Event-Driven Management Automation in the ALBM Cluster System

Dugki Min[1] and Eunmi Choi[2]

[1] School of Computer Science and Engineering, Konkuk. University,
Hwayang-dong, Kwangjin-gu, Seoul, 133-701, Korea
dkmin@konkuk.ac.kr
[2] 2School of Business IT, Kookmin University,
Chongnung-dong, Songbuk-gu, Seoul, 136-702, Korea
emchoi@kookmin.ac.kr **

Abstract. One of major concerns on using a large-scale cluster system is manageability. The ALBM (Adaptive Load Balancing and Management) cluster system is an active cluster system that is scalable, reliable and manageable. We introduce the event-driven management automation by using the ALBM active cluster system. This architecture is based on an event management solution that is composed of event notification service, event channel service and event rule engine. Critical system state changes are generated as events and delivered to the event rule engine. According to the predefined management rules, some management actions are performed when a specific condition is satisfied. This event-driven mechanism can be used to manage the system automatically without human intervention. This event management solution can also be used for other advance management purpose, such as event correlation, root cause analysis, trend analysis or capacity planning. In order to support the management automation possibility, the experimental results are presented by comparing adaptive load balancing with non-adaptive load balancing mechanism. The adaptive scheduling algorithm that uses the event management automation results in a better performance compared to the non-adaptive ones for a realistic heavy-tailed workload.

1 Introduction

Future Internet services, such as Web Services[1] and ubiquitous services[2], become more dynamic and various in clients population size and in service pattern, due to their characteristics of dynamic integration. The unpredictable characteristic of Internet services requires their service platform architecture to be scalable and reliable. A cluster of distributed servers is a popular solution architecture that is scalable and reliable as well as cost-effective: we are able to easily add economical PC servers for more computing power and storages[3,4].

** This work was supported by the Korea Science and Engineering Foundation (KOSEF) under Grant No. R04-2003-000-10213-0. This work was also supported by research program 2004 of Kookmin University in Korea.

One of major concerns on using a large-scale cluster system is manageability[5]. Internet service providers normally have a number of clusters consisting in several tens of servers up to several hundreds of servers, which might have heterogeneous platforms. Managing a huge number of distributed servers is not an easy task. Even basic management operations such as monitoring resource status, upgrading O.S. and deploying a new service, are tasks that takes lots of efforts due to lack of global knowledge and controller, and the limitation of networked computers. Therefore, a management tool is necessary to manage a number of distributed clusters effectively.

The ALBM (Adaptive Load Balancing and Management) cluster system is an active cluster system that is scalable, reliable and manageable[6]. We developed this system for various research purposes, such as active traffic management, content-based delivery, middleware services for distributed systems, and proactive distributed system management. It is composed of L4/L7 active switches for traffic distribution and management agents and station for cluster system management. This system provides a single point of management console that shows system configuration as well as system states in real time. Using this consol, we can monitor the status of all resources and also control services on distributed nodes.

In this paper, we present an event-driven management automation architecture that is used in the ALBM cluster. This architecture is based on an event management solution that is composed of event notification service, event channel service and event rule engine. Critical system state changes are generated as events and delivered to the event rule engine. According to the predefined management rules, some management actions are performed when a specific condition is satisfied. This event-driven mechanism can be used to manage the system automatically without human intervention. This event management solution can also be used for other advance management purpose, such as event correlation, root cause analysis, trend analysis or capacity planning. In order to support the management automation possibility, the experimental results are presented by comparing adaptive load balancing with non-adaptive load balancing mechanism. The adaptive scheduling algorithm that uses the event management automation results in a better performance compared to the non-adaptive ones for a realistic heavy-tailed workload.

This paper is organized as follows. Section 2 describes the architecture of the ALBM cluster system. In the next section, we present the event management solution architecture. The event management solution is composed of three subsystems: event notification service, event channel service and event rule engine. In section 4, an experimental result of performance is given to illustrate the benefit of employing the event-drive management automation mechanism for adaptive workload scheduling. We conclude in the last section.

2 The ALBM Active Cluster Architecture

As introduced in our previous research[6], the ALBM (Adaptive Load Balancing and Management) active cluster system is composed of active switches, application servers, and the management station.

The Traffic Manager (TM) is an active switch that customizes traffic packets by controlling static or dynamic services. When client traffic arrives, the TM routes 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 decide traffic routing, it collects the status information of collaborated servers periodically by contacting with the Node Agents from servers. Our TM provides several scheduling choices, 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).

In a server node, a Node Agent (NA) runs as a system-level service. 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. Using the L7 APIs, the application service can access information of cluster configuration or the current states of cluster nodes to dynamically make a decision. Also, the NA finds the dynamic information of application states through the L7 APIs. This dynamic application information is used for system operation, such as load balancing, and for other performance management. The NA is implemented in Java to maximize portability and platform-independent characteristics.

The Management Station (M-Station) with a Web-based console provides a single point administration and the management environment of the entire ALBM active cluster system. Communicating with NAs on managed nodes, it monitors states of the system resources and application services of nodes and controls them for various management purposes, such as creating a cluster or stopping an application service. The major cluster administration task is the management tool governed by human system administrators with the help of 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. 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.

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 balance 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 Event Management Solution

In this section, we introduce the overall architecture of event management solution: functionality and features of event notification service, event channel service, and event rule engine.

3.1 Event-Driven Management Automation Architecture

Event-driven management automation architecture finds the root cause of faults based on events occurred in several minutes and hours, and with the help of the event rule engine it resolves the faulty situation so that the human system administrator would not involve the system management manually. As shown in Figure 1, the management automation architecture manages the system at three levels in terms of management time: short-term, medium-term, and long term managements. Short-term management concerns real-time monitoring and immediate reaction. It monitors the system state changes, detecting system or service faults. Critical events are notified and the predefined corresponding management actions are automatically performed in real time. Event notification service and event rule engine are used at this level. Next, medium-term management concerns management intervention based on hourly information. In this

level, event log accumulated in hours are analyzed to find event correlations and root causes of faults. In this analysis process, high-level events are generated and used by the event rule engine to perform management interventions automatically or human-interactively. Long-term management automation concerns analyzing and predicting the trend of system usage and capacity needed for the future. This long-term management uses the historical log data of system states and events over a couple of weeks and months.

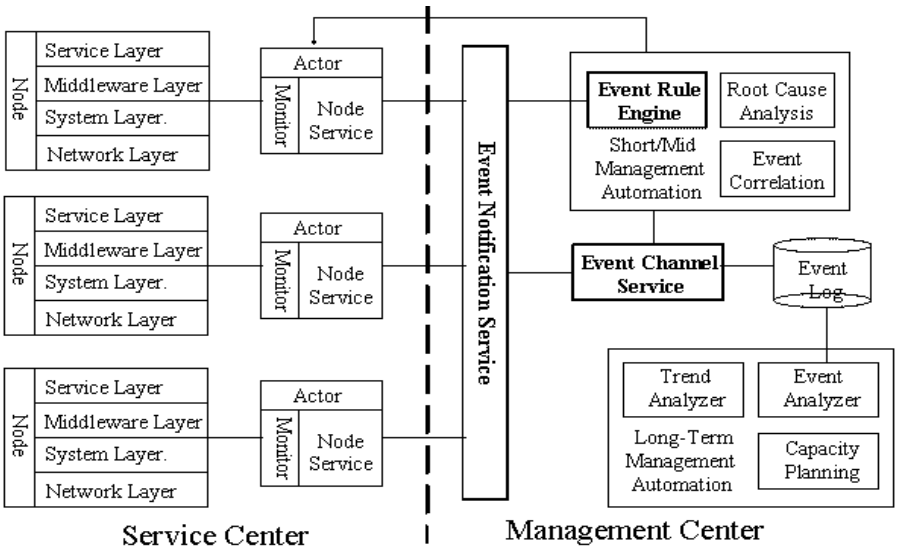


Fig. 1. Event-Based Management Automation Architecture

Figure 1 shows the system architecture of management automation. The architecture is decomposed into two subsystems. One is a service center that has a number of clusters, each of which is composed of a number of distributed servers. In each server, a NA(Node Agent) explained in Section 2 is running. It works as a management agent, monitoring and controlling system elements or application services on the node. The other is an event management center that manages the overall system. In our system, the event management center is in the M-Station. The event management center is composed of three event management solutions: Event Notification Service for event delivery, Event Channel Service for event asynchronous transmission, and Event Rule Engine for management automation. In this section, we describe the architectures of three event management services in detail.

3.2 Event Notification Service

Figure 2 shows the architecture of event notification service. It consists of three components: event communication infrastructure, event dissemination process, and event client. The event communication infrastructure is a communication infrastructure that facilitates transmitting events in various protocols and message formats. In Figure 2, the CI stands for the Communication Infrastructure. Determining a specific protocol and a message format to be used depends on the application type. In our current implementation, we provide three network protocols, i.e. TCP, UDP and a Reliable-UDP, and three message formats, i.e. a payload format, java object serialization, and a XML format. An event client is an event supplier that generates an event and sometimes becomes an event consumer that consumes an event. The event dissemination process is a service process that is shared by a number of event clients for disseminating events. The event dissemination is performed based on subject-oriented processing. In other words, an event supplier sends an event with a subject to an event dissemination process without specifying its target event consumers. The event dissemination process is responsible to deliver the subject-oriented event to appropriate target event consumers listening to the subject. By employing this shared dissemination process, individual event client can reduce a burden of disseminating tasks and thus improve the overall performance of event communication.

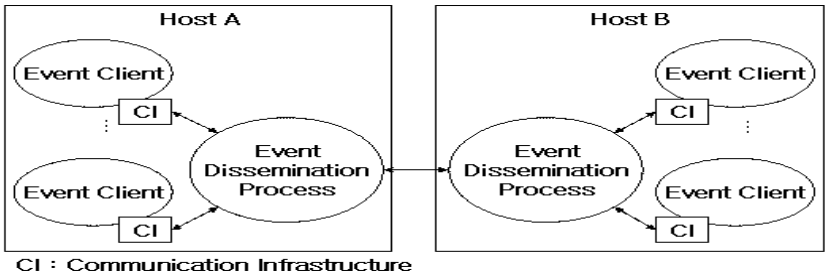


Fig. 2. Event Notification Service

Event Communication Infrastructure: Event communication infrastructure provides fundamental APIs of event transmission to event clients. Figure 3 (a) shows the structure of event communication infrastructure. Two main objects are Communication Object and DocFormat Object. *Communication Object (CO)* provides a unified communication environment that hides an underlying communication protocol and message format. The CO is implemented on top of TCP/IP protocol. Our current implementation provides communication of TCP, UDP and a reliable version of UDP. The *DocFormat Object* is used for message formatting of a CO. The DocFormat Object is in charge of converting

an event object into a message format. The current version of DocFormat Object supports three kinds of message formats: a XML format, a Payload format, and an object serialization format. The lifecycle of Communication Object and DocFormat Object is managed by the *Communication Manager*. The *Configurator* is in charge of configuration management of all these components by using configuration information stored in a XML file.

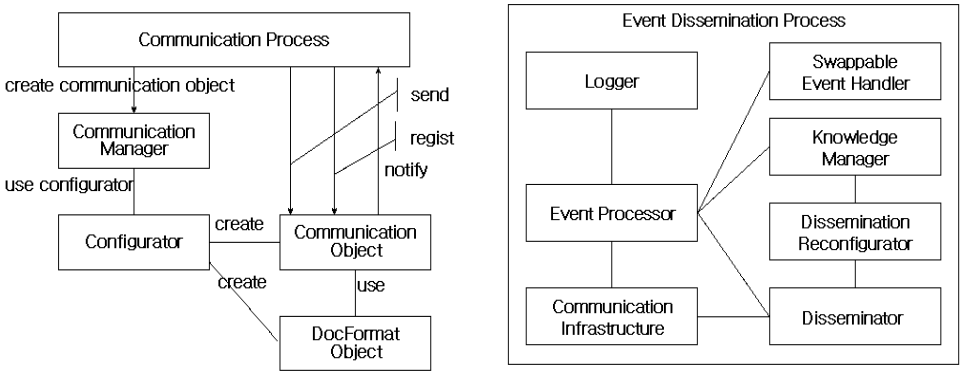
Event Dissemination Process: The event dissemination process disseminates an event from an event supplier to multiple event consumers distributed in a number of hosts. Figure 3 (b) shows the structure of the event dissemination process that is composed of the following objects: Event Processor, Swappable Event Handler, Disseminator, Knowledge Manager, Dissemination Reconfigurator, Logger, Communication Infrastructure.

The *Event Processor* is the core object of the event dissemination process. It receives an event from an event supplier through the CO and activates filtering and dissemination logics. It also leaves log information. The *Swappable Event Handler* judges whether its sending event has a meaningful message. In order to judge the semantic of an event, a filtering logic is applied. A filter is implemented as a swappable component so that new filters can be added later on demand of future need. The *Disseminator* executes actual dissemination for a given event. It decides the destinations of the given event according to the event subject, and distributes the event to the target destinations. Dissemination information and rules used in the Disseminator are managed by the *Knowledge Manager*. This information can be changed by an administrator UI, or by the system environment that is dynamically changed over times. The *Dissemination Re-Configurator* is in charge of updating dissemination rules. The *Logger* records logs during processing event dissemination.

Our event dissemination process has three major characteristics. First, a supplier can disseminate events asynchronously. Asynchronous event dissemination implies that an event supplier can sends the next event without blocking as soon as it sends the previous event. It is because the event dissemination process runs on a separate proc-ess that is independent of the supplier process. The second characteristic is that a basic dissemination rule is based on the event subjects. This is, an event is transmitted without specifying its destinations. Where to be transmitted is decided by the dissemination process according to the event subject and the system environmental knowledge. The last characteristic is contents-based message filtering. During event handling, useless events can be filtered according to predefined filtering rules. This filtering process needs little computing power, but can reduce the wasted network bandwidth as well as computing resources. The rate of saving depends on the correctness of filtering rules and the situation of event generation.

3.3 Event Channel Service

The event notification service provides synchronous event communication: events are delivered to the destinations in real-time. However, this synchronous event



(a) Event Communication Infrastructure **(b) Event Dissemination Process**

Fig. 3. Sub-components of Event Notification Service

communication is not useful when event consumers are not ready to receive. Thus, we need another communication mechanism that transmits events asynchronously. The *Event Channel Service* is such an asynchronous communication service that delivers events in a stored-and-forward mechanism. The advantage of using the event channel service is that event receivers can be decoupled from the event senders. Thus, the event receivers independently subscribe a number of event channels from which interesting events can be received. This event channel service is more valuable when the distributed servers of the cluster system are located over a number of network segments or some event receivers are available discontinuously in nature, such as mobile devices.

The event channel service has the structure as shown in Figure 4. The main components of this service are channels and its channel factory. A *channel* contains an event queue where events are stored and subscribed. The channel decouples event suppliers from event consumers, such that events can be delivered to whom subscribes the channel even though the event supplier does not know about any information of event consumers, such as their existences and locations. The *channel factory* is a factory that can create various types of channels according to QoS parameters. Each event consumer or event supplier accesses a channel through its own proxy. A proxy decides the type of event delivery: a push proxy delivers event in a push style and a pull proxy in a pull style. It also has filters inside so that an event customer can filter out a specific type of events. The proxies are created by *SupplierAdmin* or *ConsumerAdmin* according to the information of proxy QoS and management parameters.

3.4 Event Rule Engine

An event engine finds in real-time pre-defined event patterns among the generated event sequences and then it performs appropriate operations for the event patterns detected according to the event rules. Our event engine is a rule-based

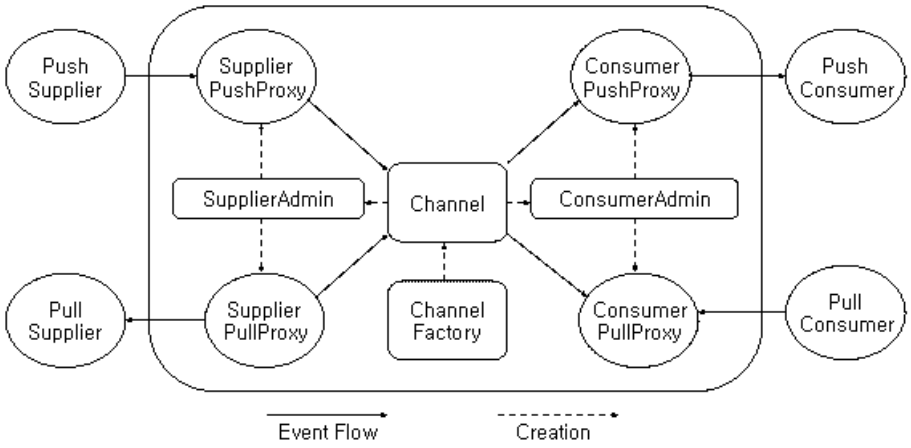
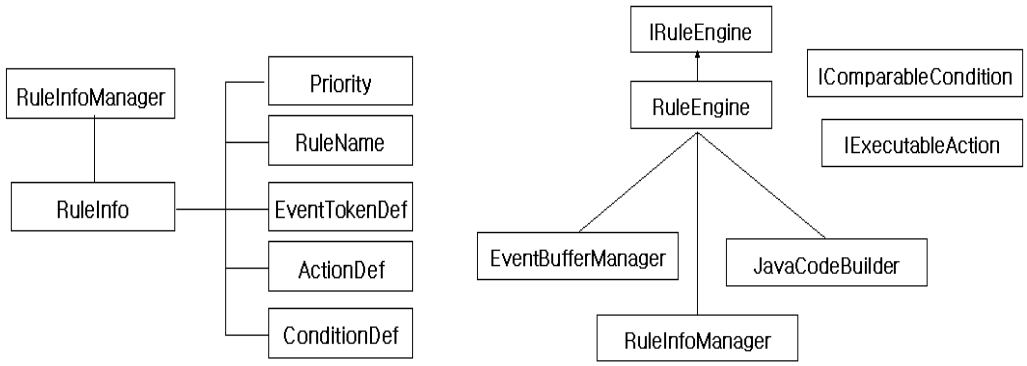


Fig. 4. Structure of Event Channel Service

one that is different from the traditional rule-based engines[7] in two ways. First, its functionality can be expanded by loading hooking classes dynamically; each event condition and action is defined as a hook class that is dynamically loaded, compiled and integrated into as a part of the engine. Since we implement the engine in a high-level object-oriented language, Java, we can develop new conditions and actions easily in an object-oriented style. Another characteristic of our rule-engine is that it uses an event-token method for finding matching rules. As a conventional compiler searches a meaningful token in a collection of strings, this approach checks only pre-defined event tokens instead of searching exhaustively. An event token is specified in general BNF operators.

The event rule engine is composed of three packages: information package, engine package, and parser package. The *information package* manages the information of the engine and its rules, shown in the Figure 5 (a). The RuleInfo defines one or more rules. A rule definition has rule name, priority, event token name, condition code, and action code in Java. The rule definition is stored in a XML file. The *engine package*, shown in Figure 5 (b), is the encore part of event rule engine. The EventBuffer-Manager manages real-time events, and removes old events after the expiration date. The RuleInfoManager manages the RuleInfo explained above. The JavaCodeBuilder converts condition or action hook classes into executable java objects when the engine initially starts. The IcomparableCondition and IExeutableAction are the interfaces that the hook classes of condition and action should implement, respectively. Finally, the *parser package* organizes a parsing table by using rules defined in the information package, and finds applicable rules by searching the occurring events in real time.



(a) Information Package **(b) Engine Package**

Fig. 5. Event Rule Engine

3.5 Experimental Results

We perform experiments to illustrate the effect of applying event-driven management automation in the ALBM cluster system. In this purpose, we apply the mechanism of event-driven management automation to the workload scheduling process. In normal situation, request traffic is distributed to the servers according to a general workload scheduling algorithm, such as Round-Robin (RR) or Least Connection (LC) [8]. However, in overloaded situation NA generates an Overloaded event, and the event is delivered to the event rule engine in the M-Station through the event notification service. According to the pre-defined event rule, the overloaded server is removed from the scheduling server list. In our experiments we employ the RR as a general scheduling algorithm. The event-driven adaptive version of RR is called E-ARR (Event-driven Adaptive RR).

We make a realistic workload that is heavy-tailed. In literature, many researchers have concluded that general Internet traffics follow heavy tail distributions [9,10]. In order to make heavy-tailed e-commerce traffic, we mix an e-commerce traffic provided by Web Bench tool[11] 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 of memory is randomly generated from 3M, 5MB, and 15MB and the memory holding duration is a random number between 0 to 10 seconds.

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-866MHz and 128 MB memory. Each client has the same system configuration. 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 6 and 7 show the experimental results of RR and E-ARR scheduling algorithms. The E-ARR achieved about 30 requests per second at 15 client threads; the RR achieved about 25 requests per second at 13 client threads in Figure 6. The E-ARR results in about 20% better performances than non-adaptive ones. For the same experiment, we present the throughput in Bytes per second in Figure 7. With the help of event-driven management automation, the adaptive mechanism could achieve better throughput by adjusting the load scheduling dynamically. According to the feature of Web Bench Tool, the next request from a client thread is generated after receiving the response of the previous request. That is, Web Bench Tool slows down sending requests, once the server starts to respond late. Due to this feature, all scheduling algorithms reduce their throughputs after reaching their peak performances. This makes points of results in the figure meaningless just after the peak performance points.

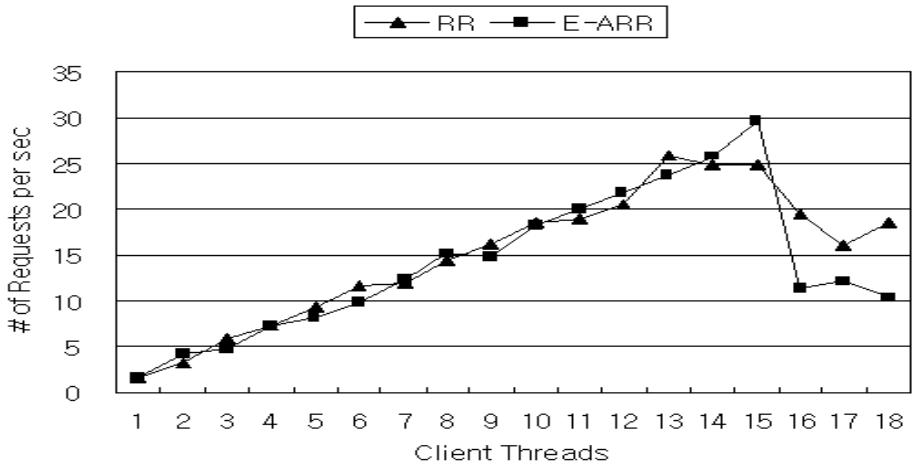


Fig. 6. Number of Requests per second of Event-driven Load Balancing

4 Conclusion

In this paper, we introduced the event-driven management automation by using the ALBM active cluster system. On top of the architecture of the ALBM cluster with its underlying components of the TM, and NAs, and M-Station introduced in Section 2, the ALBM cluster system provides the event management solution. The event-driven management automation architecture, event notification service, and event channel service, and event rule engine are introduced as the management service involved.

To support the management automation processing, the experimental results are presented by comparing adaptive load balancing with non-adaptive load

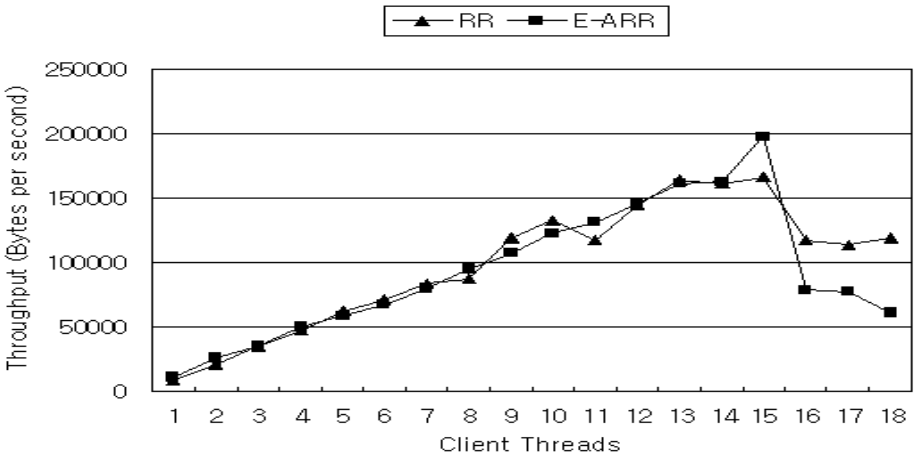


Fig. 7. Throughput (Bytes per second) of Event-driven Load Balancing

balancing mechanism. The adaptive scheduling algorithm that uses the event management automation results in a better performance compared to the non-adaptive ones for a realistic heavy-tailed workload.

References

1. Patlak, C.; Bener, A.B.; Bingol, H.: Web service standards and real business scenario challenges, Euromicro Conference, 2003. Proceedings. 29th (2003), 421 - 424
2. YamazakiK.: Research directions for ubiquitous services, Applications and the Internet, 2004. Proceedings. 2004 International Symposium on , 26-30 Jan. (2004) 12
3. Trevor Schroeder, Steve Goddard, Byrav Tamamurthy: Scalable Web Server Clustering Technologies. IEEE Network, May/June (2000) 38-45
4. Rod Gamache, Rob Short, Mike Massa: Windows NT Clustering Service. IEEE Computer, Oct. (1988) 55-62
5. Valeria Cardellini, Emiliano Casalicchio, 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. Eunmi Choi, Dugki Min: A Proactive Management Framework in Active Clusters, LNCS on IWAN, December (2003)
7. Appleby, K., Goldszmidt, G., Steinder, M., "Yemanja - a layered event correlation engine for multi-domain server farms ",Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on , 2001 ,Page(s): 329 -344
8. Jeffray S. Chase: Server switching: yesterday and tomorrow. Internet Applications (2001) 114-123
9. Martin F. Arlitt, Carey L. Williamson: Internet Web Servers: Workload Characterization and Performance Implications. IEEE/ACM Transactions on Networking, Vol. 5, No. 5, October (1997) 631-645

10. Mor Harchol-Balter: Task Assignment with Unknown Duration. IEEE Distributed Computing Systems, Proceedings. (2000) 214 ?224
11. Web Bench Tool, http://www.etestinglabs.com