Using Distributed Diagnosis to Deploy Highly-Available Web Servers

**Roverli P. Ziwich, Egon Hilgenstieler, Emerson F. F. Carara,
Elias P. Duarte Jr., Luis C. E. Bona**

Federal Uniersity of Paraná, Dept Informatics,
P.O. Box 19018 - 81531-980, Curitiba PR Brazil
{roverli,egon,emerson,elias,bona}@inf.ufpr.br

Abstract. *This work presents SAPOTI (Dependable TCP/IP Application Servers - in Portuguese: Servidores de APlicações cOnfiáveis Tcp/Ip), a distributed tool that guarantees the availability of TCP/IP application servers, in particular of Web servers. The tool is based on the SNMP framework and is executed on a group of Web servers running on a set of hosts that are monitored by a dependable distributed network management tool based on the hierarchical diagnosis algorithm Hi-ADSD with Timestamps. One server is elected to be responsible for the service. After this server becomes faulty and this event is diagnosed, the service is automatically recovered by electing another server among those that are fault-free. A priority scheme based on identifiers is defined. The service is available even if only one host/server is fault-free. Experiments are described obtained from a implementation of SAPOTI on a LAN with real Apache servers. Experiments that involved the injection of 210 faults distributed among a group of six servers were run, the measured availability was at least 97.3%. In another experiment with five servers, where 27 faults were injected, the availability was 99.5% during the whole experiment time.*

1. Introduction

Organizations and individuals have become increasingly dependent on the correct behavior of Web systems. It is thus important to guarantee the availability of these systems. This work presents SAPOTI (Dependable TCP/IP Application Servers – in Portuguese: *Servidores de APlicações cOnfiáveis Tcp/Ip*), a tool that allows the implementation of highly-available TCP/IP [1] (Transfer Control Protocol/Internet Protocol) application servers, more specifically, allowing the deployment of fault-tolerant Web servers. SAPOTI is implemented on top of a dependable distributed network monitoring system, that is based on hierarchical distributed diagnosis [2, 3]. The system employs algorithm *Hi-ADSD with Timestamps* (Hierarchical Distributed System-Level Diagnosis with Timestamps) [4] for fault-tolerant network resource monitoring.

The objective of distributed system-level diagnosis is to identify which system units are faulty and which are fault-free [5]. When diagnosis is distributed [5] each node has the ability to complete locally the diagnosis of the whole system. When the diagnosis is adaptive, the tests that each node performs are based on rounds and each round is achieved based on the previous rounds results. When the diagnosis is hierarchical, the nodes are grouped in clusters and in each testing round the size of the clusters increase. *Hi-ADSD with Timestamps* [4] is a hierarchical, adaptive and distributed system-level diagnosis algorithm.

SAPOTI employs the group abstraction in order to provide continuous, fault-tolerant service. A set of Web servers, called a server group, replicate the same content. The group employs passive replication [6], so that at any time one group member, called the primary, is responsible for serving all client requests. If the primary becomes faulty, after this event is diagnosed, the service is automatically recovered by the election of another primary among those group members that are fault-free. A priority scheme based on identifiers is defined. The service is available even if only one host/server is fault-free.

The management tool on top of which SAPOTI was implemented is based on the Simple Network Management Protocol (SNMP) protocol [7], the Internet standard framework for network management. A network management system built with SNMP consists of management entities that communicate using the management protocol. The framework includes the definition of a MIB (Management Information Base) [7], which is a collection of management objects that are useful for network monitoring and control.

Experiments were also performed and are described. The results show that in a network configuration with six nodes where 210 faults were injected distributed among all nodes, the availability of the service was at least 97.3%. In another network configuration with five nodes where some of these nodes became faulty 27 times, the availability of the Web server was at least 99.5%.

978-0-7695-3397-1/08 $25.00 © 2008 IEEE
DOI 10.1109/LA-WEB.2008.11

The rest of this work is organized as follows. Section 2 presents a general view of the dependable and distributed management tool based on hierarchical diagnosis. Section 3 details the architecture and implementation of SAPOTI. Experimental results are described in section 4. Section 5 presents related work and section 6 concludes the paper.

2. The Dependable and Distributed Network Management Tool

SAPOTI obtains diagnostic information from the afore mentioned dependable and distributed network management tool [2, 3]. This management tool implements algorithm *Hi-ADSD with Timestamps* [4], a hierarchical adaptive and distributed system-level diagnosis algorithm. A distributed system-level diagnosis algorithm allows the system's fault-free nodes to determine the state of all other nodes in the system [4, 5]. A diagnosis algorithm is called adaptive if the next tests are determined based on the previous tests results. When the diagnosis is hierarchical, nodes are grouped in clusters and in each testing round the size of the cluster increases. Initially tests are performed for clusters with size 1. The cluster size doubles in consecutive testing rounds. The maximum cluster size is $n/2$ where n is the number of nodes in the system. At each testing round, each node obtains diagnostic information about the whole cluster. It is assumed that fault-free nodes are able to execute tests in a reliable way and that the system is fully connected.

The management tool is implemented using SNMP. The SNMP standard defines entities that are traditionally called managers and agents. The agents make available management information through a MIB (Management Information Base) [7]. The managers are sets of user applications that can be constructed to perform tasks such as fault, configuration and accounting management.

The management tool consists of a set of agents. Each agent – also called a tester – keeps a *Test-MIB*. The *Test-MIB* keeps information about the tests that are executed by each corresponding agent. It also keeps information about the state of all testers and system components. The testers are identified by the management tool with sequential identifiers. The *Test-MIB* allows the configuration of secondary tests through the specification of procedures. These secondary tests are constructed specifically for each unity that must be tested.

A test procedure can be configured to monitor network services or devices. A network service is any process executing a tester and a device is any resource accessible through the network. When a secondary test is configured for a device and the corresponding tester becomes faulty, the algorithm assigns another tester to replace the faulty one. If the secondary test is employed for testing a service, the algorithm does not replace a faulty tester. In this case the state of the tested service is considered to be *unknown*.

Tests are executed periodically in pre-defined testing intervals. The *Test-MIB* allows this interval to be configured for each test. Each MIB keeps information about testing intervals and about the number of tests executed by each tester. The MIB also keeps statistics such as the mean time a managed object remains fault-free and the mean time a monitored object takes to be repaired after it becomes faulty.

3. SAPOTI: Tool Architecture

SAPOTI can be used to guarantee the high availability of TCP/IP applications, in particular of Web servers. The tool was implemented in Linux [8]. The Web server used was the Apache [9] server. SAPOTI is a distributed tool that is executed on a set of hosts that provide the dependable service. The tool requires the Web server to be installed in a group of hosts that together provide the dependable service.

SAPOTI uses the sequential identifiers assigned by the management tool. These identifiers are used to determine the priority of group members. The fault-free server with highest priority is responsible to provide the service. The lowest the identifier the highest the priority.

Each host running a group member must have a secondary test set up in its corresponding *Test-MIB*. This secondary test is employed to check the state of the Web server running in the node. This secondary test consists of a complete procedure that checks whether the Web server is fully available. From this test result and also using network diagnostic information obtained from the *Test-MIB*, SAPOTI configures the fault-free server that has the highest priority to serve Web requests.

As SAPOTI runs in a distributed way, all nodes executing the tool know the priority of the node that is currently providing the service. For the next steps, consider that a node i is executing the SAPOTI. If node i identifies that there are no other nodes with higher priority providing the service and if it is also not yet providing the service, then node i becomes the service provider. On the other hand, if node i is currently providing the service and it identifies that there is another node with higher priority that is also currently providing the service, node i stops its Web server. All diagnostic information required is obtained from the local *Test-MIB*. Thus, SAPOTI does not execute any additional network monitoring. As each node independently identifies if it should provide or not the service, the service remains available even if only one node running SAPOTI is fault-free.

SAPOTI employs a unique virtual IP address which is assigned to the host providing the service. This strategy allows the group of servers to be transparent to the client, i.e. the interface is identical to the interface of a single

server. A virtual IP address can be configured in Linux with the IP aliasing technique [10]. Besides the virtual IP address, the host also has its own IP address. Thus, in order for a machine to start providing the service it sets up the virtual IP address and starts the Apache Web server. The algorithm in pseudo-code is presented bellow.

```
The SAPOTI algorithm that runs on each node:

my_id := (local node id);
time_out := (seconds to wait for each
             iteration of the algorithm);
repeat forever
    server_id := (the lowest node id that is
                  currently providing the Web server);
    if (server_id > my_id) or (server_id is null)
    then
        if local node is not with the virtual IP up
        then bring the virtual IP interface up;
        if local node is not providing the Web server
        then start the Web server;
    else
        if local node is with the virtual IP up
        then take the virtual IP interface down;
        if local node is providing the Web server
        then stop the Web server;
    end_if;
    sleep time_out seconds;
end_repeat;
```

Figure 1 shows a group of four hosts running SAPOTI to implement a dependable Web server. In the figure the nodes with id 1 and 2 are faulty. As the node with id 3 has the highest priority among fault-free nodes, it sets up the virtual IP address and starts the Web server.

The nodes must keep the Web server files updated. This is done with the RSYNC [11] application. The RSYNC application synchronizes any change in the files content of a group of network hosts. Furthermore, the RSYNC application is efficient in the sense that it only transfers file changes. Another alternative to this task is use NFS (Network File System) [12] to share the files.

4. Experimental Results

In this section the results of two experiments are described. The experiments were executed to measure the recovery latency and the availability of a Web server in environments with a relatively high number of faults. In the first experiment 6 hosts were employed. The system was observed during an interval of 12 hours and 23 minutes. Table 1 shows the identifiers of the hosts used in the first experiment. Faults were injected so that the probability that a given host is fault-free state at a specific instant of time varied between 30% and 85%. A total service collapse was also simulated in this experiment, i.e. occasionally all hosts could become faulty, making the Web server completely unavailable.

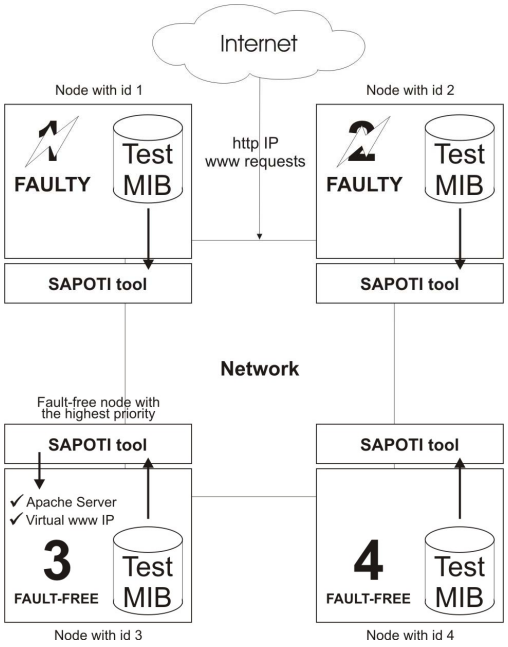

Figure 1: A group of four hosts running the SAPOTI.

In the second experiment 5 hosts were employed and the results were collected during an interval of 12 hours and 40 minutes. Table 2 shows the hosts' identifiers and how faults were injected. The fault configuration of this second experiment was specified so that there would be at least one fault-free machine during the whole experiment. Double consecutive events were also avoided – a host becomes faulty and immediately another host becomes fault-free – or vice-versa.

In both experiments a host fault was simulated by killing its SNMP agent process. This fault simulation was enough because the management tool is based on the SNMP protocol, i.e. without the SNMP agent the management tool cannot obtain status information about the machine, considering it as unavailable. The hosts were connected in an Ethernet 100 Mbps network using NFS. All hosts run Linux Debian [14] system and version 4.2.1 of the NET-SNMP [7]. An Apache Web server was executed on each host. In both experiments the testing interval employed by the management tool and the test interval of SAPOTI were set to 10 seconds. In order to obtain experimental data, during the entire experiment the state of the hosts and the state of the Web servers were sampled each 5 seconds.

Tables 3 and 4 show for both experiments the number of times that each host became faulty and the percentage of time the hosts remained faulty, in comparison with the total observation time. It is possible to show that in the first experiment 210 faults occurred distributed among all hosts and that in the second experiment 27 faults occurred.

First Experiment		
Host	**ID**	**Fault Configuration**
Puma	1	5 minutes fault-free / 10 minutes faulty
Kenny	2	8 minutes fault-free / 9 minutes faulty
Stan	3	13 minutes fault-free / 7 minutes faulty
Kyle	4	17 minutes fault-free / 6 minutes faulty
Cartman	5	21 minutes fault-free / 5 minutes faulty
Lenoc	6	25 minutes fault-free / 4 minutes faulty

Table 1: The host names, ID and fault configuration in the first experiment.

Second Experiment		
Host	**ID**	**Fault Configuration**
Puma	1	20 minutes fault-free / 40 minutes faulty
Kenny	2	40 minutes fault-free / 20 minutes faulty
Stan	3	Always fault-free
Kyle	4	Always fault-free
Cartman	5	Always fault-free

Table 2: The host names, ID and fault configuration in the second experiment.

First Experiment		
Host	**Faults**	**Faulty Time in %**
Puma	50	66.4
Kenny	44	46.6
Stan	37	35.2
Kyle	32	26.3
Cartman	29	19.2
Lenoc	26	13.7

Table 3: The number of faults and the percentage of time in which hosts remained faulty in the first experiment.

Seconds Experiment		
Machine	**Faults**	**Faulty Time in %**
Puma	14	68.5
Kenny	13	51.4
Stan	0	0
Kyle	0	0
Cartman	0	0

Table 4: The number of faults and the percentage of time in which hosts remained faulty in the second experiment.

During the first experiment the service was transferred from a host to another 177 times. This is less than the number of faults injected, 210, because some faults occurred in machines with lower priority compared to the machine currently responsible for the service. In 72 cases the Web server became unavailable. In the other 105 cases the service was assumed by a server with higher priority.

Figure 2 shows the recovery latencies measured in the first experiment. It is possible to notice that the fastest recovery took 6 seconds and that the slowest recovery took 53 seconds. The average latency was 16.4 seconds. Figure 3 shows the accumulated recovery latency of this experiment. 54% of the measured recovery latency was less than 10 seconds. It is also possible to notice that in 90% of the recovery time was less than 30 seconds.

During the second experiment the Web server was transferred from one host to another 41 times. But, because of the same reason as described above, the Web server could be repaired 26 times.

Figure 4 shows the recovery latency measured in second experiment. The fastest recovery latency was equal to 7 seconds and the slowest recovery latency took 29 seconds. The average latency was 14 seconds. Figure 5 shows the accumulated recovery latency for this experiment. 64% of the measured recovery latencies were less than 10 seconds. It is also possible to notice that 100% of the measured latencies were less than 30 seconds.

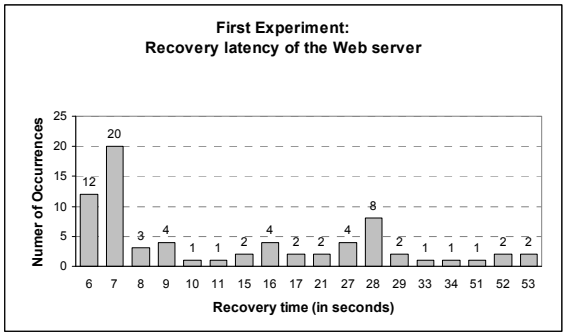

Figure 2: Recovery latency: first experiment.

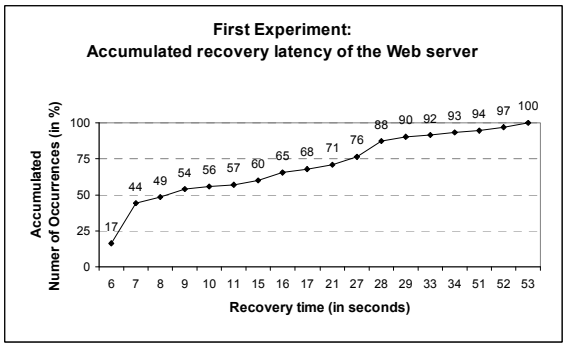

Figure 3: The accumulated recovery latency: the first experiment.

Figure 6 shows the number of times that each host started a Web server in the first experiment. It is possible to notice that even in this highly dynamic environment where 210 faults occurred distributed among six machines – and the Web server changed 177 times from one machine to another – the measured availability of the Web server was 97.35%. Therefore during the 12 hours and 23 minutes of the first experiment the Web server was unavailable for less than 20 minutes. It important to highlight that during part of this experiment faults were injected that caused all hosts to become faulty. In the second experiment there was always at least one fault-free host that could provide the service.

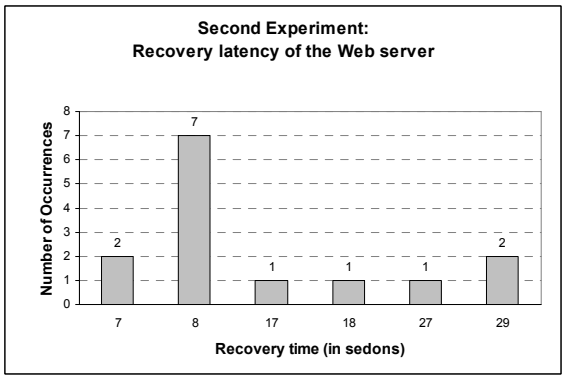

Figure 4: Recovery latency: second experiment.

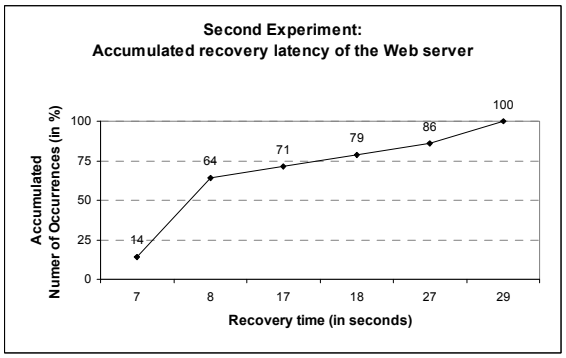

Figure 5: The accumulated recovery latency: the second experiment.

Figure 7 shows that in the second experiment, even with 27 faults distributed among the five machines – and the Web server changed 41 times from one machine to another – the availability of the Web server was 99.58% considering the experiment time. Therefore during the 12 hours and 40 minutes of the first experiment the Web server unavailability was 0.42% of the observation time, i.e. it was less than 4 minutes.

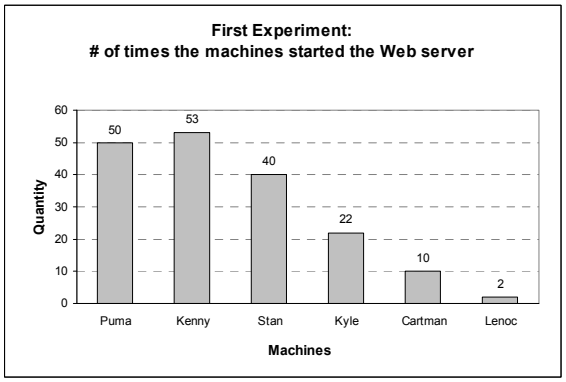

Figure 6: Number of times each machine started a Web server: first experiment.

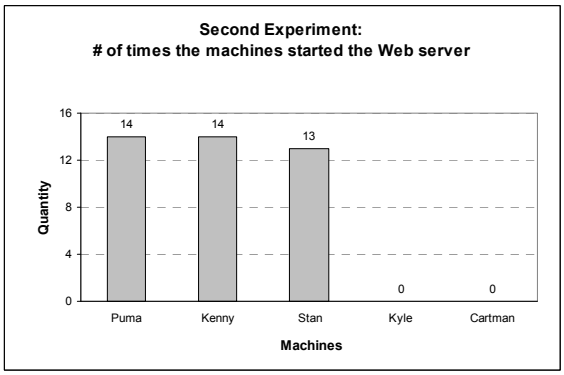

Figure 7: Number of times each machine started a Web server: the second experiment.

It is important to make clear that the presented results depend on the testing intervals (10 seconds) and sampling interval (5 seconds) employed. It is possible to configure these intervals, from milliseconds to hours. Small intervals lead to more precise results.

5. Related Work

Although to the best of our knowledge this is the first work that employs distributed diagnosis in order to implement highly-available Web servers, other approaches do exist. In [15] a heartbeat monitoring strategy is used to implement a highly-available Apache Web server using two hosts. Heartbeats are short messages employed by a process to inform other processes that it is alive. The monitoring component that generates heartbeats executes a test procedure that allows the detection of a faulty service. Even if one fault-free node is presented in the system, the Web server is supposed to be available. A key difference to our proposed system is that the number of heartbeats generated is quadratic, while it is logarithmic on average in SAPOTI.

Another solution presented in [16] describes a prototype of a highly-available Web server for IBM SP-2 systems. The solution consists of a set of hosts running Web servers connected by a switch. This switch is a central component that performs load balancing and guarantees the availability. This work is based on a combination of TCP-redirection and DNS (Domain Name Server) features. The availability is provided by detecting node or server failures and redirecting requests accordingly. Another approach that uses a central component to implement clusters of Web servers is presented in [17]. In this approach DNS is used as a centralized dispatcher to distribute the requests among the available fault-free servers.

Most related work focus not on the dependability of Web servers but on load balancing [18, 19, 20, 21, 22]. In [23] the authors propose a cluster of Web servers that can dynamically recruit non-dedicated processors when load

bursts occur. Recently in [24] virtual machines are used for deploying replicated servers. Other approaches [25, 26] make possible the construction of cluster Web servers to improve performance using the Socket Cloning (SC) strategy.

6. Conclusion

This work presented SAPOTI, a distributed tool for deploying highly-available TCP/IP application servers, in particular Web servers. SAPOTI runs on top of a management tool based on *Hi-ADSD with Timestamps,* a distributed diagnosis algorithm. The management tool allows hosts and processes to be monitored in an efficient way. SAPOTI uses diagnostic information to monitor a set of hosts that may run the Web server. If the host running the server at a given instant of time becomes faulty, SAPOTI re-starts the server on a fault-free host. The service is available even if there is only one fault-free host. Experimental results show that SAPOTI is able to consistently improve the availability of the system.

Future work includes deploying SAPOTI on a wide-area network, which requires another distributed diagnosis algorithm. Strategies to further improve the availability are also under investigation. The application of SAPOTI to other services is also planned.

7. References

[1] D. E. Comer, *Internetworking with TCP/IP – Vol. I Principles, Protocols, and Architectures*, Prentice Hall, 5th ed., 2005.

[2] E. P. Duarte Jr., and L. C. E. Bona, "A Dependable SNMP-Based Tool for Distributed Network Management," *IEEE/IFIP International Conference on Dependable Systems and Networks*, 2002.

[3] Luis C. E. Bona, Elias P. Duarte Jr., "A Flexible Approach for Defining Distributed Dependable Tests in SNMP-Based Network Management Systems," *Journal of Electr. Testing Theory and Apps.*, Vol. 20, No. 4, 2004.

[4] E. P. Duarte Jr., A. Brawerman, and L. C. P. Albini, "An Algorithm for Distributed Hierarquical Diagnosis of Dynamic Fault and Repair Events," *Proceedings of the IEEE International Conference on Parallel and Distributed Systems*, 2000.

[5] G. Masson, D. Blough, and G. Sullivan, *System Diagnosis in Fault-Tolerant Computer System Design*, ed. D. K. Pradhan, Prentice-Hall, 1996.

[6] P. Jalote, *Fault Tolerance in Distributed Systems*, Prentice Hall, 1994.

[7] *The NET-SNMP Project Home Page*, http://www.net-snmp.org. Acessed in June 2008.

[8] *The Linux Home Page at Linux on Line*, http://www.linux.org. Acessed in June 2008.

[9] *The Apache Software Foundation*, http://www.apache.org. Acessed in June 2008.

[10] *IP-Alias*, http://www.ibiblio.org/pub/Linux/docs/HOWTO/mini/IP-Alias. Acessed in June 2008.

[11] *RSYNC*, http://www.rsync.org. Accessed in June 2008.

[12] B. Callaghan, *NFS Illustrated*, Addison-Wesley, 2000.

[13] *PHP Hypertext Preprocessor*, http://www.php.net. Accessed in June 2008.

[14] *Debian GNU/Linux, The Universal Operating System*, http://www.debian.org. Accessed in June 2008.

[15] *High-Availability Middleware on Linux, Part 1: Heartbeat and Apache Web server*, http://www.ibm.com/developerworks/library/l-halinux. Acessed in June 2008.

[16] D. M. Dias, W. Kish, R. Mukherjee, and R. Tewari, "A Scalable and Highly Available Web Server," *Proceedings of the 41st IEEE International Computer Conference*, 1996.

[17] V. Cardellini, M. Colajanni, and P. S. Yu, "DNS Dispatching Algorithms with State Estimators for Scalable Web-Server Clusters," *World Wide Web*, Vol 2, No. 3, 1999.

[18] X. Gan, T. Schroeder, S. Goddard, and B. Ramamurthy, "LSMAC vs. LSNAT: Scalable Cluster-Based Web Servers," *Cluster Computing*, Vol. 3, No. 3, 2000.

[19] Q. Zhang, A. Riska, W. Sun, E. Smirni, and G. Ciardo, "Workload-Aware Load Balancing for Clustered Web Servers," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 16, No. 3, 2005.

[20] A. Riska, W. Sun E. Smirni, and G. Ciardo, "ADAPTLOAD: Effective Balancing in Clustered Web Servers Under Transient Load Conditions," *Proceedings 22nd International Conference on Distributed Computing Systems*, 2002.

[21] G. Teodoro, T. Tavares, B. Coutinho, W. Meira Jr., and D. Guedes, "Load Balancing on Stateful Clustered Web Servers," *Proceedings 15th Symposium on Computer Architecture and High Performance Computing*, 2003.

[22] S. Sharifian, M. K. Akbari, and S. A. Motamedi, "A Novel Intelligence Request Dispatcher Algorithm for Web Server Clusters," *IEEE-Eurasip Nonlinear Signal and Image Processing*, 2005.

[23] C.-S. Yang, M.-Y. Luo, "Building an Adaptable, Fault Tolerant, and Highly Manageable Web Server on Clusters of Non-Dedicated Workstations," *Proceedings of the Proceedings of the 2000 International Conference on Parallel Processing*, 2000.

[24] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "Remus: High Availability via Asynchronous Virtual Machine Replication," *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pp. 161-174, 2008.

[25] Y.-F. Sit, C.-L. Wang, and F. lau, "Socket Cloning for Cluster-Based Web Servers," *Proceedings of the IEEE International Conference on Cluster Computing*, 2002.

[26] Y.-F. Sit, C.-L. Wang, and F. lau, "Cyclone: A High-Performance Cluster-Based Web Server with Socket Cloning," *Cluster Computing*, Vol 7, No. 1, 2004.