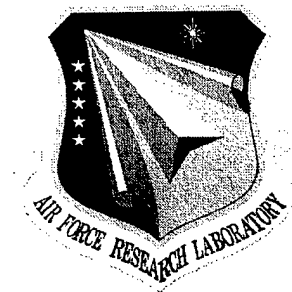


AFRL-IF-RS-TR-2000-24
Final Technical Report
March 2000



VIRTUAL DISTRIBUTED COMPUTING ENVIRONMENT

Syracuse University

Salim Hariri, Dongmin Kim, Yoonhee Kim, and Ilkyeun Ra

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

20000420 145

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2000-24 has been reviewed and is approved for publication.

APPROVED:



JON B. VALENTE
Project Engineer

FOR THE DIRECTOR:



WARREN H. DEBANY, JR., Technical Advisor
Information Grid Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFGA, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE MARCH 2000		3. REPORT TYPE AND DATES COVERED Final Jul 95 - Sep 98
4. TITLE AND SUBTITLE VIRTUAL DISTRIBUTED COMPUTING ENVIRONMENT			5. FUNDING NUMBERS C - F30602-95-C-0104 PE - 62702F PR - 5581 TA - 21 WU - AK	
6. AUTHOR(S) Salim Hariri, Dongmin Kim, Yoonhee Kim, and Ilkyeun Ra				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Syracuse University Office of Sponsored Programs 113 Bowne Hall Syracuse NY 13244-1200			8. PERFORMING ORGANIZATION REPORT NUMBER N/A	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFGA 525 Brooks Road Rome NY 13441-1200			10. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2000-24	
11. SUPPLEMENTARY NOTES Air Force Research Laboratory Project Engineer: Jon B. Valente/IFGA/(315) 330-3241				
12a. DISTRIBUTION AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) 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 report, we present the design and the experimental results with the Virtual Distribute Computing Environment (VDCE) and the Adaptive Distributed Virtual Computing Environment (ADVice) being developed at the University of Arizona and Syracuse University				
14. SUBJECT TERMS Virtual Environments, Application Environments, Virtual Machine, High-Performance Computing			15. NUMBER OF PAGES 56	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

Contents

1	Introduction	1
2	Related Work	3
3	Overview of VDCE Software Architecture	5
3.1	Application Editor	6
3.2	Application Scheduler	8
3.3	VDCE Runtime System	12
4	VDCE Testbed: Experimental Results and Discussion	15
4.1	Experiment 1: Using VDCE as a Parallel Evaluation Tool	16
4.2	Experiment 2: Using VDCE as a Problem Solving Environment	19
5	ADAPTIVE DISTRIBUTED VIRTUAL COMPUTING ENVIRONEMNT (ADViCE)	20
5.1	Introduction	20
5.2	Related Work	21
5.3	Parallel and Distributed Software Development Issues	21
5.4	Mobile Computing Issues	22
5.5	Overview of ADViCE Architecture	23
5.5.1	Visualization and Editing Server (VES)	25
5.5.2	Control and Management Server (CMS)	27
5.6	ADViCE Adaptation Approach	28
6	ADViCE Testbed: Experimental Results and Discussion	34
6.1	Experiment 1: Using ADViCE as a Parallel Evaluation Tool	34
6.2	Experiment 2: Using ADViCE as a Problem Solving Environment	37
6.3	Experiment 3: Evaluation of the ADViCE Adaptation Approach	38
7	Conclusion	40
	References	41

List of Figures

1	Virtual Distributed Computing Environment (VDCE)	5
2	Interactions Among the VDCE Modules	6
3	VDCE Authentication Window	7
4	Building the Linear Equation Solver Application with the Application Editor	8
5	Interactions Among the Control Virtual Machine Components	14
6	Setting Up the Application Execution Environment	14
7	The configuration of the VDCE Testbed	16
8	Execution Time of Matrix Multiplication Task Using VDCE	17
9	Execution Time of Matrix Multiplication Task Using p4	18
10	Adaptive Changes in the ADViCE environment.	24
11	The Main Components of the ADViCE Architecture.	25
12	An Application Flow Graph Example.	26
13	The Performance of each Application Task.	27
14	ADViCE Adaptation Algorithm	30
15	ADViCE Change Detection Procedures	31
16	Verification and Analysis Procedures	32
17	Adaptation Plan Procedures	33
18	The configuration of the current ADViCE Testbed.	34
19	The Performance of the ADViCE Implementation of the Matrix-Vector Multiplication.	36
20	The Performance of the P4 Implementation of the Matrix-Vector Multipli- cation.	36
21	An Example of Fault Tolerant Distributed Application.	39
22	An Example of the ADViCE Adaptation Algorithm.	40

List of Tables

Table 1	Symbols and their meaning.....	9
Table 2	The performance comparison of matrix multiplication task for each software phase...	18
Table 3	Performance comparison of linear equation solver application for each software phase.....	20
Table 4	The performance comparison of the matrix-vector multiplication task for each software Development phase.....	37
Table 5	Performance comparison of the linear equation solver application for each software Development phase.....	38

Abstract

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 project, we present the design and the experimental results with the Virtual Distributed Computing Environment (VDCE) and the Adaptive Distributed Virtual Computing Environment (ADViCE) being developed at The University of Arizona and Syracuse University.

The VDCE 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. The VDCE task libraries relieve end-users of tedious task implementations and also support reusability. The VDCE software architecture is described in terms of three modules: a) the Application Editor, a user-friendly application development environment that generates the Application Flow Graph (AFG) of an application; b) the Application Scheduler, which provides an efficient task-to-resource mapping of AFG; and c) the VDCE Runtime System, which is responsible for running and managing application execution and for monitoring the VDCE resources. We present experimental results of an application execution on the VDCE prototype for evaluating the performance of different machine and network configurations. We also show how VDCE can be used as a problem-solving environment on which large-scale, network-centric applications can be developed by a novice programmer rather than by an expert in low-level details of parallel programming languages.

The ADViCE which is an extension of the VDCE aims at supporting mobile computing and communication resources. 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 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.

1 Introduction

Grand challenge problems have computational and storage resource requirements that are beyond the capacities of a single computing environment. Additionally, emerging net-

work technologies such as fiber-optic transmission facilities and the Asynchronous Transfer Mode (ATM) enable data to be transferred at the rate of a gigabit per second (Gbps). Since high-speed networks have become more common and provide low latency communication services that are close to those offered by massively parallel processors (MPPs), there is a growing interest in combining the computational and storage resources that are available over the wide area networks to build a new execution environment called *metacomputing* [21]. New software tools and techniques are required to utilize the metacomputing resources, which are not fully supported by existing parallel or distributed software. The heterogeneous and dynamic nature of a metacomputing environment limits the use of existing parallel computing tools; similarly, the existing distributed systems may not provide the high performance that is a key target in a metacomputing environment.

In this report we present the design of the Virtual Distributed Computing Environment (VDCE) currently that has been developed at Syracuse University. In addition, we also present the design and the experimental results of the Adaptive Distributed Virtual Computing Environment (ADViCE) which is an extension of the VDCE to support mobile computing and communication resources.

VDCE [8, 9] provides an efficient mechanism to execute large-scale applications on distributed and diverse platforms. The main goal of the VDCE project is to develop an easy-to-use, integrated software development environment that provides software tools and middleware software to handle all the issues related to developing parallel and distributed applications, scheduling tasks onto the best available resources, and managing the Quality of Service (QoS) requirements.

VDCE is a three-tiered software architecture that consists of an Application Editor to assist in application development and specification, an Application Scheduler to perform transparent application scheduling and resource configuration, and a VDCE Runtime System to run and manage the application execution. The Application Editor is a web-based graphical user interface that helps users to develop parallel and distributed applications. In VDCE the application development process is based on a dataflow programming paradigm. The Application Editor generates its output in terms of an Application Flow Graph (AFG) in which the nodes represent task computations and links denote communication and/or synchronization among the nodes (tasks). The Application Editor provides menu-driven, functional building blocks of task libraries. A node of an AFG is a well-defined function or task selected from a given task library. VDCE provides a large set of task libraries grouped in terms of their functionality, such as matrix operations, Fourier analysis, C^3I (command, control, communication, and information) applications, etc.

VDCE provides a distributed runtime scheduler, the Application Scheduler, which provides efficient task-to-resource mapping of application flow graphs. The Application Scheduler uses performance prediction of individual tasks to achieve efficient resource allocations. The schedule decision is based on the task specifications (i.e., hardware/software requirements) in the application flow graph, locations and configurations of resources, and up-to-date resource loads. The VDCE Runtime System consists of two parts: the Control Virtual Machine (CVM), and the Data Virtual Machine (DVM). The CVM is responsible

for monitoring the VDCE resources, setting up the execution environment for a given application, monitoring the execution of the application tasks on the assigned computers, and maintaining the performance, fault tolerance, and quality of service (QoS) requirements. The DVM is responsible for providing low latency and high-speed communication and synchronization services for inter-task communications.

The main goal of the ADViCE project is to extend the current VDCE 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.

The rest of the report is organized as follows. Section 2 is a summary of the related work. In Section 3 we present the design and implementation issues of the VDCE software architecture. Section 4 presents experimental results and evaluation of the current VDCE prototype. Section 5 presents the architecture and experimental results with ADViCE. Section 6 presents Concluding remarks and future work.

2 Related Work

In this section we provide a review of related work on the software development process, followed by related work on metacomputing. 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.

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 visualization of results and program behavior. Generally, writing parallel/distributed programs overwhelms users due to the difficulty of explicitly expressing communication and synchronization among the computations [7]. 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) [11], Heterogeneous Network Computing Environment (HeNCE) [12], and Zoom [13]. 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 representation abstraction for describing heterogeneous applications. Zoom representation of an application can be translated into a HeNCE program for execution [12]. On the other hand, application development tools and environments are being modified to support web-based user interfaces, since the World Wide Web is becoming a low-cost, standard interface mechanism with which to access the computational resources that are

distributed all over the world [29].

After a parallel/distributed application is developed, the tasks of the application are assigned to the available 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 applications. One of the few research groups targeted on a general scheduling framework is the APPLoS [14] group. APPLoS proposes *application-level scheduling* in which everything about the system is evaluated in terms of its impact on the application. APPLoS develops a customized schedule for each application by including user-specific, application-specific, system-specific, and dynamic information in its scheduling decision. The Network Weather Service component provides dynamic information. The Heterogeneous Application Template provides specific information about the structure of the application. User-supplied information is entered into the system with a user-specification file. There are resource management systems to provide load sharing and resource allocation, one of which, developed at the University of Wisconsin, is the Condor [31] project, a distributed batch system for sharing the workload of compute-intensive jobs in a pool of UNIX workstations connected by a network.

The application execution and runtime phase executes the developed and configured application and produces the required output. This stage integrates the assigned resources that will be involved in execution and supports inter-module communications, which are based on either a message-passing tool such as PVM [23], P4 [25], MPI [24], and NCS [26] or on a distributed shared memory (DSM) model. During the execution of the application this stage accepts data from different computing elements and combines them for proper visualization. It intercepts the error messages generated and provides proper interpretation. Some of these message-passing tools may be used in a metacomputing environment, although they were initially developed for parallel and distributed applications. In the first I-WAY metacomputing testbed, Nexus and MPI communication libraries were used within the prototype implementations of Globus communications.

In addition, there are a few projects targeted toward providing a metacomputing environment on diverse resources. The earliest metacomputer, the NCSA Metacomputer [27], was an integration of several MPPs, mass storage units, visualization and I/O devices. Globus [21] and Legion [28] are among the most recent projects targeted toward solving metacomputing problems. A low-level *toolkit* in the Globus environment provides mechanisms such as communication, authentication, and network information. These mechanisms can be used to construct higher-level metacomputing services such as parallel programming tools, schedulers, etc. On the other hand, Legion is a distributed-object metacomputing environment that is targeted to support a wide set of tools, languages, and programming models. The major objectives of the Legion project are site autonomy, an easy-to-use seamless computational environment, high performance via parallelism, security for users and resource owners, management and exploitation of resource heterogeneity, multiple language support and interoperability and fault tolerance. Additionally, there are several web-based metacomputing projects [29], 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.

3 Overview of VDCE Software Architecture

The main design philosophy of VDCE is to provide a general software development environment to build and execute large-scale applications on a network of heterogeneous resources. VDCE is composed of geographically distributed computation sites (domains), as shown in Figure 1, each of which has one or more VDCE Servers. The words "site" and "domain" are used interchangeably in this paper. Each domain consists of several clusters, each of which includes heterogeneous resources in terms of type, speed, or the configuration. At each site the VDCE Server runs the server software, called *site manager*, which handles inter-site communications and bridges VDCE modules to the web-based site repository. The site manager is part of the Control Virtual Machine that was explained in Section 3.3.

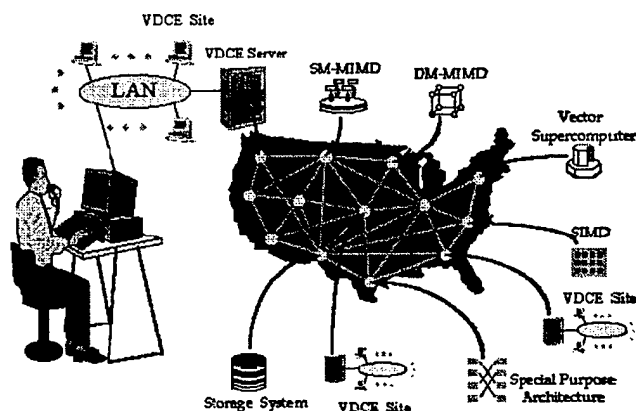


Figure 1: Virtual Distributed Computing Environment (VDCE)

The site repository consists of four different database tables. The *user-accounts table* is used to handle user authentication. In the user-accounts table, each VDCE user account is represented by a 5-tuple: user name, password, user ID, priority, and access domain type. The *resource-performance table* provides the resource (machine and network) performance attributes/parameters. These attributes are grouped into two parts: a) *static performance attributes* stored in the database once during the initial configuration of VDCE: host name, IP address, architecture type, operating system type, and total memory size, the computing weight (which will be described later in the Application Scheduler section) of each processor with respect to a base processor; and b) *dynamic performance attributes* that are updated periodically: CPU load, network latency, network bandwidth, and available memory size, number of processes, etc. The *task-performance table* provides performance characteristics for each task in the system and is used to

predict the performance of the task on a given resource. Each task implementation is specified by some parameters: computation size, communication size, and required memory size. For each task in VDCE, the task-performance table includes an entry for the measured execution time of benchmarking the task per machine type as well as the CPU loads when the measurements are taken. In order to find the location of a task's executable, VDCE stores location information of each task (i.e., the absolute path of the task executable) as well as other restrictions that might be related to the task execution for each host in the *task-constraints table*. Due to specific library requirements or other license restrictions, some task executables may reside only on a subset of the VDCE hosts.

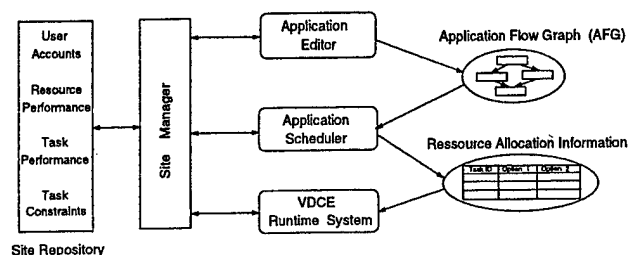


Figure 2: Interactions Among the VDCE Modules

The software development cycle for network applications can be viewed in terms of three phases: application development and specification phase, application scheduling and configuration phase, and execution and runtime phase. The functionality of these three phases is handled by the Application Editor, Application Scheduler, and VDCE Runtime System, respectively. Figure 2 shows the interaction of the VDCE modules within a site. In the following subsections we describe in detail the design and prototype implementation issues of the three main software modules.

3.1 Application Editor

The Application Editor is a web-based graphical user interface for developing parallel and distributed applications. The end-user establishes a URL connection to the VDCE Server software within the site (the *Site Manager*), which runs on a VDCE Server (see Figure 3). The Site Manager implementation is based on JAVA Web server technology, which uses servlets (i.e., server site JAVA applets) that relieve the startup overheads and run on any platform. After user authentication (as shown in Figure 3), the Application Editor, which was implemented in JAVA, will be loaded into the user's local web browser so that the user can develop his/her application.

The Application Editor provides menu-driven task libraries that are grouped in terms of their functionality, such as the matrix algebra library, C^3I (command and control applications) library, etc. A selected task is represented as a clickable and draggable graphical icon in the active editor area. Each such icon includes the task name and a set of markers for logical ports. Color coding used in this visual representation helps to distinguish input ports from output ports. Operationally, the Application Editor can be in *task mode*, *link*

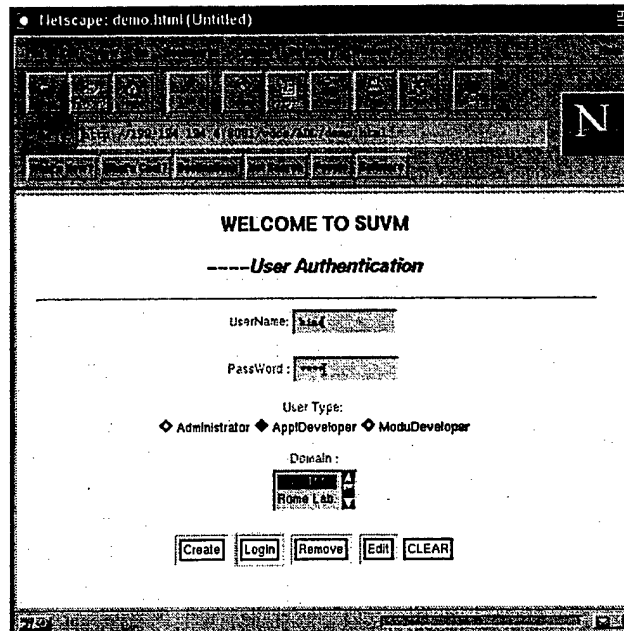


Figure 3: VDCE Authentication Window

mode, or *run mode*. In *task mode*, the user can select/add new tasks, and/or click/drag icons to position them conveniently in the active editor area. In *link mode*, the user can specify connections between tasks. In *run mode*, Editor submits the graph for execution and visualizes the performance and runtime characteristics of an ongoing computation.

The process of building a VDCE application with the Application Editor can be divided into two steps: building the application flow graph (AFG), and specifying the task properties of the application. The application flow graph is a directed acyclic graph, $G = (T, L)$, where T is the set of tasks in the application and L is a set of directed links among tasks. A directed link (i, j) between two tasks, T_i and T_j , of the application indicates that T_i must complete its execution before T_j begins to run. Figure 4 shows the application flow graph of a Linear Equation Solver (based on LU Decomposition) developed using the Application Editor. In this application, the problem is to find the solution vector x in an equation $Ax = b$, where A is a known $N \times N$ matrix and b is a known vector. With LU Decomposition, any matrix can be decomposed into the product of a lower triangular matrix L and upper triangular matrix U . Once LU Decomposition is solved, the solution vector, x , is derived with $x = U^{-1}(L^{-1}b)$. To construct the flow graph of this application, the user creates nodes by selecting LU_Decomposition, Matrix_Inverse(2), and Matrix_Multiply(2) tasks from the Matrix_Operations menu.

After the application flow graph is generated, the next step in the application development process is to specify the properties of each task. A double click on any task icon generates a popup panel that allows the user to specify optional preferences such as com-

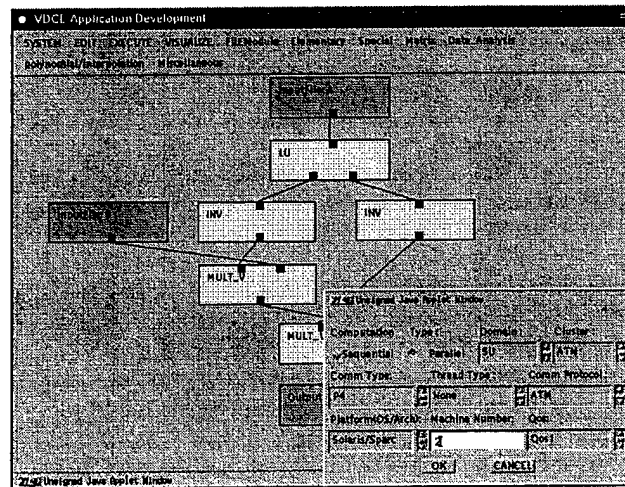


Figure 4: Building the Linear Equation Solver Application with the Application Editor

putational mode (sequential or parallel); domain type (Syracuse University or Rome Lab); cluster type (HPDC cluster, CAT cluster, TOP cluster; Rome Lab Cluster); communication type (P4, socket, MPI, DSM, NCS, PVM); thread type (none, pthread, qthread, cthread), communication protocol type (TCP/IP, ATM); machine type (SUN SPARC, RS6000, Pentium PC, HP) and the number of processors to be used in a parallel implementation of a given task (see the right part of Figure 4). In this figure, for the MULT task of the Linear Equation Solver the user has selected the parallel execution mode using two nodes of Sun SPARC machines interconnected by an ATM network. When the task properties are specified the user may either submit the application for execution in the VDCE or store the application flow graph for future use.

3.2 Application Scheduler

The main function of the Application Scheduler module in VDCE is to interpret the application flow graph and to assign the current best available resources for running application tasks in order to minimize the total execution time in a transparent manner. This module is based on application-based scheduling framework [14, 15] that is currently being implemented. VDCE provides distributed scheduling in a wide-area system in which each site consists of its own Application Scheduler running on the VDCE server. The Application Scheduler has two scheduling algorithms explained at the following pages: *site scheduler algorithm* and *host selection algorithm*. The schedule of an AFG is determined by the VDCE server at the local site, which runs the site scheduler algorithm, and a set of selected remote sites that execute the Host Selection Algorithm. Table 1 gives the meanings of the symbols used in the algorithms.

The *site scheduler algorithm* and *host selection algorithm* are based on the *list scheduling* [16, 17, 18] heuristic. In list scheduling each node (task) of the graph is assigned a priority and stored in an ordered list. In this paper node and task terms are used inter-

Table 1
Symbols and their meanings

Symbol	Meaning
AFG	Application Flow Graph.
$Site_List$	The list of sites that will be part of the scheduling process.
S_{local}	The site that has received the application execution request.
S_{remote}	The set of selected k neighbor sites of S_{local} .
$BW(S_i, S_j)$	The network bandwidth between sites S_i and S_j .
$LT(S_i, S_j)$	The network latency between sites S_i and S_j .
$Pred_Time(task_i, S_j)$	The <i>best</i> predicted execution time of $task_i$ at S_j .
$pred_time(task_i, P_j)$	The predicted execution time of $task_i$ on P_j .
$EST(task_i, S_j)$	The earliest start time of $task_i$ at site S_j .
$EFT(task_i, S_j)$	The earliest finish time of $task_i$ at site S_j .
$Predecessor(task_i)$	The set of nodes that are immediate predecessor of $task_i$.
$Exec_time(task_i, P_{test})$	The measured execution time of $task_i$ on P_{test} for the trial run.
$C_Load(P_j)$	The recent CPU load of P_j .
$M_Load(P_{test})$	The CPU load of P_{test} at the time of the trial run.
$Weight(P_j)$	The computing weight of P_j with respect to a base processor.

changeably. Whenever a processor is available for execution the highest priority task in the list is assigned to this processor. This process is repeated until all nodes of the graph are covered. The difference among the list scheduling heuristics is the way in which they assign priorities to nodes. The different priority assignment methods lead to different selection orders that result in different schedules.

We use the level of each node to determine its priority [17]. The *level* of a node is defined by the length of the longest path from the node to a terminal (or exit) node. The *length* of a path in the task graph is measured by the summation of all node weights and edge weights along the path. The node weight is the predicted execution time of the task, and edge weight is the predicted intertask communication time. Some of the previous works do not consider the edge weight when calculating the level of a node. For a node weight, we use the execution time of the task (node) on a predefined base-processor within the site. The weight of an edge between task i and task j is measured by dividing the data size to be sent from task i to task j , $D(i, j)$, to a base communication-link bandwidth, BW_{base} . We assume that each AFG has only one root node and one exit node.

Site Scheduler Algorithm

In this algorithm, the next step after initializing the *Task_List* with level values of AFG nodes is to select a set of remote sites that will be part of the scheduling process and that may possibly be part of the execution process. If the *update_request* flag is true, it indicates that one or more sites in the S_{remote} have high network traffic (or down). In this case, the remote sites are selected according to the network bandwidth between the remote site and the local site (shown in steps 4–8). Otherwise, the previously stored set is used. Then, AFG and *Task_List* are multicast to the involved sites for bidding, after

which the *Host_Selection_Algorithm* is executed at each site (step 12).

The Site Scheduler Algorithm receives the bidding from each site for each task in AFG (step 12), i.e., the best available processor, and the predicted execution time on the best available processor. Step 14 assigns the root task to the site that minimizes the predicted execution time. Step 19 calculates the earliest start time (EST) of the current task ($task_i$) at each site (S_j). To obtain the EST value of $task_i$, the summation of the earliest finish time (EFT) and the communication cost is calculated for each immediate predecessor task of $task_i$ in the graph. The EFT of a task at a site is calculated by the summation of its EST value and the predicted execution time of the task at the current site (step 20). As shown in step 22, the best site of a node is the one that minimizes the EFT value. The best available site for the current task is determined at each iteration of the while-loop from step 16 to step 25. For an application flow graph $AFG(v, e)$ with v nodes and e edges the while-loop takes $O(v)$ to compute the EST value of a node on a site (steps 15 and 16). We assume AFG to be a dense graph in which the number of edges are proportional to $O(v^2)$. Since there are v nodes in AFG and k sites involved in the scheduling process, the while-loop takes $O(kv^2)$ time; hence the time complexity of the site scheduler algorithm is $O(kv^2)$, since the while-loop is the dominant part. The value of k will be much smaller than v ; thus the worst case complexity of the algorithm is $O(v^2)$.

Site_Scheduler_Algorithm(AFG)

- Step 1 Compute the *level* for all nodes in AFG.
- Step 2 Initialize *Task_List* according to a non-increasing order of node *level*.
- Step 3 Read S_{remote} list and the *update_request* flag from resource performance table.
- Step 4 If *update_request* flag is true then
- Step 5 Select k nearest neighbor sites of S_{local} that maximize the network bandwidth and store them in a set, S_{remote} .
- Step 6 $update_request \leftarrow false$.
- Step 7 Update S_{remote} , and *update_request* in the resource performance table.
- Step 8 endif
- Step 9 $Site_List \leftarrow S_{local} \cup S_{remote}$
- Step 10 For each site $S_j \in Site_List$ do
- Step 11 Send AFG and *Task_List* for bidding.
- Step 12 $\{Pred_Time(task_i, S_j), Best_Resource(task_i, S_j)\} \leftarrow Host_Selection_Algorithm(Task_List) \quad \forall task_i \in Task_List$.
- Step 13 endfor
- Step 14 $Resource_Alloc_Table(task_1) \leftarrow S_m$, such that:
 $Pred_Time(task_1, S_m) \leftarrow \min\{Pred_Time(task_1, S_i)\}, \forall S_i \in Site_List$.
- Step 15 Remove $task_1$ from the *Task_List*.
- Step 16 while *Task_List* is not empty do
- Step 17 $task_i \leftarrow$ the first task in *Task_List*.

Step 18 For each site, S_j , in the *Site_List* do
 Step 19 $EST(task_i, S_j) \leftarrow \max \{ EFT(task_k, S_m) + (LT(S_m, S_j) + \frac{D(k,i)}{BW(S_m, S_j)}) \}$
 $\forall task_k \in Predecessor(task_i)$, such that:
 $S_m \leftarrow Resource_Alloc_Table(task_k)$.
 Step 20 $EFT(task_i, S_j) \leftarrow EST(task_i, S_j) + Pred_Time(task_i, S_j)$.
 Step 21 endfor
 Step 22 Select *Best_Site*, such that:
 $EFT(task_i, Best_Site) \leftarrow \min \{ EFT(task_i, S_j) \}, \forall S_j \in Site_List$.
 Step 23 $Resource_Alloc_Table(task_i) \leftarrow Best_Resource(task_i, Best_Site)$
 Step 24 Remove $task_i$ from the *Task_List*.
 Step 25 endwhile
 Step 26 Multicast the *Resource_Alloc_Table* to the relevant sites.

Host Selection Algorithm

The Host Selection Algorithm determines the task assignments of AFG tasks on the available processors within each site. The calculation of the EST is similar to the previous algorithm. In this algorithm, base communication-link bandwidth, BW_{base} , is considered for all connections within a site (step 4). Additionally, the latency within a site is negligible if it is compared with the latency between the different sites. The communication cost between a task and its immediate predecessor is zero if they are scheduled to the same processor. The core of the Host Selection Algorithm is the performance prediction phase. The execution time prediction of a task on a given resource is based on the current load of the processor, load of the test processor at the time of trial run, measured execution time for the trial run, and computing weights (step 5).

The measured execution time and the load value for the trial runs are retrieved from the task-performance table, as explained in the Site Repository section of this paper. $Weight(P_j)$ is the computing weight [19, 20] of processor P_j with respect to the base-processor at the site. To calculate the weight of each processor, trial runs of a set of task implementations are executed on each processor. The ratio of average execution time of the trial runs on a processor P_i to the average execution time on the base-processor gives the computing power weight of P_i . In step 6, the EFT value is the summation of the EST and the predicted execution time. For each task, the processor that minimizes the EFT value is selected as the best resource in this site. An iteration of the while loop takes $O(pv)$ times, where v is the number of nodes in AFG and p is the number of processor in the *Processor_List*. Thus the time complexity of the Site Scheduler Algorithm is $O(pv^2)$.

Host_Selection_Algorithm(Task_List)

Step 1 while *Task_List* is not empty do
 Step 2 $task_i \leftarrow$ the first task in *Task_List*.
 Step 3 For each available processor, P_j , in the *Processor_List* do
 Step 4 $EST(task_i, P_j) \leftarrow \max \{ EFT(task_k, P_m) + Comm_Cost(task_k, task_i) \}$

$\forall task_k \in Predecessor(task_i)$ such that:
 $P_m \leftarrow Best_Resource(task_k)$ and
 $Comm_Cost(task_k, task_i) \leftarrow \begin{cases} \frac{D(k,i)}{BW_{base}} & P_m \neq P_j \\ 0 & otherwise \end{cases}$

Step 5 $Pred_Time(task_i, P_j) \leftarrow \frac{C_Load(P_j)}{M_Load(P_{test})} \times$
 $Exec_Time(task_i, P_{test}) \times \frac{Weight(P_{test})}{Weight(P_j)}$

Step 6 $EFT(task_i, P_j) \leftarrow EST(task_i, P_j) + Pred_Time(task_i, P_j)$

Step 7 **endfor**

Step 8 $Best_Resource(task_i) \leftarrow P_k$, such that:
 $EFT(task_i, P_k) \leftarrow \min\{EFT(task_i, P_j)\}, \forall P_j \in Processor_List.$

Step 9 **endwhile**

Step 10 Return $Pred_Time(task_i, Best_Resource)$ and $Best_Resource(task_i)$ to S_{local}
for each task.

3.3 VDCE Runtime System

The VDCE Runtime System sets up the execution environment for a given application and manages the execution to meet the hardware/software requirements of the application. The VDCE Runtime System separates control and data functions by allocating them to the Control Virtual Machine (CVM) and Data Virtual Machine (DVM), respectively. CVM measures the loads on the resources (hosts and networks) periodically and monitors the resources for possible failures. CVM daemons control the execution of the application tasks on the assigned resources based on the performance and quality of service requirements. Application visualization (real-time or post-mortem) services are provided by CVM. DVM provides an execution environment for a given VDCE application by binding tasks so that they can interact and communicate efficiently. DVM supports socket-based point-to-point connections for inter-task communications.

Control Virtual Machine (CVM)

The functionality of CVM is provided by the following four processes: Site.CVM, Local.CVM, Monitor, and Cluster Manager (see Figure 5). Each VDCE machine runs a Local.CVM process and a Monitor daemon. Additionally, one of the machines within each cluster executes the Cluster Manager process. Each site (domain) has a Site.CVM process located at the VDCE Server machine. The main functions of the stated CVM processes are given below:

- *Retrieving Resource Performance Parameters.* VDCE resources are periodically monitored to collect up-to-date values of processor and network parameters that were given in the Site Repository subsection of this paper. The *Monitor* daemon of each machine periodically measures the up-to-date parameters every 30 seconds and updates its fields at the Cluster leader machine shown in Figure 5. The Cluster Manager daemon gathers the parameters of machines within the cluster in a table and periodically forwards the table to the Site.CVM every 60 seconds. In the future implementation the Cluster Manager will be modified to send only the workloads

of the resources that have changed considerably from the previous measurement. The workload of a resource is significantly changed if the up-to-date measurement is higher or lower than the summation of the previous measurement and the width of the confidence interval [22].

- *Updating the Site Repository.* The Site.CVM periodically updates the resource-performance table at the site repository with the parameters that are collected from Cluster Managers. The execution time and load measurement of benchmarking runs of tasks are stored at the task-performance table.
- *Monitoring the VDCE Resources.* When a Monitor daemon of a processor stores its parameters, it reads the random number that was generated by the Cluster Manager and updates its `alive.check` field with this value. Every 60 seconds the Cluster Manager compares its `alive.check` field with each cluster machine's `alive.check` field. The machines with a different value are marked as *down*; others are marked *alive*. After the comparison, the Cluster Manager assigns a new random number for its `alive.check` field. The monitor information is forwarded to the Site.CVM with the resource parameters to be stored at the site repository. The machines that are marked as *down* at the resource-performance table are not selected by the Application Scheduler.
- *Sending the Related Portion of the Resource Allocation Table.* After the resource allocation table is generated by the Application Scheduler, the Site.CVM multicasts it to the Cluster Managers that will be involved in the execution. If a machine in a cluster is assigned for a task execution, the Cluster Manager sends an execution request message and related parts of the resource allocation table to the Local.CVM of the machine.
- *Inter-site Coordination.* As explained in Section 3.2, the Application Scheduler at the local site selects a subset of remote sites and multicasts the application flow graph to these sites. The remote sites run the *Host Selection Algorithm* locally and transfer the mapping decisions to the sender site. The inter-site coordination and message transfer are handled by Site.CVMs.
- *Initialize the Application Execution Environment.* After the Local.CVM receives an execution request message from the Cluster Manager, it activates the DVM. The DVMs on the assigned machines set up the application execution environment by starting the task executions and creating point-to-point communication channels for inter-task data transfer. Figure 6 shows the part of the execution environment of the Linear Equation Solver application discussed in Section 3.1. Machine 1 will execute the LU-Decomposition task, which is followed by the execution of Matrix-Inversion tasks on Machine 2 and Machine 3. When all the required acknowledgments are received, an execution startup signal is sent to start the application execution.
- *Managing the application execution.* The Local.CVM monitors the application execution on the assigned machines and maintains the performance, fault tolerance, and QoS requirements of the application tasks. If the current load on any of these machines is more than a predefined threshold value, the Local.CVM terminates

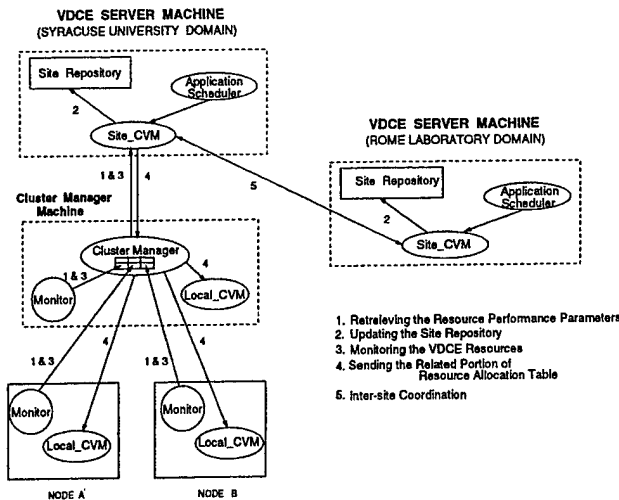


Figure 5: Interactions Among the Control Virtual Machine Components

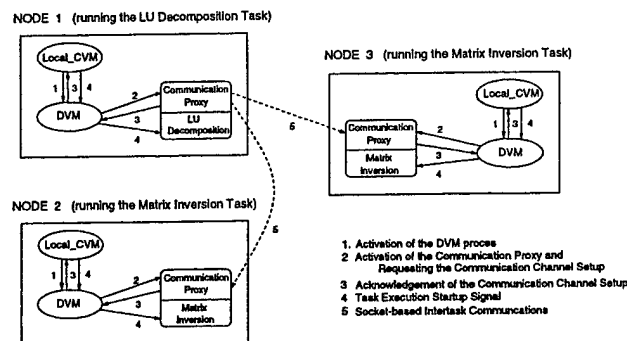


Figure 6: Setting Up the Application Execution Environment

the task execution on the machine and sends a task rescheduling request to the Site_CVM through the Cluster Manager.

Data Virtual Machine (DVM)

DVM is a socket-based, point-to-point communication system for inter-task communications. Therefore, any machine that supports socket programming can be part of VDCE. As shown in Figure 6, the DVM activates the communication proxy and sends the resource allocation information, including the socket number, IP address for target machine, etc., that will be used for the communication channel setup. After the setup is completed successfully, the communication proxy sends an acknowledgment to the Local_CVM. The execution startup signal is sent to start the task executions.

On the other hand, for a thread-based programming environment, the Data Manager

consists of three threads that are initiated by the communication proxy: send thread, receive thread, and compute thread. After the communication channel is established, the send and receive threads are activated for data transfer and the compute thread performs the task execution. The control transfer between the Local.CVM and the DVM (or any other control transfer on the same machine) are based on an inter-process communication mechanism (i.e., pipes or shared-memory paradigm). The data transfer among the communication proxies (or between send and receive threads for multithreaded systems) uses a socket-based, message-passing mechanism.

Since user tasks can be programmed in various message-passing tools, the VDCE Runtime System supports multiple message-passing libraries such as P4, PVM, MPI, NCS. Additionally, the VDCE Runtime System provides data conversions that might be needed when an application execution environment includes heterogeneous machines. The VDCE Runtime System provides several user-requested services such as I/O service, console service, and visualization service. A user can request these services while developing his/her application with the Application Editor. I/O Service provides either file I/O or URL I/O for the inputs of the application tasks. The user can suspend and restart the application execution with the console service. The VDCE visualization service provides both real-time and post-mortem visualizations. There are three types of visualizations provided in VDCE:

- **Application Performance Visualization:** The execution time of tasks in an application is visualized.
- **Workload Visualization:** Up-to-date workload information on VDCE resources is visualized.
- **Comparative Visualization:** VDCE makes it possible for an end user to experiment and evaluate his/her application for different combinations of hardware and software medium by providing the comparative performance visualization.

4 VDCE Testbed: Experimental Results and Discussion

The current VDCE prototype consists of two sites, one at Syracuse University and the other at Rome Laboratory, that are connected by the NYNET ATM Wide Area Network, as shown in Figure 7. Each site or domain has a VDCE server, a Site Repository and several computing clusters. 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 the Ethernet. The Rome Lab site consists of three clusters that include SUN, Digital, and HP workstations.

In this section we discuss and evaluate the performance of the current VDCE prototype in implementing two important tasks: 1) The use of VDCE as an evaluation tool

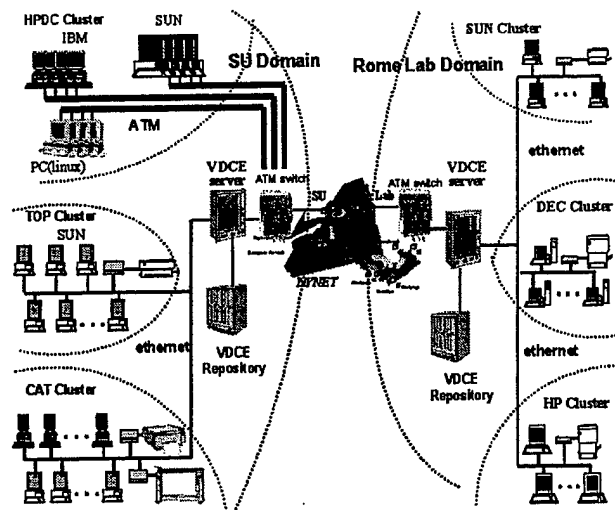


Figure 7: The configuration of the VDCE Testbed

for the parallel implementations of the VDCE library tasks using different numbers of workstations, and different networks to connect them (e.g. ATM or Ethernet); and 2) The use of VDCE as a problem-solving environment for large-scale VDCE applications.

4.1 Experiment 1: Using VDCE as a Parallel Evaluation Tool

In this experiment we used the matrix multiplication (MULT) task as a running example to show the use of the VDCE for experimentation and to evaluate 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 VDCE. We benchmarked the sequential and parallel algorithms of Matrix multiplication (MULT) based on various machine and network configurations and problem sizes. The parallel implementation of MULT ($A \times B = C$) 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 result matrix C and sends it back to the host process.

The VDCE provides a web-based, user-friendly interface that allows a novice programmer to experiment with and evaluate different parallel configurations of each VDCE task in 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 using parallel processing and message passing and visualization tools. With VDCE, once a task library is registered to the VDCE site repository, any VDCE user can use that task or any existing VDCE task by just clicking on the task name in the Application Editor. Once the task is selected, the user can click on one button to determine the problem size, the number of computers to be involved in the computation, and the

network to be used to connect them. Selecting the VDCE task and specifying how it will be implemented can be done in a few minutes. Once that is done, the task configuration can be run and its execution time visualized immediately without any effort other than clicking on the execute and visualize buttons.

Figure 8 shows the execution times of the VDCE-based, matrix multiplication algorithm for 512×512 and 1024×1024 . The result for p4-based implementation of the same multiplication algorithm is given in Figure 9. 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 MULT 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 processing algorithms when there is a change in problem size, number of nodes, or network type. As an example, for the p4-based, matrix multiplication algorithm, we can determine from Figure 9 that eight nodes provide the best performance among the test cases.

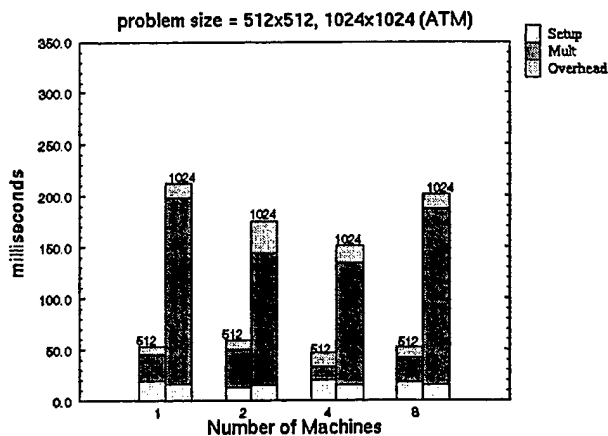


Figure 8: Execution Time of Matrix Multiplication Task Using VDCE

Table 2 compares the times required to develop, compile, execute, and visualize a Matrix Multiplication task using p4 and VDCE 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 about it and to develop an application. For VDCE, even if the user does not have any knowledge about 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 application design window of the Application Editor. Additionally, he/she can easily define the I/O for a task using the Application Editor. The total time for developing a VDCE MULT application is 2.10 minutes.

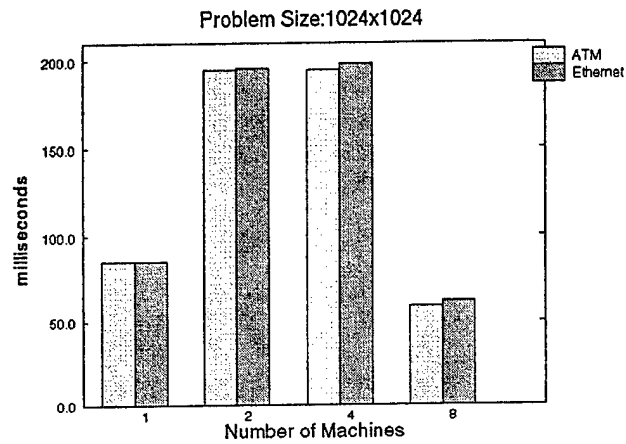


Figure 9: Execution Time of Matrix Multiplication Task Using p4

Table 2

The performance comparison of matrix multiplication task for each software phase

Phase	p4	VDCE
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.

There is no compilation time in VDCE after the VDCE MULT application is designed. The location of the executable for MULT 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 program takes 7.01 seconds for compilation. The runtime setup time in VDCE is for the CVM to transfer the activation and resource allocation information to DVM and to wait for the acknowledgment, which takes 15 milliseconds for the MULT task on the selected resource. For a p4 application, the user creates a configuration file, i.e., procgroup file, and manually links it to the p4 application which takes 980 milliseconds. VDCE runs the application automatically with the "Execute Application" button and generates the results in the selected output file. The execution time of MULT task is 136 milliseconds when it is executed on four nodes over the ATM. The execution time is 194 milliseconds using a p4 program with the same configuration.

VDCE provides dynamic and post-mortem visualization of the application. A VDCE user monitors the load of all machines dynamically in the domain and he/she can consider the load information to select an appropriate machine and/or a cluster. In addition, the execution time of each module within an application is visualized in VDCE. It takes 95 milliseconds to invoke the VDCE visualization window for the MULT 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.

4.2 Experiment 2: Using VDCE as a Problem Solving Environment

In this experiment we demonstrated how the VDCE 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 experts on 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 scratch. The VDCE provides a web-based interface to enable users to develop, configure, execute, and visualize such a distributed application in a few minutes. However, to perform the same tasks in a non-VDCE case, the user or team developers need to develop techniques to interact and communicate the modules running on different computers, and they need to develop or integrate techniques to run and manage the execution of the distributed application, as well as collect and visualize the required performance results.

To solve these difficulties, VDCE provides an integrated problem solving environment to enable novice users to develop large-scale, complex, distributed applications using VDCE tasks. The Linear Equation Solver (LES) application has been selected as a running example. Figure 4 shows the AFG of Linear Equation Solver, which consists of an LU Decomposition (LU) task, two Matrix Inversion (INV) tasks and Matrix Multiplication (MULT) tasks. The problem size for this experiment is 1024×1024 using four

Table 3
Performance comparison of linear equation solver application for each software phase ¹

Phase	p4			VDCE		
	LU	INV	MULT	LU	INV	MULT
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.	

nodes, which are SUN SPARCs and IBM RS6000 machines that are connected by an ATM network.

Table 3 compares the timing of several software phases for a Linear Equation Solver application using p4 and VDCE. When a user has enough knowledge about parallel programming and the p4, he/she will spend 838 minutes for an LU task, 1314 minutes for an INV task, and 862 minutes for MULT task. The total time to develop the application for a non-VDCE version is approximately 3014 minutes, (i.e., around 50 hours). Using VDCE, a novice user spends around six minutes to develop such an application. There is no compile time for VDCE, but a p4 application needs 21 seconds for compilation. The VDCE setup time for a Linear Equation Solver application is 43 milliseconds. The p4 user should create all proggroup files and launch them in order, which takes around eight seconds.

Since the VDCE is based on the data flow model and executes tasks automatically, there may be overlap among task executions that causes the total execution time of the VDCE application, including the setup time, to be less than the summation of all individual task execution times. In our experiment with the Linear Equation Solver application, the total execution time of p4 parallel execution using four nodes is 1691 milliseconds. A VDCE-based execution with the same configuration takes 1451 milliseconds, which outperforms the p4 by 16%.

5 ADAPTIVE DISTRIBUTED VIRTUAL COMPUTING ENVIRONEMNT (ADViCE)

5.1 Introduction

With the proliferation of wireless networks, metacomputing services can be extended to include mobile users and resources. A mobile metacomputing environment allows users

¹The last two rows of the table are for the total time of the application.

²It is the total setup time for a VDCE-based linear equation solver application.

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 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.

5.2 *Related Work*

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

5.3 *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 [7]. 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) [11], Heterogeneous Network Computing Environment (HeNCE) [12], and Zoom [13]. 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 [12]. Currently, there is an increased interest in developing web-based application development tools and environments because of the explosive use of internet applications [29]. 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 [14]. 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 [31]. 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 [23], P4 [25], MPI [24], and NCS [26] or based on distributed shared memory (DSM) [3, 4, 5, 6]. In addition, there are a few projects targeted toward providing a metacomputing environment on diverse resources. The earliest metacomputer, the NCSA Metacomputer [27], was an integration of several MPPs, mass storage units, visualization and I/O devices. Globus [21], Legion [28], and VDCE [8, 10] targeted toward the development of metacomputing environments. Additionally, there are several web-based metacomputing projects [29], 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.

5.4 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 constraints of mobile computing are [1, 2]: 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 [1]: 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.

5.5 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 10. 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 10). 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.

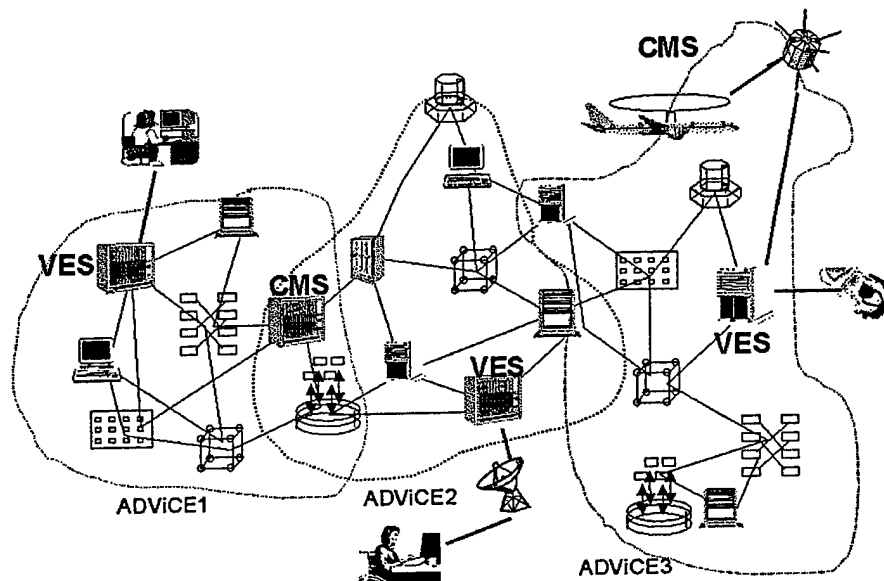


Figure 10: Adaptive Changes in the ADViCE environment.

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.

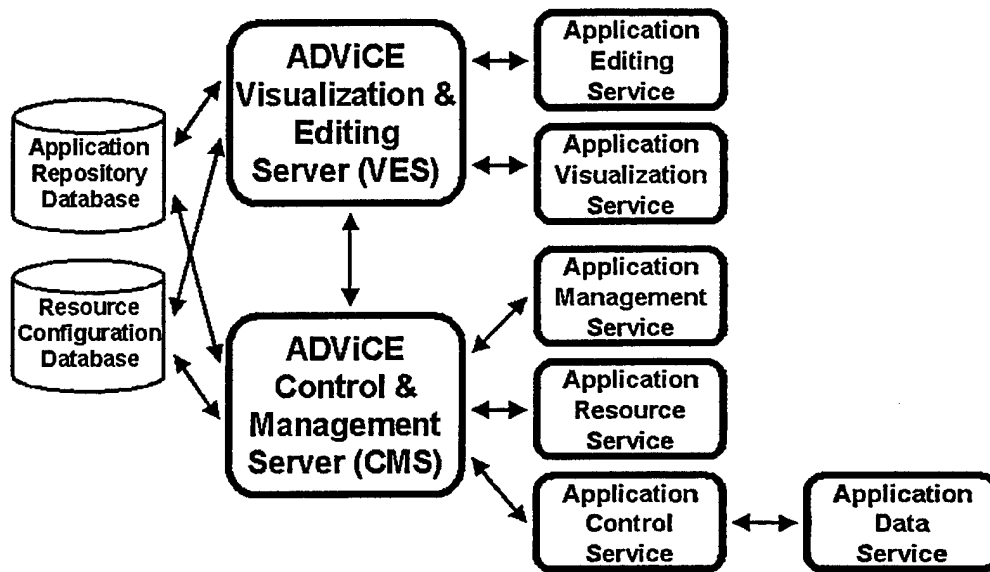


Figure 11: The Main Components of the ADViCE Architecture.

5.5.1 Visualization and Editing Server (VES)

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

description 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 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 12).

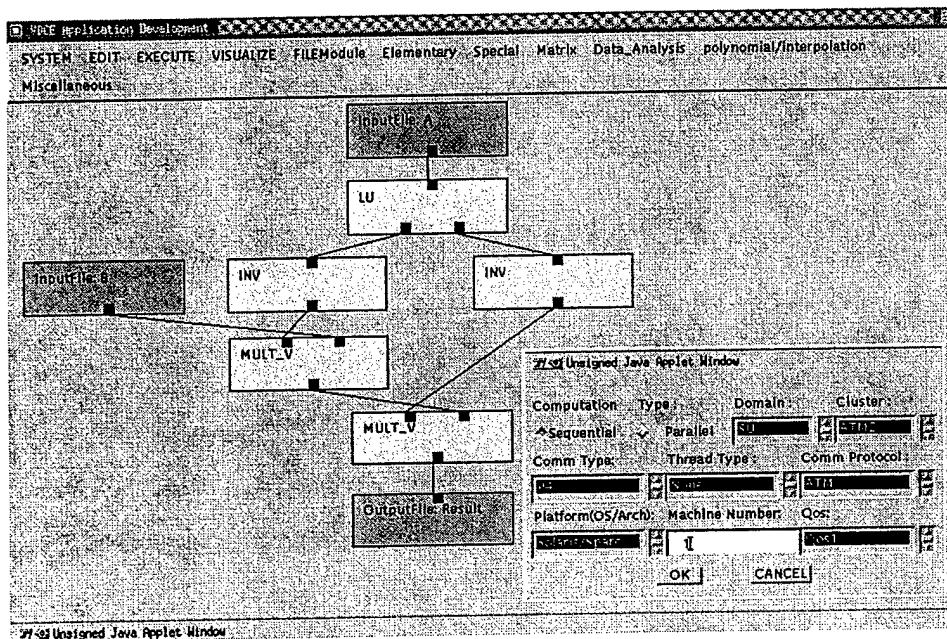


Figure 12: An Application Flow Graph Example.

description Application Visualization Service (AVS)

This service enables the user to visualize the application execution time and system runtime parameters. For example, Figure 13 shows the execution time

for each task in the application shown in Figure 12. In addition, the AVS shows the total execution time of the application and the setup time of the application execution environment.

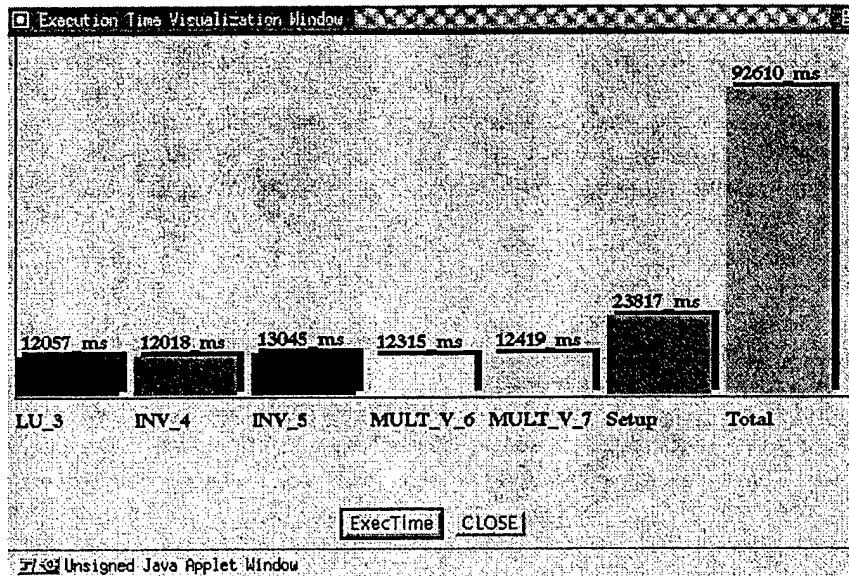


Figure 13: The Performance of each Application Task.

5.5.2 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.

description 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.

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

description 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.

description 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.

5.6 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 [32]. 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 14, 15, 16, and 17 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 14), the ACS monitors all the ADViCE services (Steps 2 through 26 in the ADViCE Adaptation Algorithm of Figure 14) 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 14). 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 15). Once that delay is detected, the Verification and Analysis procedure for that service is invoked (Step 6 in the ADViCE Adaptation Algorithm of Figure 14). In a similar manner, we devise detection algorithms for each service offered by the ADViCE servers (VES and CMS) as shown in Figure 16.

The Verification and Analysis procedures shown in Figure 16 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 15). This event could be caused by the machine being down or task failure (Step 11 through 18 in Verification_Analysis_ADS of Figure 16). 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 14 (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 17).

```

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

```

Figure 14: 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       :
8   else
9       EventType = Normal
10  endif
11  return(EventType)
12  end of Change_Detection_AES
13  :
14  :
15  procedure Change_Detection_ADS()
16  if inter task communication delay  $> D_{max}$ 
17      EventType = inter task communication delay
18  else if broken pipe detected
19      EventType = link failure
20      :
21  else
22      EventType = Normal
23  endif
24  return(EventType)
25  end of Change_Detection_ADS

```

Figure 15: ADViCE Change Detection Procedures

```

1  procedure Verification_Analysis_AES(EventType, AE(Appi))
2      case EventType = unacceptable delay to VES
3          verify delay to VES
4          if measure the delay to VES >  $D_{max}$ 
5              check if the delay is caused by the location of VES
6              EventCause = location change of VES
7              check if the delay is caused by the location of the user
8              EventCause = user's location change
9              check if the delay is caused by heavy load VES
10             EventCause = heavily loaded VES
11         endif
12     case EventType = VES down
13         verify VES down by AMS MIB
14         if true
15             check if VES down is caused by the VES machine failure
16             EventCause = VES machine down
17         endif
18     case EventType = database down
19         verify database down by AMS MIB
20         if true
21             check if database down is caused by database machine down
22             EventCause = Application Repository database machine down
23             check if database down is caused by database server down
24             EventCause = Application Repository database server down
25         endif
26     :
27     return(EventCause)
28 end of Verification_Analysis_AES
29 :
30
31 procedure Verification_Analysis_ADS(EventType, AE(Appi))
32     case EventType = inter task communication delay
33         verify the communication delay
34         if measure inter task delay >  $D_{max}$ 
35             check if the delay is caused by heavy network traffic
36             EventCause = heavy traffic
37             check if the delay is caused by heavy load
38             EventCause = heavy CPU load
39         endif
40     case EventType = link failure
41         verify link failure by AMS MIB
42         if true
43             check if link failure is caused by machine down
44             EventCause = machine down
45             check if link failure is caused by task down
46             EventCause = task down
47         endif
48     :
49     return(EventCause)
50 end of Verification_Analysis_ADS

```

Figure 16: Verification and Analysis Procedures

```

1  procedure Adaptation_Plan_AES(EventCause, AE(Appi))
2      case EventCause = location change of VES or
3          EventCause = user's location change or
4          EventCause = heavily loaded VES or
5          EventCause = VES machine down
6          | access the default VES
7          | locate a new VES
8          | transfer the information from current VES to a new VES
9      case EventCause = Application Repository database machine down
10         | choose alternative Application Repository database
11     case EventCause = Application Repository database server down
12         | start the database
13         :
14     end of Adaptation_Plan_AES
15     :
16
17     procedure Adaptation_Plan_ADS(EventCause, AE(Appi))
18         case EventCause = heavy traffic or
19             EventCause = heavy load or
20             EventCause = machine down
21             | invoke ARS to assign a new machine
22             | if migration required
23                 task migration
24             | endif
25             | if partial recovery is possible
26                 resume from the stopped task
27             | else
28                 resume from the task check pointed state
29             | endif
30         case EventCause = task down
31             | if partial recovery is possible
32                 resume from the task check pointed state
33             | else
34                 start the application from the beginning
35             | endif
36         :
37     end of Adaptation_Plan_ADS

```

Figure 17: Adaptation Plan Procedures

6 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 18. 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](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.

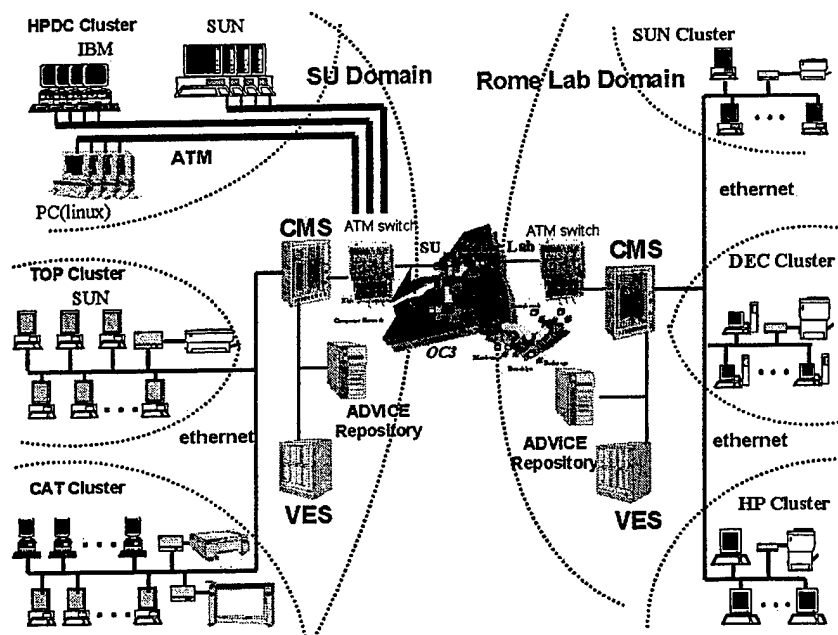


Figure 18: The configuration of the current ADViCE Testbed.

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.

6.1 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 19 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 20. 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 20 that eight nodes provide the best performance among the test cases.

Table 4 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

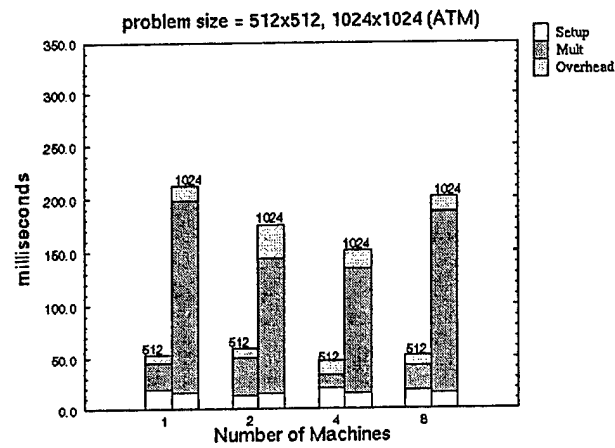


Figure 19: The Performance of the ADViCE Implementation of the Matrix-Vector Multiplication.

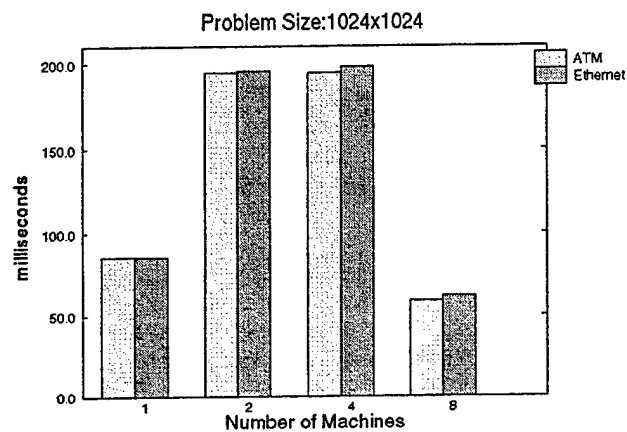


Figure 20: The Performance of the P4 Implementation of the Matrix-Vector Multiplication.

Table 4

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.

6.2 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 5

Performance comparison of the linear equation solver application for each software development phase ³

Phase	p4			ADViCE		
	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 ⁴	
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 12 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 5 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 proggroup 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.

6.3 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

³The last two rows of the table are for the total time of the application.

⁴It is the total setup time for a ADViCE-based linear equation solver application.

fault tolerant distributed application that is shown in (Figure 21).

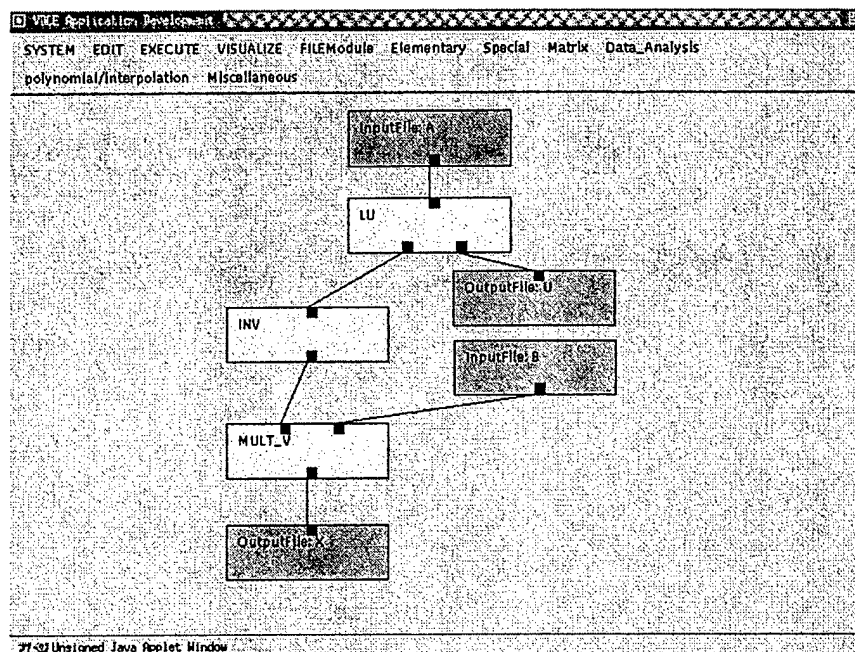


Figure 21: An Example of Fault Tolerant Distributed Application.

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 21. 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 22). 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 16). 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 22). 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

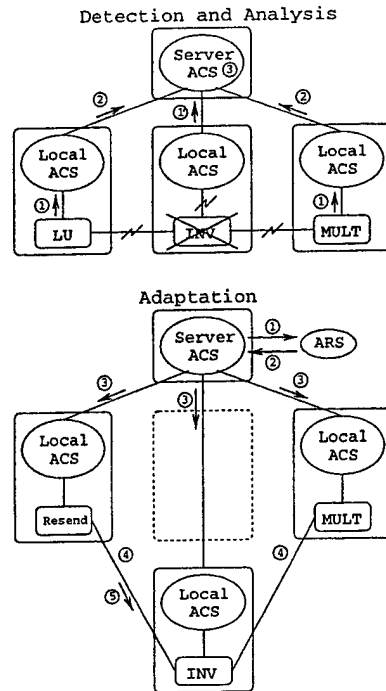


Figure 22: An Example of the ADViCE Adaptation Algorithm.

to perform the operations specified in the adaptation plan associated with the affected service. For the example shown in Figure 22, 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.

7 Conclusion

We have presented the design and evaluation of the Virtual Distributed Computing Environment (VDCE) and the Adaptive Distributed Virtual Computing Environment that have been developed at Syracuse University and the University of Arizona.

The VDCE consists of three main modules: Application Editor, Application Scheduler, and VDCE Runtime System. The Application Editor provides users with all the software tools and library functions required to develop a VDCE application. The main function of the Application Scheduler is the initial task-to-resource mapping and any necessary dynamic rescheduling. The VDCE Runtime System is based on the Control Virtual Machine (CVM) and the Data Virtual Machine (DVM). CVM provides a seamless interconnection of the resources and monitors the resources. DVM enables a high-performance communication medium among the application tasks.

We have successfully implemented a proof-of-concept prototype that supports all major components of the VDCE architecture. We are currently working on extending the current prototype in several ways: a) develop and implement an application programming interface (API) that enables users to add VDCE library tasks; b) add more sites to increase the computing services offered by VDCE; and c) develop and integrate mobile computing technology into VDCE so that users can access VDCE resources using mobile hosts and mobile interconnection networks.

We have also extended the VDCE prototype to support mobile computing and communication resources by developing the ADViCE prototype. 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.

We are currently investigating efficient techniques to achieve proactive control and management of all services offered by ADViCE that will include transparent performance, fault-tolerance, and security services for ADViCE applications/users.

References

- [1] M. Satyanarayanan, Fundamental Challenges in Mobile Computing, *Fifteenth ACM Symposium on Principles of Distributed Computing*, Philadelphia, May, 1996.
- [2] George H. Forman, John Zahorjan, The Challenges of Mobile Computing, *IEEE Computer*, Vol. 27, pp. 33-47, April, 1994.
- [3] S. Ahuja, N. Carriero, and D. Gelernter, Linda and Friends, *IEEE Computer*, vol. 18, No. 8, pp. 26-34, August, 1986.
- [4] P. Keleher, S. Dwarkadas, A. Cox and W. Zwaenepoel, Treadmarks: Distributed shared memory on standard workstations and operating systems, *Proceedings of the 1994 Winter Usenix Conference*, pp. 115-131, January, 1994.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] 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.
- [9] H. Topcuoglu and S. Hariri, A global computing environment for networked resources, in

- Proceedings of International Conference on Parallel Processing*, 1997, pp. 493–496.
- [10] 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.
 - [11] P. Newton, J. C. Browne, The CODE 2.0 graphical parallel programming language, in *Proceedings of ACM International Conference on Supercomputing*, 1992.
 - [12] R. Wolski, C. Anglano, J. Schopf, F. Berman, Developing heterogeneous applications Using Zoom and HeNCE, in *Proceedings of the Forth Heterogeneous Computing Workshop*, 1995.
 - [13] 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.
 - [14] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, Application-level scheduling on distributed heterogeneous networks, in *Proceedings of Supercomputing 96*, 1996.
 - [15] J. Weissman, A. Grimshaw, A federated model for scheduling in wide-area-systems, in *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, 1996, pp. 542–550.
 - [16] T.L. Adam, K. Chandy, and J. Dickson, A comparison of list scheduling for parallel processing systems, *Communication of ACM*, 17 (1974) 685–690.
 - [17] H. El-Rewini, H. Ali, T. Lewis, Task scheduling in multiprocessing systems, *IEEE Computer*, 28(12) (1995) 27–37.
 - [18] Y. Kwok, I. Ahmad, Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors, *IEEE Transactions on Parallel and Distributed Systems*, 7 (1996) 506–521.
 - [19] Y. Yan and X. Zhang, An effective and practical performance prediction model for parallel computing on nondedicated heterogeneous NOW, *Journal of Parallel and Distributed Computing*, 38 (1996) 63–80.
 - [20] M. Zaki, W. Li and M. Cierniak, Performance impact of processor and memory heterogeneity in a network of machines, in *Proceedings of the Forth Heterogeneous Computing Workshop*, 1995.
 - [21] I. Foster and C. Kesselman, Globus: a metacomputing infrastructure toolkit, in *Proceedings of the Workshop on Environment and Tools for Parallel Scientific Computing*, 1996.
 - [22] H. Casanova and J. Dongarra, Netsolve: a network server for solving computational science problems, in *Proceedings of Supercomputing 96*, 1996.
 - [23] 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.
 - [24] Message Passing Interface Forum, MPI: A message-passing interface standard, version 1.0 May 1994.
 - [25] R. Butler and E. Lusk, User's guide to the p4 programming system, Mathematics and Computer Science Division, Argonne National Laboratory.
 - [26] 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.
 - [27] L. Smarr and C. Catlett, Metacomputing, *Communications of the ACM*, 35, 6, (June 1992) 44–52.
 - [28] 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.
 - [29] 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.

- [30] J. Gehring and A. Reinefeld, MARS - A framework for minimizing the job execution time in a metacomputing environment, *Future Generation Computing Systems*, (1996).
- [31] Mike Litzkow, Miron Livny Experience with the condor distributed batch system, in *IEEE Workshop on Experimental Distributed Systems*, 1990.
- [32] 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.

***MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)***

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.