Experimental Results and Evaluation of the Proactive Application Management System (PAMS)

Yoonhee Kim*, Salim Hariri **, and Muhamad Djunaedi**

* EECS Dept. Syracuse University
Syracuse, NY 13244
yhkim@top.cis.syr.edu

** HPDC Laboratory
Dept.of Electrical and Computer Engineering,
University of Arizona
Tucson, AZ 85721
{Hariri,djunaedi}@ece.arizona.edu

ABSTRACT

Management of large-scale Network-Centric Systems (NCS) and their applications is an extremely complex and challenging task due to factors such as centralized management architectures, lack of coordination and compatibility among heterogeneous network management systems, and the dynamic characteristics of networks and application bandwidth requirements, just to name a few. The goal of our research is to develop a hierarchical framework to achieve end-to-end intelligent proactive network management system that can be used to manage large scale network centric systems and their applications. This framework will provide the appropriate tools to write management programs to control and manage any function or property (performance, high assurance, fault, quality of service, etc.) of the network-centric systems and their applications during all the phases of their operations. In this paper, we present a framework to develop proactive and adaptive management services and an implementation of a Proactive Application Management System (PAMS) based on that framework. Our implementation approach utilizes delegated mobile agents to implement the management functions required by any network-centric system and/or application. We also present experimental results and evaluation of the management services offered by the PAMS prototype.

1. Introduction

Most of the current network management technologies focus on collecting management information and providing a graphical-user interface to assist network managers in visualizing the collected management information and carrying out their management tasks (passive management). Furthermore, the type of information collected is not appropriate to achieve end-to-end proactive management functions. There has been little work done to make network management systems proactive and intelligent. By making network management systems proactive, all management functions will be improved and the network can respond in a timely manner to any changes in application requirements and available resources. The development of programmable application management schemes has not been investigated thoroughly and is not well understood. The main goal of our research is to remedy this problem and to develop a management framework to achieve end-end proactive application centric management.

Network management products have taken a long road to the current state. They started with the Simple Network Management Protocol (SNMP) version 1 and then version 2 [Case90, Case93], followed by the Management Information Base (MIB) version I and then version II, and Remote Monitoring (RMON) version I and version II[Mccl90a, Mccl90b,Mccl91,Wald95]. Recently, there has been an intensive effort to use web-based technologies (JMAPI and WBEM) to build network management tools [Sun96, micr97a, micr97b]. The limitations and problems with network management are:

- Most of the commercial network management systems collect management information about packet throughput, delay, and packet errors at input and output of network interfaces. For end-to-end proactive application and network management, we need to collect management information relevant to applications and computing resources such as the current loads on computers, the types of processes accessing the file systems, types of users and their access pattern profiles, security information, and so on. Furthermore, we need to have the ability to program the type of information to be collected and for what period of time. This approach enables us to dynamically collect any required management information and for the required period of time rather

being running all the time and consuming unnecessary computing and storage resources.

- The amount of information collected for large networks (enterprise networks) is huge. It is very difficult for network managers to efficiently utilize the overwhelming amount of management information to improve network performance, utilization and applications. This problem becomes extremely complex when the size of the network increases to hundreds or thousands of nodes spanning several organization domains or countries. We need to develop techniques that can process raw management information and produce concise management information filters. These management filters can then be used to achieve efficient and robust analysis of large-scale networks and their applications.
- The literature is rich with algorithms and techniques to dynamically route packets, automatic reconfiguration, adaptive scheduling of resources, dynamic fault tolerance, etc. However, very little is done to apply these algorithms/techniques to achieve proactive real-time management of networks and their applications.
- Most of the management functions such as configuration, resource allocations and scheduling are done manually by network managers. This makes the management process slow, not scalable, and not cost-effective. Furthermore, this manual management scheme can not meet the stringent real-time requirements of some critical applications.

We do need to develop novel management techniques that eliminate these problems and provide scalable management capabilities to efficiently, intelligently, and cost-effectively manage any network application running on any network of any size and at any time. In this paper, we present a framework for a network management system that provides management services to bridge the gap between application development and network management as well as build management ready applications. Our approach for the implementation of the network management framework is hierarchical and consists of three layers: Network and Protocol Management (NPM), Management Computing System (MCS), and Application Centric Management (ACM). The NPM is responsible for the collection of management information not only about the network devices, but also information related to computer processes, file systems, user access information and patterns, and protocols. The NPM will also perform tasks to manage the network devices, protocol functions, computer processes and file systems. The MCS provides the core management functions to manage system-wide resources from a system perspective rather than component perspective as is done in NPM. The ACM provides the capability to program MCS functions to control and proactively manage a given network application during all the life cycles of any network application. We present also the architecture of a Proactive Application Management System (PAMS) and evaluate the performance of some of the management services offered by the PAMS prototype.

The organization of the paper is as follows. In Section 2, we overview the current research of intelligent mobile agents. In Section 3, we describe our network management framework and the main software modules to implement this framework. In Section 4, we present the architecture of a proactive application management system implemented based on our management framework. In Section 6, we evaluate the performance of some of the management functions offered by PAMS prototype. In Section 6, we present a summary and concluding remarks.

2. Framework for Application Centric Management

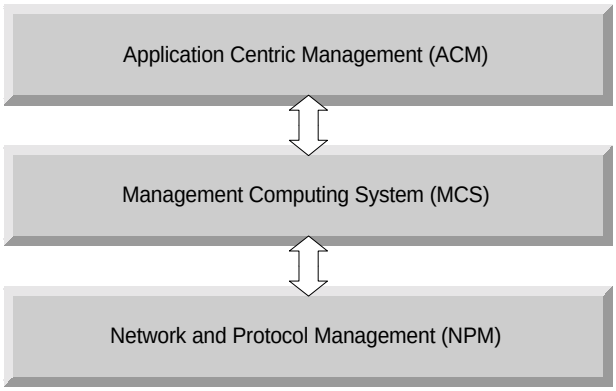

Figure 1 A framework for end-to-end proactive network management system.

The management framework we are developing can be viewed in terms of three systems: Network and Protocol Management (NPM), Management Computing System (MCS), and Application-Centric Management (ACM). The NPM is responsible to collect management information not only about the network devices, but also information related to computer processes, file systems, user access information and patterns, and protocols. The NPM will also perform tasks to manage the network devices, protocol functions, computer processes and file systems. The MCS provides the core management functions to manage the whole system resources from system perspective rather than component level perspective. In order to achieve that, the management information collected at the lower level (NPM) will be analyzed and abstracted into suitable data structures or format to perform efficient system level management functions. The MCS design concept is analogous to the operating system in computing systems. The operating system manages the computing system resources

(memory, I/O, CPU, and processes). Similarly, the MCS acts as an automatic system manager that provides management functions to achieve application centric management tasks.

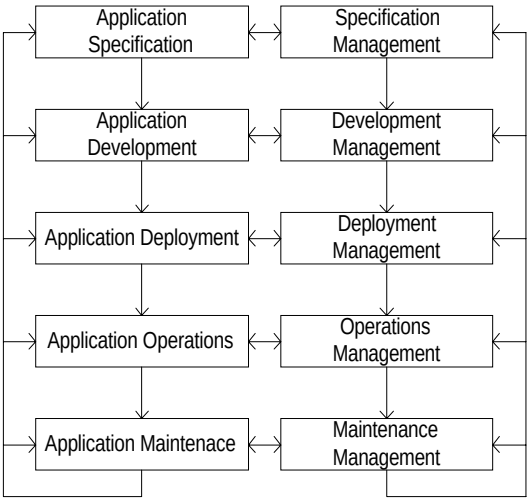

Figure 2. Software Development cycle with Management Activity

The ACM provides two main functions: Assist in the development of application management routines, and provide intelligent proactive management for a wide range of network applications. The number and type of network applications become increasingly large and their computing, storage, and network requirements differ widely. In addition to the difficulty that can be contributed to the complexity, heterogeneity, and size of the emerging network applications, the development of such applications do not take into consideration the management issues and requirements. Currently, the management of such applications follows force-fitting approach that utilizes the commercial network management services that are based on SNMP or CMIP standards.

Our approach is to develop system management functions (provided by the MCS) that can be programmed by applications to meet their requirements during all the life cycles associated with any application (e.g., specification, development, deployment, operations, and maintenance). Figure 2 shows how we can integrate the software development life cycle of an application with the management activities of that application [Hari98].

3. Architecture of the Proactive Application Management System (PAMS)

In this section, we first overview the architecture of a Proactive Application Management System (PAMS) being implemented at the HPDC Laboratory at the University of Arizona. Then, we benchmark the use of mobile agent technologies to control and management of several distributed applications. The architecture of PAMS is shown in Figure 3. The main key components of PAMS can be described in term of three services: ACM Service, MCS Service, and NPM Service.

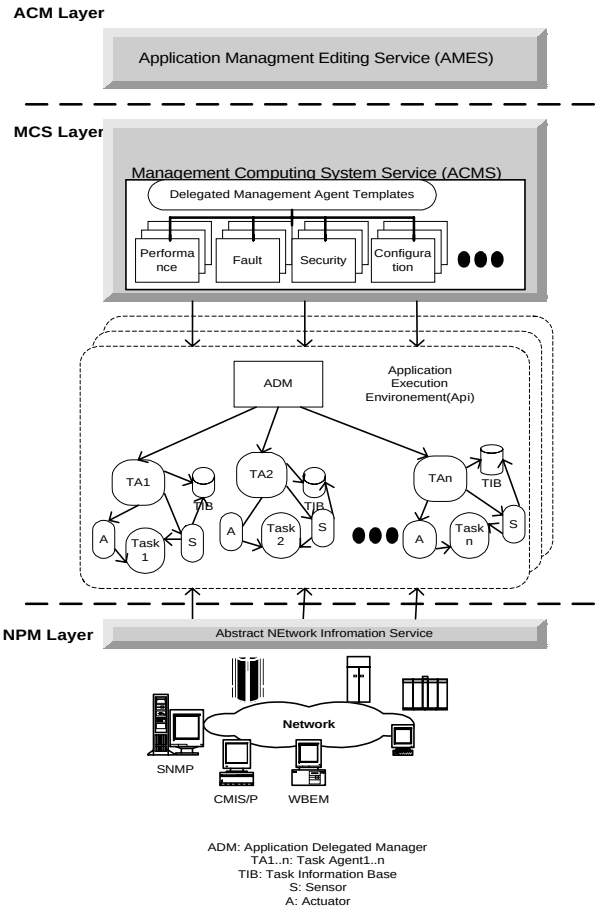

Figure 3. The architecture of PAMS

The ACM Service provides the user with the tools required to describe and characterize the management requirements of any network-centric system or application. The MCS provides the management services to automatically configure the application or system resources, monitor and control the execution of an NCS application. The NPM service provides the appropriate interface to existing network management systems and utilize their services in order to proactively manage and control the operations of the NCS or its applications. The management service categories are classified in the paper as ADMs, Sensors and Actuators [Hari20].

When a user develops an application using PAMS user interface subsystem, the application requirements are

described and characterized by an application graph as shown in Figure 3. This application graph is then interpreted by the application generator and fed into the MCS server. The MCS server checks the Management Agent (MA) templates and generates the appropriate Application Delegated Manager (ADM) appropriate for application configuration, monitoring and to manage and control the application execution during runtime. Once the application ADMs required to execute and manage the application are identified, the next step is to download the Task Agents (TAs) and the appropriate execution codes into the selected computing resources. On each machine selected, an Task Agent (TA) is activated in order to start the execution of the monitored task on that machine.

4. Proactive Application Management Algorithms

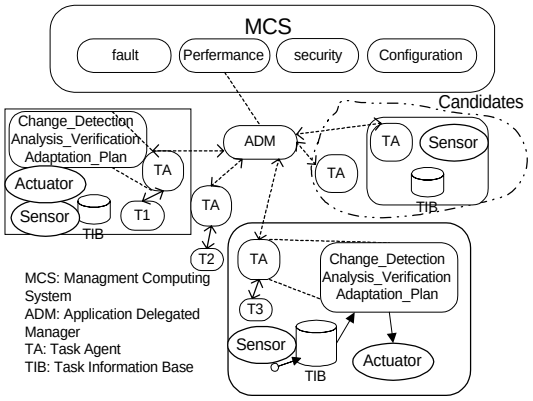

Figure 4. The Runtime Architecture of the Proactive Application Management System

In this section, we present a general approach to actively manage any desired property of an application and/or a system. The Management Computing System Service (MCS) services include the management of application performance, fault, security, accounting, resource configuration and application configuration (see Figure 4.). For each management service, there is a Delegated Management Agent (ADM) that will be responsible for the deploying the appropriate Task Agents (TAs) on the machines selected to run the application tasks under consideration. The TAs are responsible to start, monitor and control the application's tasks and communicate with the ADM. The ADM needs to interact with MCS to obtain global states of the system and allocate new resources to run the application tasks whenever it is required to maintain the quality of service requirements of the managed application. To monitor the execution of the application's tasks, we use sensors to periodically monitor the task's execution and the state of the machine running the application task, and then store this information in an Task Information Base (TIB). We also use actuators to provide the TA with the capability to control the execution of the application task (suspend, resume, migrate, etc.) and migrate the task to run on another machine. The main management activities of an TA can be abstracted into three procedures or functions: Change_Detection, Analsis_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, the task or the machine running the task encounters software or hardware failures). The Analysis_Verification algorithm is invoked whenever a change is detected in the operation of application tasks to make sure that these changes are real and not false alarms. Once the alarming event is verified and its type is identified, the Adaptation Plan procedure is invoked to execute the appropriate adaptation scheme to basically fix the problems detected during the application execution. Figure 5 shows the general Proactive Application Management Algorithm for the PAMS prototype.

Proactive_Application_Management Algorithm

```
1       While (AE(Api) is running) do
2               For each Service Si∈ MCS(Api),
3               Si ∈ {Sft, Sperf, Ssecurity, Sconfig, Saccount}
4                       Start Service Si(Api),
5                       Monitor Si(Api)
6               EndFor
7       EndWhile
End Proactive_Application_Management_Algorithm
```

Figure 5 Proactive Application Management Algorithm

The application Execution Environment *(AE(Ap_i))* refers to all the resources allocated to run a give application Ap_I. While the application is running (step 1 in the Proactive Application Management Algorithm of Figure 5), the MCS starts all the PAMS services (Step 2,3 in the algorithm of Figure 5) associated with that application and then monitor the execution of that application to detect any changes or deterioration while it is running.

Figures 6, 7, 8 and 9 show procedures required to achieve proactive performance management (S_{perf}). While the application is running, the application's tasks are monitored by the TAs (step 3 through 14 in Figure 6). In the current implementation, we use the CPU utilization metric to predict the expected task performance. Once any unacceptable change in the CPU load is detected as in the Change_Detection procedure (Figure 7), the Analysis_Verification procedure (Figure 8) evaluates and predicts the total application execution time. If the predicted application execution time is not acceptable to meet the application requirements (e.g., the application can not meet it is own deadline requirements), the

procedure recommends certain actions (e.g., migrate the task to another machine). In the Adaptation_plan procedure (Figure 9), the adaptation plan is executed (step 10 in Figure 6).

```
Procedure Sperf
1       While AEE(Api) is runnning do
2               Set up Monitoring Environment
3               For each Task Ti ∈ Api do
4               Start Sensor (TIB(Ti))
5                       While (Ti is running) do
6                               CPU_load =
Change_Detection(CPU_load)
7                               If  (CPU_load > Threshold)
then
8                                       EventType =
Analysis_Verification (CPU_load)
9                                       If (eventType =
migration) then
10
        Adaptation_plan (EventType, Candidate)
11                                      Endif
12                              Endif
13                      EndWhile
14              EndFor
15      EndWhile
Endof Procedure Sperf
```

Figure 6 Performance Management Service Algorithm

```
Procedure Change_Detection(TIB)
1       monitor CPU_load from TIB
2       If  CPU_load change then
3               Retrun CPU_load
4       Else return No_Event
5       Endif
End of Procedure Change_Detection
```

Figure 7 Change Detection Algorithm

```
Procedure Analysis_Verification (Eventload)
        /* Perform Application Performance Prediction */
1       Remaining_time ← Base_time – Elapsed_time
```

$$Predicted\_remaining\_time = \frac{Remaining\_Time}{Baseload} * Eventload(t)$$

```
2       Remaining_deadline ← Deadline – Elapsed_time
4       If Predicted_remaining_time > Remaining_deadline then
        /*Ask ADM to get migration and candidiate K */
5               Migration ←Get_Candidate ( Remaining_time,
                Predicted_remaining_time, Remaining_deadline)
6               If Migration then return (migration)
7       Else     return (noaction)
8       Endif
End of Verification_Analysis
```

Figure 8 Analysis and Verification Algorithm

```
Procedure Adaptation_Plan(event,candidate)
        If event= migration then
                A.suspend(Ti)
                state ← Restore_state(Ti);
                A.migrate(Ti, Candidate, state)
        endif
End of Adaptation_plan
```

Figure 9 Adaptation Plan Algorithm

The Analysis_Verification procedure shown in Figure 8 involves analyzing the current state of the system resources and predict the performance of its task execution. The task performance analysis takes into consideration the current load conditions in the network and the computing resources and how much time has been spent in the application execution. Once we obtain the CPU utilization(Eventload(t)), we could estimate the remaining execution time for the task as,

$$Predicted\_remaining\_time = \frac{Remaining\_Time}{Baseload} * Eventload(t)$$

where *Remaining_time* is the average execution time under *Baseload*. In step 3 of the algorithm, we use the Predicted_remaing_time equation to predict the remaining execution time. In Step 4, we check to see if the task can meet its deadline requirements. In this case, there are three possible scenarios: 1) Task Meets its QOS requirements (line 7); 2) Task Can not meet its QOS requirements (step 6), but can be met by migration to a backup resource (line 5); and 3) The system fails to meet the task QOS requirements (line 6) if there is no candidate.

4.1 Benchmarking of PAMS Performance Management Service

In this experiment, we benchmarked an application with three tasks running on a cluster of workstations (e.g., SUN SPARCstations). We measured the overhead associated with implementing PAMS performance management service for two application types: small application with an average execution time of 30 seconds and a large application with an average execution time of 450 seconds. Figure 10 shows the PAMS overhead for the small application when the system loads are under 5% for two machines and 24% for the third machine. If during the application execution, the load on two machines has suddenly increased to 99% CPU utilization, PAMS will detect this change and eventually decides on migrating the tasks running on the loaded machines to less loaded machines. If the migration is performed, PAMS was able to improve the performance by 25% as shown in Figure 11. For large-scale applications (350 ~500 seconds), PAMS is more efficient in maintaining the performance of the application. Figure 12 shows that by using PAMS to manage the performance, we can achieve a 75% performance gain when the load on the machine running task 1 is increased to a 99% CPU utilization.

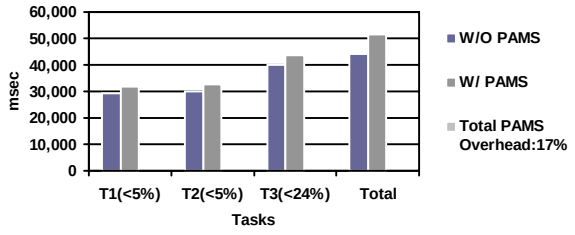

Figure 10 Scenario 1:Application Execution with 3 Tasks (Size : Small Status : Normal)

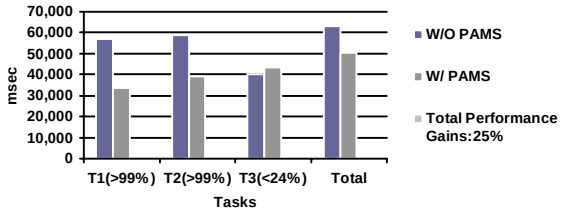

Figure 11 Scenario 2: Application Execution with 3 Tasks (Size : Small Status : Processor of T1 and T2 are overloaded)

5. Adaptation Algorithm for PAMS Fault Management Service

This service consists of three major modules: MCS, Delegated Management Agent and Application Agent. MCS handles the message flows in the system. It displays the monitoring result from each agent. MCS provides the all types of mobile agents that are required to maintain any application service. In our current implementation, we used RMI scheme to implement mobile agents. The management agents communicate with Application Agents in each machine using UDP/IP protocol.

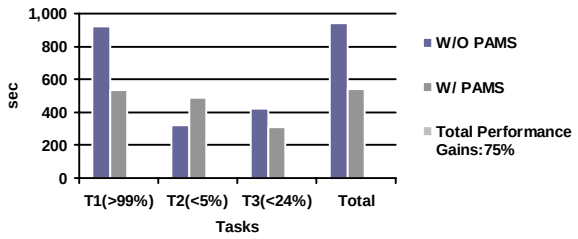

Figure 12 Scenario 3: Application Execution with 3 Tasks (Size : Large Status : Processor of T1 is overloaded)

We apply the same proactive application management algorithm to manage the fault-tolerance service of any application. In the current implementation, the fault model assume fail-stop model; the task once it is crashed, it stop the execution. The application is abstracted as a set of tasks. In PAM fault-tolerance service, we assign an application agent to each task in the application. The TA monitors the task execution and is responsible for fault-detection and recovery. Similar to the application performance management algorithm, there are three procedures: change detection, verification and incorporating adaptation plan as shown in Figure 13. Figure 14 shows the main steps followed by the algorithm to achieve fault-tolerance execution of an application with two tasks. The sensor maintains the status of the task execution in an Task Information Base (TIB) (steps 1 and 2). Once the sensor detects the failure in a task, say Task 1, it reports this events to the Task Agent (TA) (Step 3). The TA can either try to recover locally by retrying the task on the same machine by invoking the services offered by the Actuator. Otherwise, it reports the events to the Application Delegated Manager (ADM) that is responsible for providing the fault-tolerance for the application (Step 4). The ADM will then interact with the MCS to determine the new machine that can run the faulty task (Step 5). The ADM will then start an application agent on the selected machine to resume the execution of the migrated task and also maintains its fault-tolerance execution (Step 6). The TA will in its turn setup the task execution environment and monitors its execution (Steps 7 and 8).

```
Procedure Fault-Tolerance Service (S_FT )
1  while ApplicationEnvironment(App_i)
2    for each T_i ∈ App_i
3       Setup_TA(T_i);
4       Start sensor(TIB(T_i));
5       while (T_i == "RUNNING")
6             execStatus <== Change_Detection(T_i);
7             if (execStatus == "TASK_STOP")
8              EventVerification <==
Analysis_Verification(execStatus(T_i));
9                 Adaptation_Plan(EventVerification(T_i), Candidate);
10            endif
11          endwhile
12      endfor
13 endwhile
EndProcedure

Procedure Setup_TA(AgentID(T_i))
  if (AgentID(T_i) == "STOP")
     Redownload agent source;
     Restart AgentID(T_i) execution;
  Endif
End Procedure

Procedure Change_Detection(T_i)
  execStatus <== Read_TIB(T_i);
  return execStatus;
EndProcedure

Procedure Analysis_Verification(execStatus(T_i))
  EventVerification <== Check_Status_TIB(execStatus(T_i));
  return EventVerification;
EndProcedure

Procedure Adaptation_Plan(EventVerification(T_i), Candidate)
  TaskPlan <== Retrieve_Adaptation_Plan(EventVerification(T_i));
  machine <== Select_Machine(TaskPlan, Candidate);
  if (machine == "LOCAL")
     Resume T_i execution;
  else
     Setup_migration_agent(T_i);
     Resume T_i execution;
  Endif
EndProcedure
```

Figure 13 Adaptive Fault Tolerance Algorithm

5.1 Benchmarking of PAMS Fault Management Service

We used four machines in evaluating the application fault-tolerance service; one machine used as the MCS management stations, and three machines to run and

manage the application tasks. We benchmarked the application fault-tolerance service using different size applications. We evaluated PAMS performance for three task granularities: 1) Small application with average task execution time of 60 seconds, 2) Medium size applications with average task execution time of 600 seconds, and 3) Large size applications with average task execution time of 6000 seconds. Figure 15 shows the overhead incurred by using PAMS fault-tolerance service for these applications when number of faulty tasks vary from one to three. PAMS overhead for small application is around 14% while it is less than 2% for large applications.

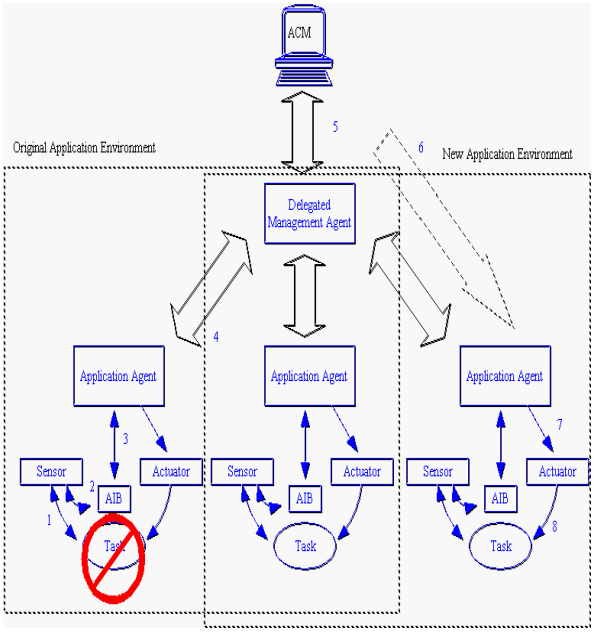

Figure 14 An Application Fault Tolerance Service Example.

6. Summary and Conclusion Remark

In this paper, we presented a general approach to develop application centric management. We also described an architecture to achieve proactive application management. Our approach is scalable and utilizes delegated agent approach to implement distributed management services. We presented preliminary results of the application performance and fault-tolerance management services offered by PAMS. Our experimental results showed that application performance can be improved significantly using PAMS' proactive management scheme. Our results showed also the low overhead incurred by PAMS to achieve application fault-tolerance execution. We are currently implementing additional services to balance the load across the network resources and maintain the system and application security requirements.

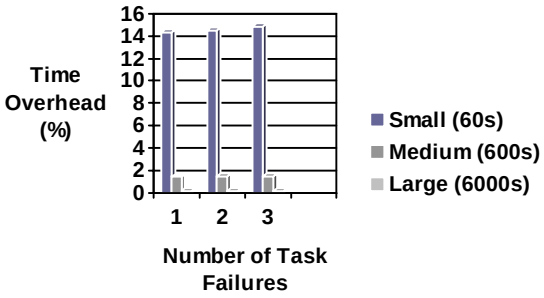

Figure 15 PAMS fault-tolerance service overhead.

7. Reference

[Case90] J. Case, M. Fedor, M. Schoffstall, and C. Davin, "Simple Network Management Protocol (SNMP)," RFC 1157, May 1990

[Case93] . McCloghrie, M. Rose, and S. Waldbusser, "Introduction to version 2 of the Network Management Framework," RFC 1441, April 1993

[Mccl90a] K. McCloghrie and M. Rose, "Management Information Base for Network Management of TCP/IP-based Internets," RFC 1156, May 1990

[Mccl90b] K. McCloghrie and M. Rose, "Structure and Identification of Management Information for TCP/IP-based Internets," RFC 1155, May 1990

[Mccl91]K. McCloghrie and M. Rose, "Management Information Base for Network Management of TCP/IP-based Internets: MIB-II," RFC 1213, March 1991

[Sun96] Sun Microsystems, Inc. "Java Management API Home Page," http://java.sun.com/products/JavaManagement/index.html, November, 1996

[Micr97a] Microsoft corporation, "Web-Based Enterprise Management Initiative Home Page," http://wbem.freerange.com, 1997

[Micr97b] Microsoft corporation, "Microsoft Web-Based Enterprise Management Professional Developers Kit." 1997

[Wald95] S. Waldbusser, "Remote Network Monitoring Management Information Base," RFC 1757, Feb. 1995

[Hari98] Salim hariri, Y. Kim, "The End-to-End Proactive Management", submitted to IEEE/IFIP 1998 Network Operations and Management Symposium(NOMS98), New Orleans, Feb, 1998

[Hari20] Salim hariri, Y. Kim, M. Djunaedi, "Design and Analysis of a Proactive Application Management System (PAMS)", submitted to IEEE/IFIP 2000 Network Operations and Management Symposium(NOMS2000), Hawaii, April, 2000