Scaling Server Selection using a Multi-Broker Architecture

Mohamed Vall O. Mohamed-Salem, University of Montreal
Gregor von Bochmann, University of Ottawa
Johnny W. Wong, University of Waterloo

Abstract

Server replication is a common approach to improving the scalability of a service on the Internet. For this approach, the task of finding an appropriate server from a set of replicas is a critical issue. We have proposed in a previous work an architecture where a broker is used to provide server selection on a per session basis. When the number of servers and/or the number of clients becomes large, a single broker may not have sufficient capacity to handle the load. An extended architecture based on the replication of brokers is therefore considered. We first discuss alternative organizations that support access to multiple brokers and the needed cooperation between brokers in order to achieve server selection effectively. We then propose a server selection policy for our multiple broker architecture and evaluate its performance by simulation.

1. Introduction

Server replication is a common approach to improving the scalability of a service on the Internet. For this approach, the task of finding an appropriate server from a set of replicas is a critical issue. Several replicated service architectures have been proposed and investigated [2,5,9,10] and some of them are currently deployed on the Internet. In [3], we presented an architecture where an independent entity called "broker" is used to provide server selection on a per session basis. Our architecture allows flexible organization of resources used by web sites. The broker could be at the server side under the same authority as the replicated servers. This case is applicable for example to sites with heavy load and a 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.

When more servers are deployed to cope with increased user demand, the broker may become the system bottleneck. It may then be necessary to scale up the brokerage service by replicating the broker. In this paper, we extend our architecture in [3] to include replicated brokers. Each broker manages its own cluster of servers. The various brokers communicate and cooperate in order to optimize the use of the resources in their respective clusters. A client first locates one of the brokers, and then interacts with this broker for the selection of a server.

The brokers represent entry points to the system of servers. Each service has a unique public name, and for a given client, this public name resolves to the address of a specific broker. Methods to access one of the brokers include domain name system (DNS) [5], connection router [10] and network layer anycast [1]. An important consideration in inter-broker cooperation is the distribution of client sessions evenly among the servers regardless of which broker is accessed as the point of entry. Of interest to this study is the design and evaluation of server selection algorithms in our extended architecture.

This paper is organized as follows. In Section 2 we discuss issues that are related to supporting the inter-broker cooperation. In Section 3 we propose a server selection policy that is capable of selecting a server globally within all the clusters. Simulation results on the performance of our proposed policy are discussed in Section 4. Section 5 presents an overview of related work. Finally, Section 6 contains a summary of our findings.

Proceedings of the 23 rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03)
0-7695-1921-0/03 $17.00 © 2003 IEEE

2. Management of Multiple Brokers

In this section, issues related to the cooperation among a group of active broker are discussed.

2.1 Joining/Leaving a Group of Brokers

Each active broker keeps track of information on members of the same group. This includes address information and status of the server cluster. When a group is first created, the first member has only status information on its own cluster. A newly activated broker joins the group by first sending a query message to one of the active brokers in order to get the list of all other members, and then broadcasts a "join" message to these members. When a broker goes offline, it informs the rest of the group by sending them a "leave" message.

2.2 Reporting Performance Information

The objective of inter-broker cooperation is to select the best server globally for a client session. To do so, brokers must be able to report the status of their clusters to each other. The following alternatives may be considered:

- Continuous reporting: A summary of the status information is periodically transmitted to all the members of the group.
- Explicit status request: Status information is sent only when explicitly requested by a broker.
- Selective continuous reporting: Status information is sent periodically to interested brokers only. A broker expresses interest by sending a subscription message to other members of the group.

2.3 Server Selection

Each broker implements a server selection algorithm within its own cluster. Examples algorithms include round robin, least active sessions and least utilization (see [3] for details of these algorithms). Different brokers may implement different algorithms. With multiple brokers, each broker has to decide when it should limit the scope of server selection to its own cluster and when it should start looking for better alternatives in other clusters. In the latter case, the broker identifies the most suitable cluster, and sends a server selection request to this remote broker. Upon receiving a response from the remote broker, the address of the selected server is forwarded to the client. If the remote broker rejects the request, the broker may repeat the same process with other brokers.

3. A Global Server Selection Method

In this section we presents a global server selection method for our replicated broker architecture. We assume that all brokers implement the same server selection policy, referred to as Global Least Utilized (GLU). This is an extension of the Least Utilized (LU) policy in [3]; it selects the server that has the lowest utilization.

As discussed in Section 2, each broker is able to gather status information on other clusters in order to build a global view of the status of the system. This global view allows overloaded brokers to identify potential candidate brokers (or clusters) to which new client requests can be directed. For this purpose, we define two zones of operation for each cluster: green and red (see Figure 1). Green means that the cluster is operating with an acceptable level of performance and can handle the incoming load. On the other hand, red means that the cluster is under heavy load, and requests for new client sessions should be redirected to other clusters.

The boundary between the two zones is defined by a threshold based on the average utilization of servers within the cluster (denoted by u). At any instant in time, each cluster is considered to be in one and only one of two sets based on its state: Green (GS) if $u \leq threshold$ and Red (RS) otherwise. Each broker keeps track of the membership of GS and RS, based on status information received.

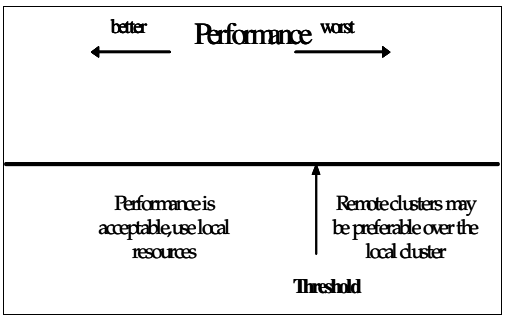

Figure 1: State of a cluster

We distinguish between two variants of our global least utilized policy (GLU), depending on whether the threshold is statically set or dynamically adjusted. We will refer to these two variants by static GLU (SGLU) and dynamic GLU (DGLU) respectively. SGLU uses a predefined threshold that is common to all clusters. Its performance depends largely on the choice of this threshold. DGLU, on the other hand, uses a dynamic threshold that depends on the status of the system and hence may not be the same at the different clusters at any instant of time. This threshold is computed as follows. Each broker periodically computes the average utilization over all clusters using the latest status information received. The resulting value is used as its threshold. With this approach, a cluster with utilization lower than the overall average will marks itself as green. Otherwise, the cluster will mark itself as red and start dispatching new requests to its peers.

The key step used by a broker, say broker k, in our GLU server selection algorithm can now be defined as follows:

if *cluster* $k \in \{GS\}$, use LU to select a server from the local cluster,

else if *GS* not empty, request the best broker in *GS* to select a server

If a server cannot be found, the client request is rejected.

4. Performance Evaluation

4.1 Simulation Model

In this section, we use simulation to evaluate the performance of our global server selection algorithm. Our simulation model consists of K brokers, each of which manages a cluster of N servers. Each cluster is situated at a different region. The load on the system is generated by a fixed number M of concurrent sessions. When a session is newly created, it has probability P_i of being from region i and uses the broker of that region as an entry point to the server side. The session length follows an exponential distribution with a mean of 36 pages in accordance with the results presented in [7]. When a session ends, a new session is immediately created so that the total number of active sessions is maintained at M.

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 some 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. As in [6], two heavy-tailed distributions, namely Pareto and Weibull, are used to model the user think time, the number of objects per page, and the processing time between object requests in a same page. Each server is modeled by a capacity parameter measured by the time required by a server to process one byte of data [5]. For example, if a server has a capacity of 10^6 bytes/sec and the average size of an object is 10,000 bytes, then the server can process on average 100 objects per second.

4.2 Simulation Experiments

In our simulation experiments, each cluster has two servers; each has a capacity of 10^6 bytes/sec. The broker uses the LU algorithm as described in [3] for server selection within its own cluster. Two levels of load are simulated: heavy load and moderate load. For the heavy load case, M = 2000 which yields an average server utilization of approximately 96%. The corresponding values for moderate load are M = 1500 and 79% respectively. A configuration with three clusters is considered. For each newly created session, the probability P_i that the session is in region i is set as follows: $P_1 = 0.2$, $P_2 = 0.3$, and $P_3 = 0.5$.

4.3 Simulation Results

In [3], it was shown that LU is capable of realizing good load balancing among servers of different capacities and under different load conditions. In this paper, our objective is to investigate the feasibility and performance of using a similar algorithm for global load balancing in a multi-broker architecture.

We first study the performance of the two variants of GLU algorithm, namely SGLU and DGLU. Figure 2 shows the response time percentiles achieved by SGLU under heavy load

conditions for four values of the threshold: 65%, 75%, 85%, and 95%. As mentioned previously, when the average server utilization exceeds this threshold, a broker starts dispatching sessions to other brokers and refuses incoming requests from these brokers.

For comparison purposes, we show in Figure 2 a curve labeled "one cluster", which is the performance achieved when all servers are included in a single cluster. This represents the ideal performance for global server selection. We also show a curve labeled "independent clusters", which is the performance achieved in the absence of any cooperation between the clusters. This represents the worst case that can be achieved by a global policy.

The results in Figure 2 lead to the following observations. For small values of the threshold, each cluster reaches the red zone quickly, leaving little room to accommodate requests from other clusters. The cooperation between brokers is then very limited and a large number of requests are served in their original regions with whatever QoS available. As the threshold increases towards values that are close to the offered load, the overall QoS improves steadily. For example, the percentile of requests completed in less than 1.5 seconds with thresholds 95%, 85%, 75% and 65% are 88%, 69%, 59% and 56% respectively. The corresponding value for the worst case of no cooperation among brokers is 50%.

Similar results were obtained for the case of moderate load conditions. These results are reported in [14].

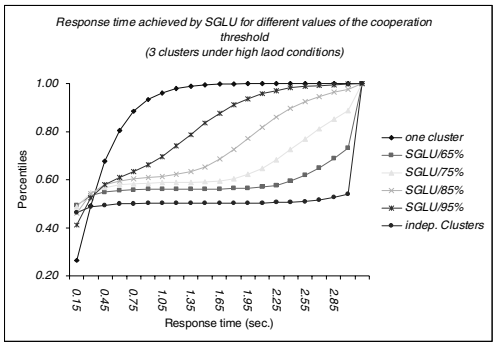

Figure 2: SGLU in high load conditions

The results in Figure 2 lead to two key observations: (1) Cooperation between brokers is beneficial when one considers global load sharing; and (2) Choosing an appropriate threshold for SGLU may be difficult because it is affected by the load on the overall system. Observation (2) has motivated the design of DGLU. In Figure 3, we show the results for DGLU under heavy load conditions. As discussed in Section 3, DGLU computes the threshold dynamically using performance data exchanged periodically between brokers. Three different values for the length of the reporting interval are used in our simulation. They are 11, 51, and 101 seconds respectively.

We observe from the results in Figure 3 that DGLU is superior to SGLU, and yields performance results very close to ideal case of a single cluster. The main advantage of DGLU is the automatic adjustment of the threshold. Our results also show that the performance of DGLU is not sensitive to the length of reporting intervals, for the three values considered.

Once again, the results for the case of moderate load conditions are similar (see [14] for details).

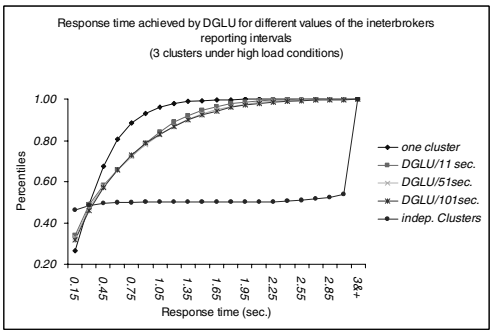

Figure 3: DGLU in high load conditions

5. Related Work

Several architectures have been proposed in the literature for the management and realization of scalable services [2,3,5,9,10]. In all cases, the issue of scalability of the server selection process was addressed. An early work on this issue was based on the use of an anycast service at the network layer for service discovery and selection [1]. The objective is to allow an application to send a request to potentially multiple recipients in order to be connected with one and preferably

just one of these recipients only. This issue of scalability of anycast was investigated in [11,12]. An application layer anycast paradigm has also been proposed for the localization of servers across the Internet [2].

A two-level server assignment approach was proposed in [8] in which DNS performs a first level dispatching of clients to servers, and if necessary an overloaded server performs a redirection to other servers. In [4], a three-level architecture based on DNS was also proposed.

In all the above proposals, the server assignment process, including the redirection by servers, is done on a per connection (or request) basis. In our architecture [3], however, server selection is performed at the session level where each client is assigned to a server for the duration of its session. Control information is centralized at brokers and servers do not need to be involved. Our approach necessitates some modification to client software. However, all information necessary for server selection decision is kept at the server side and at the brokers. This gives service providers the flexibility to implement server selection policies that suit their specific needs.

6. Conclusions

In this paper we have investigated the scalability of the server selection process based on an extension of the architecture that we have proposed previously [3]. This extended architecture consists of multiple brokers where each broker manages a server cluster and cooperates with other brokers to achieve global load balancing. We have also defined a group management protocol that allows multiple brokers to cooperate. This protocol provides support for membership management, exchange of performance data, and an inter-broker negotiation.

We have developed a global server selection policy that can be used by a set of cooperating brokers. This policy is based on the definition of a performance threshold. When this threshold is exceeded at a cluster, new requests for server selection are sent to other clusters. Two variants of this policy, depending on whether the threshold is set statically or adjusted dynamically, have been evaluated by simulation. It was found that that both variants yield a significant performance improvement when compared to the case of totally independent clusters. We also observed that the dynamic variant has by far better performance than its static counterpart. This is mainly due to the fact that the best value for the threshold depends largely on the load on the system.

Our architecture is quite flexible; it allows clients to select servers by specifying their own QoS requirements. Our algorithm can also be adapted easily to include different classes of clients. In this case, the global selection may be limited to a certain class of clients and a cluster may serve specific classes only [13].

References

[1] C. Patridge, T. Mendez and W. Milliken, “Host Anycasting Service”, Internet RFC 1546, 1993.

[2] S. Bhattacharjee, M.H. Ammar, E.W. Zegura, V. Shah and Z. Fei, “Application Layer Anycasting”. Proc. IEEE INFOCOM 97, 1997.

[3] Mohamed-Vall M. Salem, J.W. Wong and G.v. Bochmann, “A Scalable Load-Sharing Architecture for Distributed Applications”, Proc. SoftCom '2001, Split, Croatia.

[4] V. Cardellini, M. Colajanni and P.S. Yu, “Geographic Load Balancing for Scalable Distributed Web Systems", Proc. Eighth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, San Francisco, 2000.

[5] V. Cardellini, M. Colajanni and P.S. Yu, “DNS Dispatching Algorithms with State Estimators for Scalable Web-Server Clusters”', World Wide Web Journal, Vol. 2, No. 3, pp. 101-113, 1999.

[6] P. Barford and M. Crovella, “Generating Representative Web Workloads for Network and Server Performance Evaluation”, Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pp. 151-160, 1998.

[7] L. Cherkasova, and P. Phaal, “Session Based Admission Control: a Mechanism for Improving Performance of Commercial Web Sites”, Proc. 7th International Workshop on Quality of Service, London, 1999.

[8] D. Andresen, T. Yang, V. Holmedahl, and O. Ibarra, “SWEB: Towards a Scalable

[9] World Wide Web Server on Multi-computers", Proc. 10th International Parallel Processing Symposium, Hawaii, 1996.

[10] E.D. Katz, M. Butler and R.A. McGrath, "A Scalable HTTP Server: The NCSA Prototype", Computer Networks and ISDN systems, volume 27, pp. 155-164, 1994.

[11] Cisco Distributed Director, available at http://www.cisco.com/warp/public/cc/pd/cxsr/dd/index.shtml.

[12] R. Engel, V. Peris, E. Basturk, V. Peris and D. Saha, "Using IP Anycast for Load Distribution and Server Location", Proc. Third Global Internet Mini-Conference in conjunction with Globecom '98, 1998.

[13] D. Katabi and J. Wroclawski, "A Framework for Global IP-Anycast (GIA)", Proc. ACM SIGCOMM 2000, Stockholm, Sweden, 2000.

[14] Mohamed-Vall M. Salem, G.v. Bochmann and J.W. Wong, "Server Selection for Differentiated Classes of Users", Proc. 2002 International Symposium on Performance Evaluation of Computer and Telecommunication Systems, San Diego.

[15] Mohamed-Vall M. Salem: "Scalable Server Selection Using a Broker Node," Ph.D. Thesis, Université de Montréal, 2002.

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, the IBM Toronto Laboratory Centre for Advanced Studies, an IBM Faculty Partnership Award, and the Natural Sciences and Engineering Research Council of Canada under a cooperative research and development grant.