A Proactive Management Framework in Active Clusters

Eunmi Choi[1] and Dugki Min[2]

[1] School of Computer Science and Electronic Engineering, Handong Global University, Heunghae-eub, Puk-ku, Pohang, Kyungbuk, 791-708, Korea
emchoi@handong.edu

[2] School of Computer Science and Engineering, Konkuk University, Hwayang-dong, Kwangjin-gu, Seoul, 133-701, Korea
dkmin@konkuk.ac.kr

Abstract. An active Web cluster system is an active network that has a collection of locally distributed servers that are interconnected by active switches, providing a Web application service. In this paper, we introduce the ALBM (Adaptive Load Balancing and Management) active cluster system that provides proactive management. The architecture of the ALBM active cluster and its underlying components are presented. We focus on system-level and service-level management of the active cluster system by presenting the corresponding proactive ALBM framework. The system-level framework considers performance counters of resource state dynamics; the service-level framework concerns service quality and proactive actions based on event occurrences. The experimental results on adaptive load balancing are presented in terms of system-level proactive management. In addition, a proactive event message service tool is introduced for providing effective services and management in terms of service-level proactive management.

1 Introduction

An active network contains network nodes that perform customized processing of packets [15,16,17]. An active Web cluster system is an active network that has a collection of locally distributed servers interconnected by active switches[5,18], providing a Web application service. It is shown as a single transparent cluster system to the user with one site name and one virtual IP address [1,2,3]. Due to their characteristics of cost-effectiveness, high scalability, and high availability, active Web cluster systems have become a typical solution to the next generation Internet services[4].

Various active Web cluster architectures have been developed in many forms. The most popular form is based on hardware L4 active switches [5,6,7]. The H/W switch-based cluster systems are easy to deploy only by connecting server nodes to the active switch box. The employed H/W L4 active switches act as traffic managers that would direct IP traffics to the highly appropriate healthy server in a cluster, performing network address translation on messages flowing through them. Since these systems are based on H/W switches, it is impossible to add any customizing schedules or new management services that contain capability to automatically recover the system from partial failures. Recently, some cluster systems are constructed on top of the Linux Virtual Server (LVS) [8,9,10,11]. The LVS is a software load balancer that directs network connections to multiple servers, so that servers can share their workload. As a S/W active switch, the LVS supports most of connection scheduling algorithms that are provided by the H/W active switches. Moreover, since the LVS is an open software, it can be customized to collaborate with other software tools or extended to add more improved scheduling algorithms for reflecting the system state of each node. Several variations of LVS[11,12,13] have been introduced to make the LVS be more adaptive.

In this paper, we introduce the ALBM (Adaptive Load Balancing and Management) active cluster that has a hybrid cluster architecture, which employs S/W active switches as well as middleware agents on server nodes. The ALBM cluster has proactive management framework that could provide both adaptive load balancing and proactive event-driven management. For adaptive load balancing, its active switch, called Traffic Manager, provides network-level server load balancing that changes the scheduling adaptively according to node state information by means of node agents. In order to analyze the dynamics of server performance due to workload, we perform several experiments with performance counters. We compare the performance results of adaptive scheduling algorithms to those of non-adaptive ones on various kinds of workload. For proactive management, the ALBM proactive framework includes service-level and server-level monitoring mechanisms for collecting performance counters. It also contains the rule-based event processing engine, and has direct and indirect event notification mechanisms for collecting and processing events.

This paper is organized as follows. Section 2 presents the ALBM active cluster system architecture and its components. Section 3 introduces the proactive ALBM management framework. In Section 4, we show experimental results on the performance of ALBM active cluster for various adaptive load balancing algorithms. In Section 5, we introduce a proactive event message service tool. We summarize in Section 6.

2 The ALBM Active Cluster Architecture

In this section, we introduce the architecture of ALBM (Adaptive Load Balancing and Management) active cluster system with its underlying components. As shown in Figure 1, the ALBM cluster system is composed of active switches, application servers, and the management station. Although the cluster consists of tens of application servers, called nodes, it is published with one site name and one virtual IP address that are assigned to one or more active switches, called Traffic Managers(TMs).

The Traffic Manager(TM)s interface the rest of cluster nodes with the Internet, making the distributed nature of the architecture transparent to Internet

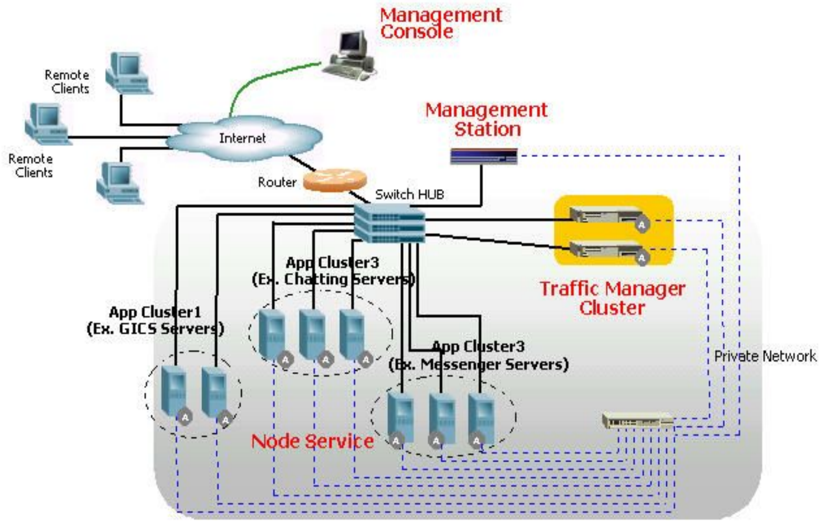


Fig. 1. Architecture of the ALBM Active Cluster System

clients. The TM is an active switch that performs customized processings on incoming and outgoging packets. When client traffic arrives, the TM routes the client packet to one of the servers according to its load balancing algorithm and policy, performing network address translation on the packets flowing through them. In order to decide traffic routing adaptively, it collects the status information of collaborated servers periodically and schedules servers in a specific order according to employing load balancing algorithm. After translating to a proper address, it routes a packet to the assigned server on the OSI Layer 4 level. Our TM provides several load scheduling algorithmss, such as Round-Robin, Least-Connected, Weighted, Response-time basis, and adaptive algorithms. Currently, our TM supports two types of L4 switching mechanisms: Direct Routing (DR) and Network Address Translation (NAT). Compared to other typical L4 switches, our TM considers current nodes states to manage the load-balancing mechanism. The TM receives information of real servers on the fly, such as state of workloads, dead node, and cluster configurations. To have the system highly fault-tolerant and scalable, two or more TMs can be organized as a cluster of TMs. In this case, each of TMs executes the Traffic Manager Node (TM-Node) Service. The TM-Node services perform membership management in an efficient way to be prepared with possible combinations of active or stand-by TM. The TM is implemented in C to achieve a high performance and take a small portion of memory.

On each of application servers, a middleware service, called Node Agent (NA), is running. The NA is a system-level service that is remotely and dynamically deployed and initiated by our middleware deployment service. The NAs are indicated by Circle As on the bottom of each node in Figure 1. The NA makes the underlying server be a member node of the ALBM system. The NA takes two types of agent roles. First, it works as an agent for managing the managed node. It monitors and controls the system elements or the application service of the node, and collects the state and performance information on its local management information basis. It interacts with the M-Station, giving the local information and receiving administrative commands for management purposes. Second, it works as an agent for clustering. Regarding membership management, it sends heartbeat messages to one another. When there is any change in membership, the membership convergence mechanism is initiated by the master node. The master node is dynamically elected by members whenever there is no master node or there exists inconsistent master information among members. Besides, the NA provides L7 APIs to application services running on the node.

The management station, called M-Station, is a management center of the entire ALBM cluster system, working together with the management console through Web-based GUIs. All administrators commands are received into and executed through the M-Station. By interacting with the master node of a cluster, the M-Station collects the dynamic state or performance information of the cluster system resources and application services. Also, the M-Station checks the system state in the service-level, and carries out some actions when values monitored from service-level quality are significantly far behind the service-level QoS objectives. According to the management strategies and policies determined by the human administrator, the M-Station takes proper management actions, such as alarming events or removing failed nodes. The M-Station is implemented in Java.

All the components in the ALBM system are interconnected with public and private networks. The public network is used to provide public services to Internet clients, while the private network is used for secure and fast internal communications among components on management purpose.

2.1 Adaptive Load Balancing Mechanism

The adaptive scheduling algorithms in the ALBM active cluster system adjust their schedules, taking into accounts of dynamic state information of servers and applications collected from servers. The ALBM algorithm is as follows. By collecting appropriate information of server states, the NAs customize and store the data depending on the application architecture, machine types, and expectation of a system manager. Each NA decides if the current state is overloaded or underloaded by using upper or lower thresholds of resource utilization determined by system configuration and load balancing policies of cluster management. Each cluster has a coordinator that is in charge of any centralized task in the cluster. We call the coordinator a Master NA, and only the Master NA communicates with TM as the representative in order to control the incoming TMs traffic. After collecting state data of all NAs in a cluster, the Master NA reports the state changes to the TM. Thus, real-time performance data are transferred to the M-Station, and the state data of servers are reported to the TM. By using the state information reported by Master NAs, the TM adjusts traffics of incoming requests properly to balanced server allocation. The TM does not allocate requests to overloaded servers, until the overloaded server state is back to a normal state. The scheduling algorithms are applied to the TM through the control of M-Station.

3 Proactive ALBM Framework

This section presents the proactive management framework on the ALBM active cluster system. This framework provides a management environment to adaptively control the system according to the system dynamics. For short-term management, it detects faults, failures, or overload states and reacts automatically to recover the system from the fault or performance degradation situation. It also accumulates the states of system and the state changes, such as events, into databases. Information stored in these databases is used for mid-term and long-term management, such as performance analysis, root cause analysis, and capacity planning.

In our proactive management framework, three types of dynamic system information are collected. One type is performance counters of resource states in a server node. Theses counters are retrieved directly from the server node in the system level. Typical examples of resource state counters are available memory in Kbytes, processor time, and NIC total bytes. The next type of system information is performance counters of service quality that are measured in the service level, i.e., collecting service counters outside the service cluster. Typical examples of service quality counters are the average response time per request and the average throughput per second. These types of system information represent the system state in different views. The third type is events that occur while serving applications and managing the system. The system listens and stores events in terms of management, since events represent the state changes of the system that might affect the performance or availability of the system.

The this section, we describe the architecture of our proactive management framework in two views: one shows the structure of how to collect the performance counters of resource states and service quality, and the other shows the structure of how to collect the events.

3.1 Architecture for Collecting Performance Counters

Figure 2 shows the architecture of modules that are related to performance counters collection and process. Performance counters of resource states are collected by the Resource Monitor in our framework. The Resource Monitor is implemented as a separate process that is independent of the NA process in a server node, since the Resource Monitor includes operations that could wait for failed resources. The Resource Monitor checks the resource status by accessing OS APIs and sends the measured information to the Resource State Collector of the NA. The Resource State Collector collects resource states from a number of resources periodically. Resource states collected by the Resource State Collector are passed to the Node State Evaluator. The Node State Evaluator evaluates the status of the server node based on the pre-defined node policies and makes the Event Delegator generate an event if necessary.

Another way to measure performances of server nodes is to monitor the service quality in service level. The Service Level (SL) Monitor checks the service quality of each node in the service level, by measuring service availability, response time, and recovery time. As a separate process, the SL Monitor can be deployed on any machine, which may locate outside of the service cluster or run on the same machine where the M-Station locates. By considering the objectives, the SL manager generates events and notifies them to the M-Station with the corresponding measured QoS information.

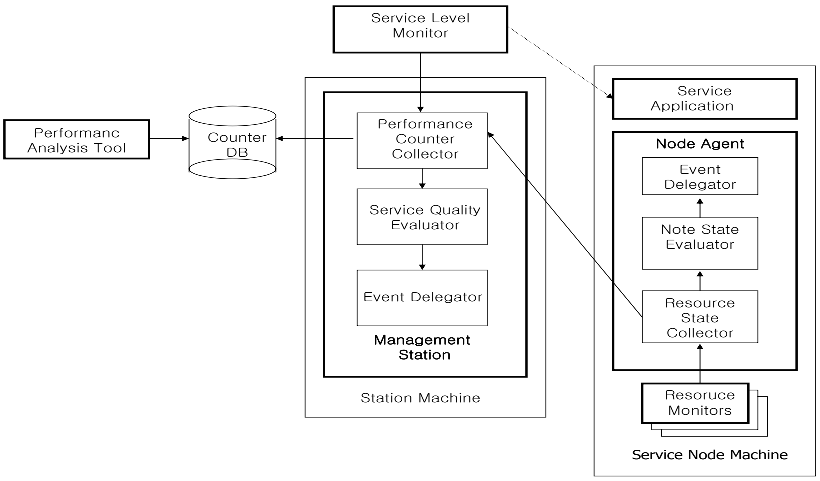


Fig. 2. Architecture of Performance Counter Collection

The resource state counters and the SL quality counters are collected by the Performance Counter Collector in the M-Station. The M-Station stores the performance counters into the Counter DB. This Counter DB is retrieved by the Performance Analysis Tool to trace system resource states and service quality states. Meanwhile, resource state counters and SL quality counters are passed to the Service Quality Evaluator for evaluating the system states and service level. If it is necessary to generate an event, the Event Delegator generates and sends the events to the appropriate system components.

The TMs interface the rest of cluster nodes, making the distributed nature of the architecture transparent to Internet clients. All inbound packets to the system are received and forwarded to application servers through the TMs. It provides network-level traffic distribution services by keeping on balancing the servers loads. On each of application servers, a middleware service, called Node

Agent (NA), is deployed. The NA is in charge of management and operation of the ALBM cluster, such as cluster formation, membership management, and adaptive management and load balancing. The management station, called M-Station, is a management center of the entire ALBM cluster system. All administrators commands typed through management console are received and delivered to the M-Station. Communicating with NAs, the M-Station performs cluster management operations, such as cluster creation/removal and node join/leave. The M-Station also collects current states and configurations of the entire system, and provides them to other authenticated components in the system. Using the server state information provided by NAs, it performs proactive management actions according to the predefined policies. Besides, the M-Station checks the state of the system in service-level, and carries on some actions when the monitored service-level quality value is significantly far behind with the service-level QoS objectives.

3.2 Architecture for Event Collection

The NA watches the service node to detect any occurrences of state change due to failures, errors, and faults of resources or service applications. According to the pre-defined node policies, the NA generates events through Event Delegator. As described in the previous subsection, the Event Delegator also generates performance degradation events guided by the Resource State Evaluator. The generated events are passed to the Event Rule Engine inside of the NA and also notified to the other system components outside of the NA through the Event Notifier. Figure 3 shows the architecture of the modules that are related to event notification and process. The Event Rule Engine, which contains rules and the corresponding conditions and actions, is in charge of event processing. When a new event is arrived, the rule engine performs the rule-matching step and processes the event according to the matched rule. Our rule engine is designed to be expandable or changeable for newly created event types and event processing conditions or actions; it can generate action or condition classes during run time from a XML rule definition schema. Some actions are performed and activated by the Event Rule Engine, but other actions are delegated to the Managed Agent.

Events are delivered toward the outside of the NA through the Event Notifier. The Event Notifier provides a direct event notification environment between two remote applications or components. As a separate service process, the Event Notifier can be shared by several applications running on the same server. Whenever a distributed component has changed its state, the component uses the Event Notifier to notify a state-change event to other related distributed components. The target where to disseminate an event is determined by the dissemination information that is dynamically updated according to the event rule definition as well as the system state and configuration. For outgoing or incoming events, simple event processing actions, such as filtering or formatting, can be applied according to the event rule definition.

In our proactive framework, if an event is so urgent that an automatic action should be performed immediately, the urgent events are directly passed to the

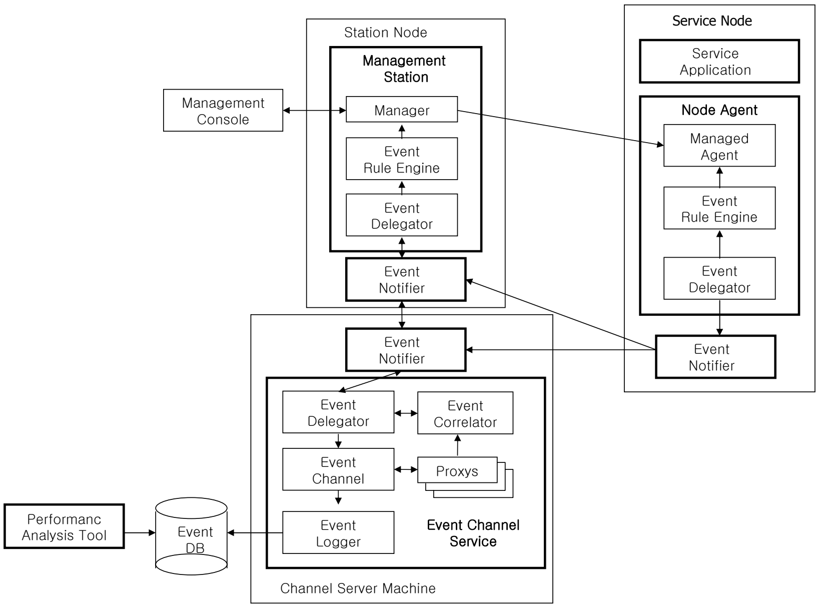


Fig. 3. Architecture of Event Collection

M-Station through the Event Notifier. The Event Delegator of M-Station collects all directly transmitted events from NAs. The Event Rule Engine of M-Station determines if the proper action should be performed after looking at the collected events. The Event Rule Engine contains a rule engine where rule matching and executing process is performed when a new event is arrived. In our rule engine system, a rule is activated by a number of correlated events that satisfy a given condition, and then an activated rule executes a predefined action. Thus, a rule is defined by a combination of a number of correlated events, called Event Token, a rule condition, a rule action, as well as by its properties, such as a rule name and a priority. The rule priority determines the order of applying rules when there are two or more rules compete. The event token name follows a format of domain-name:event-type-name. The condition and action codes are written in Java. The Manager of M-Station carries out the action to the associated managed agents. The process and result are reported to the Management Console.

The Event Notifier also sends all the events including the urgent events to the Event Channel Service that provides the indirect event communication service. The Event Channel Service provides indirect event transmission service that decouples event consumers from event producers. For each event customer or producer group, the Event Channel Service provides different quality of services or different event transmission mechanisms (push or pull mode). Also, the Event Channel Service is used as an event collection center where all the events are collected from NAs and various event producers in the cluster system. Using the aggregated events, we can perform an event correlation processing for root-cause analysis or a trend analysis for capacity planning.

The Event Channel Service is composed of Event Channel, Event Logger, and Event Correlator. An Event Channel performs various event-processing services, such as event scheduling for QoS-based transmission or event filtering. It also delivers events to the Event Correlator and the Event Logger for further processes of event analysis. The Event Correlator obtains events from channels and applies rules to find correlations between events. The Event Correlator is used to find a root-cause of node failure and to generate knowledge-based high-level events. The Event Logger sends event logs to the remote event log server to collect all the events for a time period. The collected events are analyzed together with the collected performance counters for system trend analysis. The channel has a flexible structure to be changed according to the QoS policy.

4 An Experiment with Adaptive Mechanism

We perform an experiment to show the benefit of applying our adaptive mechanism to the non-adaptive load scheduling algorithms. As for load scheduling algorithms, two commonly used ones are employed: the Round-Robin(RR) algorithm that is a popular static algorithm, and the Least Connection (LC) algorithm that is a popular dynamic algorithm. The adaptive version of RR and LC are called ARR and ALC, respectively. In our adaptive mechanism, the cost of adaptive operation is negligible since reporting the state information of servers to the TM is not a periodic processing and it occurs only when the state changes. It is very few compared to the amount and frequency of incoming requests.

We make a realistic workload that is heavy-tailed. In literature, many researchers have concluded that general Internet traffics follow heavy tail distributions [8,9,10]. In order to make heavy-tailed e-commerce traffic, we mix an e-commerce traffic provided by Web Bench tool[7] and a memory-intensive traffic at the rate of 80% and 20%, respectively. The e-commerce traffic contains mixed requests of text pages, image files, and CGI requests. The memory-intensive traffic contains memory requests of random size and random duration. The random size and duration are randomly selected between 5MB to 15MB and between 0 to 20 seconds, respectively.

The workload requests are generated by tens of client machines, which are interconnected with a cluster of server nodes in the same network segment. Each server has PIII-900MHz dual CPU and 512 MB memory. Each client has PIV 1.4GHz CPU and 256MB memory. The network bandwidth is 100MB. The number of connections per client thread is 4. The total running time is 2 minutes, think time between requests in a client is 10 seconds, and ramp-up time is 20 seconds.

Figure 4 shows the experimental result; adaptive ARR and ALC algorithms achieve about 10% better performances than non-adaptive ones. The best performance is achieved with the ALC algorithm. Due to the active characteristic of traffic manager (i.e. network switch), the adaptive mechanism could achieve better throughput by adjusting the load scheduling dynamically. According to the feature of Web Bench Tool, the next request is generated after the response of the previous request has received. That is, Web Bench Tool slows down sending requests when the server responds late. Due to this feature, all scheduling algorithms reduce their throughputs after reaching their maximum performances. This makes results after the peak points meaningless.

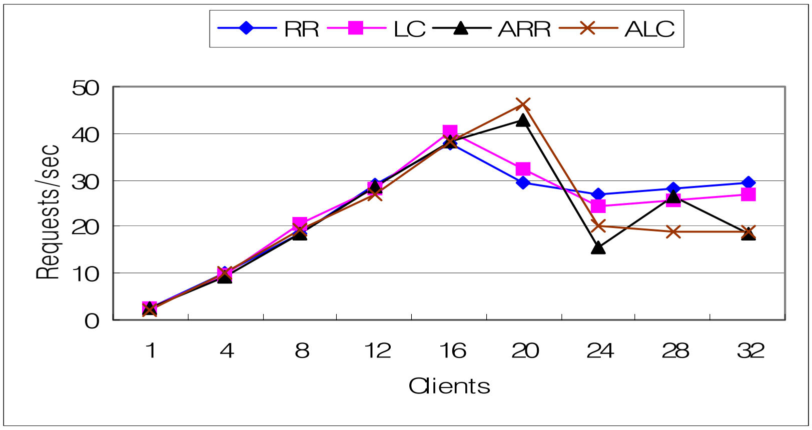


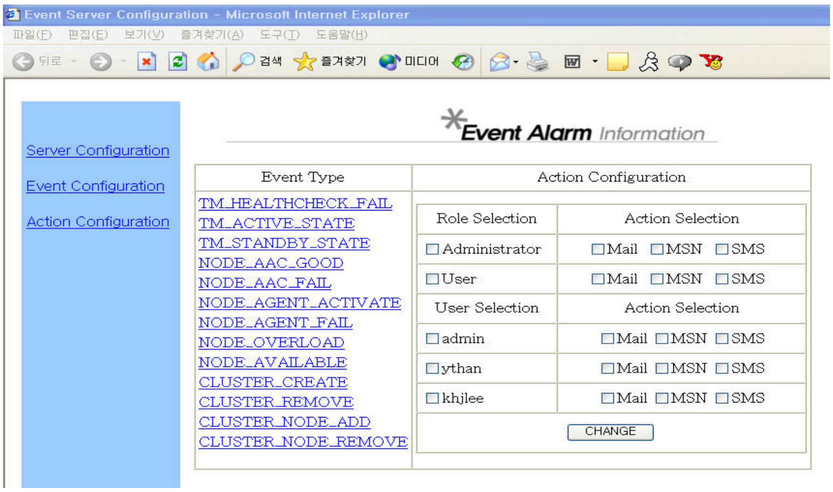
Fig. 4. Experimental Comparison

5 Proactive Event Messaging Service Tool

In addition to adaptive load balancing, the dynamic state information is used in the M-Station for proactive management. The M-Station provides proactive actions in two levels: system level and service level. In system level, the M-Station receives system state transition events from NAs and takes actions according to the rules managed by an event rule-engine. Events are transmitted to the M-Station through underlying event notification and channel services. In service level, a service-level (SL) monitor takes the similar role as NAs. The SL monitor checks the service quality of the ALBM cluster in service level, such as availability, response time, and recovery time. By considering the QoS objectives, the SL monitor generates events and notifies them to the M-Station with the corresponding measured QoS information. The SL monitor can be deployed on any place and computer in Internet.

In this section, we present a proactive event messaging service is implemented on top of the proactive management framework. The proactive event messaging service sends a message to the system manager according to the predefined event rules when an event occurs. Figure 5 shows the GUI to set up actions according to the event type. When an event occurs, the setting action is triggered and performed as desired. It is possible to send an e-mail to a human administrator, an

MSN message passing, or an SMS cellular phone message delivery. The message can be delivered to one person or a collection of people with a specific role.


Event Server Configuration - Microsoft Internet Explorer
파일(F) 편집(E) 보기(V) 즐겨찾기(A) 도구(T) 도움말(H)
뒤로
검색
즐겨찾기
미디어
Server Configuration
Event Configuration
Action Configuration
CHANGE

Event Alarm Information

Event Type	Action Configuration
TM_HEALTHCHECK_FAIL TM_ACTIVE_STATE TM_STANDBY_STATE NODE_AAC_GOOD NODE_AAC_FAIL NODE_AGENT_ACTIVATE NODE_AGENT_FAIL NODE_OVERLOAD NODE_AVAILABLE CLUSTER_CREATE CLUSTER_REMOVE CLUSTER_NODE_ADD CLUSTER_NODE_REMOVE	

Role Selection	Action Selection
☐ Administrator	☐ Mail ☐ MSN ☐ SMS
☐ User	☐ Mail ☐ MSN ☐ SMS
User Selection	**Action Selection**
☐ admin	☐ Mail ☐ MSN ☐ SMS
☐ ythan	☐ Mail ☐ MSN ☐ SMS
☐ khjlee	☐ Mail ☐ MSN ☐ SMS

Fig. 5. The GUI of Event Action Set-up

6 Conclusion

In this paper, we introduced the ALBM active cluster system that provides proactive management. The architecture of the ALBM active cluster and its underlying components are presented. As an active switch, the TM is in charge of routing the client packet to one of the servers according to its scheduling algorithm and policy, performing network address translation on the packets flowing through them. In order to applying system-level and service-level management to the active cluster system, a proactive ALBM framework is introduced. It collects system information and processes them according to system state changes. The system-level framework considers performance counters of resource states by probing, processing, making a decision, and performing an action based on state changes of resource utilization. The experimental results on adaptive load balancing are presented in terms of system-level proactive management. Adaptive scheduling algorithms result in a good performance compared to the non-adaptive ones for a realistic heavy-tailed workload. Besides, a proactive event message service tool is introduced for providing effective services and management.

References

1. Gregory F Pfister: In Search of Clusters, 2nd Ed. Prentice Hall PTR (1998)
2. Daniel A. Menasce: Trade-offs in Designing Web Cluster. IEEE Internet Computing, Volume:6 Issue:5 Sep/Oct (2002) 76 ?80
3. Ana-Maria Cretu, Voicu Groza, Abdul Al-Dhaher, Rami Abielmona: Performance Evaluation of a Software Cluster. IEEE Information and Measurement Technology Conference May (2002) 1543-1548
4. Jeffray S. Chase: Server switching: yesterday and tomorrow. Internet Applications (2001) 114-123
5. Valeria Cardellini, Emiliano Casaliccho, Michele Colajanni, Philip S. Yu: The State of the Art in Locally Distributed Web-server Systems. IBM Research Report, RC22209(W0110-048) October (2001) 1-54
6. Ronald P. Doyle, Jeffrey S, Chase, Syam Gadde, Amin M Vahdat: The trickle-down effect: Web caching and server request distribution. In Proceedings of the Sixth International Workshop of Web Caching and Content Distribution, 2001
7. Lee Breslan, Pei Cao, Li Fan Graham Phillips, Scott Shenker: Web Caching and Zipf-like distributions: Evidence and implications. In Proceedings of IEEE infocom Mar (1999)
8. TurboLinux: Turbo Linux Cluster Server 6 user guide. http://www.tubolinux.com, (2002)
9. LVS documents, http://www.linuxvirtualserver.org/Documents.html
10. Wensong Zhang, Shiyao Jin, Quanyuan Wu: Scaling Internet Service by LinuxDirector. High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition, Volume: 1, (2000) 176-183
11. Wensong Zhang: Linux Virtual Server for Scalable Network Services. Linux Symposium 2000, July (2000)
12. M. Wangsmo: White paper: Piranha ? load-balanced web and ftp clusters. http://www.redhat.com/support/wpapers/piranha (1999)
13. J. Trocki: mon: Service monitoring daemon. http://www.kernel.org/software/mon
14. Eunmi Choi: Performance Test and Analysis for an Adaptive Load Balancing Mechanism on Distributed Server Cluster Systems. 2004 Future Generation Computer Systems, Elsevier Science, Vol. 20. No. 1, (2004, will be appeared)
15. M. Brunner: A Service Management Toolkit for Active Networks, IEEE Network Operations and Management Symposium(NOMS) April (2000) 265-278
16. Marcus Brunner: Active Networks and its Management, IEEE Universal Multiservice Networks (ECUMN) Oct (2000) 414-424
17. Marcus Brunner, Rolf Stadler: Service Management in Multiparty Active Networks, IEEE Communications Magazine, Vol:38 Issue:3, March (2000) 144-151
18. Jiani Guo, Fang Chen, Laxmi Bhuyan, and Raj Kumar: A Cluster-Based Active Router Architecture Supporting Video/Audio Stream Transcoding Service, IEEE Proceedings of International Parallel and Distributed Symposium (IPDPS) (2003) 44