A global computing environment for networked resources

Haluk Topcuoglu
Syracuse University

Salim Hariri
Syracuse University

A Global Computing Environment for Networked Resources*

Haluk Topcuoglu and Salim Hariri
Department of Electrical Engineering and Computer Science
Syracuse University, Syracuse, NY 13244-4100.

Abstract

Current advances in high-speed networks and WWW technologies have made network computing a cost-effective, high-performance computing alternative. New software tools are being developed to utilize efficiently the network computing environment. Our project, called Virtual Distributed Computing Environment (VDCE), is a high-performance computing environment that allows users to write and evaluate networked applications for different hardware and software configurations using a web interface. In this paper we present the software architecture of VDCE by emphasizing application development and specification, scheduling, and execution/runtime aspects.

1 Introduction

The new trends in networking protocols (including ATM and Fast Ethernet) and emerging WWW technologies have enabled the development of a cost-effective, high-performance, distributed computing environment, *network-based computing.* The target of current research on software tools and problem solving environments is to exploit fully the underlying network-based computing framework. We are developing a network-based computing environment called Virtual Distributed Computing Environment (VDCE). VDCE is composed of distributed sites, 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 site databases. 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.

In this paper we present the VDCE-based application development, which can be divided into a pipeline of three phases: application design and specification, scheduling, and execution/runtime. VDCE provides a web-based graphical user interface, the Application Editor, that helps users to design and build parallel and distributed applications. The VDCE Application Scheduler component is a distributed runtime scheduler that uses performance prediction of individual tasks of an application to achieve efficient resource allocations. The third component, the VDCE Runtime System, is responsible for monitoring the networked resources, setting up the execution environment of a given application, monitoring the task executions on the assigned resources, and providing communication and synchronization services for intertask communications.

The rest of the paper is organized as follows. In Section 2 we present the application design and specifications issues. The Application Scheduler is explained in Section 3. We present the VDCE Runtime System in Section 4. Concluding remarks and future work are given in Section 5.

2 Application Design and Development

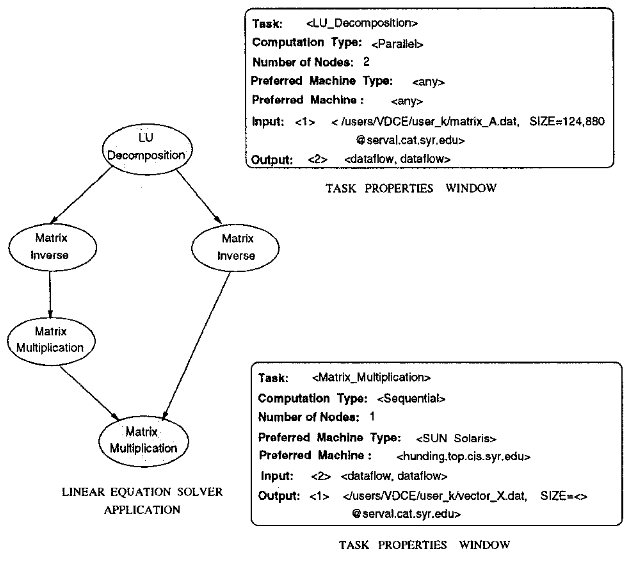
The Application Editor component of VDCE is a web-based, graphical user interface for developing parallel and distributed applications. The end-user establishes a URL connection to the VDCE Server software within the site (*Site Manager*), which runs on a VDCE Server. After user authentication, the Application Editor is 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. The process of building an application with the Application Editor can be divided into two steps: building the application flow graph (AFG), and specifying the

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

task properties of the application.

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), input/output files, machine type, and the number of processors to be used in a parallel implementation of a given task. If an input of a task is supplied by its parent tasks, the file entry is marked as *dataflow*. Figure 1 shows the application flow graph of the Linear Equation Solver and the contents of the task properties window for LU_Decomposition and Matrix_Multiplication tasks.



Task: <LU_Decomposition>
Computation Type: <Parallel>
Number of Nodes: 2
Preferred Machine Type: <any>
Preferred Machine : <any>
Input: <1> < /users/VDCE/user_k/matrix_A.dat, SIZE=124,880 @serval.cat.syr.edu>
Output: <2> <dataflow, dataflow>

Task: <Matrix_Multiplication>
Computation Type: <Sequential>
Number of Nodes: 1
Preferred Machine Type: <SUN Solaris>
Preferred Machine : <hunding.top.cis.syr.edu>
Input: <2> <dataflow, dataflow>
Output: <1> </users/VDCE/user_k/vector_X.dat, SIZE=<> @serval.cat.syr.edu>

Figure 1. Application Flow Graph of Linear Equation Solver

3 Application Scheduling

The main function of the Application Scheduler module in VDCE is to interpret the application flow graph and to assign the most suitable available resources for running the application tasks in order to minimize the schedule length (total execution time) in a transparent manner. Our scheduling heuristic is based on *list scheduling* [2, 3, 4]. In list scheduling, each node (task) of the graph is assigned a priority before the scheduling process. The VDCE scheduling heuristic uses the level [4] 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 the path from the node to an exit 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 VDCE the level of each node of an application flow graph is determined before the execution of the scheduling algorithm. 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 nearest remote sites, the resource allocation table is generated and transferred to the Site Manager running on the VDCE server. Application tasks are scheduled within a site (or within the nearest-neighbor sites) to decrease inter-task communication time. The Application Scheduler, which is based on [1, 5], has two built-in algorithms: *site scheduler algorithm* and *host selection algorithm*, as shown in Figure 2 and Figure 3, respectively. When the Application Scheduler receives the execution request of an application, it runs 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. 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,

```
1. Receive application flow graph from 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 $Host\_Selection\_Algorithm$ (local and remote sites).
5. Receive the outputs of $Host\_Selection\ Algorithm$,
     from each $S_i$ in $S_{remote}$.
6. Initialize $ready\_tasks = \{task_i | task_i$ 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 input
       •Assign $task_i$ to $S_j$, 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 $S_j$, which min. $Time_{total}(task_i, S_j)$.
    Store resource allocation information for $task_i$.
    Update the $ready\_tasks$ set by removing $task_i$, and adding
     children nodes of $task_i$.
```

Figure 2. 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,
   $R_{set} = \{R_1, R_2, \dots, R_m\}$, from *resource-perf. 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 $task_i$,
      $Predict(task_i, R_i)$, for all $R_i$ in $R_{set}$.
     •Assign $task_i$ to $R_j$, which minimizes the performance
      prediction time, $Predict(task_i, R_j)$.
```

Figure 3. Host Selection Algorithm

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 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. For parallel tasks, the host selection algorithm is updated to select the number of machines required within the site. The core of the given built-in scheduling algorithms is the performance prediction [6] phase, which is provided by separate function evaluations of each task on each resource.

Each site has a site repository for storing user-accounts information, task and resource parameters that are used by the scheduler. A *user-accounts database* is used to handle 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. A *resource performance database* provides resource (machine and network) attributes or parameters such as host name, IP address, architecture type, OS type, total memory size of the machine, recent workload measurements, and available memory size. A *task performance database* provides performance characteristics for each task in the system and is used to predict the performance of a task on a given resource. Each task implementation is specified by several parameters such as computation size, communication size, required memory size, etc. A *task constraints database* is used to store the location information of each task (i.e., the absolute path of the task executable) for each host.

4 Application Execution and Runtime Support

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 Data Manager provides low-latency and high-speed communication and synchronization services for inter-task communications.

4.1 Control Manager

Functionally, the Control Manager services are grouped into two modules: the *Resource Controller*, and the *Application Controller.*

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. In what follows we summarize the functions of the Resource Controller components shown in Figure 4.

The *Monitor* daemon periodically measures the up-to-date resource parameters, i.e., CPU load and memory availability and sends the values to the Group Manager. The Group Manager sends to the Site Manager only the workloads of the resources that have changed considerably from the previous measurement [7].

Another function of the Group Manager is to periodically check all hosts in the group by sending echo packets to hosts and waiting for their responses. When a failure of a host is detected, the Group Manager passes this information to the Site Manager. The host is then marked as "down" at the site's resource-performance database.

The Site Manager component of the Resource Controller 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), and it updates the task-performance database with the execution time after, an application execution is completed.

Another function of the Site Manager is to multicast the resource allocation table to the Group Managers that will be involved in the execution. Each Group Manager sends an execution request message and the related portion of the resource allocation information to the Application Controller of the related machines. Additionally, the inter-site coordination

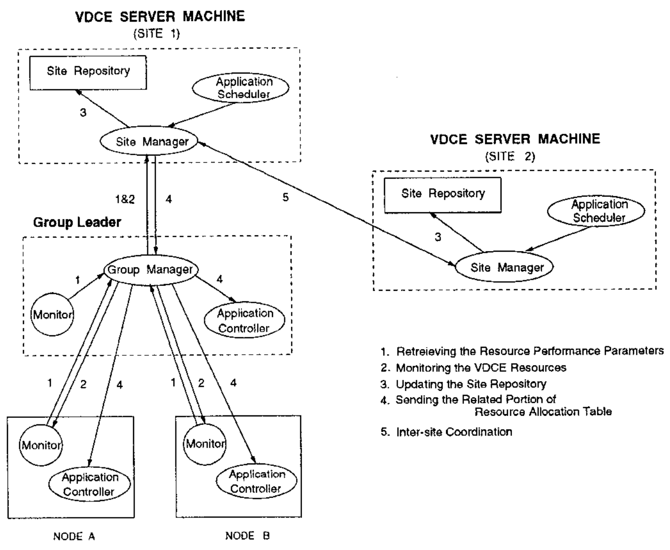


Figure 4. Interactions Among the Resource Controller Components

and message transfer (for scheduling and monitoring purposes) are handled by Site Managers.

Application Controller The Application Controller sets up the execution environment and manages the services provided by interacting with the Data Manager. 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. When all the required acknowledgments are received an execution startup signal is sent to start the application execution.

The Application Controller monitors the application execution on the assigned machines. 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.

4.2 Data Manager

The VDCE Data Manager is a socket-based, point-to-point communication system for inter-task communications. 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, as explained in the previous section.

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 application performance and workload visualizations.

5 Conclusion

We have presented the design of the Virtual Distributed Computing Environment (VDCE) for networked resources. We have successfully implemented a proof-of-concept prototype on campus-wide resources that supports the application design, scheduling, and runtime aspects. We are improving the current implementation of the VDCE so that it can support accesses to several geographically distributed sites. We are also implementing a distributed shared memory model that will allow VDCE users to describe their applications using a shared memory paradigm.

References

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

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

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

[4] 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.

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

[6] 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.*

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