Chapter 1

A PROBLEM SOLVING ENVIRONMENT FOR NETWORK CENTRIC COMPUTING [1]

Salim Hariri
Dept. of Electrical and Computer Engineering
The University of Arizona, Tuson, AZ 85721
hariri@ece.arizona.edu

and Dongmin Kim, Yoonhee Kim, IlKyeun Ra, Haluk Topcuoglu
Dept. of Electrical Engineering and Computer Science
Syracuse University, Syracuse, NY 13244-4100
kimd,yhkim,ikra, haluk@ecs.syr.edu

Jon Valente
Distributed Information Systems Branch, AF Research Laboratory/Rome Research Site, Rome, NY 13441

Abstract *The current advances in high-speed networks and WWW technologies have made network computing a cost-effective high performance computing environment. The new software development models and problem solving environments must be developed to utilize the network computing environment efficiently. In this paper we present the Virtual Distributed Computing Environment (VDCE) as a problem solving environment for high-performance distributed computing over wide-area networks. The VDCE enables scientists to develop parallel and distributed applications without knowing the detailed architecture of the underlying computing and communication resources. The VDCE provides well-defined library functions that relieve the end users from tedious task implementations and it supports software reusability. The VDCE software architecture consists of two modules: The Application Editor, and the VDCE Run-*

[1]This research is supported by AF Research Laboratory contract number F30602-95-C-0104.

time System. The Application Editor is a Web-based graphical user interface that helps user to develop network applications and specifies the computing and communication properties of each application task. The VDCE Runtime System schedules the individual tasks of the application to the best available resources, runs, and manages the application execution. We also present how the VDCE can be used as a problem solving environment and how the users can experiment with and evaluate the performance of their applications for different hardware and/or software configurations.

Keywords: problem solving environment; cluster computing; metacomputing; web-based computing;

1. INTRODUCTION

The current advances in networking protocols, software development tools, and the emerging WWW technologies have enabled the development of a cost-effective, high-performance distributed computing environment, *network-based computing* [1]. The network-centric computing can provide the computational and storage requirements for solving large scale applications that used to run only on massively parallel processors and supercomputers.

The available software tools for a high-performance computing environment still requires detailed understanding of the underlying architecture and the components of the application. Writing a parallel/distributed program overwhelms most of the users due to the complexity of the communication and synchronization issues. Therefore, over the last few years, a number of application development and representing tools have become available to relieve users from this tedious process, including Code [2], HeNCE [5], and Zoom [5, 6], which are all graph-based programming environments. The graph representation of parallel programming simplifies programming, debugging, and visualization phases. However, most of these tools do not provide users the ability to experiment with and evaluate parallel/distributed programs with different underlying software and hardware components.

In this paper, we present an overview of the Virtual Distributed Computing Environment (VDCE), a metacomputing environment that is currently being developed at The University of Arizona and Syracuse University. We also present how the VDCE can be used as a problem solving environment for large scale network applications. Our problem solving environment enables users to focus on the solution approach to the problem rather than worrying about the parallel/distributed com-

puting and network implementation issues.

The rest of the paper is organized as follows. Section 2 briefly discusses the issues of software development process for network-centric computing and related work. In Section 3, we present an overview of the VDCE architecture. In Section 4, we demonstrate through several examples how the VDCE can be used to develop parallel/distributed applications as well as experiment with and evaluate their performance. Summary and concluding remarks are given in Section 5.

2. SOFTWARE DEVELOPMENT ISSUES

Parallel and distributed software development is a non-trivial process and requires a through understanding of the application and the underlying computer and communication architectures [12]. The software development process is described as a set of stages where each stage requires different kinds of supports, such as portable application description/design support, program implementation and run-time support, visualization support and reusability support. Software development process can be broadly defined in terms of three stages: *Application Development*, *Application Scheduling*, and *Application Execution*.

2.1 APPLICATION DEVELOPMENT

In this stage, the application is specified (represented) in the form of a functional flow description of the application and its computation and communications requirements. Each node (termed as a functional module) in the functional flow diagram is a black-box and contains information about (1) its input(s), (2) the function to be performed, (3) the desired output(s) and (4) the requirements at each node. The output of this stage is a detailed process flow graph where the nodes represent the functional components and the edges represent interdependencies.

Designing and implementing parallel/distributed programs overwhelms most users due to the difficulty of expressing communication and synchronization among the computations [7]. 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 parallel programs difficult to understand.

However, a graph-based programming environment provides simple and easy-to-use mechanisms for expressing the interaction of multiple threads within a parallel program [7, 8]. Over the last few years a number of graph-based application development and representation tools have become available, including Code [2], HeNCE [5], and Zoom [5, 6]. However, most of these tools do not provide users the ability to experiment with and evaluate the generated parallel/distributed programs with different underlying software and hardware components, different implementation techniques, interconnection networks and message-passing tools.

2.2 APPLICATION SCHEDULING

This stage determines the hardware and software requirements of the application by interpreting the application specifications and assigns current best available resources for running the application tasks in order to minimize the total execution time. This stage includes mapping module and estimation module. The mapping module has two consecutive parts according to it hierarchy: domain-mapping part, resource-mapping part. The estimation module is a critical component of the application development stage. This module evaluates different options available and identifies the option that provides the best performance.

Since the general form of scheduling problem is NP-hard, researchers have been working on near-optimal scheduling algorithms. Most of these algorithms are not general; instead they are targeted for specific application and specific resource types. There are some research projects that target application-level resource allocation issues such as APPLeS [11], and MARS [10] projects.

2.3 APPLICATION EXECUTION

This stage handles the execution of the developed and scheduled application. The runtime stage integrates the assigned resources that will be involve in execution, and supports inter-module communications, which is based on either a message-passing tool such as PVM, p4, MPI, NCS or a distributed shared memory (DSM).

In the next section, we describe how these software development issues are implemented in the VDCE prototype.

3. VDCE ARCHITECTURE OVERVIEW

The objective of the Virtual Distributed Computing Environment is to provide a general software development environment to develop large scale parallel and distributed applications. As a problem-solving environment, the VDCE provides users with a set of task libraries to solve one class of applications. These task libraries are used to compose any application as an application flow graph. The VDCE consists of several geographically distributed computational sites, each of which runs its own VDCE Server. An end user views the underlying VDCE resources that are interconnected by a global wide area network as a single seamless computation resource (see Figure 1.1).

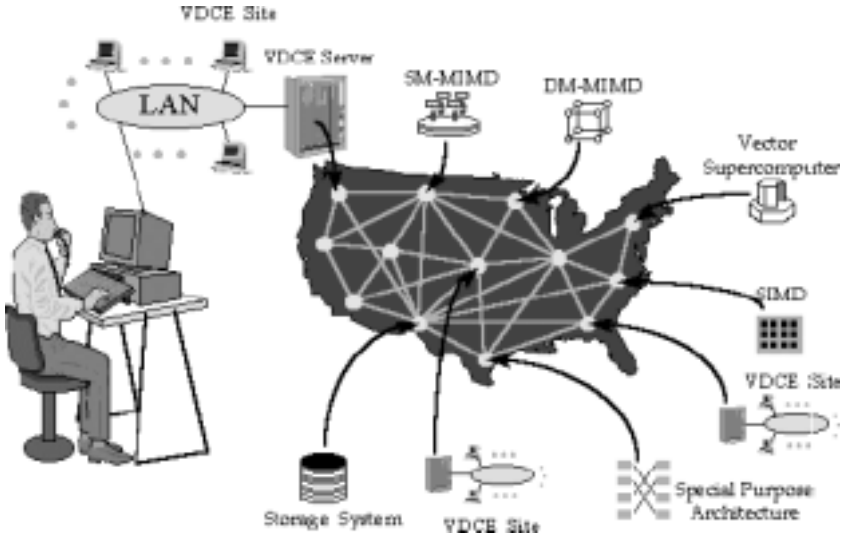

Figure 1.1 An example of a Virtual Distributed Computing Environment (VDCE)

To develop an application, the end user establishes a URL connection to the VDCE Server within the site. After the proper user authentication, the Application Editor will be loaded into the user's local web browser, so that, the user can develop his/her network application. The process of building a network application with the Application Editor can be grouped into two steps: building the application flow graph (AFG), and specifying the task properties of the application. Figure 1.2 shows the development of the application flow graph of a Linear Equation Solver using the VDCE Application Editor. 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 the

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 Figure 1.3). In this figure, for the LU task of the Linear Equation Solver, the user has selected the parallel execution mode using 2 nodes of Solaris machines that are interconnected by an ATM network.

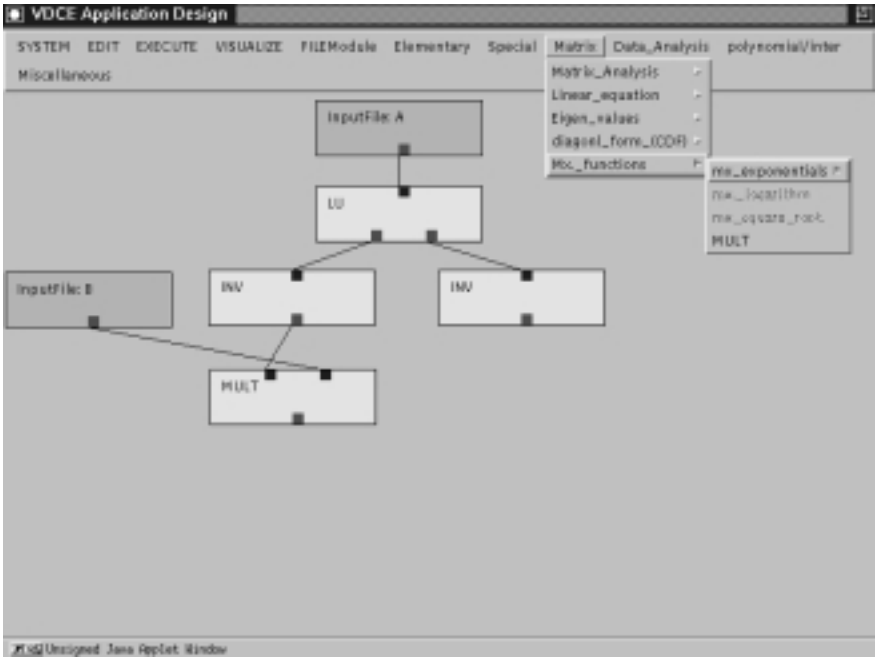

Figure 1.2 Building Application Flow Graph of Linear Equation Solver Application

Apart from the application development process, the Application Editor provides different visualization supports that are grouped as: Module/Application Execution Visualization, and Workload Visualization for all the available VDCE resources as well as for those selected to run a given application.

3.1 VDCE RUNTIME SYSTEM

The functionality of the VDCE Runtime System can be viewed in terms of two machines: The Control Virtual Machine (CVM) and the Data Virtual Machine (DVM). *The Control Virtual Machine* is responsible for scheduling the application, monitoring and managing the execu-

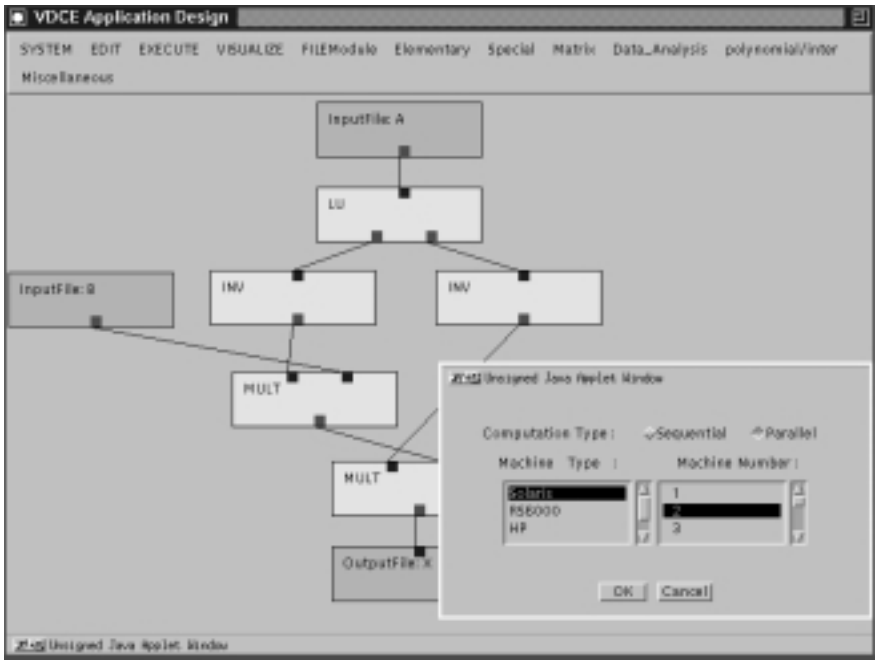

Figure 1.3 Snapshot of the Application Editor while Specifying Task Properties

tion of the application on the assigned resources. The CVM handles the load measurement of resources (hosts and networks) and periodically updates the *resource-performance database*. The CVM supports real-time and post-mortem visualization of the application execution. For further detail about the architecture of the VDCE and its implementation approach can be found in [3, 4].

4. VDCE APPLICATION DEVELOPMENT

In this section, we present how the VDCE can be used as a problem solving environment for matrix algebra applications and use the Linear Equation Solver application as a running example. 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. The Linear Equation Solver can be solved based on Gauss-Jordan elimination, Gaussian elimination with back-substitution, or LU Decomposition. With LU Decomposition any matrix can be decomposed into the product of a lower triangular matrix L and upper triangular matrix U. Once LU Decom-

position is solved, the solution vector, x, is derived as shown below:

$$\begin{aligned} Ax &= b \\ (LU)x &= b \\ Ux &= L^{-1}b \\ x &= U^{-1}(L^{-1}b) \end{aligned}$$

Figure 1.2 shows the corresponding application flow graph to solve the Linear Equation Solver. Figure 1.3 shows how the user can specify the task properties (i.e., machine type, sequential or parallel execution, number of machines for parallel execution case).

4.1 EXPERIMENTAL RESULTS

The VDCE provides support to experiment with and evaluate the application performance for different underlying hardware/software configurations. In what follows we discuss the performance results of the Linear Equation Solver for different network types and machine configurations. The problem size considered in our analysis is 128×128. Furthermore, each task was run sequentially and in parallel on 2 and 4 solaris machines interconnected by both ATM and Ethernet networks. Table 1.1 shows the benchmarking results of the individual tasks (LU (LU Decomposition), INV (matrix inversion) and MULT (matrix multiplication)).

Table 1.1 Performance of Linear Equation Solver tasks on ATM and Ethernet (msec)

No. of nodes	**LU** Ethernet	**LU** ATM	**INV** Ethernet	**INV** ATM	**MULT** Ethernet	**MULT** ATM
1 node	226073	217191	280626	278534	49903	48392
2 nodes	236180	233573	276193	273654	49205	44091
4 nodes	253731	253089	274421	270139	53088	50311

The performance of the LU task is best when it runs sequentially because it is a communication-bound. For the INV task, the parallel execution on 4 nodes gives the best performance while the parallel execution of the MULT task is best when it runs on two nodes. The ATM-based network gives better results than Ethernet-based case for

all the task implementations. According to these results, if the user chooses 1 node for the LU task, 4 nodes for the INV task and 2 nodes for the MULT task, the application performance will be better than all the other configurations.

The linear equation solver example presented here serves as a proof-of-concept on the benefits that can be gained from allowing users to experiment with different implementation algorithms to determine the configuration that lead to better performance results.

5. CONCLUSION

The VDCE problem solving environment enables scientists to develop parallel/distributed applications without specialized knowledge of the underlying computer and communication hardware and software. In the VDCE,the users use their web-browsers to write parallel/distributed applications and experiment with and evaluate the application performance for the different combinations of hardware and/or software tools. The VDCE architecture can be described in terms of two main modules: The Application Editor module and the VDCE Runtime System. The Application Editor provides users with all software tools and library functions required to develop a parallel/distributed application. The VDCE Runtime System maps the tasks of the application onto the resources, monitors the resources, establishes a high-performance communication among the application tasks, sets up the application execution environment and manages the application execution. We are improving the current implementation of the VDCE to support both mobile as well as fixed computing and communication resources.

References

[1] D. k. Panda, and L. M. Ni, "Special Issue on Workstation Clusters and Network-Based Computing", Journal of Parallel and Distributed Computing, 40, pp 1-3, 1997.

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

[3] 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.

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

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

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

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

[8] M.F. Kleyn, J.C. Browne, "A High Level Language for Specifying Graph Based Languages and Their Programming Environment (Draft)", University of Texas at Austin

[9] K. Dincer and G. C. Fox, "Design Issues in Building Web-based Programming Environments", To appear in Proceedings of the Sixth IEEE Int.Sym. on High Performance Distributed Computing (HPDC-6), 1997.

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

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

[12] M. Parashar, S. Hariri, T. Haupt, G. Fox, "A Study of Software Development for High Performance Computing" Programming Environments for Massively Parallel Distributed Systems, 1994.