A Self-configuration Mechanism for High-Availability Clusters*

Hocheol Sung[1], Sunyoung Han[1,**], Bok-Gyu Joo[2], Chee-Wei Ang[3], Wang-Cho Cheng[3], and Kim-Sing Wong[3]

[1] Department of Computer Science and Engineering, Konkuk University, Korea
{bullyboy, syhan}@cclab.konkuk.ac.kr
[2] Department of Computer and Communications, Hongik University, Korea
bkjoo@hongik.ac.kr
[3] Networking Department, Institute for Infocomm Research, Singapore
{angcw, chengwc, wongks}@i2r.a-star.edu.sg

Abstract. Most high-availability (HA) solutions currently used are based on the pre-configuration done by human administrators. If the configuration between machines participating in the HA cluster can be automated, services clustered can be provided more efficiently. For realizing this concept, the server agent and service description server (SDS) are designed and implemented in this paper. The server agent exchanges several messages with other servers and SDS. SDS is the central server that manages information needed for each machine to do "self-configuration". We also implement a web-based monitoring tool to watch the status of the overall system.

1 Introduction

Most high-availability (HA) solutions depend on human administrators to do "pre-configuration" [1][2]. For example, in the Novell clustering solution, administrators have to assign available secondary servers to a primary server. For this reason, the performance of the overall system depends on the expertise of the administrators and administrators' error can reduce overall system reliability. Moreover, in mobile network environments, doing "pre-configuration" is very difficult because servers can move to another network [3]. Another problem caused by "pre-configuration" is the quality of services clustered. The state of the secondary server can be changed during the course of service provisioning. However, in the pre-configured cluster, the state of the secondary is not reflected in configuration of cluster.

Considering this point of view, in this paper, we propose the high-availability system that is able to self-configure. Each server joins a network looks for machines that can act as its secondary server and assigns the optimal machine to its secondary server dynamically. And each server must advertise its services to other machines in the network so that they can know "who can be my secondary server in this network".

* Research based on the EDEN2 project (a collaboration between Institute for Infocomm Research, Singapore and Konkuk University, Korea).
** Corresponding author.

V.N. Alexandrov et al. (Eds.): ICCS 2006, Part IV, LNCS 3994, pp. 260 – 263, 2006.

For these purpose (service discovery and advertisement), we newly introduce a central server called the Service Description Server (SDS) in this paper.

2 Operational Overview

Essentially, all participating servers including SDS need to exchange messages for self-configuration. First, when server 1 is up, it sends a registration message to SDS. Whenever receiving a registration message, SDS has to send a table update message to each server on the network. The table update message contains the list of server that can back up the primary service of the server 1 and each backup server's status information, "priority". For deciding "priority" of each server, SDS should receive a loadlevel message periodically from each server registered with SDS. After some times, server 2 is started, it will send a registration message to SDS and receive a table update message from SDS. After server 2 registered, server1 will receive another table update message from SDS. It may contain backup server information, if server 2 can be a secondary server for server 1's primary service. After getting the table update message, server1 will send an assignment message to server 2, telling server 2 to be its secondary server and listen to its keep-alive message. Similarly, when server 3 is started, server1 and server 2 will get table update message from SDS and assign secondary server for its own primary service. In this case, server 1 should choose its secondary server between server 2 and server 3 based on the status of each server. Of course, server 2 may assign server3 to its secondary server and send a keep-alive message to server 3 continuously.

In case, server 1 is down, server 2 that is secondary server will not receive a keep-alive message from server 1. After some times, server 2 will know server 1's down and send a takeover message to SDS informing SDS that it will run the service provided by server 1. As receiving a take over message, SDS will update its database and send a table update message to each server on the network. After getting the table update message form SDS, server 2 will assign another secondary server (server 3 in this scenario) for the service taken over and send a keep-alive message to server 3. After running for a while, server 1 is restarted and does registration with SDS. When server 1 receives a table update message from SDS, it will know who run its primary service and send a release service message to server 2 that took over the service from server 1 in previous scenario. When server 2 receives the release service message, it will stop the service that is the server 1's primary service and send a remove assignment message to server 3 informing server 3 of no need to listen to a keep-alive message from server 2. After getting acknowledgment from server 2, server 1 will starts the primary service and send a takeover message to SDS to update its database. After receiving the table update, server 1 will repeat the secondary assignment as mentioned earlier.

3 Experimental Results

For testing our implements, we set up test-bed in a local network as shown in Fig. 1. Although we use 5 Linux systems for our test, only 3 application servers such as the HTTP, FTP and EMAIL server participate in the self-configuration.

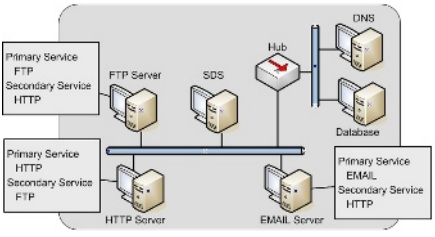

Fig. 1. Local test-bed

The database shown in Fig. 1 is for storing all the server necessary files such as configuration file, web page files and etc. Although not mentioned earlier, we also need a DNS server in local test-bed. Because most users use the FQDN to reach the server, each server has to register its name to the DNS. When the primary service is taken over by the secondary server, it should send dynamic update request to the DNS to point the service FQDN to the secondary server's IP address [4].

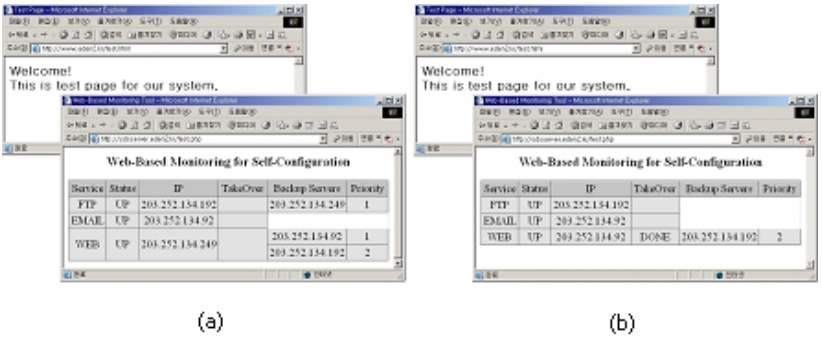

Fig. 2. (a) All application servers are normally in operation and (b) the HTTP server is down and service failover is initiated

Fig. 2 (a) shows when all application servers are registered and normally in operation. In this case, as the priority of the EMAIL server is higher than that of the FTP server (lower value is higher priority), the EMAIL server is assigned to be a secondary for HTTP service. Next, Fig. 2 (b) shows when the HTTP server is down and service failover is initiated. Because the HTTP server is down, the secondary server (the EMAIL server) for HTTP service is now providing the service and appoints another secondary server (the FTP server) for HTTP service. Finally, when the HTTP server recovers from fail and service failback is initiated, the secondary server releases the http server and revokes the assignment of another secondary server. Also, FTP server will be a secondary for HTTP server as shown in Fig 1 (a).

For evaluating the performance of the system, we simulated the actions of clusters and defined 2 types of models for this simulation, ***static clustering model*** and ***dynamic clustering model***. The static clustering model is for pre-configured clusters.

In this model, one of backup servers is assigned to a secondary server for all service time. The dynamic clustering model is for self-configured clusters. In this model, a primary server can select an optimal backup server as a secondary server at any time.

For this simulation, we configure the cluster with one primary server and 2 backup servers and run the service for 24 hours (simulation time). During the simulation, the primary server dies and recovers randomly according to its load and we calculate the response time for client's requests. From the client's point of view, the response time for service requests is the major metric for evaluating the service availability [5]. Fig.3 shows the simulation result. For static clustering model, as server's load average increase, the response time for service requests also increase. But, for dynamic clustering model, the response time for service requests doesn't increase.

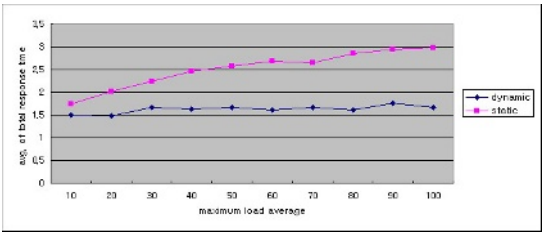

Fig. 3. Relation between each server's load average and response time for service requests

4 Conclusion

This paper presents a self-configuration concept for high-availability clustering system. Each machine participating in self-configuration can assign the secondary server for its primary service dynamically, based on the status of each backup server. As a result, when the primary server fails, its service can be automatically failover by the optimal backup server and the overall quality of service can be maintained well.

References

1. The High-Availability Linux Project, http://www.linux-ha.org
2. Novell, http://www.novell.com
3. Hocheol Sung, Sunyoung Han: Server Mobility using Domain Name System in Mobile IPv6 Networks, ICCS 2004 - LNCS3036, June 2004
4. Vixied (Ed.), P., Thomson, S., Rekhter, Y. and J. Bound: Dynamic Updates in the Domain Name System, RFC 2136, IETF, April 1997.
5. Enrique Vargas: High Availability Fundamentals, Sun BluePrints™ OnLine, November 2000.