The Software Architecture of a Virtual Distributed Computing Environment*

Haluk Topcuoglu Salim Hariri Wojtek Furmanski Jon Valente† Ilkyeun Ra
Dongmin Kim Yoonhee Kim Xue Bing Baoqing Ye
Department of Electrical Engineering and Computer Science
HPDC Laboratory
Syracuse University
Syracuse, NY 13244-4100.
{haluk, hariri}@cat.syr.edu

† Rome Laboratory, Rome NY

Abstract

The requirements of grand challenge problems and the deployment of gigabit networks makes the network computing framework an attractive and cost effective computing environment with which to interconnect geographically distributed processing and storage resources. Our project, Virtual Distributed Computing Environment (VDCE), provides a problem-solving environment for high-performance distributed computing over wide area networks. VDCE delivers well-defined library functions that relieve end-users of tedious task implementations and also support reusability. In this paper we present the conceptual design of VDCE software architecture, which is defined in 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 monitoring the VDCE resources.

1. Introduction

Grand challenge problems have computational and storage resource requirements that are beyond the capacities of a single computing environment. Additionally, emerging network 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). A high-speed network of geographically distributed heterogeneous resources represents a cost-effective, network-based computing environment for solving large-scale problems addressed by grand and national challenges. New software development models and problem solving environments are being developed to utilize efficiently the network computing environment.

The software development process of parallel and distributed applications can be broadly described in terms of three phases: a) application development and specification, b) application scheduling and resource configuration, and c) application execution and runtime. Most of the related work so far has focused only on one or two of these phases; only a very few projects have completely addressed all phases of software development.

The first phase, i.e, parallel and distributed application development and specification phase, overwhelms most users because of the difficulty of expressing communication and synchronization among computations [3]. Some text-based parallel programming environments support the data-parallel paradigm, which requires advanced compilation techniques and compilers. Most of the other environments require explicit insertion of communication and synchronization primitives within the programs, which makes programs difficult to understand. Over the last few years a number

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

of graph-based application development and representation tools have become available, including Code [1], HeNCE [2], and Zoom [2, 4]. A graph-based programming environment provides simple and easy-to-use mechanisms for expressing the interaction of multiple processes within a parallel/distributed program [3]. 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 [19] with which to access the computational resources that are distributed all over the world.

After a parallel/distributed application is developed, the tasks of the application are assigned at the second phase to the existing 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. There are also some research projects that target application-level resource allocation issues such as APPLeS [10] and MARS [9] projects. 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 [6], P4 [5], Express [8], MPI [7], and NCS [18] 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. The runtime system handles dynamic load-balancing, application-level and resource-level fault tolerance capabilities.

In this paper we present our approach for developing a software environment, which we refer to as a Virtual Distributed Computing Environment (VDCE). VDCE provides a problem-solving environment for high-performance distributed computing over high-speed wide area networks, i.e, the NYNET Testbed in New York state and the National Information Infrastructure (NII). 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 software architecture consists of three separate parts: Application Editor, Application Scheduler, and VDCE Runtime System. 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 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 a 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 the configurations of the resources, and up-to-date resource loads. The VDCE Runtime System consists of two managers: the Control Manager and the Data Manager. The Control Manager 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 Data Manager is responsible for providing low latency and high-speed communication and synchronization services for inter-task communications.

The rest of the paper is organized as follows. In Section 2 we present the design and prototype implementation issues of the VDCE software architecture including the three modules of the system. Concluding remarks and future work are given in Section 3.

2. Overview of VDCE Software Architecture

The main design philosophy of VDCE is to provide a general software development environment in which to build and execute large-scale applications on a network of heterogeneous resources. VDCE is composed of geo-

graphically distributed computation sites, as shown in Figure 1, each of which has one or more VDCE Servers. At each site, the VDCE Server runs the server software, called *site manager*, which handles the inter-site communications and bridges the VDCE modules to the web-based repository.

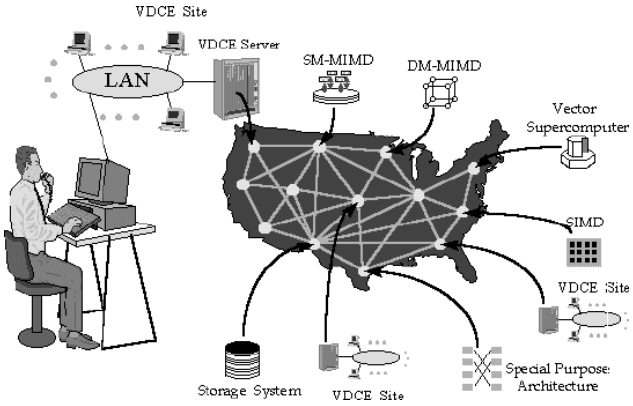

Figure 1. Virtual Distributed Computing Environment (VDCE)

Site repository, the web-based storage environment within a VDCE site, consists of four different databases.*User-accounts database* is used to handle the user authentication. In user-accounts database, each VDCE user account is represented by a 5-tuple: user name, password, user ID, priority, and access domain type. The *resource-performance database* provides the resource (machine and network) attributes/parameters. These attributes are grouped into two parts: a) *static attributes* stored in the database once during the initial configuration of VDCE such as: host name, IP address, architecture type, OS type, and total memory size; and b) *dynamic attributes* that are updated periodically, such as recent load measurement and available memory size. The *task-performance database* 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 several parameters such as computation size, communication size, required memory size, etc. In order to find locations of a task's executables, VDCE stores location information of each task (i.e., the absolute path of the task executable) for each host in the *task-constraints database*. Due to specific library requirements, some task executables may reside only on some of the hosts.

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 what follows we describe in detail the design and prototype implementation issues of the three software modules.

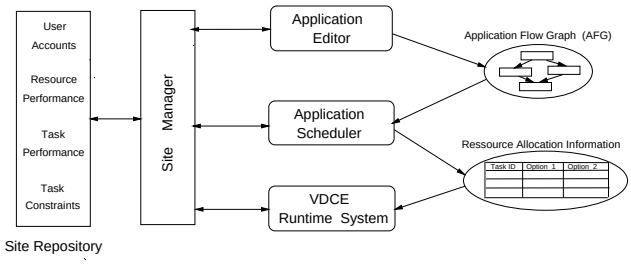

Figure 2. Interactions Among the VDCE Modules

2.1. 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. The Site Manager implementation is based on JAVA Web server technology which uses servlets (i.e., server site JAVA applets) that relive the startup overheads and run on any platform. After user authentication, 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 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.

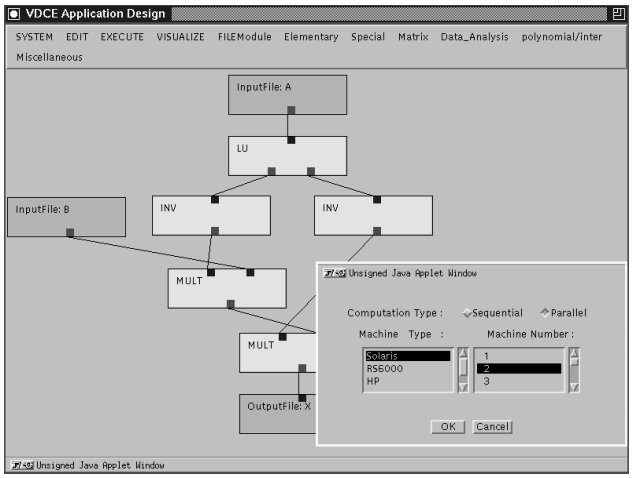

Figure 3. Building the Linear Equation Solver Application with the Application Editor

The process of building an HPDC 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 3 shows the building of an application flow graph of a Linear Equation Solver with the help of 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. LU Decomposition is one of the several methods for solving linear equations. The nodes of this application, i.e, LU decomposition, matrix inversion, matrix multiplication, etc., are selected from the matrix operations menu and linked to form the application flow graph.

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 computational mode (sequential or parallel), machine type, and the number of processors to be used in a parallel implementation of a given task (see the right part of Figure 3). In this figure, for the LU Decomposition task of Linear Equation Solver the user has selected parallel execution mode using two nodes of Solaris 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 he/she may store the application flow graph for future use.

2.2. 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 the application tasks in order to minimize the schedule length (total execution time) in a transparent manner. We provide an application-based scheduling framework that provides and guarantees Quality-of-Service (QoS) of a given application. The Application Scheduler considers both software and hardware requirements of an application before selecting the best schedule.

Our scheduling heuristic is based on *list scheduling* [11, 12, 13]. In list scheduling, each node (task) of the graph is assigned a priority before the scheduling process. The first step of the scheduling process is to select the node with the highest priority. The next step is to select the best available processor to run the selected task. These steps are repeated until all nodes of the graph are covered.

The VDCE scheduling heuristic uses the level [11] of each node to determine its priority. The node (task) with a higher level value will have a higher priority for scheduling. The level of a node in the graph is computed as the largest sum of computation costs along a path from the node to an exit node. The exit node is the node that does not have a child node. For the computation cost, the task (node) execution time on the base processor, which is already measured and stored in the *task-performance database* at site repository, is used. In order to select a task for scheduling, the node must be a *ready* node with the highest priority. A *ready* node has no *parent* nodes, or its *parent* nodes were already scheduled. In VDCE the level of each node of an application flow graph is determined before the execution of the scheduling algorithm.

2.2.1 Built-in Scheduling Algorithms

VDCE provides distributed scheduling in a wide-area system, in which each site consists of its own Application Scheduler running on the VDCE server. After the

best schedule of the whole application is determined by the local site and a set of remote sites, the resource allocation table is generated and transfered to the Site Manager running on the VDCE server. The Application Scheduler, which is based on [10, 14, 15], has two built-in algorithms: *Site Scheduler Algorithm*, and *Host Selection Algorithm*, as shown in Figure 4 and Figure 5, respectively.

The Application Scheduler at the local site, i.e., the site at which the VDCE receives the execution request of an application, runs the *site scheduler algorithm*. Once the application flow graph (AFG) is accessed by the *site scheduler algorithm*, a subset of remote sites is selected and the AFG is multicast to these sites, at which the Application Schedulers will run the *host selection algorithm*. In order to decrease the search space for scheduling, only a subset of remote sites is selected. Additionally, a site can be a local site for some of the applications and it can be a remote site for some of the others running in the VDCE system.

The built-in host selection algorithm at each remote site determines the best available machine within the site for each task, which minimizes the predicted execution time. Then each site sends the mapping information of each task, i.e, machine name and predicted execution time, to the local site. For the entry tasks that have no parents, or the tasks that do not require any input file for execution, the site scheduler algorithm selects the site (the resource within the site) that minimizes the prediction time for the task. For other cases the local-site scheduler algorithm selects the best site, based on the summation of predicted execution time and transfer time of the task input files. The site at which a parent task is scheduled, the parent's site, is determined to evaluate the transfer time. The inter-task transfer time is based on the network transfer time between a site and the parent's site, and the size of the transfer. The input size of the application can be used for the transfer size parameter. If the site is the same as the parent site, then the total inter-task transfer time will be zero.

The idea behind this algorithm is to schedule the application tasks within a site (or within the nearest neighbor sites) to decrease the inter-task communication time. Although these built-in algorithms are designed for application tasks that request a single machine, it is not difficult to extend the algorithm for parallel tasks. For parallel tasks, the host selection algorithm is updated to select the number of machines required within the site. By scheduling the parallel task execution within a site, the inter-site communication overhead for parallel tasks is removed.

The core of the given built-in scheduling algorithms is the performance prediction phase. Since the heterogeneous nature of the resources and time-sharing make the scheduling difficult, the performance prediction of tasks guides the scheduler in finding the most efficient resource allocations. As discussed in [16, 17], the performance of the processors changes from one application to another; i.e., a processor may give the best execution time for a specific application, but it may give the worst time for another application. Therefore, in VDCE we provide separate function evaluations, $Predict(task_i, R_j)$, to predict the performance of each task, $task_i$, on each resource, R_j.

The performance prediction functions are based on a combination of analytical modeling and measurements of experimental runs. This provides an accurate and fast approach to predict the performance of a given task on a particular machine. The input parameters of the prediction functions include: $Measured\_Time(task_i, R_{base})$, which is the execution time of $task_i$ on a dedicated base processor, R_{base}, for unit_size input; $Weight(task_i, R_j)$, which is the computing power weight [16, 17] of R_j with respect to the base processor, R_{base}, for $task_i$; $Mem\_Req(task_i)$, which is the memory requirement of $task_i$; $Memory\_Avail(R_j)$, which gives the available memory size on the machine; and $CPU\_load(R_j)$, which is the current load on R_j. The required parameters for prediction are stored at *task-performance* and *resource-performance* databases. Trial runs are required to obtain the computing power weights of processors for each task. The current workload parameters are computed using forecasting techniques based on a window of most recent workload measurements.

2.3. 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 Manager and Data Manager, respectively. The Control Manager measures the loads on the resources (hosts and networks) periodically, and monitors the resources for possible failures. The Control Manager daemons operate execution of the application tasks on the assigned resources by maintaining the performance and quality of service requirements.

1. Receive application flow graph (AFG) from local Application Editor.
2. Select k nearest VDCE neighbor sites, $S_{remote} = \{S_1, S_2, \ldots, S_k\}$, for local site S_{local}.
3. Multicast application flow graph to each S_i in S_{remote}.
4. Call *Host_Selection_Algorithm* (for local site and selected remote sites).
5. Receive the outputs of *Host_Selection Algorithm*, i.e, the selected machine and performance prediction time pairs of all tasks, from each S_i in S_{remote}.
6. Initialize the set for the ready tasks : $ready\_tasks = \{task_i | task_i \text{ is an entry\_node}\}$.
7. For each $task_i$ in $ready\_tasks$ set:

   If the $task_i$ is an entry task or $task_i$ does not require any input file from its parent node tasks,
   Assign $task_i$ to the site S_j, which minimizes $Predict(task_i, R_j)$.
   Else
   Determine the site(s), S_{parent}, which is assigned for one or more of the parent nodes of $task_i$.
   For each site S_j in S_{remote} evaluate:
   $Time_{total}(task_i, S_j) = transfer\_time(S_{parent}, S_j) \times file\_size + Predict(task_i, R_j)$
   Assign $task_i$ to the site S_j, which minimizes $Time_{total}(task_i, S_j)$.
   Set resource allocation table entry of the $task_i$ with the assigned resource.
   Update the ready_tasks set by removing $task_i$, and adding children nodes of $task_i$.

Figure 4. Site Scheduler Algorithm

1. Retrieve task-specific parameters of AFG tasks from *task-performance database*.
2. Retrieve resource-specific parameters of a set of resources,
   $R_{set} = \{R_1, R_2, \ldots, R_m\}$, from *resource-performance database*.
3. Set $task\_queue = \{task_i | task_i \text{ in AFG}\}$.
4. For each $task_i$ in task_queue
   Evaluate the performance prediction time of the $task_i$, $Predict(task_i, R_i)$,
   for every resource, R_i in R_{set}.
   Assign $task_i$ to the resource, R_j, which minimizes the performance prediction time, $Predict(task_i, R_j)$.

Figure 5. Host Selection Algorithm

The Data Manager provides low latency and high-speed communication and synchronization services for inter-task communications. The I/O and application visualization (real-time or post-mortem visualizations) services are provided by the Data Manager.

2.3.1 Control Manager

Functionally, the Control Manager services are grouped into two modules: the *Resource Controller*, which manages the VDCE resources, and the *Application Controller*, which manages the application execution.

The Resource Controller. The Resource Controller within a site contains three different processes: a Site Manager, a Group Manager for each group leader machine, and a Monitor daemon for each VDCE resource, as shown in Figure 6. The main functions of the Resource Controller are:

- *Retrieving Resource Performance Parameters.* VDCE resources are periodically monitored to collect up-to-date values of processor and network parameters. Each VDCE machine has a *Monitor* daemon that periodically measures the up-to-date processor parameters, i.e., CPU load and memory availability. The measured values are sent to the group leader machine. The Group Manager, shown in Figure 6, periodically receives the up-to-date values from hosts. Group Manager sends only the workloads of the resources that have changed considerably from the previous measurement to the Site Manager. 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 [20]. The Site Manager stores/updates the relevant VDCE database with the received values.

- *Monitoring the VDCE Resources.* The Group Manager periodically checks to see if all hosts in the group are alive by sending echo packets to hosts and waiting for their responses. These packets are used to detect the node and network failures and to measure the network parameters, i.e., network latency and transfer rate within a group. When a failure of a host is detected, the Group Manager passes this information to the Site Manager. Then the host is marked as "down" at the site's resource-performance database.

- *Updating the Site Repository.* As explained above, the Site Manager periodically updates the resource-performance database at the site repository with the monitoring information (i.e, the workload measurement and failure detection information of the resources). After an application execution is completed, the newly measured execution time of each application task is stored in the task-performance database. The Site Manager also updates the site repository whenever a resource is added or removed from the VDCE. The Application Scheduler retrieves the required parameters from databases through the Site Manager.

- *Sending the Related Portion of the Resource Allocation Table.* After the resource allocation table is generated by the Application Scheduler, the Site Manager multicasts it to the Group Managers that will be involved in the execution. If a machine in a group is assigned for a task execution, the Group Manager sends an execution request message and related parts of the resource allocation table to the Application Controller of the machine.

- *Inter-site Coordination.* As explained in Section 2.2, 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 Managers.

Application Controller. The execution environment setup and management services are provided by the Application Controller by interacting with the Data Manager.

- *Initialize the Application Execution Environment.* After the Application Controller receives an execution request message from the Group Manager, it activates the Data Manager. The Data Managers 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 7 shows the part of the execution environment of the Linear Equation Solver application discussed in Section 2.1. Machine 1 will execute the LU_Decomposition task, which is followed by execution of Matrix_Inversion tasks on Machine 2 and Machine 3. After the Application Executor receives the acknowledgment from Data Manager for the communication channel setup, it forwards the acknowledgment to the Site Manager. When all the required acknowledgments are received an execution startup signal is sent to start the application execution.

- *Managing the application execution.* The Application Controller 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 Application Controller terminates the task execution on the machine and sends a task rescheduling request to the Group Manager. If any assigned machine does not respond to the keep-alive packets from the Group Manager, the machine is marked as "down" and the Site Manager is informed in order to prevent further task mappings on the machine until it is up.

2.3.2 Data Manager

The VDCE Data Manager 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 7, the Data Manager 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 communication channel setup. After the setup is completed successfully, the communication proxy sends an acknowledgment to the Application Controller. 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 Application Controller and

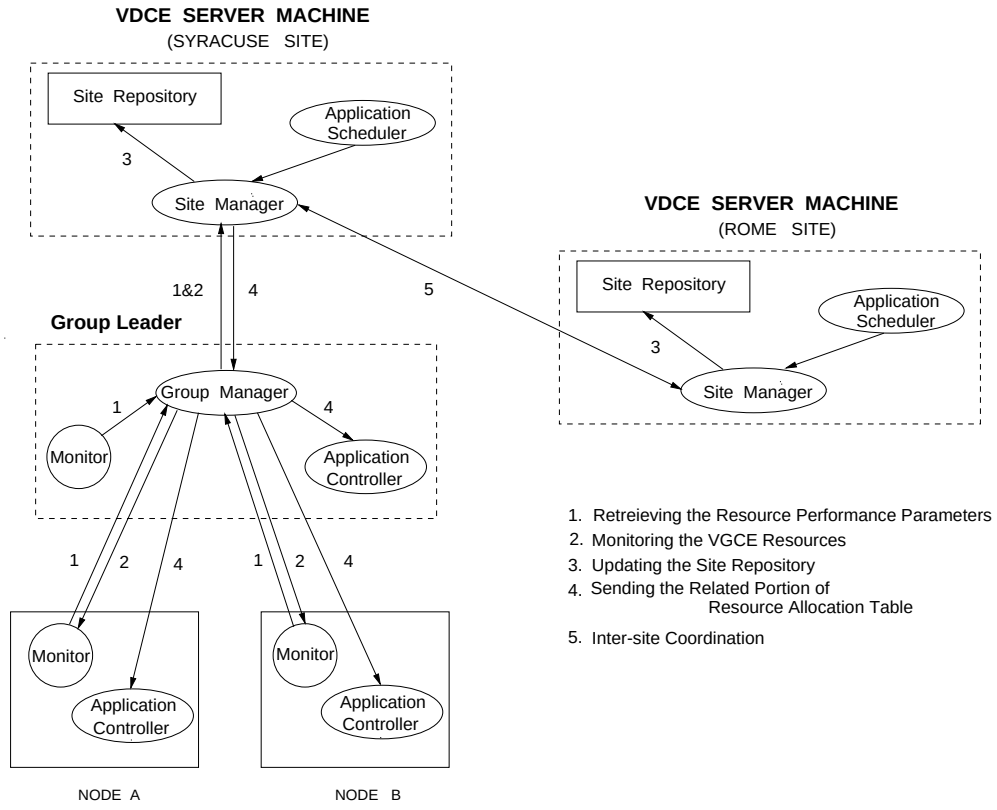

Figure 6. Interactions Among the Resource Controller Components

the Data Manager (or any other control transfer on the same machine) are based on 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 application (or another user-defined performance measure) 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.

3. Conclusion

We have proposed a problem-solving environment called Virtual Distributed Computing Environment (VDCE) for high-performance, distributed computing over wide-area networks. In this paper we provided the software architecture for VDCE, which consists of three main modules: Application Editor, Application Scheduler, and VDCE Runtime System. The Application Editor provides users with all the software tools and li-

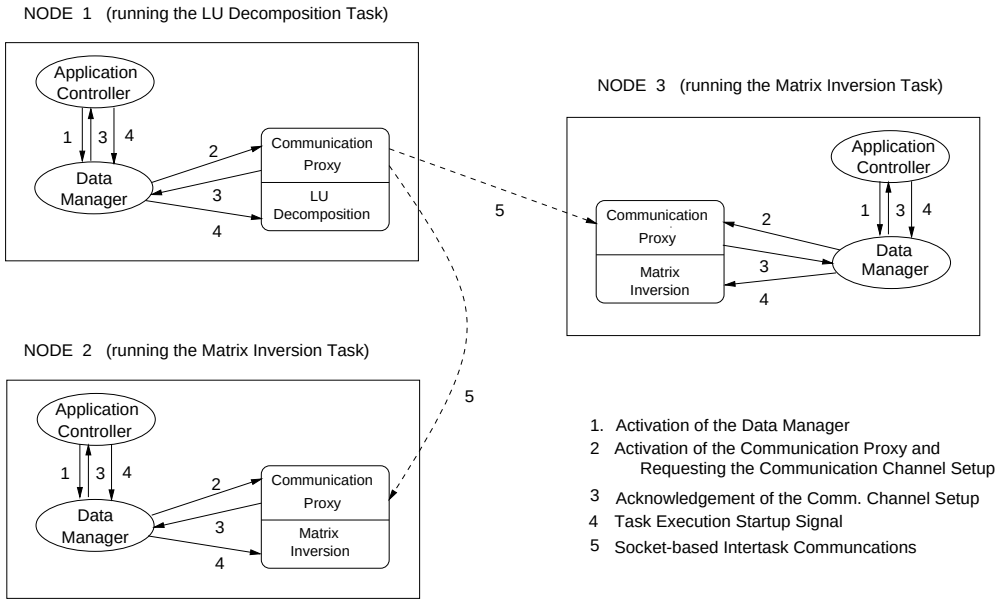

Figure 7. Setting Up the Application Execution Environment

brary functions required to develop an network application. The major function of the Application Scheduler is the initial task-to-resource mapping and any necessary dynamic rescheduling. The VDCE Runtime System is based on two managers, the Control Manager and the Data Manager. The Control Manager provides a seamless interconnection of the resources and it monitors the resources. The Data Manager enables a high-performance communication medium among the application tasks.

We have successfully implemented on campus-wide resources a proof-of-concept prototype that supports all major components of the VDCE architecture. We are improving the current implementation of the VDCE so that it can support access to several geographically distributed sites. We are also implementing a distributed shared memory model that will allow VDCE users to describe their applications using shared-memory paradigm.

Acknowledgments

We would like to thank Kivanc Dincer for the discussions on the design. We further thank Elaine Weinman for proofreading this manuscript.

References

[1] P. Newton, J. C. Browne, "The CODE 2.0 Graphical Parallel Programming Language," *Proceedings ACM International Conference on Supercomputing*, July 1992.

[2] R. Wolski, C. Anglano, J. Schopf, F. Berman, "Developing Heterogeneous Applications Using Zoom and HeNCE," *Proceedings of Heterogeneous Workshop IPPS 95.*

[3] J. C. Browne, S. Hyder, J. Dongarra, K. Moore, P. Newton, "Visual Programming and Debugging for Parallel Computing," *IEEE Parallel and Distributed Technology*, Spring 95.

[4] C. Angalano, J. Schopf, R. Wolski, F. Berman, "Zoom: A Hierarchical Representation for Heterogeneous Applications," UCSD CS Technical Report, CS95-451.

[5] R. Butler, and E. Lusk., "User's Guide to the p4 Programming System," Mathematics and Computer Science Division, Argonne National Laboratory.

[6] 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.

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

[8] Parasoft Corporation, Pasadena, CA. Express user's guide, version 3.2.5, 1992.

[9] J. Gehring and A. Reinefeld, "MARS - A Framework for minimizing the job execution time in a metacomputing environment," *Future Generation Computing Systems*, 1996.

[10] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, "Application-Level Scheduling on Distributed Heterogeneous Networks," *Proceedings of Supercomputing 96*, November 1996.

[11] T.L. Adam, K. Chandy, and J. Dickson, "A Comparison of List Scheduling for Parallel Processing Systems," Communication of ACM, Vol 17, no 12, pp 685-690, Dec 1974.

[12] H. El-Rewini, H. Ali, T. Lewis, "Task Scheduling in multiprocessing systems," *IEEE Computer*, December 1995.

[13] Y. Kwok, I. Ahmad, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, Vol 7, pp 506-521, 1996.

[14] H. G. Dietz, W. E. Cohen, B. K. Grant, "Would you run it here... or there? (AHS: Automatic Heterogeneous Supercomputing," *1993 International Conference on Parallel Processing*, Vol II, pp 217-221, 1993.

[15] J. Weissman, A. Grimshaw, "A Federated Model for Scheduling in Wide-Area-Systems," *Proceedings of HPDC5*, pp 542-550, 1996.

[16] Y. Yan and X. Zhang, "An Efficient and Practical Performance Prediction Model for Parallel Computing on Non-dedicated Heterogeneous NOW," To appear in *Journal of Parallel and Distributed Computing*.

[17] M. Zaki, W. Li and M. Cierniak, "Performance Impact of Processor and Memory Heterogeneity in a Network of Machines," *Proceedings of 4th Heterogeneous Computing Workshop*, Santa Barbara, CA, April 1995.

[18] 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 HPDC5*, 1996.

[19] K. Dincer and G. C. Fox, "Design Issues in Building Web-based Programming Environments," To appear in *Proceedings of HPDC6*, 1997.

[20] H. Casanova, J. Dongarra, "Netsolve: A Network Server for Solving Computational Science Problems," *Supercomputing 96*, November 1996.