

Project title	VPC - Virtual Private Cluster
Student researcher	Sweta Patel
Advisor	Prof. Silvia Figueira
Institution	Santa Clara University

Goals and Purpose of the Project

The VPC's purpose is to allow distributed parallel applications to execute securely and at full speed on a network of workstations, by isolating the machines executing an application from other machines and users.

This project was divided in two parts. The first goal was to create VPCs (Virtual Private Clusters), in which distributed parallel applications can execute in clusters of workstations securely, without the interference of other applications. This part of the project included the development of functions to create and release a VPC. Each VPC contains the nodes requested by an application. During the existence of a VPC, the VPC machines are isolated from the other machines, and communication is only allowed within the cluster. Local firewalls are used to block messages coming from machines outside the VPC. This guarantees that an application can execute securely and at full speed, i.e., with no contention for CPU in each machine. Also, during the existence of the VPC, the login capability of the machines are turned off to make sure that the application executes in dedicated mode, using all the capacity of the machines.

The second goal was to create an authentication mechanism to eliminate the need for the ".rhosts" file. Currently, the execution of MPI applications depends on the existence of a ".rhosts" file, which allows users to login from one machine to another without a password. The existence of this file is a serious security problem. When there is a ".rhosts" file, once an intruder breaks into an account, this intruder has access to all the machines listed in the file.

Account of Process Used in Research

The plan was to have the VPC work in a network of Linux machines. Linux clusters have been used extensively for parallel computing, and the Linux system has firewall features embedded in the kernel. In fact, Iptables, which is a firewall program, comes standard with the most popular Linux distributions and is the firewall of choice among Linux administrators. It is used to set up, maintain, and inspect the tables of IP-packet filter rules in the Linux kernel. There are several different tables which may be defined, and each table contains a number of built-in and user-defined chains. Each chain is a list of rules that are checked against packets. Each rule has a *target*, which specifies what to do with a packet that matches. For example, a *target* may be a jump to a user-defined chain in the same table. The program is open source and is always being updated. The VPC software uses an API (Application Program Interface) to communicate with Iptables.

I began my research for this project by talking to my project advisor about the security issues concerning the clustered computing environment. I then read research papers and articles on Grid computing and the security measures that are employed in there. This reading helped me get a bigger picture of my project. After that I went to the actual design and implementation of the VPC. But, before I could work on Linux and the firewalls, I had to familiarize myself with the operating system and the Iptable modules that come with Red Hat Linux. After that, I learned how to install and use SSH and MPICH together to enable the execution of parallel applications without the ".rhosts" file on the computers.

After understanding the basic installation and use of all the major technologies, I used the C programming language to develop an interface that allows the users to input application name and password. This way they authenticate themselves for using the applications that are registered with the VPC. After the user is authenticated, the interface allows the user to enter their node requirement for application execution. The VPC tool then checks to see if there are enough nodes to execute the application. If the nodes are available, the interface asks the user to enter the data set. During this process, the VPC tool establishes

firewalls between the individual selected nodes and the master node, where the request was initiated. This way, the nodes talk to the master node only, thereby eliminating the race condition for acquiring the nodes by other computers using similar clustering tools. After deciding that there are enough nodes, the master node dissolves the individual firewalls and establishes a bigger firewall around all the nodes so that they talk to each other and to the master node, but not to other computers. When the firewall is set, the master node executes the application using MPICH. Before the execution begins in the firewalled environment, the master node is authenticated by each node that will be used by the application. This process uses SSH public/private keys. At the end of execution, the VPC software dissolves the bigger firewall to make the nodes available for other requests.

Conclusions and Results Achieved

The end result of this over-a-year long research is that I have learned new technologies and have combined security and resource management into a tool that can be used for clustered parallel computing.

Project Title	CS_Lite: A Lightweight Computational Steering System
Student Researcher	Sonia Bui
Advisor	Dr. Silvia Figueira
Institution	Santa Clara University

Goals and Purposes

The main goal was to design and implement a simpler, more general, and easier to use computational steering environment. A computational steering environment (CSE) allows users to interact with normally non-interactive batch processes. Being able to interact with their programs allows researchers to gain insights into the execution behavior of programs. Many computational steering environments have been developed, but most are too complex, making them difficult to use. In addition, CSEs were developed specifically for one type of application, so a certain CSE developed for one application could not be easily used for a different type of application. The goal of my project, CS_Lite, is to provide to the user (any programmer wishing to interact with his or her program, usually a researcher) a simpler, easier to use CSE that can be used in a wide variety of applications. All the user needs to do in order to use CS_Lite is to incorporate the necessary CS_Lite libraries and add specific CS_Lite function calls, very similar to using MPI.

Account of Process Used in Research

The first step in my research involved reading papers about existing computational steering environments. I had to first understand how computational steering environments worked, gaining ideas on how to develop CS_Lite. I also found out what the flaws in the other CSEs were, namely their complexity and application-specificity.

The next step was the design process of CS_Lite. CS_Lite involved a client-server architecture. The clients were the nodes executing the program (augmented with CS_Lite function calls); the servers monitored and steered the program. Monitoring involves the outputting of various program variables during the course of program execution; steering is the modification of variable values inputted by the user during execution. The clients communicated with the servers through the network.

CS_Lite can be broken up into three parts: the server components, the client components, and the user interface. The monitoring server was designed to accept UDP packets being sent by a client program. Each packet contained the node number of the executing machine, the message number, and the actual value of a variable. The monitoring server writes the contents of the packet into a log. The steering server was designed to accept TCP packets. TCP was used because of the reliability required in modifying program variables. Both servers are multi-threaded, allowing them to handle multiple requests coming from

different clients at the same time (since parallel programs could involve clients executing at the same time).

The client component of CS_Lite communicates with the monitoring and steering servers. The client component would abstract from the user all the low-level details of socket communication. The user would just need to know how to call the correct function to communicate with which server. Different function calls would be needed for different data types of the variables. In addition, setup functions would be needed to establish the name and port number of the two servers. Also, a shutdown function would shut the servers down after the program has finished monitoring and steering. By adding the necessary client functions to their code, users can make the programs more interactive.

The user interface was designed to be simple and easy to use. The user would start the servers on the command line, specifying the port number used in the setup function of the client. To view the monitoring log, the user would open a browser to a webpage that would display in the browser the contents of the log. To steer the program variables, the user would open another browser window to a webpage that would provide the user with the name of the variable and an area for inputting the new value. The new value would be sent back to the steering server, which sends the new value back to the executing program. All webpage interactions employ CGI.

These components were implemented separately. Each component followed the iterative model, in which the components evolved and gained functionality in each iteration of the traditional waterfall cycle. Although the components were developed separately, tests of the interaction of the components occurred frequently. All components were built from scratch. The target environments were Unix systems. All code was written in C to promote portability.

Results and Conclusions

The end product of CS_Lite was able to accomplish the specified goals for this project. CS_Lite did provide the needed monitoring and steering functionality of other more complex CSEs, but unlike the complex CSEs, CS_Lite gave a much friendlier interface to the user: monitoring and steering functionality in a program was achieved with simple function calls to the CS_Lite library, and monitoring and steering occurred in the commonly used web browser interface. This simple interface feature of CS_Lite also promotes its use in a wide variety of applications. In addition, the resulting CS_Lite system also added a bonus feature of recovery if the steering server went down. Future additions to CS_Lite may include secure communications between servers-clients and servers-webpages, and compatibility with Linux systems.