서비스 지향적인 효율적인 클러스터 서버 구축 및 관리

채 희 성[†] · 송 하 윤[††] · 김 한 규[††] · 이 기 철[†††]

요 약

현대의 서버 시스템은 대부분의 경우 클러스터 시스템으로 이루어지어 가능한 많은 사용자를 지원할 뿐만 아니라 가능한 많은 어플리케이션을 지원하는 것을 목적으로 하는 서비스 지향 클러스터 시스템이다. 클러스터 시스템 아키텍처의 발전으로 자바 프레임웍에 기반한 미들웨어 어프로치가 발전하고 있다. 미들웨어에 의한 방법은 서버 시스템의 성능과 어플리케이션의 활용도를 보장하면서도 서버 시스템 구축을 위한 대부분의 노력을 덜어준다.

본 연구에서는 JMX를 이용하여 클러스터 시스템의 손쉬운 구현과 관리를 달성할 수 있는 새로운 클러스터 시스템을 소개한다. 일단 클러스터 시스템의 구축과 구성단계를 보이며, 어플리케이션과 시스템 양자에 걸쳐서 손쉬운 구축과 확장, 그리고 관리가 미들웨어에 기반한 시스템에서 이루어짐을 보인다. 덧붙여 서비스 지향 클러스터 시스템이 미들웨어에 기반하여 우수한 성능을 보임을 성능 평가 실험 결과를 통하여 검증하였다. 기본적인 성능 평가 결과는 서버의 가용성, 그리고 로드 밸런싱과 스케줄링 알고리즘의 효율성을 검증하였다. 특히, 우리의 서비스기반 스케줄링 방법이 정상부하시 적재불균형문제와 과부하시의 대처능력에서 타 방법에 비해 우수함을 보였다.

키워드 : 클러스터 관리, 자바 매니지먼트 익스텐션, 미들웨어, 서비스 지향 클러스터 서버

Efficient Cluster Server Construction and Management for Service Orientation

Hee Seong Chae[†] · Ha Yoon Song[††] · Han-gyoo Kim[††] · Kee Cheol Lee[†††]

ABSTRACT

Modern server systems are usually composed in the form of cluster systems in order to serve not only as many users but also as many kinds of applications as possible. The progression of the cluster system architecture leads in a middleware approach based on the Java framework. The middleware approach alleviates the efforts for the construction and the management of a server system but still preserves its performance and applications on the server. In this research, we introduce a new clustering scheme for the easy construction and maintenance of a cluster server system with the Java Management Extensions. We first demonstrate the construction and configuration process. Our experiment sets can verify that it is easy to construct, expand and manage a middleware based cluster system as well as the applications which reside on it. In addition, we can achieve reasonable performance on our service oriented clustered system with the help of state-of-the-art middleware. The experimental results of performance demonstration contain the availability of a server, and the effectiveness of load balancing and scheduling mechanisms. Especially, our service oriented scheduling mechanism was shown to successfully manage load imbalance under the normal load and cope with the overloaded situations, compared with other known scheduling mechanisms.

Key Words : Cluster management, Java management extensions, Middleware approach, Service oriented cluster servers

1. Introduction

The rapid development of the wired and wireless network technologies has made various computing services indispensable. Any user request is transferred to the corresponding server through wired or wireless networks. As the numbers of users and the types of applications are increasing, server capacities are required to progress more and more. Generally speaking, it is not possible for a single node (processor) server to process such huge amounts and various kinds of application demands. Accordingly the internal structure of a server is required to be a cluster system with many computing nodes with connections between nodes.

Cluster systems provide four primary benefits over single larger machines: high scalability, availability,

† 준 회 원 : 홍익대 컴퓨터공학과 박사과정
†† 정 회 원 : 홍익대학교 컴퓨터공학과 부교수
††† 종신회원 : 홍익대학교 컴퓨터공학과 교수
논문접수 : 2007년 3월 30일, 심사완료 : 2007년 9월 13일

performance and cost effectiveness. And they may show high reliability in extreme situations.

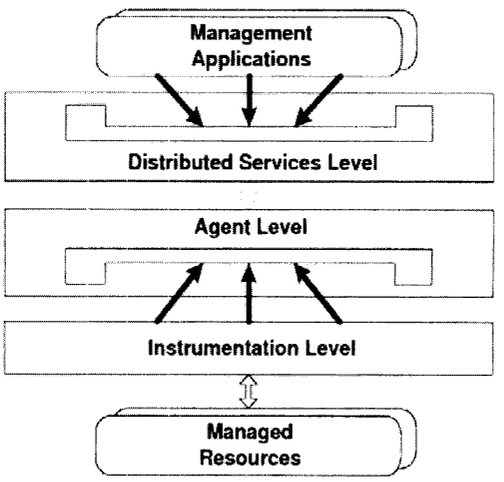
Previous studies such as [2, 3, 5, 9, 17, 18] show the characteristics of cluster systems extensively. The capabilities required for a cluster system to be a server can be summarized as follows:

-Must serve as many users as possible: User serviceability.
-Must serve as many kinds of applications as possible: Application serviceability.
-Must be easy to construct: Constructability.
-Must be easy to expand: Expandability.
-Must easily remove and recover any malfunctioning node without severe system overhead: System manageability.
-Must easily manage any server performance-related elements: Performance.
-Must easily manage applications on the server: Application manageability.

The efforts have long been made to construct the cluster systems that meet the above requirements. The system capabilities that the cluster systems must hold have already been sufficiently studied in the field of distributed computing and GRID computing. However, other than the computational GRID, a cluster system for general applications requires various capabilities. The current issue is how easily a cluster system can be built and how flexibly it can be managed. CORBA[15] has been a representative framework for these purpose, but still very out-of-date for the present construction and management requirements. Considering that the service handlers for providing various application services are being developed in Java, a Java-based management framework is required. In the cluster system environments, the Java Management Extensions(JMX) shows very efficient functionalities in managing service handlers written in Java[14, 23].

We have designed and constructed a cluster system with JMX[13] with enough care in order to meet aforementioned requirements, and we will verify in this paper that our clustered servers can fulfill these requirements. Our experiences show that the cluster systems meet all the requirements suggested. For example, two man-months were consumed for the construction of a server system, and as service requests were added, its performance remained satisfactory.

We will describe detailed issues as follows: In Section 2 we will review the Java Management Extensions. In Section 3 we will show the overall construction and configuration process of our cluster system. Section 4 will show the management details. Sections 5 and 6 will show the availability test and the effect of various loads balancing mechanism of our cluster system, respectively. We will conclude this paper in the final section.



(Fig. 1) Overall structure of JMX.

2. Related Work: Introduction to Java Management Extensions

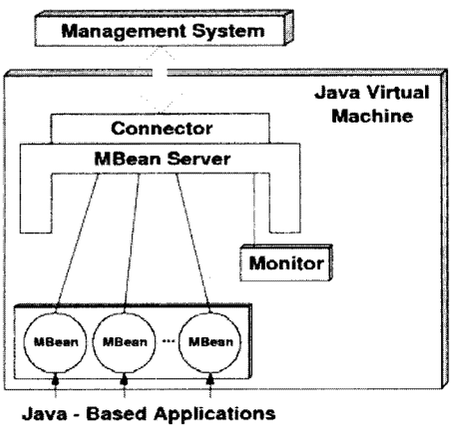
The Java Management Extensions (JMX) is a management framework, so called a middleware, suggested by Sun Microsystems[14, 21, 23]. JMX basically consists of three hierarchical levels: Instrumentation level, Agent level, and Distributed Services level. The data and application programs, which must be managed, can be separated independent of the Manager level. JMX also has a set of APIs for the management protocol. The basic structure of JMX is shown in (figure 1).

Since JMX is based on the Java technologies, it can seamlessly cooperate with other Java based technologies such as EJB, Jini, JDMK (Java Dynamic Management Kit), and so on. JMX also supports APIs for the network management such as SNMP so that every network management feature can be available. We introduced JMX since heterogeneous server architectures can be a seamless part of a cluster system only if the servers incorporate the Java technologies. The unit of the management object for JMX is an application that enables Load Balancer to distribute jobs over servers easily for the efficient load balancing. With these features a cluster

system with JMX can be regarded as a viable solution to configure and manage a cluster system based on the Java technologies. And it is true that most of the emerging applications written in Java thus require Java compatible environments[1, 10].

2.1 Instrumentation Level

The Instrumentation level provides any Java technology based object with instant manageability. This level is aimed at the entire developer community that utilizes any Java technology. This level provides the management of Java technologies which are standard across all the industries[21]. The components of this level are MBeans (Managed Beans), Notification Model, and MBean Metadata Classes. MBeans are categorized by Standard MBeans, Dynamic MBeans, Open MBeans, and Model MBeans.



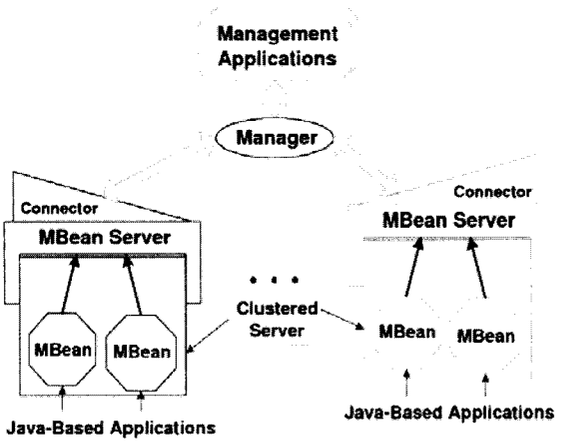
(Fig. 2) Cluster server internals.

2.2 Agent Level

The Agent level provides management Agents. JMX Agents are containers that provide core management services which can be dynamically extended by adding JMX resources. This level is aimed at the management solution development community, and provides management through the Java technologies. MBean Server and Agent Services are the core parts of this level. MBean Server, a component for MBean registration, supports a management interface for each MBean so that the management system can recognize each MBean. Agent Service is an object of the management operation for MBeans registered in the server. It has Dynamic Class Loading, Monitors, Timers, and the Relation Service.

2.3 Distributed Services Level

The so-called Manager level provides management components that can operate as Manager or Agents for the distribution and consolidation of management services. This level is aimed at the management solution development community, and completes the management through Java technologies provided by the Agent level. Manager and Agents can communicate through the adapters with management protocol APIs, or via a connector client to contact the connector server in the Agent. Available APIs are SNMP, IIOP protocol adaptor, and WBEM[4, 7, 20, 23].



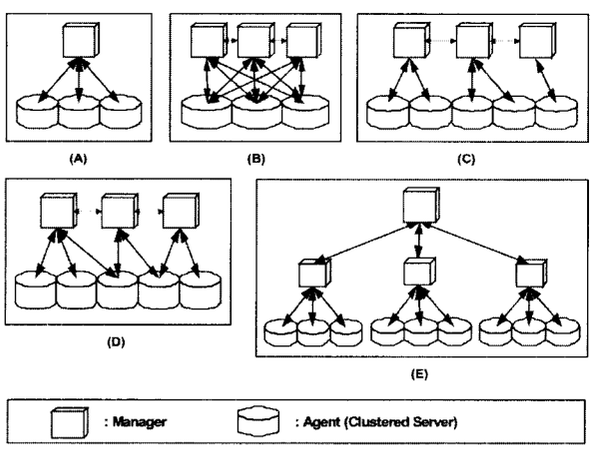
(Fig. 3) Conceptual structure of cluster.

3. Cluster System: Construction then Configuration

In this section, we will show the architecture to build up a cluster system and then present a cluster configuration. We used JMX as a middleware for our cluster system construction. The manageable object for JMX is an application program based on the Java technologies such as EJB (Enterprise Java Beans). Each application program will be mapped on an *MBean*, and each MBean will be registered on the MBean server that resides on the same Java virtual machine where the application program is under execution. In addition, the Monitor of this MBean server (one of the Agent Services) manages the applications with their status information. The Manager system can recognize and resolve any erroneous situation by the reception of an event from Monitor whenever Monitor senses an exceptional situation such as the halt of an MBean or overloading of an MBean, etc. For this purpose, Agent should be able to communicate with the Manager system. The communication can be made by APIs for SNMP[19, 20]. Alternatively, the connector server on Agent and the connector client on the Manager system can build communication between Agent and the Manager system.

For the actual system implementation, the second communication scheme was used.

(Figure 2) shows the inside of the server, which is based on EJB and managed by JMX. (Figure 3) shows the overall structure of the cluster system in the simplest configuration. There is one Manager in the system, which manages the whole cluster system. This simple structure can be used to configure the cluster system without any performance degradation. However, the SPoF (Single Point of Failure) problem can occur whenever the Manager system becomes faulty, which is an example of a fatal situation with very low availability[11]. However, our approach allows advanced configuration of a cluster system as the following subsection shows.

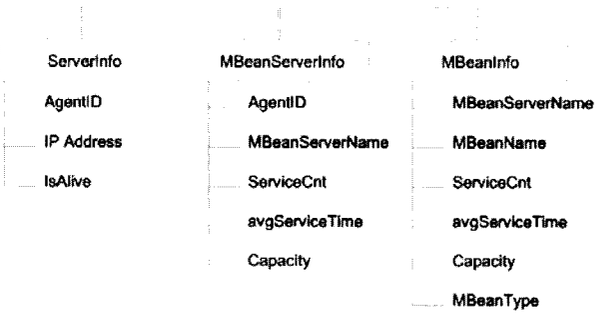


(Fig. 4) Alternative configuration of cluster.

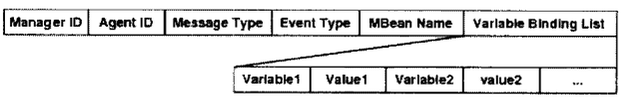
3.1 Dynamic Configuration of the Cluster

In (figure 4), five other configurations are suggested for dynamic clustering. Each configuration has its own cons and pros, which are trade-offs between the availability and resources usage efficiency.

(A) is the basic architecture model discussed in (figure 3), bearing the possibility of SPoF problem. (B), (C), and (D) are alternative architectural models to solve the possible SPoF problem. Because the manager of a conventional cluster system is only a load balancer, the extension to the multiple manager scheme such as (B), (C) and (D) has been rarely discussed[6, 8, 12]. On the other hand, our approach is ready for the extension to (B), (C) and (D), and a multiple manager scheme can be a solution to the SPoF problem. Several Manager systems are employed to guarantee the basic fault tolerance in case of the failure of a Manager system. However, these models have the problems such as the redundancy of management information and the high communication overhead between servers and Managers, which may cause the resource extravagance. (E) shows a hierarchical model with a topmost Manager system that manages each sub-Manager system. This model is much more scalable than the other models, but the extra hierarchy requires additional computing and communication resources. In addition, the sub-Manager system must be extended in a way that the nodes with the sub-Manager can bear both Manager and the top level of Agent.



(Fig. 5.) Structure of management of objects.



(Fig. 6) Message format

3.2 Managed Objects

Manager manages three sorts of objects: ServerInfo, MBeanServerInfo, and MBeanInfo. ServerInfo has the information of a node where Agent resides. MBeanServerInfo has the information of the MBean server created by the Agent. MBeanInfo stands for the management information of each MBean registered on the MBean server. (Figure 5) shows the structure of the Manageable object and attributes of each object. We omit detailed explanation about the attributes since attributes of each object are self descriptive.

3.3 Message Formats and Types

(Figure 6) shows the message format for the communication between Manager and Agents defined for cluster management in this paper. <Table I> shows the descriptions of each message format and the six message types defined in this paper.

3.4 Cluster System Implementation

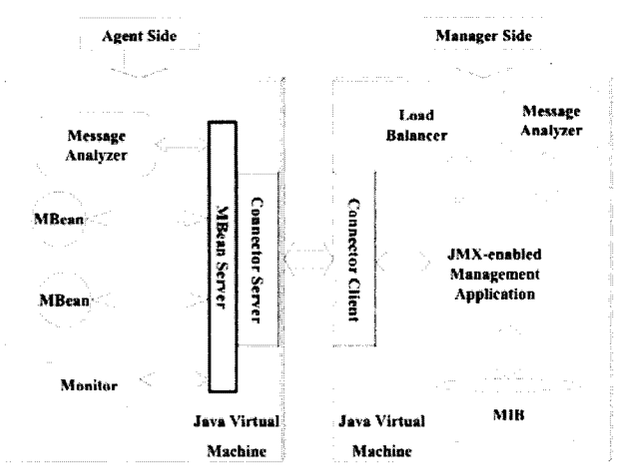
The actual cluster system suggested in this paper was implemented and tested. The configuration is basically similar to (figure 4-(A)). The implemented testbed is composed of normal computers (i.e., without Manager) and computers with Manager. Each node has heterogeneous

〈Table I〉 Message format and type descriptions

Message Format Descriptions		Message Type Descriptions	
ManagerID	The identifier of Manager for communication.	MBean Information Request	Manager's request of MBean information managed by Agent.
AgentID	The identifier of Agent for communication.	MBean Information Transfer	Agent's reply of MBean information requested by Manager.
Message Type (with six sub-fields)	Message type is one of the followings. (1) MBean information request (2) MBean information transfer (3) Event notification (4) MBean server status information (5) MBean server status information transfer (6) IsAlive	Event Notification	Transfer of an event from a MBean or the MBean Server to Manager.
EventType (thrown by a MBean or the MBean server)	Event type is one of the followings. (1) MBean Capacity Full (2) MBean Server Capacity Full (3) MBean Capacity Free (4) MBean Server Capacity Free	MBean Server Status Information Request	Manager's request of the status of a MBean server to Agent.
MBean Name	The record of MBeanName.	MBean Server Status Information Transfer	Agent's transfer the MBean server status information.
Variable Binding List	Contains the MBean or MBean server information.	IsAlive	Manager's check the living status of nodes.

processors and operating systems (e.g., Windows 2000 Server, Windows 2000 Professional, and Linux). The Manager server has a connector client for the communication with Agent. The detailed component diagram is shown in (figure 7).

The Manager server obtains management information and notifies it to the management application. Finally the administrator acquires the overall status of the cluster system. Each normal node executes its user applications. Each application is mapped on an MBean and registered on the MBean server. For the implementation for this paper, each node has one MBean server to manage MBeans. The MBean server registers MBeans and Agent services that manage application programs. Among various Agent services, Monitor plays the key role to sense erroneous situations and to throw events for the notification of errors. Whenever each application starts a new service for a client, a new MBean will be created and registered to the MBean server. Monitor checks the number of MBeans on the server or other related information specific to MBeans, and throws an situation-related event so that the whole system recognizes the error occurrence.



(Fig. 7) Implementation details.

4. Cluster Management

With the JMX based cluster system technologies, a cluster system can cope with various situations requiring management as fluent as possible. The most representative situations have been researched and can be distinguished. We will see the management solution for each case one by one. With the support of JMX, it is very easy to complete these solutions.

We choose the *polling* scheme from Manager to nodes intentionally since we can alleviate the extra load of management to a node. An interrupt scheme from a node to Manager can burden computational nodes since an interrupt is a kind of an active mechanism to the node. Therefore we choose a passive mechanism of polling from the view of a node. We can also control the period of polling by simply trimming the polling timer on one Manager instead of the individual interrupt timers on

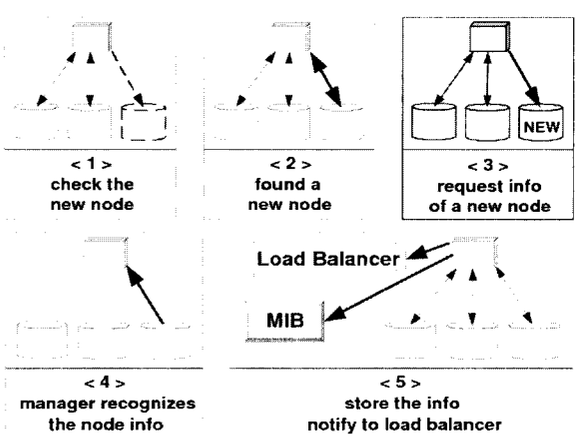
multiple nodes. This simplified scheme works very well in our experiments and can strengthen a node's serviceability.

-New server addition: a new server can be added anytime.
-Server removal: any server can be removed.
-Server overload management: the server overloaded by the concentration of related requests must be managed.
-Server failure management: the faulty server must be removed.

4.1 Addition of a new node : (Figure 8)

Servers can be added anytime to cope with the concentrated user requests. Manager does a major role for the addition mechanism in cooperation with Load Balancer.

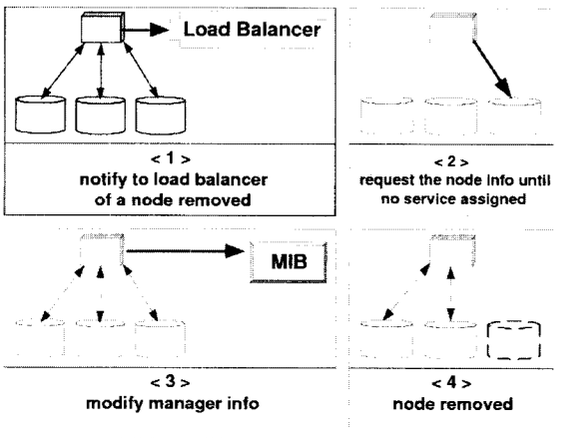
(1) Manager: checks the aliveness of its nodes and their management record by sending IsAlive message. A new node must reply.
(2) Manager: finds a new node starting, assigns a new ID to the Agent of the node, and sends a message requesting the information of MBeans and the MBean server on the server.
(3) Manager: records the information received from the server, modifies the management information on MIB of the node, and notifies the addition of the new node to Load Balancer.
(4) Load Balancer: starts sending jobs to the added node.



(Fig. 8) Addition of a new node

4.2 Removal of an existing node : (Figure 9)

Any node can be removed anytime for the management purpose by server administrator. Manager does a main role.

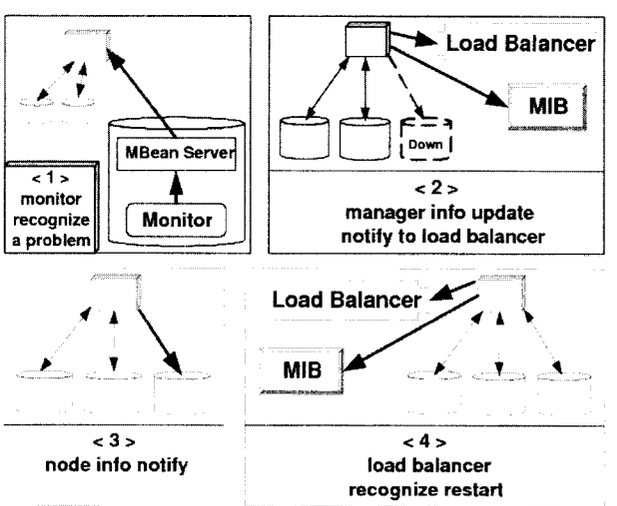


(Fig. 9) Removal of an existing node

(1) Administrator: sends the information of a to-be-deleted node to Manager.
(2) Manager: notifies Load Balancer that the node is out of service - no more jobs can be assigned.
(3) Manager: requests the information of MBeans to the node to check if the node has jobs on processing.
(4) The requests of the MBean information are repeated until there are no jobs on the node.
(5) Manager: updates the management information of the node.
(6) The node can be physically deleted.

4.3 Management of an overloaded node (load balancing) : (Figure 10)

An overloaded server must be handled in order to maximize the overall server performance. Otherwise, an overloaded node will be a critical bottleneck for the server system and maybe a faulty one in final.



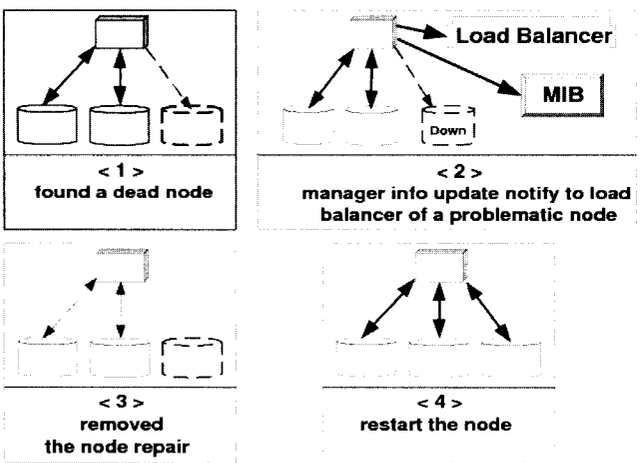
(Fig. 10) Handle of an overloaded node

(1) Situation: Either the MBean capacity or the MBeanServer capacity on a node exceeds its predefined capacity.
(2) Monitor: recognizes an overloading situation and notifies to Manager through an event.
(3) Manager: informs Load Balancer that the node cannot receive any more service request.
(4) Manager: requests ServiceCnt information to check if other nodes can process more jobs. Each node has its own avgServiceTime, and must check if it can reduce the avgServiceTime. Nodes having ServiceCnt less than their capacity can process additional requests.
(5) Manager: notifies Load Balancer that these nodes are available to serve additional requests.

4.4 Management of a node failure : (Figure 11)

A failed node must be removed and/or replaced to keep the overall server performance.

(1) Monitor: checks the aliveness of nodes by sending an IsAlive message periodically, and reports to Manager.
(2) Manager: notifies Load Balancer that a node is halted.
(3) Load Balancer: stops the assignment of new jobs to the node.
(4) Manager: notifies Administrator.
(5) Administrator: removes the node from the cluster.



(Fig. 11) Handle of a node failure.

5. Availability Verification

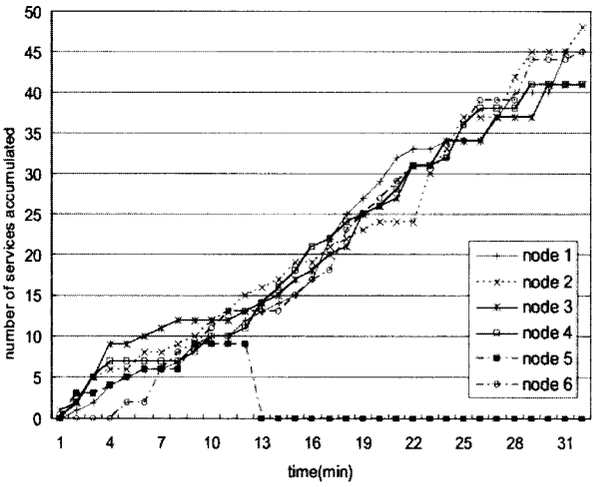
How the system responds for the construction and management of a cluster system will be eventually reflected in the load balancing performance. Availability can be achieved by the efficiency of load balancing in our cluster server. Load balancer of Manager in (figure 7) does basic load balancing based on MBeanServerInfo.ServiceCnt which is an attribute of management objects. A new request will be assigned by the load balancer to a node with the smallest MBeanServerInfo.ServiceCnt. We call it a Service-Oriented (SO) scheduling since it does not sacrifice resources in order to collect the current status of clustered nodes. Instead, every possible system resource is designed to devote to the service for user requests.

We designed various situations to check the availability of the overall server cluster. We will show three representative results in the following subsections. The graph of time versus the accumulated number of allotted jobs shows the load balancing results. In case a node is removed or down, the accumulated number for the node is set to zero to indicate the operation of the node stopped. These experiments are designed for each node with tens of jobs alloted and suppose overloaded situations with the duration of less than an hour. In order to verify the availability and load balancing (containing scheduling) respectively, we designed primitive experiments for each category. We believe that this approach can verify the performance of our cluster system in each performance category.

Other complicated load balancing and scheduling schemes incorporated with our cluster server will be discussed and the results will be demonstrated in Section 6.

5.1 Case I

(Figure 12) shows the experimental results for the most common case. A new node 6 was added at minute 3, and



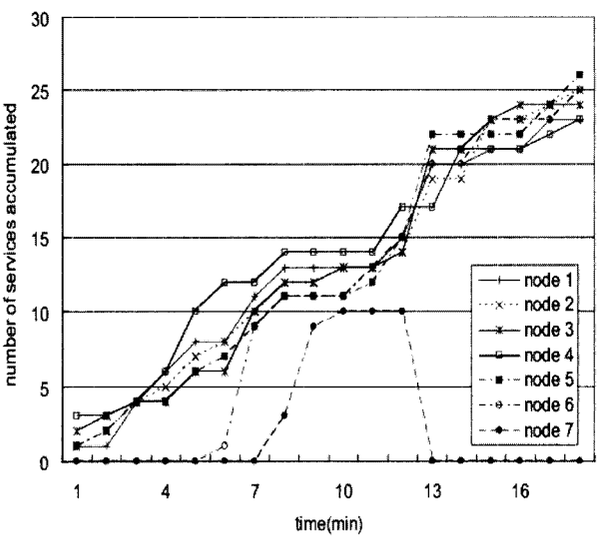
(Fig. 12) one node added and one node removed

jobs started to be assigned to the node between minutes 4 and 5. At minute 9, node 5 was ordered to be removed, and instantly no new jobs were allotted to the node, and all the remaining jobs in the node were serviced, and the node was eventually removed between minutes 12 and 13. We need to look at what happened to the other nodes. Especially for node 3, its application or MBean node processing power was exceeded between minutes 8 and 12, and also frequently exceeded after minute 22, resulting in no request allotment for the node. We also need to note minutes 4 to 7. The job allotments to a new node became busy because it had less requests being processed. Contrarily, the allotments to the other nodes slowed down.

5.2 Case II

(Figure 13) is for the case of two nodes added and one node removed. In addition, the processing capacities of the applications and MBean nodes of each node were set to a small number. It can be done by setting MBeanServerInfo.Capacity and MBeanInfo.Capacity and the number of MBeans on a node to small numbers. In other words, the overall server capacity was intentionally set to low in order to simulate the overload situation.

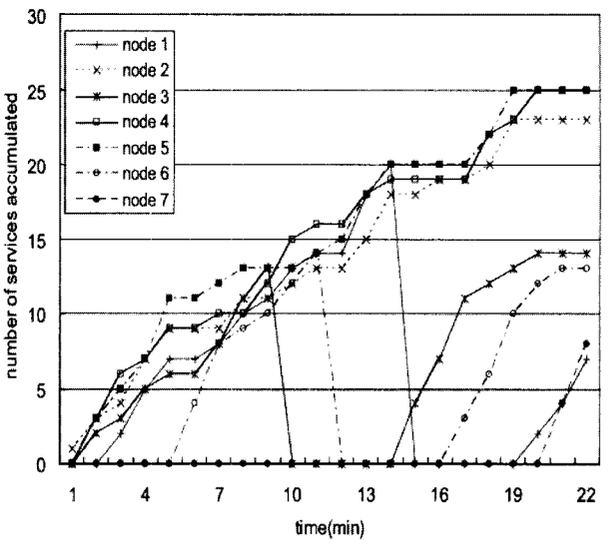
Two nodes were added between minutes 3 and 5. For the next 3 minutes, those two nodes started to be allotted but almost no jobs were allotted to the other nodes. At minute 10, node 7 was ordered to be removed, and no new jobs were allotted to it. After it was completely removed at minute 13, new job allotments to each node were delayed for 2 or 3 minutes because of limited node capacities and a small number of active nodes.



(Fig. 13) Two nodes added and one node removed

5.3 Case III

(Figure 14) shows the results for a more special condition. Three nodes got down, and recovered and added back to the cluster. Additionally two nodes were added at minutes 3 and 17, respectively, and they two started to be allotted 2 or 3 minutes later, limiting the allotments to the other nodes for the time being. Three nodes got down at minutes 9, 11, and 14, and added back to the cluster at minutes 10, 12, and 15, respectively. As a node got down, this figure clearly shows that more jobs were allotted to the other nodes. As a node was newly added or added back, the job allotments to the other nodes were shown to be smoothened.



(Fig. 14) Three nodes got down, three nodes recovered, and two nodes added.

6. Load Balancing and Scheduling

Second set of experiments was carried out to prove that the scheduling of our method works well. The Load Balancer distributes jobs over cluster system with a specific scheduling policy. As described in Section 4, the cluster management and load balancer can handle the load balancing between nodes with communications to and from the JMX manager as a part of the cluster management (subsection 4.3). In this section, our service-oriented (SO) scheduling mechanism was compared with the Round-Robin(RR) scheduling[16] and the Least-Connection(LC) scheduling[24].

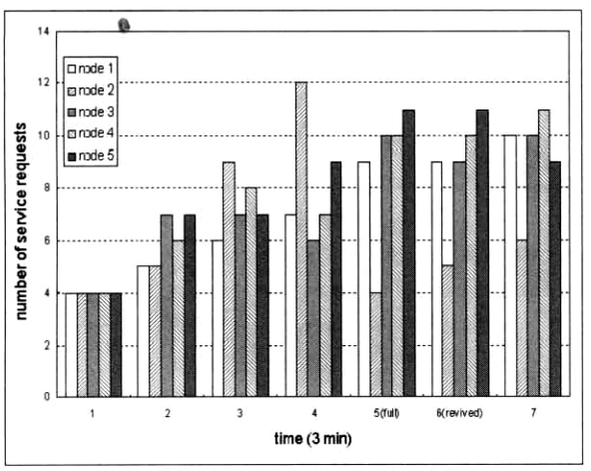
The brief description of SO scheduling is based on the descriptions of other scheduling mechanisms. In RR scheduling, the status of each node is not considered, and jobs are distributed to nodes in turn. In LC scheduling,

for each service request, Load Balancer sends a notification of new service request to the manager system that collects the most recent node information from each node (containing ServiceCnt information), and sends the latest node information back to Load Balancer. Accordingly, LC scheduling assigns a job to the node with least jobs.

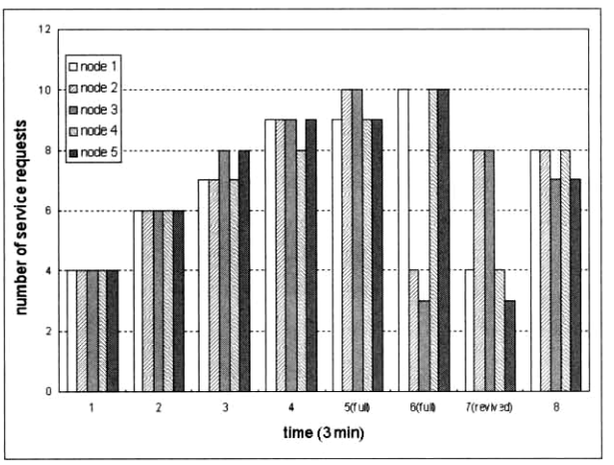
In our SO scheduling, when the manager system receives a request from Load Balancer, it does not collect the latest information of nodes. Instead, it just sends its current, already collected, information of the nodes in MIB back to Load Balancer. RR scheduling does not utilize yet the existing information while LC scheduling requires extra network traffic in order to bring up the up-to-date status of each node. SO scheduling utilizes relatively recent and already collected information only, not sacrificing resources and time to collect the most recent cluster status. Thus, SO scheduling shows the asymptotically optimized performance of LC scheduling but with low scheduling overhead.

In these experiments, the same service requests were generated for each of the scheduling methods, and the number of jobs being processed by each node was checked every three minutes. Service request profiles are designed to cause load imbalance with relatively big jobs and a small number of jobs in short duration of service time.

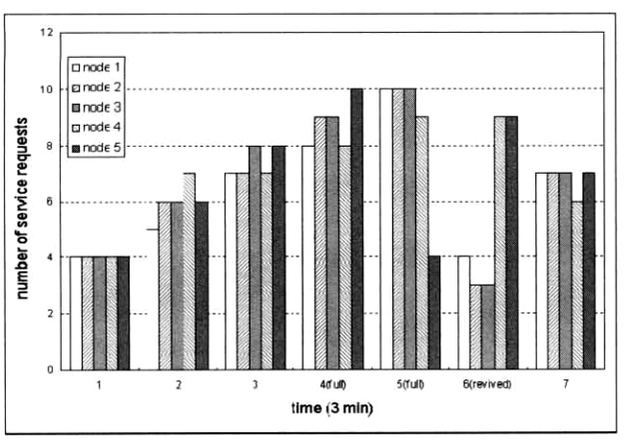
Figures 15 to 17 show the experimental results for an extreme situation where specific nodes reach their processing limits, 10 requests per node in these experiments. In RR scheduling, as shown in (figure 15), once some limitation is reached (more than 10 requests per node at time 4), the numbers of jobs assigned are not balanced among nodes. On the contrary, in our service oriented scheduling in (figure 16) and LC scheduling as shown in (figure 17), right after some node limitation is reached, it can be confirmed that the numbers of jobs get almost equally distributed among nodes.



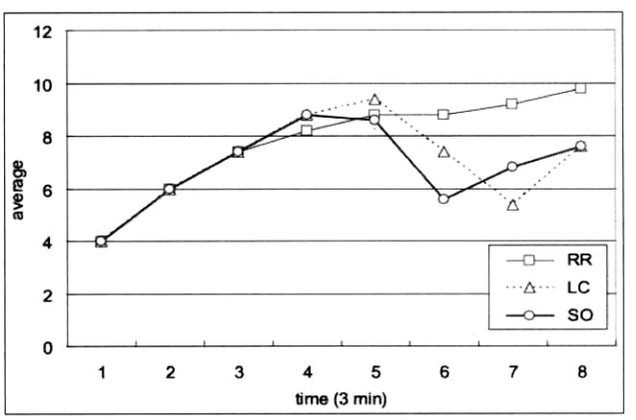
(Fig. 15) Round-Robin Scheduling



(Fig. 17) Least-Connection Scheduling



(Fig. 16) Service-Oriented Scheduling

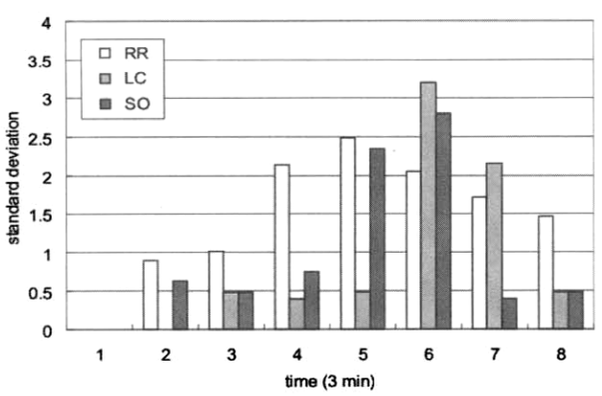


(Fig. 18) Average Number of Jobs per Node

Considering that our simple, the service oriented scheduling method can save most of the information collection and network traffic time which LC method must use, our method seems very attractive in terms of scheduling time and efficiency. We investigated the average number of allotted jobs per node and the standard deviation of number of allotted jobs per node.

The average number of allotted requests per node is depicted in (figure 18). Under the normal load, three load balancing mechanisms show the similar responses. Under the overload situation after time 4, SO scheduling reacts instantaneously to reduce the average node loads, while RR scheduling remains the same and LC scheduling shows the delayed reaction. These results show that the low overhead of SO scheduling mechanism leads in the prompt load balancing in the overloaded situations.

(Figure 19) shows the standard deviation of number of allotted requests per node. The standard deviation of number of allotted jobs per node stands for the measure of load imbalance. We can verify that RR scheduling does not concern about load balance while two others consistently concern about load balancing. The LC scheduling can minimize the load imbalance under the normal load but it shows abrupt and prolonged load imbalance under the overload situation. Our SO scheduling successfully manages load imbalance under the normal load and successfully copes with the overloaded situations. SO scheduling successfully recognizes and suppresses the load imbalance instantly and distributes loads over time to smoothen the overloaded requests.



(Fig. 19) Standard Deviation of Number of Jobs per Node

7. Conclusion and Future Works

In this paper we demonstrated the fast construction, the flexible configuration, and the fluent management of a cluster system with JMX middleware, which is an up-to-date Java based resource management framework.

- -Fast construction: it takes only two man-month to construct our cluster server system.
- -Flexible configuration: the configuration of a cluster system is easy and service oriented.
- -Fluent management: our system can also dynamically cope with the various situations of a cluster management for the high manageability and availability with very reasonable performance.

For the availability issues, the several scenarios of the cluster management and application processing were introduced and the availability of our server system was tested under each scenario. Since the availability highly depends on the load balancing mechanism of the cluster, the load balancing issues have been further studied with different scheduling mechanisms. It showed our service oriented load balancing approach was eligible compared to other complicated mechanisms.

However, the JMX specification still has several limitations for the cluster system managements. For example, one Agent cannot have multiple MBean servers. We hope that the next version of JMX will be improved so that the much dedicated management can be available by multiple MBean servers on one Agent for better clustering environments.

Apart from the suggested cluster system, which is based on LAN environments, WAN (Wide Area Network)-based cluster systems can be studied in the future. The ultimate integration of worldwide networks will lead to a new topic of the global management. One of the examples can be the technology that enables Beans based on the Java technologies, especially the EJB technology, to be managed in the global environments. For the global management, the naming of Beans or servers will be of another issue. We expect Jini will be one of the possible solutions[22].

References

[1] J. Batheja and M. Parashar, "Adaptive cluster computing using javaspaces," IEEE International Conference on Cluster Computing, pp.323-330, 2001.

[2] E. A. Brewer, "Lessons from giant-scale services," IEEE Trans. Internet Computing, Vol.5, No.4(July/Aug.), pp.46-55, 2001.

[3] E. V. Carrera and R. Biancini, "Efficiency vs. portability in cluster-based network servers," Proceedings of the 8th Symposium on Principles and Practice of Parallel Programming, Snowbird, UT, 2001.

[4] J. D. Case, M. Fedor, M. L. Schoffstall, and C. Davin, "Simple Network management Protocol ({SNMP})," Internet RFC 1157, 1990.

[5] G. Chen, C. Wang, and F. Lau, "A scalable cluster-based web server with cooperative caching support," Computation and Currency: Practice and Experience 15, 7-8 (June/July), pp.681-705, 2003.

[6] B. Elbert and B. Martyna, 'Client/Server Computing,' Artech House, 1994.

[7] J. A. Farrell and H. Kreger, "Web services management approaches," IBM Systems Journal, Vol.41, No.2, 2002.

[8] S. Goddard and T. Schoeder, "The sasha architecture for network-clustered web servers," High Assurance Systems Engineering, 2001.

[9] S. D. Gribble, E. A. Brewer, J. M. Hellerstein, and D. Culler, "Scalable, distributed data structures for internet service construction," Proceedings of the 4th USENIX Symposium on Operating Systems Design and Implementation (OSDI2000), 2000.

[10] K. A. Hawick and H. A. James, "Dynamic cluster configuration and management using javaspaces," IEEE International Conference on Cluster Computing. pp.145-148, 2001.

[11] S. Horman, "Creating linux web farms (linux high availability and scalability)," Tech. Rep., http://www/vergenet.net/linux/has/, Nov. 2000.

[12] D. Ingram, S. Shrivastava, and F. Panzieri, "Constructing dependable web services," IEEE Trans. Internet Computing, Vol4, No.1(Jan./Feb.), pp.25-33, 2000.

[13] H. Kim, H. Y. Song, and K. C. Lee, "Dynamic configuration and management of clustered system with JMX," Lecture Notes in Computer Science 2662, pp.858-867, 2003.

[14] H. Kreger, "Java management extensions for application management," IBM Systems Journal, Vol.40, Vol.1, 2001.

[15] D. S. Linthicum, "CORBA 2.0?," Open Computing, Vol.12, No.2(Feb.), 68-, 1995.

[16] J. B. Nagle, "On packet switches with infinite storage," IEEE Trans. Communications, Vol.35, No.4(Apr.), pp.435-438, 1987.

[17] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum, "Locality-aware request distribution in cluster-based network servers," Proceedings of the 8th ACM Conference on Architectural Support for Programming Languages and Operating Systems. San Jose, CA, pp.205-216, 1998.

[18] Y. Saito, B. N. Bershad, and H. M. Levy,, "Manageability, availability and performance in porcupine : A highly scalable internet mail service," Proceedings of the 17th ACM Symposium on Operating Systems Principles, Charlston, SC, 1999.

[19] W. Stalling, 'SNMP, SNMP v2, and CMIP,' Addison Wesley, 1993.

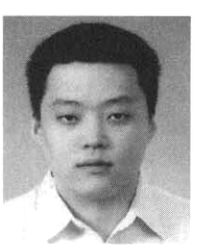
[20] Sun-Microsystems, 'Java management extensions SNMP manager APIs,' 1999a.

[21] Sun-Microsystems, 'JMX white paper,' 1999b.

[22] Sun-Microsystems, 'Jini architecture specification,' Vol.1, No.2, 2001

[23] Sun-Microsystems, 'Java management extensions specifications,' Vol.1, No.4, 2003.

[24] W. Zhang, "Linux virtual server for scalable network services," Ottawa Linux Symposium, 2000.



채 희 성

e-mail : winkatu@empal.com
2001년 홍익대 컴퓨터공학과(학사)
2003년 홍익대 컴퓨터공학과(석사)
2005년-현재 홍익대 컴퓨터공학과 박사과정
관심분야 : 웹 학습, 분산 시스템

송 하 윤

e-mail : hayoon@wow.hongik.ac.kr
1991년 서울대학교 계산통계학과(학사)
1993년 서울대학교 전산과학과(석사)
2001년 University of California - Los Angeles Computer Science Department (박사)
2001년~현재 홍익대학교 컴퓨터공학과 부교수
관심분야 : 분산 센서 시스템, 고고도 네트워킹

김 한 규

e-mail : hkim@ximeta.com
1981년 서울대학교 기계공학과(학사)
1984년 서울대학교 기계공학과(석사)
1994년 University of California-Berkeley 전산학과(박사)
1994년~현재 홍익대학교 컴퓨터공학과 부교수
관심분야 : 네트워크, 분산 시스템

이 기 철

e-mail : klee1@hongik.ac.kr
1977년 서울대학교 전자공학과(학사)
1979년 한국과학원 전산학과(석사)
1987년 University of Wisconsin-Madison 전기 및 컴퓨터공학과(박사)
1989년~현재 홍익대학교 컴퓨터공학과 교수
관심분야 : 기계학습, 시스템프로그래밍 등