Evaluation of PAMS' Adaptive Management Services

Yoonhee Kim,
Department of Electrical Engineering and Computer Science,
Syracuse University
Syracuse, NY 13244
yhkim@ecs.syr.edu

Salim Hariri, and Muhamad Djunaedi
Department of Electrical and Computer Engineering, University of Arizona
Tucson, AZ 85721
{hariri, djunaedi}@ece.arizona.edu

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 bandwidth requirements. The development of an integrated network management framework that is proactive, scalable and robust is a challenging research problem. In this paper, we present our approach to implement a Proactive Application Management System (PAMS). PAMS architecture consists of two main modules: Application Centric Management (ACM) and Management Computing System (MCS). The ACM module provides the application developers with all the tools required to specify the appropriate management schemes to manage any quality of service requirement or application attribute/functionality (e.g., performance, fault, security, etc.). The MCS provides the core management services to enable the efficient proactive management of a wide range of network applications. The services offered by the MCS are implemented using mobile agents. Furthermore, each MCS service can be implemented using several techniques that can be selected dynamically by invoking the corresponding mobile agent template for the service implementation. In this paper, we present our preliminary results of evaluating PAMS management services to manage the performance and fault tolerance execution of three applications of different sizes (small, medium and large). The experimental results demonstrate that our agent-based approach can lead to significant gains in the performance and low overhead fault management of parallel/distributed. For example, the overhead incurred in the application fault management to tolerate one task failure, two task failures, and three task failures in a medium to large size application is less than 0.02%.

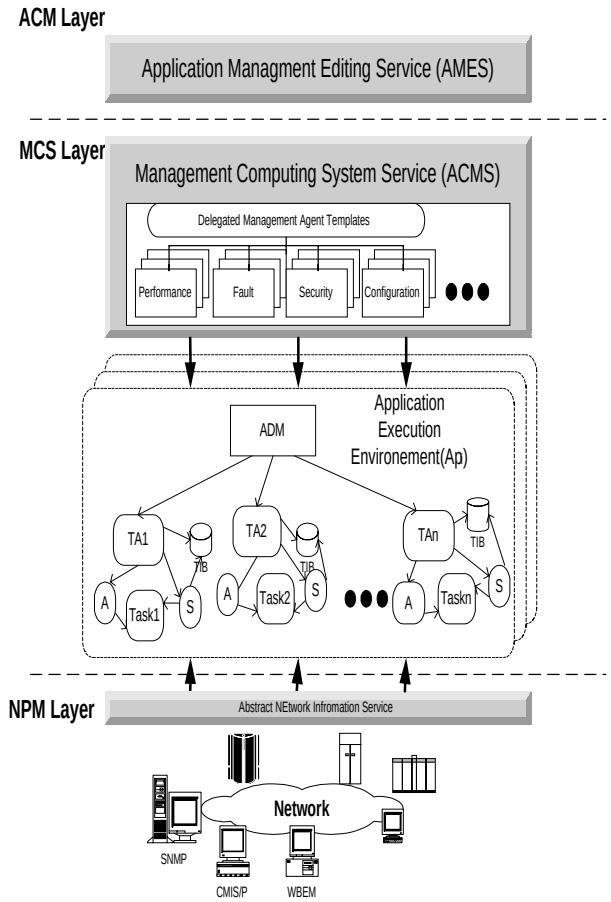
1. Introduction

The emerging high speed networks and the advances in computing technology are important driving forces to merge the communications and computing technologies that will result in an explosive growth in network complexity, size and networked applications. Furthermore, we are observing an explosive growth in network applications that use computing, networking and storage resources that can be accessed from global national and/or international networks. The management of such networks and their distributed applications has become increasingly complex, and unmanageable. Unfortunately, the current network management technologies focus on collecting management information and manually manage the network using platform-specific products. There has been little research toward the development of intelligent, efficient, proactive end-to-end management of large networks and their applications.

The increased importance of network management for large-scale networks has stimulated research on novel approaches to reduce the management complexity and cope with dynamic management change. Instead of a centralized manager, multi-managers and their communication protocols are proposed such as Management by Delegation (MbD)[4] and Code Mobility[5]. Another approach replaces the manger-agent relationship among managers and agents with peer-to-peer relationship using the Common Object Request Broker Architecture (CORBA) has been studied in the area of Telecommunications Information Networking Architecture (TINA) framework [2]. A few web-based approaches to network management have emerged recently (JMAPI, WEBEM). [3].

However, distributed network management of applications over heterogeneous has not fully studied and is becoming increasingly important. Recently, Application Management MIB [7] and MIB for Application [6] have been proposed to collect and store common application management information in

IETF. Common Information Model (CIM) by DMTF is proposed a similar process information definition for WBEM [Patrck98]. Still, there has been little work done to achieve programmable application management schemes and is not well understood.



ADM: Application Delegated Manager
TA1..n: Task Agent1..n
TIB: Task Information Base
S: Sensor
A: Actuator

Figure 1. The Runtime Architecture of the Proactive Application Management System.

In this paper, we present the design and evaluation of a Proactive Application Management System (PAMS) prototype being developed at the University of Arizona. PAMS provides adaptive application management services to dynamically manage the performance and fault of parallel/distributed applications in an unreliable and heterogeneous computing environment. PAMS implementation is based on using mobile agents that can be programmed to maintain the quality of service requirements of distributed applications. We have evaluated three adaptive techniques to manage the performance and fault tolerance of distributed applications. The first approach is based on using active redundancy to improve performance and tolerate faults. The second approach is based on passive redundancy in which a set of machines is designated as backup machines to be used to replace any of the machines assigned to the application tasks in order to improve performance or to tolerate software/hardware failures. The third approach does not introduce redundancy in the system and it requires task migration to another machine in order to improve performance or to tolerate software/hardware failures. The preliminary results of applying these techniques demonstrate that our agent-based approach can lead to significant gains in the performance and low overhead fault management of parallel/distributed application. The organization of the paper is as follows. In Section 2, we give a brief overview of the PAMS prototype. In Section 3, we discuss our approach to benchmark and evaluate the adaptive performance management services offered by PAMS. In Section 4, we benchmark and evaluate the adaptive fault management service.

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

The architecture of PAMS is shown in Figure 1. The ACM layer provides application developers with the tools required to specify and characterize the application requirements in terms of performance, fault, security, and also specify the appropriate management scheme to maintain the application requirements. Once the application management requirements are defined using the ACM tools, 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.). 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.

The main management activities of 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 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. Once the change event is verified and its type is identified, the Adaptation Plan procedure is invoked to execute the appropriate adaptation scheme.

```
Proactive_Application_Management Algorithm
1 For each Ap Api∈ ACM(Api),
2       Assign Application Delegated Manager ADM
(Api)
3       Lunch ADM (Api)
4       While (AEE(Api) is running) do
5               For each Service Si∈ APi
6                       Si ∈ {Sft, Sperf, Ssecurity, Sconfig}
7                       Start Service Si(Api),
8                       Monitor Si(Api)
9               EndFor
10      EndWhile
End Proactive_Application_Management_Algorithm
```

Figure 2 Proactive Application Management Algorithm

Figure 2 shows the general Proactive Application Management Algorithm for the PAMS prototype. The application Execution Environment (AEE(Api)) refers to all the resources allocated to run a give application ApI . While the application is running (step 4 in the Proactive Application Management Algorithm of Figure 2), the ADM starts all the task agents required to manage the application requirements (performance, security, fault, etc.) (Step 7,8 in the algorithm of Figure 2) and then monitor the execution of that application to detect any changes or deterioration while it is running. In what follows, we discuss PAMS approach to use mobile agents to manage the performance and fault tolerance of parallel/distributed applications.

3. Adaptive Performance Application Management

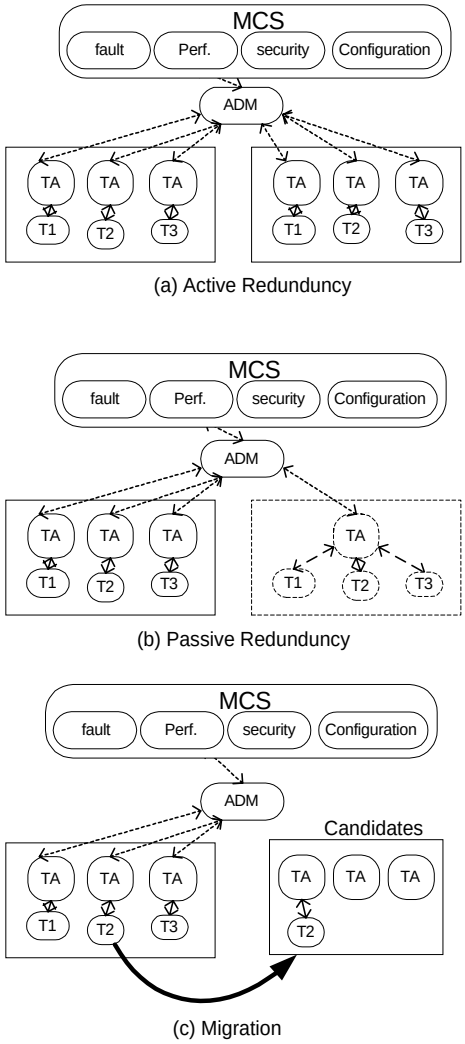


Figure 3 Controlling Techniques of Performance Management

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 [9][10], and resource indicators for different levels of performance. Adjusting threshold schemes [13] and polling intervals [14] are the main issues in implementing the performance monitoring function. Performance statistics can be used to recognize potential bottlenecks or failures before they cause problems. Five major prediction models for performance predictions for parallel or distributed applications are discussed in [10]. With performance prediction, performance management schemes can proactively manage large and complex systems. Dynamic load-balancing [12] and process migration [11] have also been studied to provide appropriate performance management.

In our application performance management, we monitor the execution times of an application as well as the resource and network utilization. In addition, we use redundancy techniques and task migration to implement the control functions required to dynamically manage the application performance. In this paper, we evaluate three techniques to manage the application performance: active redundancy, passive redundancy and migration. Each technique is implemented as an agent template as shown in Figure 3.

The active redundancy scheme duplicates the execution of the application on two machines (see Figure 3 (a)). In this scheme, the task agent will pick up the results from the first machine that completes the task execution. This approach has several advantages. First, lead to better performance because we always pick up the results from the faster machine. Second, it simplifies the performance management since no need to perform task migration or load balancing in the system due to load changes or bursty traffic conditions.

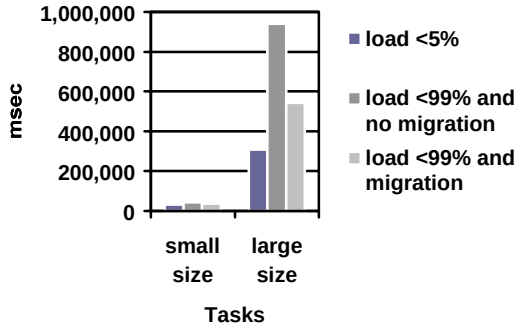


Figure 4 Application Execution with migration scheme

The passive redundancy assigns each task to a primary machine that will run the task and another machine to be used as a backup whenever the task performance deteriorates on the assigned machine (see Figure 3 (b)). The backup machine is kept-up-to-date in order to be ready to resume the task execution from the last updated checkpoint. The main advantage of this approach is that it needs less resources than the active redundancy approach. In this scheme, one backup machine can be used as a backup machine to several tasks.

The third approach does not introduce redundancy and improves the performance by task migration (see Figure 3 (c)). However, the overhead of task migration is high and it should be used only for large task granularities where the migration overhead is relatively small when compared to the task execution time.

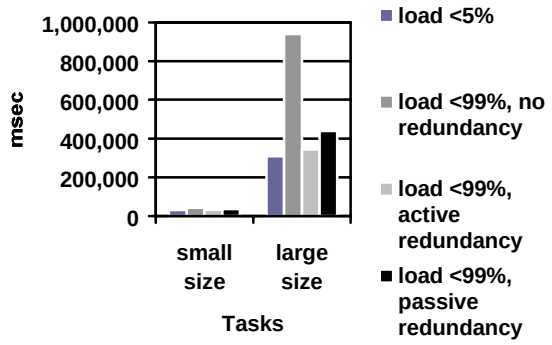


Figure 5 Application Execution with Redundancy policies

We benchmarked the overhead associated with implementing PAMS performance management service for two application types: a small application with an average execution time of 30 seconds and a large application with an average execution time of 450 seconds. We evaluated the use migration, active redundancy and passive redundancy techniques to dynamically mange the performance of these two applications. If, during the application execution, the load on a machine suddenly increased to 99% CPU utilization, the migration approach was able to improve the performance by 25% for the small size application (approximately 40 seconds) and by 75% for the large application (approximately 308 seconds) as shown in Figure 4. The active redundancy technique achieved a 31% performance gain for the small application and 174% for the large application as shown in Figure 5. Similar results were achieved in the passive redundancy approach, where a 22% performance gain was achieved for the small

application and a 114% performance gain for the large application.

4. Adaptive 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 [15]. 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 [17], a design methodology is proposed to introduce support techniques to reduce the resource cost of fault-tolerant software, both in space and time, by providing designers with a flexible redundancy architecture in which dependability and efficiency can be adjusted dynamically at run time. In another work [18], the use of mobile agents to support adaptive fault tolerance is implemented. In our adaptive application fault-tolerance approach, we use mobile agents to efficiently manage the redundancy. We evaluate two redundancy techniques: Passive and Active redundancy.

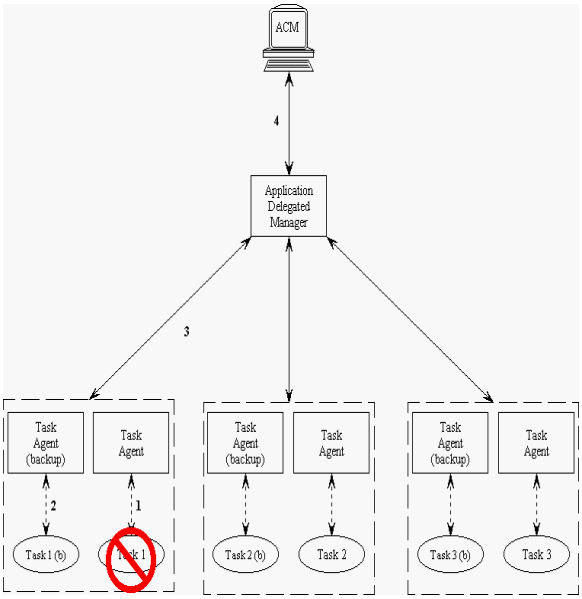


Figure 6 Active Redundancy Techniques for Fault Management

In the active redundancy technique shown in Figure 6, we assign two identical tasks to two machines that are managed by two Task Agents (TAs); one task is designated as the primary task while the second one is referred to as the secondary task. In this scenario, the ADM doesn't need to determine the adaptation plan when a fault occurs. 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 addition to reducing the time for fault detection, active redundancy technique simplifies the communication between task agents. Figure 8 shows the overhead incurred by applying this redundancy scheme to adaptively manage the faults of three applications with three tasks each. In the small application case (execution time is around 60s), the overhead incurred in using our scheme to detect and recover from one task failure, two task failures, and three task failures are 0.10%, 0.18%, and 0.22%, respectively (see Figure 7). For medium and large applications, the overhead in managing one, two or three task failures is very small (less than 0.02%).

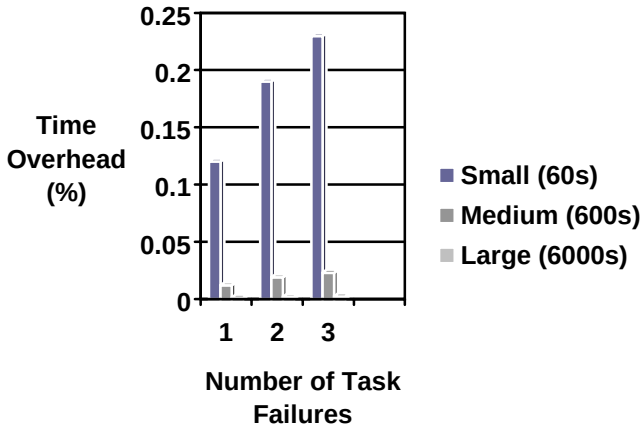


Figure 7 The overhead of Active Redundancy Technique

The second approach is based on using passive redundancy in managing the application faults (see Figure 8). In this scenario, we assign the task to two machines: one is designated as the primary machine while the second machine is designated as the backup machine. The backup machine does not run the task as is done in the active redundancy case, but it is kept up-to-date about the task execution periodically so it can resume the task execution from the last checkpoint (update) if a fault occurred in the primary task. Furthermore, the backup machine could be assigned as

a backup machine for more than one task. This improves the utilization of the system resources. Figure 9 shows the overhead incurred in applying this redundancy technique to manage the faults of three applications. For a small application with three tasks, the overhead incurred to manage one task failure, two task failures, and three task failures are 0.18%, 0.26%, and 0.42%. For a medium to large size application, the overhead to manage one, two or three task failures is very small (less than 0.02%).

It is clear from the experimental results that our approach is very efficient, especially, for large parallel/distributed applications. Furthermore, the use of mobile agents and agent templates, we can dynamically select the appropriate redundancy technique at runtime depending on the system load and number of available resources.

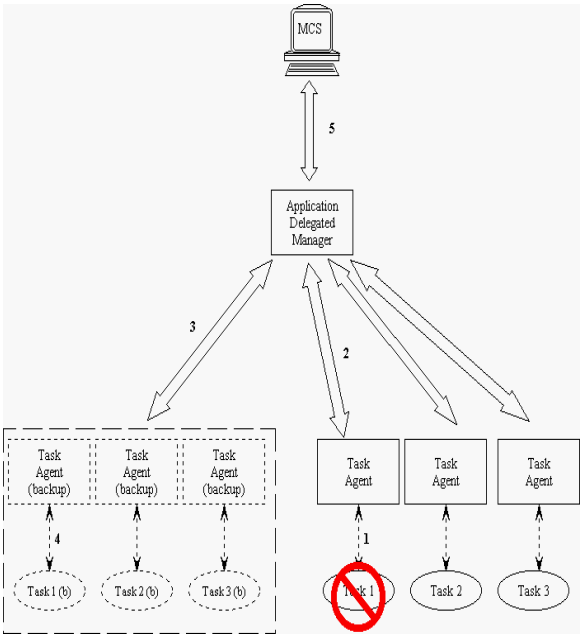


Figure 8 Passive Redundancy Techniques for Fault Management

5. Conclusion

In this paper, we presented our approach to implement a Proactive Application Management System (PAMS). The PAMS architecture is based on integrated management framework being developed at the University of Arizona [8]. The experimental results of the PAMS management services to manage the performance and fault tolerance execution of three applications of different sizes (small, medium and large demonstrate that our agent-based approach can lead to significant gains in performance and low overhead in fault management. We are currently implementing additional services to balance the load across the network resources and maintain the system and application security requirements.

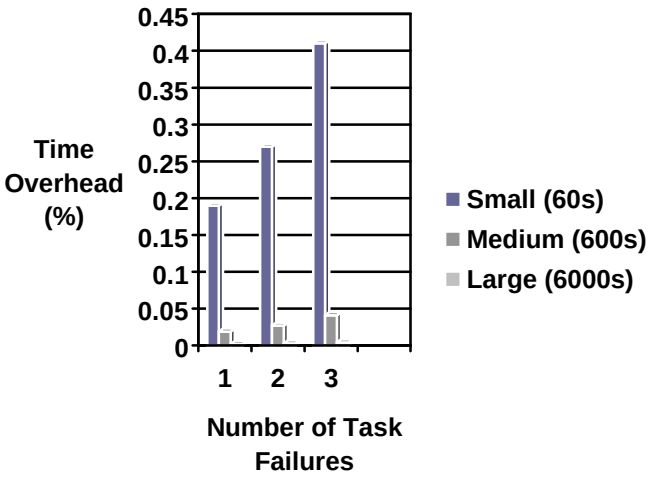


Figure 9 The overhead of Passive Redundancy Technique

6. Reference

[1] S. Waldbusser, Remote Network Monitoring Management Information Base RFC1757, Feb. 1995.

[2] J. Pavon and J. Tomas, CORBA for Network and Service Management in the TINA Framework, IEEE Communication Magazine, March 1998.

[3] J. P. Thompson, Web-Based Enterprise Management Architecture, IEEE Communication Magazine, March 1998

[4] G. Goldszmidt and Y. Yemini, Distributed Management by Delegation, in 15th international Conference on Distbuted Computing, June 1995.

[5] M. Baldi, S. Gai and G. Picco, Exploiting Code Mobility in Decentalized and Flexible Network Management, In First International Workshop, MA97, Berlin, Germany, April 97.

[6] C. Krupczak and J. Saperia, Definition of System-Level Managed Objects for Applications, RFC2287, Feb 1998.

[7] C. Kalbfleisch, C. Krupczak, R. Presuhn, and J. Saperia, Application Management MIB, Internet-draft, Nov. 98.

[8] S. Hariri, Y. Kim, P.Varshney, R.Kamiski. D haugue, C Maciag, The End-to-End Proactive Management. IEEE/IFIP 1998 Network Operations and management Symposium, Feb. 1998
[9] Michael Katchabaw, Stephen Howard, Andrew Marshall, Michael Bauer, "Evaluating the Cost of Management: A Distributed Application Management Testbed," Proceeding of the 1996 CAS conference(CASCON'96) Toronto, Canada, Nov.12-14 pp29-41.
[10] Tomas Fahringer, "Automatic Performance Prediction of Parallel Programs" Kluwer Academic Publishers, 1996
[11] Michael Litzkow, "Supporting Checkpointing and Process Migration outside the Unix Kernel," Usenix Winter Conference, San Francisco, California, 1992
[12] Mohammed Zaki, Wei Li, Srinivasan Parthasarathy "Customized Dynamic Load Balancing of a Network of Workstations,"Technical Report 602, Dec. 1995
[13] Marina Thottan, Chuanyi Ji, "Adaptive Thresholding for Proactive Network Problem Detection, Procedding of the 1998 international workshop for Systems Management, Newport, April, 1998.
[14] P Dini, G. Bochmann,T. Koch, B. Kramer, "Agent based Management of Distributed Systems with Variable Polling Frequency Policies,"
[15] A. Avizienis. "Fault-Tolerant Systems." IEEE Transactions on Computers, C-25(12):1304-1312, December 1976.
[16] P. Jalote. "Fault Tolerance in Distributed Systems." Prentice Hall, 1994
[17] J. Xu, A. Bondavalli, F. D. Giandomenico. "Dynamic Adjustment of Dependability and Efficiency in Fault-Tolerant Software", in "Predictably Dependable Computing Systems", B. Randell, J. C. Laprie, H. Kopetz and B. Littlewood Ed., Springer-Verlag, 1995, pp.155-172.
[18] S. Bagchi, K. Whisnant, Z. Kalbarcyzk, R.K. Iyer. "Chameleon: Adaptive Fault Tolerance Using Reliable, Mobile Agents", The 27th Fault Tolerance Computer Symposium, Munich, Germany, June 23-25 1998

Yoonhee Kim is currently a Ph.D. candidate in the department of Electrical Engineering and Computer Science at Syracuse University and work in a research engineer position at the University of Arizona. She received her M.S. degree in Computer Information Science from Syracuse University, New York at 1996. Her research interests include system, network and application management, distributed and parallel computing systems, and software architecture. Email: yhkim@ece.arizona.edu

Dr. Salim Hariri is currently an Associate Professor in the Department of Electrical and Computer Engineering at The University of Arizona. Dr. Hariri received his Ph.D. in computer engineering from University of Southern California in 1986, and a M.Sc. degree from The Ohio State University. He is the Director of the Center for Advanced TeleSysMatics (CAT): Next-Generation Network-Centric Systems. His current research focuses on high performance distributed computing, agent-based proactive and intelligent network management systems, design and analysis of high speed networks, benchmarking and evaluating parallel and distributed systems, and developing software design tools for high performance computing and communication systems and applications. Dr. Hariri is the co-Editor-In-Chief for the *Cluster Computing*. Dr. Hariri served as the General Chair of the IEEE International Symposium on High Performance Distributed Computing (HPDC).

Muhamad Djunaedi received the B.S. degree in computer and electrical engineering from Purdue University in 1995. Since 1998, he has been studying for M.S. degree in electrical and computer engineering department at University of Arizona. His research interests include mobile agent, fault tolerance, distributed system and management of information system. Email: djunaedi@ece.arizona.edu