

CATALINA: A SMART APPLICATION CONTROL AND MANAGEMENT ENVIRONMENT

Salim Hariri, Muhamad Djunaedi, Yoonhee Kim, Rinda P. Nellipudi,
Ashok K. Rajagopalan, Prasad Vadlamani, Yeliang Zhang
High Performance Distributed Computing Laboratory
Department of Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85721

C. S. Raghavendra
Department of Electrical Engineering Systems
University of Southern California
Los Angeles, CA 90089

ABSTRACT

Management of large-scale parallel and distributed applications is an extremely complex task due to factors such as centralized management architectures, lack of coordination and compatibility among heterogeneous network management systems, and dynamic characteristics of networks and application requirements. The development of an integrated network management framework that is proactive, scaleable and robust is a challenging research problem. In this paper, we present our approach to implement a smart application control and management (CATALINA) that provides dynamically programmable control and management services to support the development and deployment of smart (intelligent) applications. The CATALINA environment provides the application developers with all the tools required to specify the appropriate control and management schemes to maintain any quality of service requirement or application attribute/functionality (e.g., performance, fault, security, etc.) and the core active management services to enable the efficient proactive management of a wide range of network applications. In this paper, we present our implementation approach and services of CATALINA environment to manage the performance, security, and fault tolerance execution of smart applications.

1. Introduction

The emerging information networking services will require the underlying infrastructure to achieve high performance access to advanced information processing, storage, intelligent information management, and value-added services such as security and network related services. These information services have different bandwidth requirements (e.g., voice bandwidth is in Kbps while high-quality video is in Mbps) and require to be transmitted for different types of users over heterogeneous networks that employ different transmission technologies (earth, air or space links), network protocols (TCP/IP, OSI, proprietary, military). The management and control of the wide range of information services (such as voice, data, imagery and video), each with different traffic characteristics and with different security, throughput, delay, reliability, and bandwidth requirements is challenging. The focus of our research is on the development of an open and scaleable framework that provides active and intelligent control and management of information networking resources and services.

Network management products have taken a long road to the current state. They started with the Simple Network Management Protocol (SNMP) [Case90], followed by the Management Information Base (MIB)[McCl90], and finally Remote Monitoring (RMON)[Wald95]. Recently, there has been an intensive effort to use web-based technologies (JMAPI and WBEM) to build network management tools [JMAPI, Thom98]. The current limitations and problems with network management are: 1) most network management systems collect management information about network events and not much about computing systems and processes; 2) they do not provide good support to interpret and diagnose the large volume of collected management information; 3) the control and management functions are mainly isolated and not intelligent; and 4) most of management functions are manually executed by the system administrators; 5) The current systems are based on a platform-centered paradigm that separates applications from the required data and service. This centralized paradigm has severe limitations when applied to manage in real-time the emerging large scale, complex multi-domain networks. These limitations make the management services slow, not scaleable, inefficient to manage large-scale networks and their applications. There is an acute need for a framework to control and manage network resources, performance, and security for application-centric management.

Mobile agents are programs that can be dispatched and executed remotely under local or remote control. There are two general approaches identified for agent-based service architecture: remote execution (or smart network) and migration (or smart message).

With these agent-based approaches, services can be provided instantly, customized, and distributed [Mage96a, Mage96b]. In [Gold95], generalized scripted and delegated agents for management has been proposed. The mobile agents are sent to remote sites where they are incorporated into

the local network management program and are used for intelligent tasks like MIB filtering. There are two general approaches to moving agents between network nodes which involves transferring different components of an agent's program context either automatically or manually by the agent programmer [Peine 97]. Some agent implementations have been proposed and developed to manage various domains such as ATM connection management [Hall98], and managing mobile connection based on client-server model [Saha97]. The use of mobile agents to achieve effective and efficient network management is becoming increasingly important.

The organization of the paper is as follows. In Section 2, we give an overview of the CATALINA architecture and implementation approach. In Section 3, we describe in further detail our approach to implement the active performance, fault and security services. In Section 4, we provide our summary and concluding remarks.

2. CATALINA Architecture

The architecture of CATALINA is shown in Figure 1. The Application Management Editor (AME) tool provides application developers with the services required specifying and characterizing the application requirements in terms of performance, fault, security, and also specifying the appropriate management scheme to maintain the smart application requirements. Once the application management requirements are defined using the AME, the next step is to utilize the management services provided by the Management Computing System (MCS) to build the appropriate application execution environment that can dynamically control the allocated resources to maintain the application requirements during the application execution. The MCS assigns one Application Delegated Manager (ADM) to manage one or more application attributes (performance, fault, security, etc.). We can perceive ADM as the operating system of the distributed application running in the system. For each task in the application, the ADM launches an appropriate Task Agent (TA) to monitor and manage the task execution. The TA monitors the task execution using appropriate task sensors and intervenes whenever the task execution on the assigned machine can not meet its requirements using the task actuators that can suspend, save task execution state, or migrate the task execution to another remote machine. Our approach supports several strategies to maintain each task attribute. For example, to manage the task performance, ADM could use active redundancy, passive redundancy, or by migrating the task execution to a faster machine when the assigned machine becomes heavily loaded. The appropriate management scheme can be selected at runtime depending on the system state and the current available resources as will be discussed in further detail later.

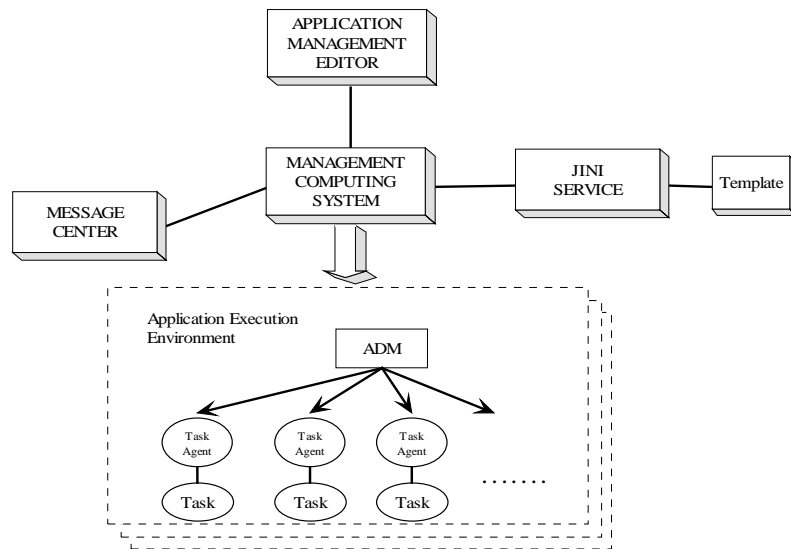


Figure 1. CATALINA Architecture

To get the appropriate application execution environment, MCS utilizes JINI Service which provides several functionality that optimizes our architecture e.g. lookup service, which is used to discover any services registered and also capability of publishing any services available during the execution. Our service is in the form of a template, which is the blueprint of the application execution environment. The template provides the skeleton of application attributes supported in CATALINA (performance, fault, security, load balancing, etc.). To coordinate the communication between the different modules in the architecture, we provide Message Center (MC), which uses the concept of distributed-shared memory, refer to the Figure 2. Every message exchange between modules is performed through MC. In MC, every task is assigned a port which acts as mailbox. Every message directed to a task is placed on this mailbox. The concept of Message Center will be discussed in further detail later.

2.1 Application Management Editor

Application Management Editor provides the application developers with all the tools required to specify the appropriate control and management schemes to maintain any quality of service requirement or application attribute/functionality (e.g. performance, fault, security, etc.). AME graphical user interface provides menu-driven task libraries that are grouped in terms of their functionality (e.g. matrix algebra library, command and control-task library, etc.) and capabilities of the task configuration. Visualization of the

environment can be achieved by constructing a model using the available libraries. The information in each node keeps tracks of the node security and gives updates periodically.

2.2 Messaging in CATALINA

The agent-to-agent and ADM-to-agent communication is based on message passing, which has been adopted from the Mach operating system. Each task is assigned a port in the message center and has the ability to queue an ordered list of messages. Each task is assigned a capability list, which lists the permissions to send and receive messages from other processes. The capability lists of all the tasks are stored in the message center. If T1 wants to send a message to T2, the message is sent to the message center daemon, which checks the validity in the capability list of T1 and queue it in the port of T2. When T2 requires a message it sends a request to the message center daemon which sends the first message from the port of T2. T2 will be in blocking mode until it receives the message. Once a message has been delivered a backup is made in the database.

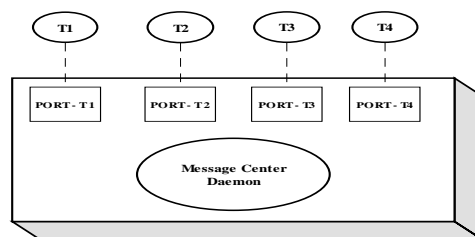


Figure 2. Message Center

3.1 CATALINA Adaptation Strategy

For reasons of dealing with the dynamic computing environment, we formulate an adaptation strategy which is the common procedure to be followed by each application attributes. In our architecture, this strategy is basically the main management activity of the Task Agent. There are three types of strategies: Change_Detection, Analysis_Verification, and Adaptation_Plan.

The Change_Detection procedure is responsible for detecting the conditions in which the monitored tasks deviates from the acceptable behavior or operation (e.g., the task performance degrades severely due to bursty traffic conditions, or due to software or hardware failures). The Analysis_Verification algorithm is invoked whenever a change is detected and to make sure that the change is real and not due to false alarms. This algorithm needs the knowledge to verify the encountered situation. Once the

change event is verified and its type is identified, the Adaptation Plan procedure is invoked to execute the appropriate adaptation scheme. To support this procedure, Task Agent utilizes a module, which provides the capabilities to suspend and resume a task execution at particular state. This procedure also incorporates the migration of task to another machine.

3.2 Checkpointing Scheme in CATALINA

CATALINA provides a user-level checkpointing library which when appended to the task, provides checkpointing and rollback recovery features. The checkpointing mechanism utilizes the shared memory concept. Checkpointing typically allows long-running tasks to save state at regular intervals or at particular stages of computing, so that they could be restarted if any interruptions, like transient failures occur to the task or if the machine, the task is running on, fails. To make the system fault-tolerant, each task, t , takes a checkpoint where the state is saved in a globally shared stable memory. So when the task t is faulty or the node/machine on which the task resides on fails, t is rolled back to the checkpoint and then the computation on t is restored.

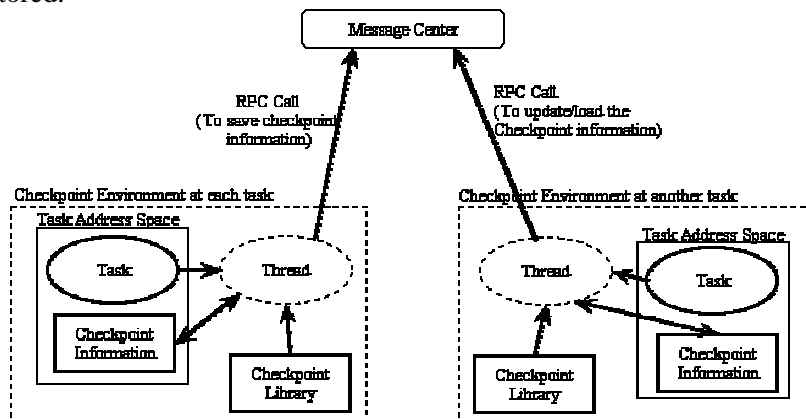


Figure 3. Model of the checkpointing scheme and working

The basic operation of the checkpointing libraries is to spin off a thread for each task. Each task typically creates a shared memory segment and the thread shares the shared memory segment with the task (refer Figure 3). So the checkpoint library threads are responsible for constantly reading the task state information and also writing this information to a central location. Multiple user-level threads to handle the various features of checkpointing cannot take advantage of a symmetric multiprocessor (SMP) because the kernel is not aware of the threads [Diet99], so we use straightforward single-thread user-level libraries. Checkpoint library also provides functions to

support communication (using RPC) with the central location (Message Center). The checkpoint library is provided in C language syntax.

Since the first piece of code in the application program/task is the checkpoint code, this is the first piece of code that will be executed. The function call *cat_create_SM()* is used to create the shared memory, that will be shared between the task and the thread to be created. *cat_create_thread()* is the function call to create a thread and attach it to the shared memory. The actual memory segment to be shared is determined by a key and during attaching of the thread to the shared memory, the key is used to identify the segment. It also invokes the checkpointing functions and makes a RPC function call to the message center to determine the last checkpoint information and update the shared memory accordingly. Until this process is done, the application program/task is made to wait. *cat_control()* is the function which receives control from the thread to proceed in the application program/task. Any change in the variables (which is in the shared memory) can be read and monitored accordingly. Periodic RPC are made by the thread to the message center to update the shared memory data (refer Figure 3). This provides for continual checkpointing, where the shared segment is constantly updated till the completion of the task. This method also provides for reconstruction in a straightforward manner, as during each task launch when the checkpoint process starts, it simultaneously initiates the segment reconstruction as well.

3.3 Smart Application for the CATALINA

The scenario of smart application is an example, which is modeled after the expert system. Since CATALINA has sets of task agents sitting on the tasks going to be finished, we can construct smart objects with different methods embedded. Each smart object has a set of knowledge base from which we can choose the method to run depending on the situation we define in the set of rules.

Fully connected policy tasks are running on different or the same machine. What we have now is 4 policy tasks, which are IP (Information Policy), AP (Activation Policy), LP (Location and Cooperation Policy) and SP (Selection Policy) (refer to Figure 4). Every policy task is connected to its individual task agent, which communicates with the message center.

IP collects information about the number of tasks in the queue, the 1-min load average and a statistical prediction of execution time. IP can choose which method to follow to collect particular information based on a parameter passed to it by other tasks. IP writes the information collected into the message center, which can be read by other tasks. SP decides which application to be chosen based on the information provided by IP. AP calculates a threshold value based on the information collected from message center and triggers the load balancing mechanism on the machine, which is

over the threshold value. LP is responsible to transfer some tasks to another machine, which is targeted by SP.

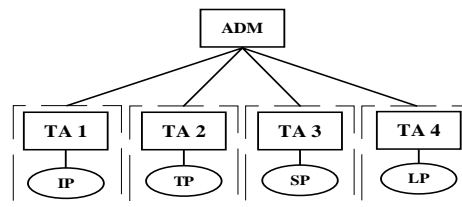


Figure 4. Smart Application Scenario

Information, which is provided by different tasks to the message center, is used as a parameter to choose method within other tasks. Hierarchy in the security mechanism enables us to choose the required authentication and authorization of the agents.

3.4 Active Application Performance Management

Performance management for distributed systems is complex due to the existence of many components that need to be monitored and controlled. Performance management techniques can be broadly characterized into two schemes: monitoring and controlling. Monitoring is the function that tracks the performance activities of the resources, networks and their applications. The controlling function enables performance management to make adjustments to improve performance. We need algorithms and techniques to derive appropriate performance metrics [Katch96][Fahr96], and resource indicators for different levels of performance. Adjusting threshold schemes [Thot98] and polling intervals [Dini97] are the main issues in implementing the performance monitoring function. Five major prediction models for performance predictions for parallel or distributed applications are discussed in [Fahr96]. With performance prediction, performance management schemes can proactively manage large and complex systems. Dynamic load balancing [Zaki95] and process migration [Litz92] have also been studied to provide appropriate performance management.

The objective of the active application performance scenario in CATALINA is to provide flexible system to support performance management in distributed applications. Research is being carried out to predict the effect of changes in bandwidth, latency, load, and TCP window size on the performance of an application. Mobile agents are being used to provide network performance management [Boho00] and [Yu00] by active node-monitoring. However little has been done to provide performance management of distributed applications using mobile agents. Capability of the

mobile agent to monitor the applications can be achieved without having to change the program code.

We use a component-based application, where each component is a process or task, which runs simultaneously in the system. To provide the system to monitor and manage performance of an application during execution we provide a checkpoint mechanism in each task to capture the execution. At each checkpoint the task agent calculates the resource utilization - ρ from the previous checkpoint and resource utilization $\rho(t)$ for the period t of all the iterations upto that instant. Then the agent will calculate the resource utilization $\rho(t+\Delta t)$ and the time required to complete the remaining part of the application $T=1/1-\rho(t+\Delta t)$ using the statistical prediction algorithm [Iver99]. The TA wraps this checkpointing information and sends a message to the ADM. ADM decides the plan for execution based on different strategies - to continue executing the task on the same node if it meets the performance requirements or to stop the execution if it fails to achieve the performance requirements and migrate the task to a different machine or to migrate the task due to a failure in the agent or the task.

The ADM decides if the application can meet the time-requirements on that particular machine, if not it will stop the execution at the checkpoint and start the application on a different machine from the same checkpoint. By saving the state of the application and migrating it and starting the execution from the checkpoint we can ensure that the time requirements for the performance are satisfied.

3. 5 Active Application Fault Tolerance

The main goal of the application fault management is to efficiently recover from hardware/software failures of the system resources. Redundancy is an important technique to detect and recover from component failures in the system. The redundancy can be in the form of hardware, software, or time [Avi76]. As the system increases its complexity, more sophisticated techniques are needed to manage those redundancies. In addition, the fault management scheme must be flexible and adaptive. In SCOP [Xu95], a design methodology is proposed provide designers with a flexible redundancy architecture in which dependability and efficiency can be adjusted dynamically at run time.

The goal of active application fault tolerant scenario in CATALINA is to provide flexible system to support fault tolerance in distributed application. Due to the increasing complexities of application programs, more sophisticated mechanisms are required to handle failures of various kinds. Research in basic concepts to build a reliable application has been performed (i.e. process surveillance [Bec91], checkpointing and recovery [KoT87]). However, little has been done to support fault tolerance in distributed application in a transparent way using mobile agents. In [Bagchi98], the use

of mobile agents to support adaptive fault tolerance is implemented. In [Bec94], another layer between application and operating system has been added to relieve programmers from instrumenting application to handle fault tolerance. Our approach is to utilize mobile agent technology to provide flexibility in choosing recovery mechanisms during run-time execution of the application.

In our approach, we have chosen a component-based application where each component can be perceived as a process or task, which runs simultaneously on the system. We assign a task agent (TA) for each task to handle all the communications (with Message Center) and management issues. In order to monitor task execution, every TA uses Sensor which capture the state of execution of the task and Actuator which Actuator facilitates TA with the capability to control task execution [Hari00].

In order to support the capability of tolerating any fault during application execution, we provide an application-level checkpoint mechanism in each task to capture the execution state. Since our goal is to make fault tolerant mechanism transparent to application, we use another layer which provides the same functional interface to the application as the kernel. The addition of this layer along with Message Center provides a better performance in our approach.

We can measure the expected performance gain using our approach by evaluating two types of recovery mechanisms: Active and Passive Redundancy. In the Active redundancy, if the fault occurs in the primary task, the results can be picked up without any delay from the secondary task that becomes the new primary task once its task agent detects the failure in the primary task due to software or hardware failures. In Passive Redundancy, a stand-by agent will continue the task as soon as the failures occurred.

Using our system, we try to implement a Smart Distributed Dynamic Load Balancing Application. This application can be composed of four major functions that govern the concept of Information, Transfer, Location and Selection policy (see Figure 4). Each policy has several methods to use depending on the current load situation. ADM has the knowledge for each policy so that it can manage the action taken in case of failure.

In case of failure in one of the tasks, TA detects the fault by using sensor. TA then notifies MC about the situation. ADM encounter this message and decide the appropriate action based on the knowledge. Using recovery mechanisms adopted, ADM decides to migrate the faulty task and rollback the other tasks related to the fault task. ADM sends a TA to new location. To synchronize the overall execution, every TA will be responsible to update the current execution state of each corresponding task. The current execution state can be obtained from MC. This way, the overall execution of tasks is consistent. We also guarantee that the stopped task will resume its execution from its last execution state.

The use of adaptation strategy allows our architecture to dynamically reconfigure the task execution when the load changes. This change will affect

the execution of task. ADM will notice this and consult the knowledge to decide appropriate action. If ADM decides to change the method of task execution it notifies the corresponding TA about this. TA updates the execution state of a task and passes this state information to the task. The task will continue the execution using other methods.

3.6 Security Architecture for CATALINA

The proposed security framework is based on certain assumptions. The internal system is supposed to be acting with trusted hosts and assumed to be secure from the outer hosts. The trusted mobile agents are moving within the system, which are provided with specified authentication and authorization schemes. The policies and features of the security system are designed in accordance with the main idea of the CATALINA architecture. A trusted entry and trusted exit is assured while the agents moving inside the system are not hassled with any kind of security transactions. The interaction between the hosts using made a secured one by using the *Java Sandbox* policy. This is used based on the above assumptions in hand.

The *Sandbox* model provides mechanisms stated below to make the system to be a trusted entry and trusted exit policy (*TETE*). The framework consists of a Code verifier, Class loader and Security Manager. Code Verifier is supported by the byte code verification schemes and thus we can make sure the authorization and the authentication of the agents to get into the system. Class loader follows up on the code verification to make sure the behavior of the agents is following the *supposed behavior*. By this we not only make sure that the agent enters in the right way, but we also see that the agent is not carrying more/wrong information while leaving. Security Manager is the one which makes sure that the resources are accessed by the right and entrusted agents thus giving a more secure mechanism for the underlying distributed shared memory policy.

4. Conclusion

In this paper, we presented a general overview of CATALINA: A smart application control and management environment. We have also described the architecture and the implementation approach with an example of a smart object application in the environment. The various services of the CATALINA environment to manage and control performance, fault-tolerance and security have been extensively dealt with. Also a novel checkpointing scheme has been presented to augment the fault-tolerance services of CATALINA. Currently, we have been developing the main components of CATALINA system and its services and will present the benchmark result in the future.

References

- [Case90] J. Case, M. Fedor, M. Schoffstall, and C. Davin. Simple Network Management Protocol (SNMP). RFC 1157; May 1990.
- [McC190] K. McCloghrie and M. Rose. Management Information Base for Network Management of TCP/IP-based Internets. RFC 1156; May 1990.
- [JMAPI] Sun Microsystems, Inc. Java Management API Home Page. <http://java.sun.com/products/JavaManagement/index.html>; November, 1996.
- [Wald95] S. Waldbusser. Remote Network Monitoring Management Information Base. RFC1757; Feb. 1995.
- [Mage96a] T. Magedanz, T. Eckardt. Mobile Software Agents A new Paradigm for Telecommunication Management. Proc. of NOMS96; Kyoto, Japan; April, 1996.
- [Mage96b] T. Magedanz. Intelligent Agents: An Emerging Technology for Next Generation Telecommunications. IEEE INFOCOM; San Francisco, California, USA; March, 1996.
- [Gold95] G. Goldsmith, and Y. Yemini, "Decentralizing Control and Intelligence in Network Management. Proc. of 4th Int'l. Symposium on Integrated Network Management; Santa Barbara; 1995.
- [Peine97] Peine H. An Introduction to Mobile Agent Programming and the ARA System. Dept. of Computer Science Technical Report; University of Kaiserslautern; Germany, 1997.
- [Hall98] D. Hall, S. Rooney. Controlling the Tempest: Adaptive management in Advanced ATM Control Architecture. IEEE Journal on Selected Areas in Communications; vol. 16, No. 3; April, 1998.
- [Saha97] A. Sahai, C. Mortin, S. Billiard. Intelligent Agents for a Mobile Network Manager (MNM), 1997 Intelligent Networks and Intelligence in Networks; 1997.
- [Bagchi98] S. Bagchi, K. Whisnant, Z. Kalbarczyk, R. K. Iyer. Chameleon: A Software Infrastructure for Adaptive Fault Tolerance. The 18th Symposium on Reliable Distributed System; 1998.
- [Bec91] T. Becker. Keeping Processes Under Surveillance. Proc. of 10th Symposium on Reliable Distributed Systems; pp. 198-205; 1991.
- [Bec94] T. Becker. Application-Transparent Fault Tolerance in Distributed Systems. Proc. 2nd Int'l. Workshop on Configurable Distributed Systems; pp. 36-45; Mar. 1994; Pittsburgh, PA.
- [Hari00] S. Hariri, Y. Kim, M. Djunaedi. Design and Analysis of a Proactive Application Management System. Proc. of NOMS2000; April, 2000.
- [KoT87] R. Koo, S. Toueg. Checkpointing and Recovery-Rollback for Distributed Systems. IEEE Transactions on Software Engineering; Vol. SE-13, No. 1; pp. 23-31; 1987.
- [Diet99] W. R. Dieter, J. E. Lumpp Jr.. A User-level Checkpointing Library for POSIX Threads Programs. 29th Annual International Symposium on Fault_tolerant Computing; 1999.
- [Gend00] E. Gendelman, L. F. Bic, M. B. Dillencourt. An Application-Transparent, Platform-Independent Approach to Rollback-Recovery for Mobile Agent Systems. Proceedings of the 20th International Conference on Distributed Computing Systems; pp. 564-571; 2000.
- [Tanaka99] K. Tanaka, H. Higaki, M. Takizawa. Checkpoints in Distributed Object-Based Systems. Proceedings of the International Workshop on Parallel Processing; 1999.
- [Boho00] C Bohoris, G Pavlov, H Cruickshank. Using Mobile Agents for Network Performance Management. Proc. of NOMS 2000; April, 2000.
- [Yu00] Y. Miyoshi, K. Kamahora, Yong-Jin Park, Y. Urano, H. Tominaga. An Advanced Network Management System with Mobile Agents.
- [Ivers 99] M. A. Iverson, F. Ozguner, L. C. Potter. Statistical Prediction of Task Execution Times through Analytic Benchmarking for Scheduling in a Heterogeneous Environment. Eighth Heterogeneous Computing Workshop (HCW'99).