Adaptive Distributed Virtual Computing Environment (ADViCE)*

Salim Hariri[1] Dongmin Kim[2] Yoonhee Kim[2] Ilkyeun Ra[2]
Baoqing Ye[2] Xue Bing[2] Haluk Topcuoglu[2] Jon Valente[3]

[1]*Department of Electrical and Computer Engineering*
The University of Arizona
Tucson, Arizona 85721-0104
E-mail: hariri@ece.arizona.edu

[2]*Department of Electrical Engineering and Computer Science*
HPDC Laboratory
Syracuse University
Syracuse, NY 13244-4100

[3]*U.S. Air Force Research Laboratory (Rome Research Site)*
Rome, NY 13441

The next generation of network-centric applications will utilize a large number of computing and storage systems that are connected by global high speed networks. We refer to the environment that provides transparent computing and communication services for large scale parallel and distributed applications as Metacomputing environment. In this paper, we present the design and the experimental results with the Adaptive Distributed Virtual Computing Environment (ADViCE) being developed at The University of Arizona and Syracuse University. The ADViCE provides an efficient web-based approach for developing, running, evaluating and visualizing large-scale parallel and distributed applications that utilize computing resources connected by local and/or wide area networks. ADViCE supports a transparent access to the development, computing and communication services that are offered regardless whether the users are connected through fixed or mobile networks. In addition, the ADViCE resources can also be connected through mobile as well as fixed networks. The ADViCE architecture consists of two independent servers: Visualization and Editing Server (VES) and Control and Management Server (CMS). These two servers provide all the services required in an efficient parallel and distributed programming environment. The ADViCE services include Application Editing Service, Application Visualization Service, Application Resource Service, Application Management Service, Application Control Service and Application Data Service. We also present the experimental results to evaluate the performance and effectiveness of the the ADViCE prototype to provide three important functions: 1) Evaluation Tool: to analyze the the performance of parallel applications with different machine and network configurations; 2) Problem-Solving Environment: to assist in the development of large scale parallel and distributed applications, and 3) Application-Transparent Adaptivity: to allow parallel and distributed applications to run in a transparent manner when their clients and resources are fixed or mobile.

*This research is supported by Rome Laboratory contract number F30602-95-C-0104.

1 Introduction

High performance distributed computing environments capitalize on the emerging high speed network technology, parallel and distributed programming tools and environments, and the proliferation of high performance desktop computers. Recently, there has been an increased interest in building large scale high performance distributed computing (Metacomputing) such as Globus [9], Legion [16], VDCE[2, 4]. These metacomputing projects provide large scale applications with computing and storage power that was once available only in traditional supercomputers. With the proliferation of wireless networks, metacomputing services can be extended to include mobile users and resources. A mobile metacomputing environment allows users not only access to information servers from mobile computers, but also enables them to develop, run, and visualize large scale parallel and distributed applications running on heterogeneous computers that are connected by wired and wireless networks.

The main goal of the ADViCE project is to extend the current VDCE [2, 4] to support mobile users and resources. ADViCE provides a parallel and distributed programming environment; it provides an efficient web-based user interface that allows users to develop, run and visualize parallel/distributed applications running on heterogeneous computing resources connected by wired and wireless networks. Consequently, the fact that some of the resources are mobile such as users, computers, storage devices and networks become transparent to the users and the application developers.

2 Related Work

In this section we provide a brief overview of the issues related to parallel and distributed programming environments and mobile computing.

2.1 Parallel and Distributed Software Development Issues

The software development process of parallel and distributed applications can broadly be described in terms of three phases: a) Application design and specification, b) Application scheduling and resource configuration, and c) Application execution and runtime.

- **Application Design and Specification:** In a well-integrated execution environment it is important to provide: a) an easy-to-use interactive user-interface to design and specify parallel distributed applications and, b) well-developed graphical utilities for the visualization of results and program behavior. Generally, writing parallel and distributed programs overwhelms users due to the difficulty of explicitly expressing communication and synchronization among the computations [1]. A graph-based programming environment, in which a program is defined as a directed graph where nodes denote computations and links denote communication and synchronization between nodes, may be used to decrease the work of programmers. Currently, there are a few visual parallel programming languages and environments, such as Computationally Oriented Display Environment (Code) [5], Heterogeneous

Network Computing Environment (HeNCE) [6], and Zoom [7]. To develop a Code or HeNCE application, a programmer first expresses the sequential computations in a standard language and then specifies how they are to be composed into a parallel program. Zoom is a hierarchical abstraction for describing heterogeneous applications. Zoom representation of an application can be translated into a HeNCE program for execution [6]. Currently, there is an increased interest in developing web-based application development tools and environments because of the explosive use of internet applications [17].
ADViCE graphical user interface is web-based GUI and has been developed using JAVA programming language and JAVA servers.

- **Application Scheduling and Resource Configuration** After the is specified and developed, the application tasks need to be assigned to the available computing and storage resources. In the literature, although the task scheduling (or resource allocation) problem has been investigated extensively, most of the algorithms and systems are valid only for specific architectures and/or certain class of applications. One interesting general scheduling framework is the APPLeS [8]. The APPLeS proposes *application-level scheduling* in which all system aspects are evaluated with respect to application performance. APPLeS develops a customized schedule for each application by including user-specific, application-specific, system-specific, and dynamic information in its scheduling decision. There are resource management systems to provide load sharing and resource allocation such as the Condor project that has been developed at the University of Wisconsin [19]. Condor is a distributed batch system for sharing the workload of compute-intensive jobs in a pool of UNIX workstations connected by a network. In ADViCE, we follow similar approach to APPLeS, where for each parallel and distributed application, the system generates at runtime an adaptive schedule that can optimize the requirements of an application such as performance, fault-tolerance, or security.

- **Application Execution and Runtime:** The application execution and runtime phase executes the developed and configured application. This stage integrates the assigned resources that have been assigned to run the application tasks. The software tools used for the execution of the application can be either based on message-passing tools such as PVM [11], P4 [13], MPI [12], and NCS [14] or based on distributed shared memory (DSM) [23, 24, 25, 26]. In addition, there are a few projects targeted toward providing a metacomputing environment on diverse resources. The earliest metacomputer, the NCSA Metacomputer [15], was an integration of several MPPs, mass storage units, visualization and I/O devices. Globus [9], Legion [16], and VDCE [2, 4] targeted toward the development of metacomputing environments. Additionally, there are several web-based metacomputing projects [17], that either use the JAVA programming language as the main computation language or provide a coordination medium based on WWW technologies or the JAVA language. There may be some drawbacks to these methods. First, they may not support the programs written in other languages such as C and Fortran. Second, they may support communication only between a server and a client, which restricts the execution of the candidate applications. The ADViCE runtime system is based on message passing tools and is implemented using P4 and NCS. We also using JAVA and web-servers to perform all the control, management and visualization functions, while

we use C, C++, Fortran, and any other language to program the application tasks. In other words, our approach is open and can support any language to implement the application tasks.

2.2 Mobile Computing Issues

Mobile computing is increasingly becoming an important programming environment and there has been very little research to address the programming issues in such an environment and how to integrate it into the current parallel distributed programming environments with stationary resources. The main characteristics and constrains of mobile computing are [22, 21]: 1) The use of wireless networks make mobile resources resource-poor relative to stationary resources and the communication performance and reliability varies widely, 2) Mobile resources complicates the issues related to resource locations and portability, and 3) Mobile resources rely on a finite energy resource. The main limitations of developing mobile parallel and distributed programming environments include the following:

- The use of wireless networks implies that applications will experience low transfer rate and unreliable communication links. We expect this limitation to ease in the future as the use of wireless technology expand and more progress is made in increasing the transfer rate over wireless networks.
- The current techniques to support dynamic task migrations and adaptive resource configurations are rigid and can not run efficiently when the computing and storage resources are fixed and/or mobile. For example, it is possible that some of the tasks associated with a parallel and distributed application could be running on several high performance computers that are connected by a fiber-optic high speed network while other tasks are running on computers that are connected by a low speed, unreliable wireless network. The performance of this application will drastically affected by the performance of the communication services offered by the wireless network.

The main goal of the ADViCE prototype is to integrate stationary parallel and distributed computing environment with mobile computing. We developed an efficient approach to support adaptive programming and services for both mobile and stationary resources. In general, there are two extremes for supporting adaptation [22]: 1) Make the adaptation is entirely the responsibility of individual applications, and 2) Make the adaptation is completely transparent to the application and thus must be supported by the system. The first approach avoids the need for system support, but it lacks the ability to resolve incompatible resource demands of different applications and to enforce limits on resource usage. The second approach since it can support adaptivity to existing applications so they can run on mobile resources without any modifications. The adaptivity approach supported in ADViCE is a combination of these two schemes. The user can specify during the application development the application adaptivity requirements. The ADViCE runtime system is responsible for maintaining the adaptivity requirements of the application during its execution.

3 Overview of ADViCE Architecture

The ADViCE can be viewed as a collection of geographically dispersed computational sites or domains, each of which has its own set of ADViCE servers as shown in Figure 1. In any ADViCE, the users, fixed or mobile, access the ADViCE servers (Visualization and Editing Server (VES) and Control and Management Server (CMS)) to develop parallel and distributed applications that can run on fixed or mobile computing resources (see Figure 1). In ADViCE, the users are provided with a seamless parallel and distributed computing environment that provides all the software tools to develop, schedule, run and visualize large scale parallel and distributed applications. In other words, ADViCE supports the following types of transparency:

- Access Transparency: The users can login and access all the ADViCE resources (mobile and/or fixed) regardless of their locations.
- Mobile Transparency: ADViCE supports in a transparent manner mobile and fixed users and resources.
- Configuration Transparency: The resources allocated to run a parallel and distributed application can be dynamically changed in a transparent manner; that is the applications or users do not need to make any adjustment to reflect the changes in the resources allocated to them.
- Fault-Tolerance Transparency: The execution of a parallel and distributed application can tolerate failures in the resources allocated to run that application. The number of faults that can be tolerated depends on the redundancy level used to run the application.
- Performance Transparency: The resources allocated to run a given parallel and distributed application might change dynamically and in a transparent manner to improve the application performance.

Due to some changes in the network traffic or failures, it might be necessary to move the execution environment of one application from one ADViCE domain to another as shown in Figure 1. During the switching from one ADViCE environment to another, one or more ADViCE servers as well as the resources allocated to run a given ADViCE application might be switched. In Figure 1, when the application execution environment is switched from ADViCE1 to ADViCE2, the VES is changed while the CMS is kept the same in both environments.

Our approach to implement the ADViCE architecture is based on identifying a set of servers that are essential to provide the required tools for any parallel and distributed programming environment. The current prototype is built using two web-based servers as shown in Figure 2: Visualization and Editing Server (VES) and Control and Management Server (CMS). The ADViCE architecture can be generalized to more than two servers. However, in our implementation, we used only two servers to simplify the implementation of the required ADViCE services. The VES provides all the editing and visualization services essential for the application development, while the CMS provides all the services required to schedule, control and manage the execution of the application so it can dynamically adapt its execution environment to maintain its quality of service requirements. In what follow, we briefly describe the basic services offered by the ADViCE servers.

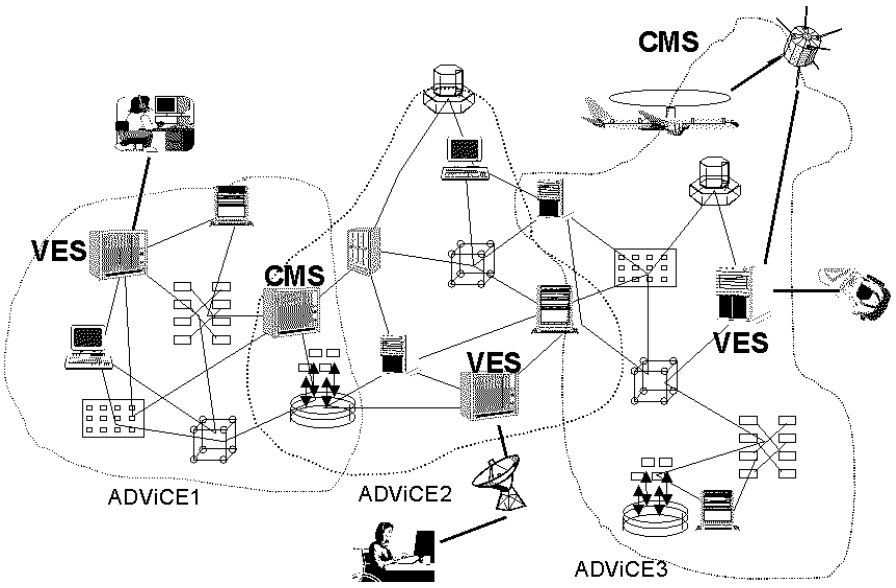

Figure 1: Adaptive Changes in the ADViCE environment.

3.1 Visualization and Editing Server (VES)

This server provides two main application development services: Application Editing Service (AES) and Application Visualization Service (AVS).

3.1.1 Application Editing Service (AES)

The AES is a web-based graphical user interface for developing parallel and distributed applications. The AES provides users with commands to develop and run a new or an existing parallel and distributed application. The main functions offered by the AES are connection establishment and application editor.

- **Connection Establishment:** Before the end-user connects to the appropriate VES, a default server is initially used to fulfill the logical-physical mapping. The default VES will determine the appropriate VES server based on user's location and current system performance parameters. Once the appropriate VES is identified, then the authorization and authentication procedures are invoked by the selected VES server before the user is allowed to use the ADViCE services. After the user passes successfully all the security procedures, the AES invokes the Application Editor window to support the user with the tools required to develop parallel and distributed applications.

- **Application Editor:** The application editor provides menu-driven task libraries that are grouped in terms of their functionality, such as matrix algebra library, command and control task library, etc. A selected task is represented as a clickable and draggable graphical icon in the active editor area. Using the application editor, the

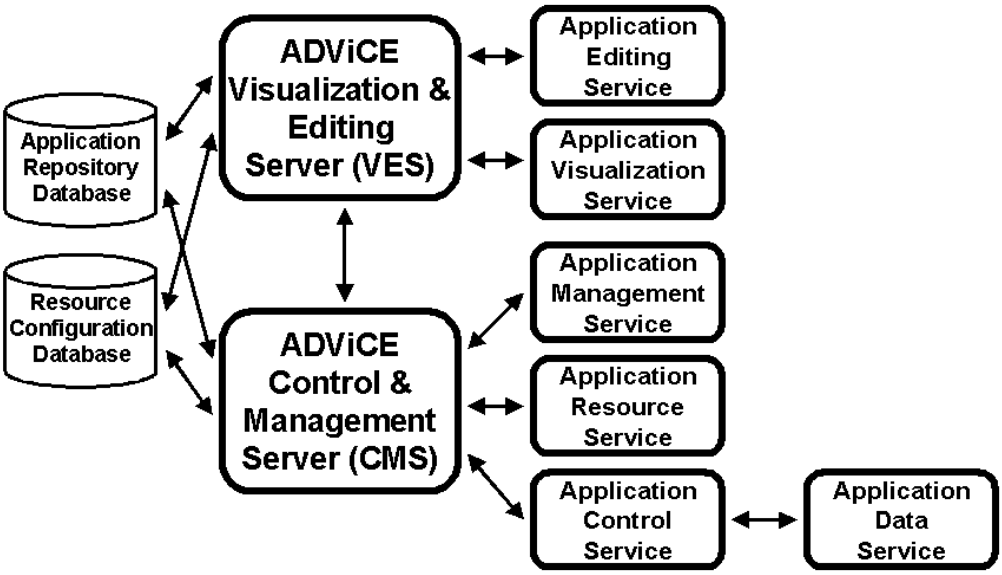

Figure 2: The Main Components of the ADViCE Architecture.

user can develop an Application Flow Graph (AFG) which is a directed graph where the nodes denote library tasks and links denote the communication/synchronization between the nodes. The application editor provides also users with the capability to specify task configuration; that is whether to run each task in sequential or in parallel, and if in parallel how many nodes to execute that task (see Figure 3).

3.1.2 Application Visualization Service (AVS)

This service enables the user to visualize the application execution time and system runtime parameters. For example, Figure 4 shows the execution time for each task in the application shown in Figure 3. In addition, the AVS shows the total execution time of the application and the setup time of the application execution environment.

3.2 Control and Management Server (CMS)

The main services of the CMS include Application Resource Service (ARS), Application Management Service (AMS), Application Control Service (ACS), and Application Data Service (ADS). In addition, the CMS maintains two databases (see Figure 2): one to store the configuration and status information about the resources available in an ADViCE domain (a domain is a distributed computing environment controlled by one organization or an administration), and one database to store the task performance information (e.g. execution times of each ADViCE library task on different computing platforms). The task performance database is used to estimate the task execution time on different computing platforms and is used by the ARS to optimize the allocation of resources to application tasks.

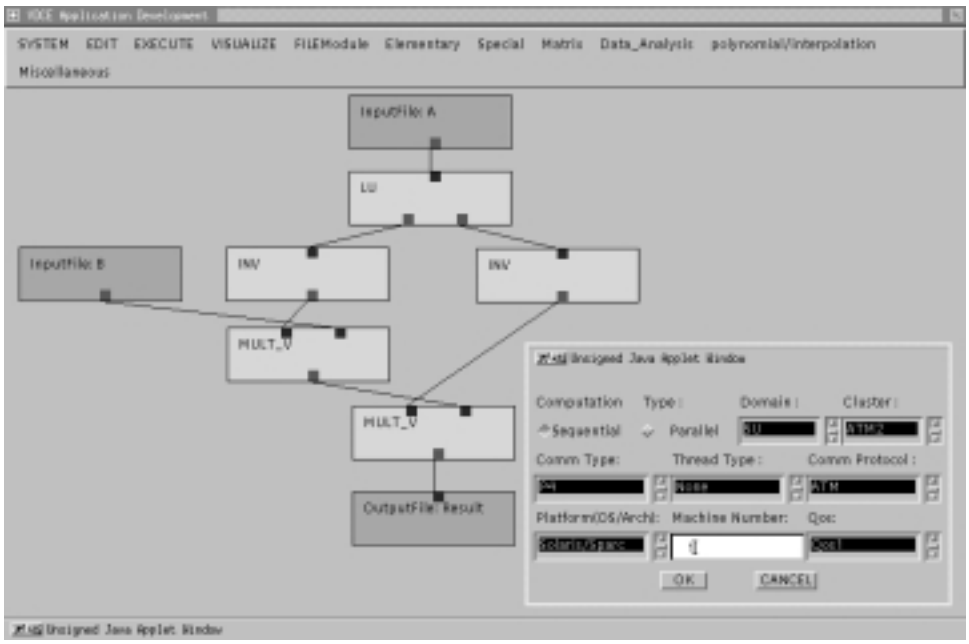

Figure 3: An Application Flow Graph Example.

3.2.1 Application Resource Service (ARS)

The main functions of the ARS is to interpret the application flow graph generated by the AES and then allocates resources to the application tasks to optimize certain criterion such as performance, fault-tolerance, or any other requirements specified by the user. The main functions of the ARS include Performance-based Scheduling, Security-based Scheduling, and Fault Tolerance-based Scheduling. The performance-based scheduling determines the mapping of tasks to resources that will maximize the application performance, while the security-based scheduling allocates to the application tasks only the resources that meet that application security requirements. Similarly, the fault tolerance based scheduling allocates redundant resources to run each application task such that the application execution can tolerate certain number of failures in the resources allocated to execute the application. In addition, the ARS provides application rescheduling capability in order to reallocate some of the application tasks whose executions have been interrupted due to some changes in network and system resources; these changes could be triggered because of the mobility of resources or software/hardware failures in the ADViCE resources.

3.2.2 Application Management Service(AMS)

The AMS utilizes standard management functions to control and manage the execution of parallel and distributed applications. The AMS provides ARS with management information about ADViCE resources to optimize the allocation of application tasks to the currently available ADViCE resources. The AMS also provides a well defined interface

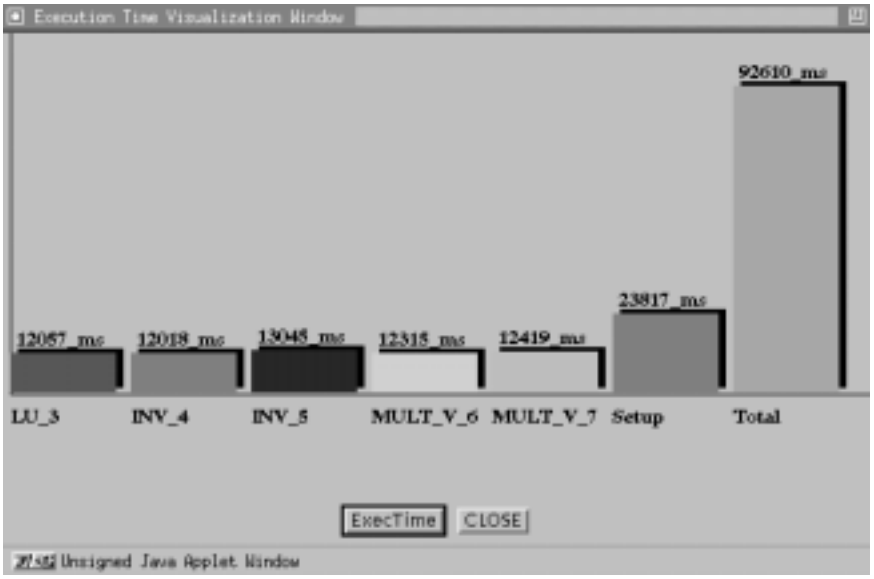

Figure 4: The Performance of each Application Task.

that enables other software modules (e.g. ARS, ACS, ADS) to access any management information required to achieve real-time adaptive services.

3.2.3 Application Control Service (ACS)

The ACS provides applications with the required services to setup, run, control and manage their execution within the ADViCE. The main ACS functions include setting up the application execution environment, monitoring the application execution, and collecting the task performance information required for the visualization of the application execution. In setting up the application execution environment, the ACS launches a proxy process (we refer to as the local-ACS) at each machine selected for the application execution according to the Allocation Channel Table (ACT) generated by the ARS. This involves setting up socket connections between the CMS and the client machines. The local-ACS periodically updates the task performance database and notifies the CMS of any runtime errors.

3.2.4 Application Data Service (ADS)

The ADS provides services to establish high speed communication data paths between the application tasks. In addition, ADS supports limited task management functions such as data conversion, task migration, handling user request exception, and periodically monitoring the task performance.

4 ADViCE Adaptation Approach

One important goal of the ADViCE is to deliver an adaptive parallel and distributed computing environment that can automatically modify its configuration based on the changes in the environment. These changes could be due to failures in hardware, software failure, mobility of resources, or bursty network traffic. The ADViCE adaptation approach follows three important phases or steps: 1) Change Detection, 2) Analysis and Verification, and 3) Adaptation Plan. This approach is similar to the adaptation approach proposed to achieve fault tolerance distributed computing [20]. For each ADViCE service (AES, AVS, ARS, ACS, ADS), we develop the appropriate algorithms to detect the changes in the service once it occurs, to analyze and verify the detected changes in the service, and finally carry out the steps defined in the adaptation plan associated with that service. Figures 5, 6, 7, and 8 show the ADViCE Adaptation Algorithm and procedures.

The Application Execution Environment ($AE(App_i)$) denotes all the resources allocated to run application App_i. While the application is running (Step 1 in the ADViCE Adaptation Algorithm of Figure 5), the ACS monitors all the ADViCE services (Steps 2 through 26 in the ADViCE Adaptation Algorithm of Figure 5) associated with that application to detect any possible changes or deterioration in the application performance. Once any change is detected, the change detection procedure associated with the service that has experienced the changes is invoked (Steps 4, 10, 16, and 22 in the ADViCE Adaptation Algorithm of Figure 5). For example, assume during the application development, the mobile user has experienced an excessive delay because the AES service is running on a VES server that is outside the current location of the mobile user. This is detected when the AES monitoring routine discovers that the communication delay to the VES server is larger than the acceptable D_{max} (Step 1 in Change_Detection_AES of Figure 6). Once that delay is detected, the Verification and Analysis procedure for that service is invoked (Step 6 in the ADViCE Adaptation Algorithm of Figure 5). In a similar manner, we device detection algorithms for each service offered by the ADViCE servers (VES and CMS) as shown in Figure 7.

The Verification and Analysis procedures shown in Figure 7 involves analyzing the current state of the system resources by using the AMS services to validate and identify accurately the event(s) that contributed to the changes if they were proven to be true and not false or transient. For example, if the change detection procedure of the ADS has determined the EventType to be "link failure" (Step 4 in Change_Detection_ADS of Figure 6). This event could be caused by the machine being down or task failure (Step 11 through 18 in Verification_Analysis_ADS of Figure 7). The verification and analysis could be simply reading the OperStatus in the interface MIB associated with each communication link used for the inter-task communications. If the status of is found to be caused by machine failure, then the EventCause is assigned as "machine down" and then the Adaptation Plan associated with ADS is invoked as shown in Figure 5 (Step 25).

The Adaptation Plan procedures involves taking the appropriate actions to enable the ADViCE to adapt to the changes that have been detected and verified. The adaptation plan procedure invoke the appropriate operations associated with the adaptation of each service. For example, the adaptation plan for the ADS associated with "task down" could be to restart the application execution from the beginning (Step 17 in Adaptation_Plan_ADS of Figure 8).

```
   procedure ADViCE_Adaptation_Algorithm
1      while ( AE(App_i) is running ) do {
2          monitor ADViCE_Services
3              monitor AES
4                  EventType ← Change_Detection_AES()
5                  if EventType ≠ Normal
6                      EventCause ← Verification_Analysis_AES(EventType, AE(App_i))
7                      Adaptation_Plan_AES(EventCause, AE(App_i))
8                  endif
9              monitor AVS
10                 EventType ← Change_Detection_AVS()
11                 if EventType ≠ Normal
12                     EventCause ← Verification_Analysis_AVS(EventType, AE(App_i))
13                     Adaptation_Plan_AVS(EventCause, AE(App_i))
14                 endif
15             monitor ACS
16                 EventType ← Change_Detection_ACS()
17                 if EventType ≠ Normal
18                     EventCause ← Verification_Analysis_ACS(EventType, AE(App_i))
19                     Adaptation_Plan_ACS(EventCause, AE(App_i))
20                 endif
21             monitor ADS
22                 EventType ← Change_Detection_ADS()
23                 if EventType ≠ Normal
24                     EventCause ← Verification_Analysis_ADS(EventType, AE(App_i))
25                     Adaptation_Plan_ADS(EventCause, AE(App_i))
26                 endif
27     } endwhile
   end of ADViCE_Adaptation_Algorithm
```

Figure 5: ADViCE Adaptation Algorithm

```
    procedure Change_Detection_AES()
1       if t_connect(VES) > D_max
2           EventType = unacceptable delay to VES
3       else if unable to locate VES
4           EventType = VES down
5       else if unable to locate the database server
6           EventType = database down
                    ⋮
7       else
8           EventType = Normal
9       endif
10      return(EventType)
    end of Change_Detection_AES

            ⋮

    procedure Change_Detection_ADS()
1       if inter task communication delay > D_max
2           EventType = inter task communication delay
3       else if broken pipe detected
4           EventType = link failure
                    ⋮
5       else
6           EventType = Normal
7       endif
8       return(EventType)
    end of Change_Detection_ADS
```

Figure 6: ADViCE Change Detection Procedures

```
   procedure Verification_Analysis_AES(EventType, AE(App_i))
1      case EventType = unacceptable delay to VES
2          verify delay to VES
3          if measure the delay to VES > D_max
4              check if the delay is caused by the location of VES
5                  EventCause = location change of VES
6              check if the delay is caused by the location of the user
7                  EventCause = user's location change
8              check if the delay is caused by heavy load VES
9                  EventCause = heavily loaded VES
10         endif
11     case EventType = VES down
12         verify VES down by AMS MIB
13         if true
14             check if VES down is caused by the VES machine failure
15                 EventCause = VES machine down
16         endif
17     case EventType = database down
18         verify database down by AMS MIB
19         if true
20             check if database down is caused by database machine down
21                 EventCause = Application Repository database machine down
22             check if database down is caused by database server down
23                 EventCause = Application Repository database server down
24         endif
                   ⋮
25     return(EventCause)
   end of Verification_Analysis_AES
           ⋮

   procedure Verification_Analysis_ADS(EventType, AE(App_i))
1      case EventType = inter task communication delay
2          verify the communication delay
3          if measure inter task delay > D_max
4              check if the delay is caused by heavy network traffic
5                  EventCause = heavy traffic
6              check if the delay is caused by heavy load
7                  EventCause = heavy CPU load
8          endif
9      case EventType = link failure
10         verify link failure by AMS MIB
11         if true
12             check if link failure is caused by machine down
13                 EventCause = machine down
14             check if link failure is caused by task down
15                 EventCause = task down
16         endif
                   ⋮
17     return(EventCause)
   end of Verification_Analysis_ADS
```

Figure 7: Verification and Analysis Procedures

```
    procedure Adaptation_Plan_AES(EventCause, AE(App_i))
1       case EventCause = location change of VES or
2            EventCause = user's location change or
2            EventCause = heavily loaded VES or
3            EventCause = VES machine down
4            | access the default VES
5            | locate a new VES
6            | transfer the information from current VES to a new VES
7       case EventCause = Application Repository database machine down
8            | choose alternative Application Repository database
9       case EventCause = Application Repository database server down
10           | start the database
                 ⋮
    end of Adaptation_Plan_AES
          ⋮

    procedure Adaptation_Plan_ADS(EventCause, AE(App_i))
1       case EventCause = heavy traffic or
2            EventCause = heavy load or
3            EventCause = machine down
4            | invoke ARS to assign a new machine
5            | if migration required
6            |   task migration
7            | endif
8            | if partial recovery is possible
9            |   resume from the stopped task
10           | else
11           |   resume from the task check pointed state
12           | endif
13      case EventCause = task down
14           | if partial recovery is possible
15           |   resume from the task check pointed state
16           | else
17           |   start the application from the beginning
18           | endif
                 ⋮
    end of Adaptation_Plan_ADS
```

Figure 8: Adaptation Plan Procedures

5 ADViCE Testbed: Experimental Results and Discussion

The current ADViCE prototype consists of two sites, one at Syracuse University and the other at Rome Laboratory, that are connected by the OC3 ATM Wide Area Network, as shown in Figure 9. We are currently setting up a new site at the University of Arizona. Each site or domain has two ADViCE servers that manage the computing and network resources available in their site. At the Syracuse University site there are three computing clusters: HPDC, CAT, and TOP. The HPDC cluster consists of several ATM switches and ATM concentrators that connect high-performance workstations and PCs at a rate of 155 and 25 Mbps, respectively (URL:http//www.atm.syr.edu). The TOP and CAT clusters have SUN SPARCs, SUN IPXs and IBM RS6000s that are connected to the ATM cluster through an Ethernet network. The Rome Lab site consists of three clusters that include SUN, Digital, and HP workstations.

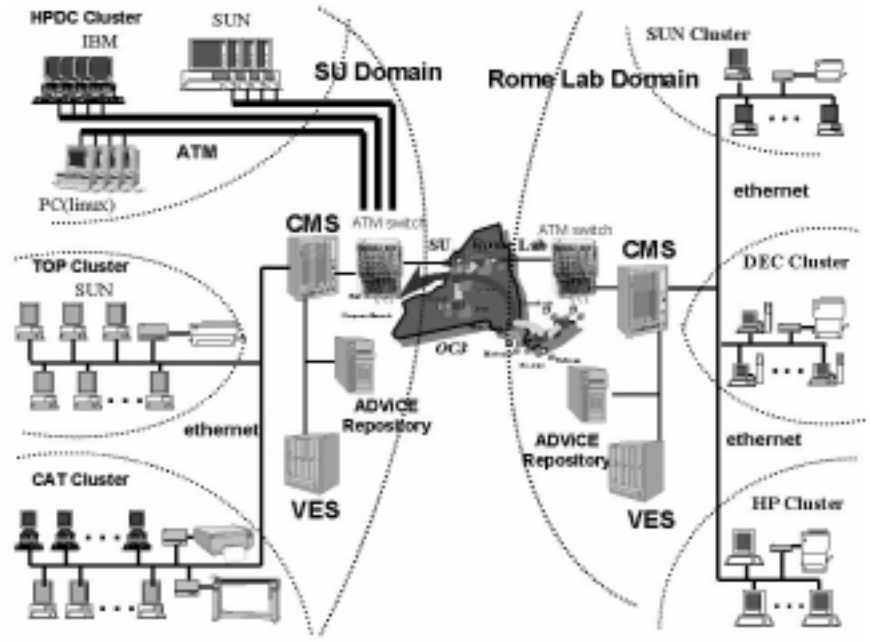

Figure 9: The configuration of the current ADViCE Testbed.

In this section we discuss and evaluate the performance of three important functions supported by the ADViCE prototype: 1) Task Performance Evaluation Tool, 2) Problem-Solving Environment, and 3) Adaptation Support.

5.1 Experiment 1: Using ADViCE as a Parallel Evaluation Tool

In this experiment we used the matrix-vector multiplication (MULT_V) task as a running example to evaluate the use of the ADViCE prototype as an evaluation tool to analyze the performance of different configurations when the number of computers, network types,

and problem sizes are changed. We compared the time and effort required to perform such tasks with and without using the ADViCE prototype. We benchmarked the sequential and parallel algorithms of matrix-vector multiplication(MULT_V) based on various machine and network configurations and problem sizes. The parallel implementation of the MULT_V ($A \times B = C$) task is based on the host-node programming model. The master process distributes the rows of matrix A evenly among the processes (where each process runs on one workstation) while all the slave processes receive the entire B matrix. Each slave process computes its part of the result matrix C and sends it back to the host process.

The ADViCE provides a web-based, user-friendly interface that allows a novice programmer to experiment with and evaluate different parallel configurations of each ADViCE task in a few minutes. We argue that performing similar evaluation tasks is almost impossible for novice programmers and requires hours and even days to be performed by an expert programmer in parallel processing, message passing and visualization tools. Using ADViCE prototype, once a task is registered in the ADViCE task library, the user can use that task or any other library tasks by just clicking on the task name in the Application Editor window. Once the task is selected, the user can specify the desirable configuration to run the selected task; specify the number of computers to be involved in the computation, and the network to be used to connect them if the task is going to run in parallel. Selecting the ADViCE task and specifying how it will be implemented can be done in a few minutes. Once that is done, the task configuration can be executed and its execution time can be visualized immediately without any effort other than clicking on the execute and visualize buttons in the Application Editor window.

Figure 10 shows the execution times of a matrix multiplication algorithm for two problem sizes, 512×512 and 1024×1024. The result for a p4-based implementation of the same multiplication algorithm is given in Figure 11. The experiments were done for one, two and four Sun SPARCs that are connected by an IP/ATM network. We also evaluated the performance of the MULT_V task on a heterogeneous cluster of four SUN SPARCs and four IBM RS6000 workstations. The objective of such an evaluation is to provide users with a better understanding of the performance of parallel algorithms when there is a change in problem size, number of nodes, or network type. As an example, for the p4-based implementation of the matrix-vector multiplication algorithm, we can determine from Figure 11 that eight nodes provide the best performance among the test cases.

Table 1 compares the times required to develop, compile, execute, and visualize the Matrix-Vector Multiplication task using p4 and the ADViCE prototype for a 1024×1024 problem size with four nodes. In the design and implementation phase, it takes around 862 minutes for a parallel programming expert to develop a p4-based multiplication program from scratch if we assume that programming speed is two minutes per line. If the programmer has no experience with p4, he/she will spend more time to learn the tool and then implement the parallel algorithm. For the ADViCE, even if the user does not have any knowledge in parallel programming, but wants to run the application in parallel, the only thing he/she needs to do is to choose the parallel option in the task configuration window of the Application Editor. The total time for developing the ADViCE MULT_V application is 2.10 minutes rather than 862 minutes if one needs to develop the application

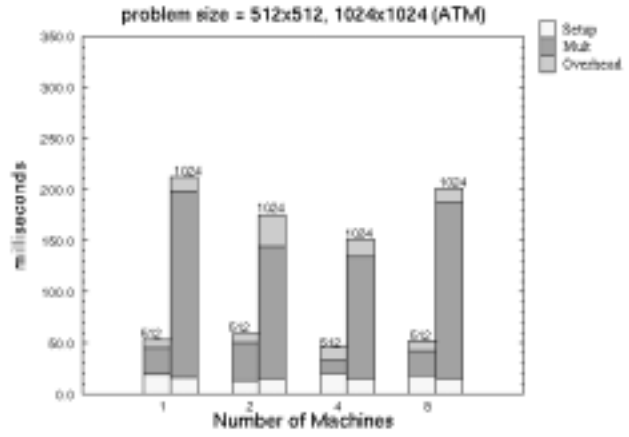

Figure 10: The Performance of the ADViCE Implementation of the Matrix-Vector Multiplication.

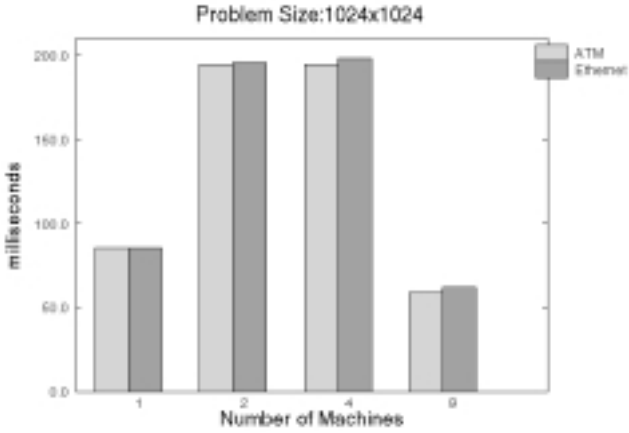

Figure 11: The Performance of the P4 Implementation of the Matrix-Vector Multiplication.

Table 1
The performance comparison of the matrix-vector multiplication task for each software development phase

Phase	p4	ADViCE
Design and development	862 min. (431 lines)	2.10 min.
Compilation	7.01 sec.	0 sec.
Runtime setup	0.980 sec.	0.015 sec.
Task execution	0.194 sec.	0.136 sec.
Visualization and evaluation	1890sec.	0.095 sec.

from a scratch.

The location of the executable for the MULT_V task on the selected resource is provided in the resource allocation information, which is retrieved from the task constraints table. The executable is then linked to the I/O module. In the p4 version the MULT_V program, it takes 7.01 seconds for compilation. The runtime setup time for the ADViCE prototype consists of the time it takes the ACS to transfer the activation and resource allocation information to the ADS and the time for the acknowledgment. This setup time takes 0.015 seconds for the MULT_V task on the selected resource. For a p4 application, the user creates a configuration file, i.e., the procgroup file, and manually links it to the p4 application which takes 0.98 seconds. ADViCE runs the application automatically with the "Execute Application" button and generates the results in the selected output file. The execution time of the MULT_V task is 0.136 seconds when it is executed on four nodes over the ATM. The execution time is 0.194 seconds using a p4 program with the same configuration.

In addition, ADViCE provides dynamic and post-mortem visualization of the application. The user can visualize the loads of all the machines in one domain and can even focus on the load information for the machines selected to run a given application. Furthermore, the execution time of each module within an application is visualized in ADViCE. It takes 0.095 seconds to invoke the ADViCE visualization window for the MULT_V task. If a p4 user wants to visualize the execution time to compare its performance with others, it is necessary to use another graphic tool. The visualization and evaluation time depends on which tool is used; as an example, "gnuplot" takes 1890 seconds.

5.2 Experiment 2: Using ADViCE as a Problem Solving Environment

In this experiment we demonstrate how the ADViCE can enable a novice programmer to develop large-scale parallel and distributed applications running on geographically distributed heterogeneous resources. Implementing such applications is currently a challenging programming problem and time consuming for even experts in parallel and distributed programming tools. A distributed application can be viewed as an Application Flow Graph (AFG), where its nodes denote computational tasks and its links denote the communications and synchronization between these nodes. Without an application development tool, a developer or development team must apply much effort and time to develop a distributed application from a scratch. To solve these difficulties, ADViCE

Table 2
Performance comparison of the linear equation solver application for each software development phase [1]

	p4			ADViCE		
Phase	LU	INV	MULT_V	LU	INV	MULT_V
Design and development	838 min. (419 lines)	1314 min. (657 lines)	862 min. (431 lines)	2.10 min.	1.57 min.	2.30 min.
Compilation	6.45 sec.	8.10 sec.	7.01 sec.	0 sec.	0 sec.	0 sec.
Runtime setup	1.200 sec.	1.580 sec.	0.980 sec.		0.043 sec [2]	
Task execution	0.386 sec.	0.556 sec.	0.194 sec.	0.801 sec.	1.360 sec.	0.140 sec
Application execution		1.691 sec.			1.451 sec.	
Application visualization		3200 sec.			0.140 sec.	

provides an integrated problem solving environment to enable novice users to develop large-scale, complex, distributed applications using ADViCE tasks. The Linear Equation Solver (LES) application has been selected as a running example. Figure 3 shows the AFG of the Linear Equation Solver, which consists of an LU Decomposition (LU) task, two Matrix Inversion (INV) tasks and Matrix-Vector Multiplication (MULT_V) tasks. The problem size for this experiment is 1024×1024 and its execution environment consists of four nodes, which are SUN SPARCs and IBM RS6000 machines that are connected by an ATM network.

Table 2 compares the timing of several software phases for the Linear Equation Solver application using p4 and ADViCE. When a user has enough knowledge about parallel programming and the p4 tool, he/she will spend 838 minutes for an LU task, 1314 minutes for an INV task, and 862 minutes for MULT_V task. The total time to develop this application is approximately 3014 minutes, (i.e., around 50 hours). Using ADViCE, a novice user spends around six minutes to develop such an application. There is no compile time in ADViCE, but a p4 application needs 21 seconds for compilation. The ADViCE setup time for a Linear Equation Solver application is 0.043 seconds. The p4 user should create all procgroup files and launch them in order, which takes around eight seconds.

Since the ADViCE is based on the data flow model and executes the application tasks concurrently, the application execution time, including the setup time, is less than the summation of all the individual task execution times. In our experiment with the Linear Equation Solver application, the total execution time of the p4 implementation using four nodes is 1.691 seconds. The ADViCE implementation of the same application with the same configuration is approximately 1.451 seconds.

5.3 Experiment 3: Evaluation of the ADViCE Adaptation Approach

One of the main features of the ADViCE prototype is its transparent adaptation support. In this experiment, we evaluate the performance of the ADViCE prototype to develop a

[1] The last two rows of the table are for the total time of the application.
[2] It is the total setup time for a ADViCE-based linear equation solver application.

fault tolerant distributed application that is shown in (Figure 12).

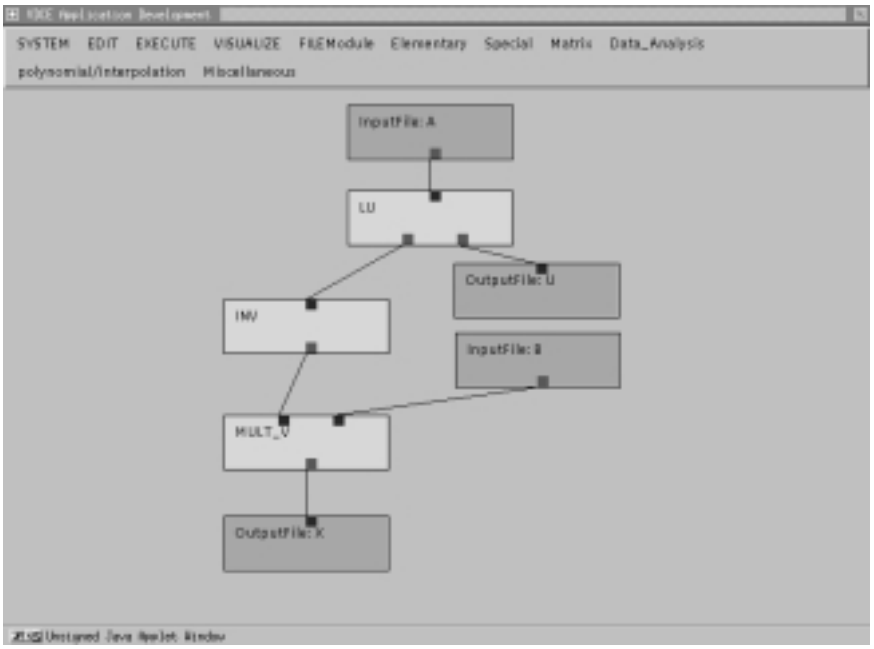

Figure 12: A Fault Tolerant Distributed Application Example.

After a user develops an application using the ADViCE Application Editor window (AES) and specifies that the application tasks should tolerate link and machine failures. During the application execution, we manually kill the process running one of the application tasks, say the INV task, as shown in Figure 12. The INV task failure is immediately detected by the Local ACS that continuously monitoring the execution of the of the INV task (Step 1 in Detection and Analysis Phase of Figure 13). The error message is reported to the Server ACS running on the CMS (Step 1' and Step 2). The next step is to invoke the Verification_Analysis_ADS procedure that is running on the Server ACS of the CMS (Step 3) that determines that the EventCause is "Task down" (Step 15 in the Verification_Analysis_ADS of Figure 7. Once that is determined, the Adaptation_Plan_ADS procedure is invoked. A simple recovery procedure could be to restart all the application tasks (LU, INV, and MULT_V). This recovery procedure involves invoking the ARS to reschedule resources to the application (see step 1 in Adaptation Phase of Figure 13). Once the ARS schedules the application tasks and passes it to the Server ACS (Step 2), the Server ACS setups the new application execution environment by starting the Local ACS on each machine selected to run the application (Step 3). Once that is done, the Local ACS starts the task execution on its machine (Step 4).

The performance of the adaptation algorithm depends on the the Change Detection Time (CDT), Verification and Analysis Time (VAT), and Adaptation Plan Time (APT). The CDT measures the time it takes ADViCE to detect the change event in any of ADViCE services. The VAT measures the time it takes ADViCE to verify the change event and determine its cause type. The APT measures the time it takes ADViCE

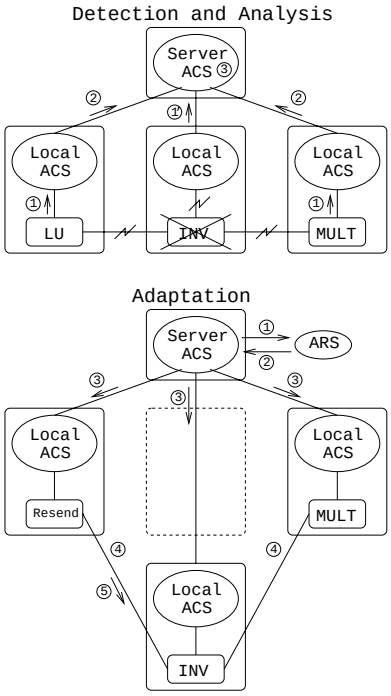

Figure 13: an Example on the ADViCE Adaptation Algorithm.

to perform the operations specified in the adaptation plan associated with the affected service. For the example shown in Figure 13, the CDT is 7.675 seconds, VAT is 5.328 seconds and APT is 18.451 seconds. We are currently evaluating different techniques to achieve efficient implementations of all the procedures identified in the three phases of the ADViCE adaptation algorithm.

6 Conclusion

In this paper, we presented an overview of the Adaptive Distributed Computing Environment(ADViCE) being developed at the University of Arizona and Syracuse University. The ADViCE consists of two main servers: Visualization and Editing Server (VES) and Control and Management Server (CMS). These two servers provide all the services required to develop parallel and distributed applications, run, control, manage, and visualize the execution of these applications. We have successfully implemented a proof-of-concept prototype of the ADViCE architecture that provides most of the ADViCE services. We also presented our experimental results and evaluation of the utility of the services supported by the ADViCE prototype to achieve efficient and seamless parallel and distributed programming environment. We are currently extending the capabilities of ADViCE to provide efficient adaptive scheduling algorithms and proactive management services.

References

[1] J. C. Browne, S. Hyder, J. Dongarra, K. Moore, P. Newton, Visual programming and debugging for parallel computing, IEEE Parallel and Distributed Technology, 3(1) (1995) 75–83.

[2] H. Topcuoglu, S. Hariri, W. Furmanski, J. Valente, I. Ra, D. Kim, Y. Kim, X. Bing, B. Ye, The software architecture of a virtual distributed computing environment, in *Proceedings of Sixth IEEE International Symposium on High Performance Distributed Computing*, 1997, pp. 40–49.

[3] H. Topcuoglu and S. Hariri, A global computing environment for networked resources, in *Proceedings of International Conference on Parallel Processing*, 1997, pp. 493–496.

[4] H. Topcuoglu, S. Hariri, D. Kim, Y. Kim, X. Bing, B. Ye, I. Ra and J. Valente, The Design and Evaluation of a Virtual Distributed Computing Environment, *J. of Networks, Software Tools and Applications (Cluster Computing)*, 1998.

[5] P. Newton, J. C. Browne, The CODE 2.0 graphical parallel programming language, in *Proceedings of ACM International Conference on Supercomputing*, 1992.

[6] R. Wolski, C. Anglano, J. Schopf, F. Berman, Developing heterogeneous applications Using Zoom and HeNCE, in *Proceedings of the Forth Heterogeneous Computing Workshop*, 1995.

[7] C. Angalano, J. Schopf, R. Wolski, F. Berman, Zoom: a hierarchical representation for heterogeneous applications, technical report cs95-451, Computer Science Department, University of California at San Diego, 1995.

[8] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, Application-level scheduling on distributed heterogeneous networks, in *Proceedings of Supercomputing 96*, 1996.

[9] I. Foster and C. Kesselman, Globus: a metacomputing infrastructure toolkit, in *Proceedings of the Workshop on Environment and Tools for Parallel Scientific Computing*, 1996.

[10] H. Casanova and J. Dongarra, Netsolve: a network server for solving computational science problems, in *Proceedings of Supercomputing 96*, 1996.

[11] A. Beguelin, J. Dongara, A. Geist, R. Manchek, and V. Sunderam, User Guide to PVM, Oak Ridge National Laboratory and Department of Mathematics and Computer Science, Emory University, 1993.

[12] Message Passing Interface Forum, MPI: A message-passing interface standard, version 1.0 May 1994.

[13] R. Butler and E. Lusk, User's guide to the p4 programming system, Mathematics and Computer Science Division, Argonne National Laboratory.

[14] S. Park, S. Hariri, Y. Kim, J.S. Harris, and R. Yadav, NYNET communication system (NCS): a multithreaded message passing tool over ATM network, *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, 1996, pp. 460–469.

[15] L. Smarr and C. Catlett, Metacomputing, Communications of the ACM, 35, 6, (June 1992) 44–52.

[16] A. Grimshaw and W. Wulf, Legion - A View from 50,000 Feet, *Proceedings of Fifth IEEE International Symposium on High Performance Distributed Computing*, 1996, pp. 89–99.

[17] K. Dincer, *World-Wide Virtual Machine: A Metacomputing Environment Integrating World-Wide Web and High Performance Computing and Communication Technologies,* Ph.D. Thesis, Syracuse University, 1997.

[18] J. Gehring and A. Reinefeld, MARS - A framework for minimizing the job execution time in a metacomputing environment, Future Generation Computing Systems, (1996).

[19] Mike Litzkow, Miron Livny Experience with the condor distributed batch system, in *IEEE Workshop on Experimental Distributed Systems*, 1990.

[20] M. A. Hiltunen and R. D. Schlichting, Adaptive Distributed and Fault-Tolerant Systems, *International Journal of Computer Systems Science and Engineering*, vol. 11, nbr. 5, pp.

125-133, September 1996.

[21] George H. Forman, John Zahorjan, The Challenges of Mobile Computing, *IEEE Computer*, Vol. 27, pp. 33-47, April, 1994.

[22] M. Satyanarayanan, Fundamental Challenges in Mobile Computing, *Fifteenth ACM Symposium on Principles of Distributed Computing*, Philadelphia, May, 1996.

[23] S. Ahuja, N. Carriero, and D. Gelernter, Linda and Friends, *IEEE Computer*, vol. 18, No. 8 , pp. 26-34, August, 1986.

[24] P. Keleher, S. Dwarkadas, A. Cox and W. Zwaenepoel, Treadmarks: Distributed shared memory on standard workstations and operating systems, *Proceedigns of the 1994 Winter Usenix Conference*, pp. 115-131, January, 1994.

[25] K. Johnson, M. Kasshoek and D. Wallach, CRL: High-Performance All-Software Distributed Shared Memory, *Proceedings of the Fifteenth Symposium on Operating Systems Principles*, December, 1995.

[26] W. Liang, C. King and E. Lai, Adsmith:An efficient object-oriented DSM environment on PVM, *Proceedings of the 1996 International Symposium on Parallel Architecture, Algorithms and Networks*, pp. 173-179, June 1996.