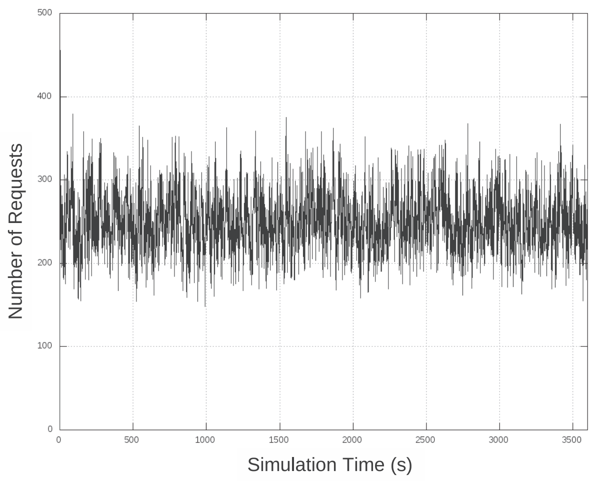

Figure 4. Example of workload generated by Packmime.

where the probability of a client to belong to the ith domain is proportional to $1/i^x$. This solution was motivated by previous work that demonstrate that if one ranks the popularity of client domains by the frequency of their accesses to a Web site, the size of each domain is a function with a large head and a very long tail. In our simulation, the total aggregate load is divided between the clients according to the Zipf's distribution. We can also vary the skewness of the distribution by changing the exponent x of the function.

4.2. Internet Latencies

We have considered a scenario with six replicas of the web server that are world wide distributed (Figure 5): one in South America (S1), one in North America (N1), two in Europe (E1 and E2), and two in Asia (A1 and A2).

We used the average of the latencies (ping RTT/2) measured on real hosts of PlanetLab[3] in Brazil, USA, Belgium, Austria, Japan, and China, to simulate the latencies among the replicated web servers. Table 1 shows the measured RTTs. We also consider that each replica serves a region and that the latency between a replica and a client of its region is 10ms.

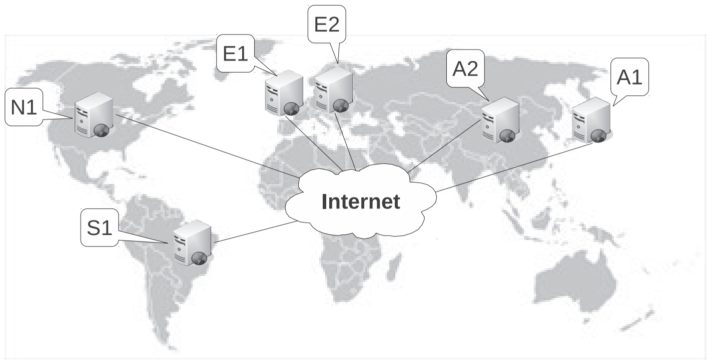

Figure 5. Web Server Replicas.

[3]http://www.planet-lab.org

Table 1. Table of latencies among the replicas (in milliseconds).

	S1	N1	E1	E2	A1
N1	89				
E1	138	48			
E2	140	58	18		
A1	193	109	151	162	
A2	272	156	114	122	68

Although we used fixed latencies in our simulations, their magnitudes are real and we believe that small variations that do not affect the magnitude of the latencies would not affect our results. Since we consider the entire response time (network + service) to make decisions in the policies, from the client's perspective, it does not matter if the differences in response time are caused by network latency variations or due to variable server workload. Thus, we believe that, varying network latencies would not change the overall results.

In order to consider the latency of the TCP protocol, we adopted the analytic model proposed by Cardwell et al. ([Cardwell et al. 2000]). This work extended previous models for TCP steady-state by deriving models for two other aspects that can dominate TCP latency: the connection establishment three-way handshake and the TCP slow start [RFC793 1981]. Therefore, the model proposed by Cardwell et al. can predict the performance of both short and long TCP flows under varying rates of packet loss.

In practice, the TCP model estimates the time needed to transfer data from one endpoint to another in terms of: (i) the size of the data to be transferred; (ii) the network latency between the two endpoints; and (iii) the packet loss rate probability.

4.3. Configuration

We compared our solution (AD) with two other server selection policies:

- Round Robin (RR): Each client sends requests to all servers in a rotative way;
- Best Server (BS): Each client uses RR to probe all servers. The server that presents the best mean response time is selected. Next, the client keeps sending all requests to the selected server until its mean response time exceeds the mean response time of other server. In this case, the client starts probing again, in order to avoid using out-of-date mean response times.

In order to present the flexibility of our solution, we performed our experiments considering two scenarios, one that favors BS and another that favors RR. In the first, the total capacity of the servers was set to 1200 requests per second (rps) divided among the servers as follows: S1 = 100 rps, N1 = 300 rps, E1 = 200 rps, E2 = 300 rps, A1 = 200 rps, and A2 = 100 rps. The clients were configured to generate approximately 72% of the total capacity. In the second scenario, the total capacity was divided equitably among the servers and the aggregated load was set to approximately 90% of the total capacity.

The parameters used in the heuristic are shown in Table 2. Note that, since DEC needs to be sufficiently large to alleviate the load of an overloaded server, we adopted a DEC proportional to P_i.

Table 2. AD Simulation Parameters.

Parameter	Value	Description
INC	0.01	Probability increment.
DEC	$0.3P_i$	Probability decrement.
t_UPDATE	1s	Time between probability updates.
WSIZE	30 requests	Window size for response time slide mean.

5. Results

Figure 6 presents the mean response times for Scenario 1. This scenario is unfavorable for RR (Round Robin), that does not have any information about server capacity. Although the total capacity of the system is much higher than the load, the server capacities are heterogeneous. Two servers (A1 and E2) have capacities lower than the equitable fraction of load distributed by RR. This lead A1 and E2 to become overloaded, making their queues grow faster than others queues and, consequently, affecting the system response times.

The bad performance of RR is also the result of its lack of adaptability to the overloading of the servers. Also, RR does not benefit from the small network latencies offered by nearby servers, because it blindly distributes the requests among all servers.

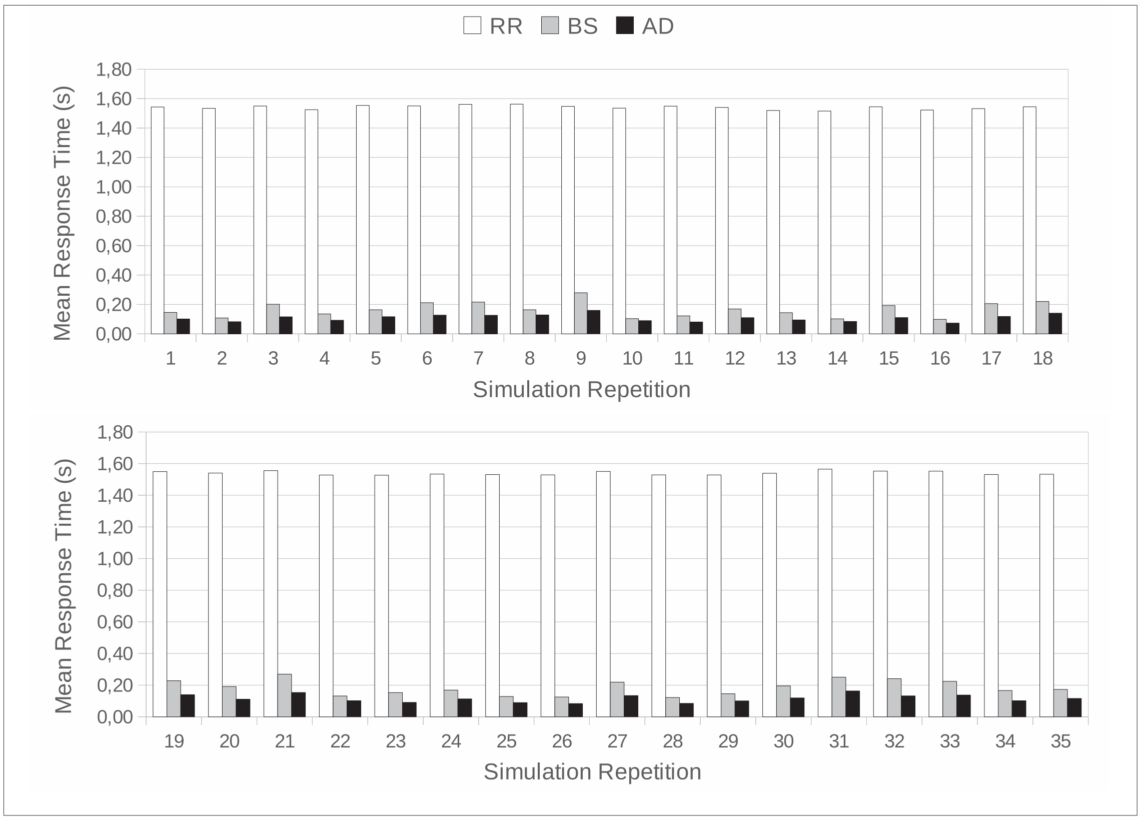

Figure 6. Mean response times for scenario 1.

For the sake of visualization, we replotted the mean response times of BS (Best Server) and AD (Adaptive) in Figure 7. As we can see, AD presented better response times in all simulations. In more than 85% of the simulations, AD presented mean response times more than 25% smaller than BS. In some cases, the difference was more than 40%.

This result was expected because, although BS tries to select the server with best response times, once the nearby server becomes overloaded, BS deviates the entire load of the client to farther servers that are lightly loaded. Although the best server decision is continuously re-evaluated by clients, the time it takes to change its decision is enough to overload the current chosen server. In the same situation, AD deviates only a fraction of the load, keeping as much requests as possible to be served by the nearby servers.

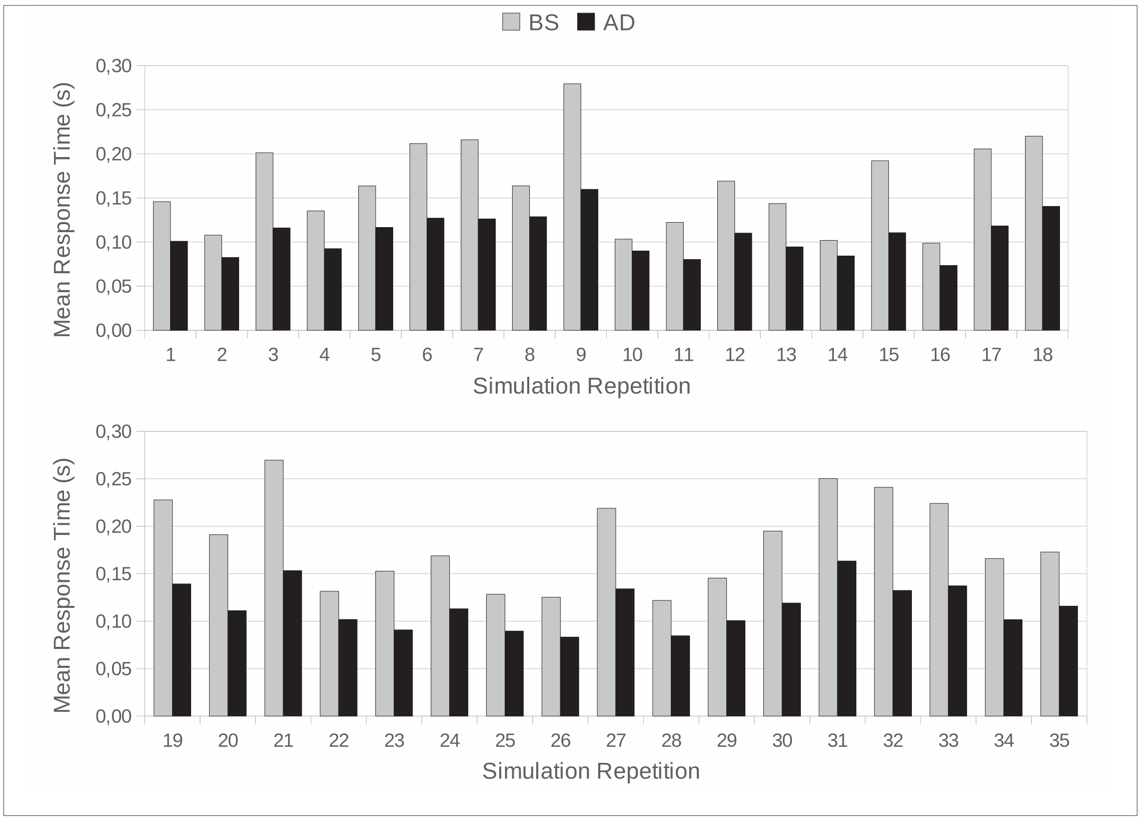

Figure 7. Mean response times for scenario 1.

The results obtained in scenario 1 may suggest that server selection policies that equitably distribute the load are useless if compared to policies that dynamically adapts to sever condition changes. However, the simulations of scenario 2 show that it is not a general rule.

In scenario 2, the system is almost saturated – the load corresponds to 90% of the capacity. Nevertheless, since the servers have similar capacities, RR's equitable distribution grants that no server becomes overloaded. On the other hand, the higher load amplifies the effect of chasing the best server implemented by BS. For this reason, BS presented the worst performance in this scenario, with mean response times 18% larger than RR and 48% larger than AD on the average.

Even though RR does not cause server overloading in this scenario, it still does not benefit from the small network latencies offered by nearby servers. This makes AD to perform better than RR, presenting mean response times more than 25% smaller than RR in more than 85% of the simulations.

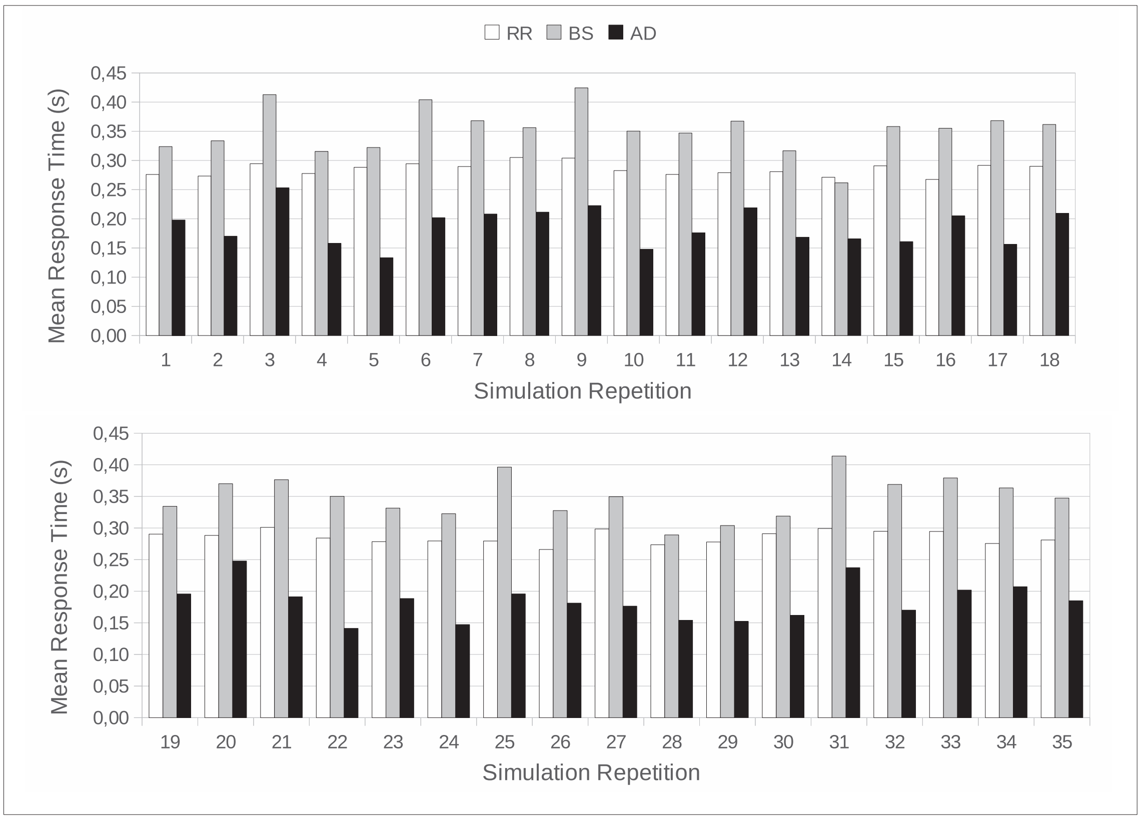

Figure 8. Mean response times for scenario 2.

While the first scenario is characterized by a lightly loaded system with heterogeneous servers, the second presents an almost saturated system with homogeneous servers. It is clear that, in the first case, due to its adaptability to server state changes, BS performed better than RR. In the second case, the equitable load distribution produced by RR outperformed BS's greedy strategy. Nevertheless, AD produced the best response times in both scenarios. This indicates that our solution successfully adapted to the system states while the other solutions did not. The results suggests that, in the considered scenarios, our hypothesis is valid.

6. Conclusion

A main advantage of client-side server selection policies is that clients can monitor end-to-end response times in a better way than server-side solutions. Besides, sometimes, client-side policies are the only option available. Most of the client-side policies proposed so far select one server to which the client should send all requests or equitably distribute the load among all of them.

Our simulations have shown that in scenarios where several clients use the same server selection policy, these two types of solution can lead to load-unbalanced states and, consequently, to the worsening of response times.

In this paper, we argue that if clients collaborate in order to balance server load they can obtain better response times. Our solution adaptively changes the fraction of load each client sends to each server giving higher priorities to nearby servers. Although this less greedy strategy of sending fractions of the load to worser servers seems to be counterintuitive, our experiments have shown that our solution overcomes the two types of policies proposed so far, even in scenarios that favors one type or another.

Future work include: (i) the evaluation of the sensitivity of our solution through an even more comprehensive set of simulations; and (ii) the deployment and evaluation of our solution in a real testbed [Nakai et al. 2009b].

7. Acknowledgements

The authors would like to thank FAPESP (n. 07/56423-6), CAPES, and CNPq for the financial support, and Schooner, Emulab, PlanetLab, and RNP (National Education and Research Network) for the infrastructure support.

References

Packmime-http: Web traffic generation in ns-2. http://www.cs.odu.edu/∼mweigle/research/netsim/packmime-nsdoc.pdf.

Barroso, L. A., Dean, J., and Hölzle, U. (2003). Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28.

Cao, J., Cleveland, W. Y. G., Jeffay, K., Smith, F., and Weigle, M. (2004). Stochastic models for generating synthetic http source traffic. In *Proceedings of INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*.

Cardellini, V., Colajanni, M., and Yu, P. S. (2003). Request redirection algorithms for distributed web systems. *IEEE Trans. Parallel Distrib. Syst.*, 14(4):355–368.

Cardwell, N., Savage, S., and Anderson, T. (2000). Modeling tcp latency. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1742 –1751.

Chatterjee, D., Tari, Z., and Zomaya, A. Y. (2005). A task-based adaptive ttl approach for web server load balancing. In *Proceedings. 10th IEEE Symposium on Computers and Communications, 2005. ISCC 2005.*, pages 877–884. IEEE Computer Society.

Colajanni, M. and Yu, P. S. (2002). A performance study of robust load sharing strategies for distributed heterogeneous web server systems. *IEEE Transactions on Knowledge and Data Engineering*, 14(2):398–414.

Colajanni, M., Yu, P. S., and Dias, D. (1998). Analysis of task assignment policies in scalable distributed web-server systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(6):585–600.

Conti, M., Gregori, E., and Lapenna, W. (2005a). Client-side content delivery policies in replicated web services: parallel access versus single server approach. *Perform. Eval.*, 59(2+3):137–157.

Conti, M., Gregori, E., and Lapenna, W. (2005b). Content delivery policies in replicated web services: Client-side vs. server-side. *Cluster Computing*, 8(1):47–60.

Conti, M., Gregori, E., and Panzieri, F. (2001). Qos-based architectures for geographically replicated web servers. *Cluster Computing*, 4(2):109–120.

Dykes, S., Robbins, K., and Jeffery, C. (2000). An empirical evaluation of client-side server selection algorithms. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1361 –1370 vol.3.

Mendonça, N. C., Silva, J. A. F., and Anido, R. O. (2008). Client-side selection of replicated web services: An empirical assessment. *The Journal of Systems and Software*, 81:1346–1363.

Moon, J.-B. and Kim, M. H. (2005). Dynamic load balancing method based on dns for distributed web systems. In *E-Commerce and Web Technologies. EC-Web*, volume 3590 of *Lecture Notes in Computer Science*, pages 238–247. Springer.

Nakai, A. M., Madeira, E., and Buzato, L. E. (2009a). DNS-based load balancing for web services. In *Proceedings of the 6th International Conference on Web Information Systems and Technologies*.

Nakai, A. M., Madeira, E., and Buzato, L. E. (2009b). Lab4WS: A testbed for web services. In *Proceedings of the 2nd IEEE International Workshop on Internet and Distributed Computing Systems (IDCS'09)*.

Pan, J., Hou, Y. T., and Li, B. (2003). An overview of dns-based server selections in content distribution networks. *Computer Networks*, 43(6):695–711.

Pathan, M. and Buyya, R. (2009). Resource discovery and request-redirection for dynamic load sharing in multi-provider peering content delivery networks. *J. Netw. Comput. Appl.*, 32(5):976–990.

Ranjan, S. and Knightly, E. (2008). High-performance resource allocation and request redirection algorithms for web clusters. *IEEE Trans. Parallel Distrib. Syst.*, 19(9):1186–1200.

RFC793 (1981). Transmission Control Protocol.

Su, A.-J., Choffnes, D. R., Kuzmanovic, A., and Bustamante, F. E. (2006). Drafting behind akamai (travelocity-based detouring). *SIGCOMM Comput. Commun. Rev.*, 36(4):435–446.

Yokota, H., Kimura, S., and Ebihara, Y. (2004). A proposal of dns-based adaptive load balancing method for mirror server systems and its implementation. In *Advanced Information Networking and Applications, 2004. AINA 2004. 18th International Conference on*, volume 2, pages 208 – 213 Vol.2.