CluMSy: A Middleware for Cluster Management

Elaine Jane E. Chua[1] Philip Chan[2] Reynaldo Jose C. Camarillo[1]
Frances Ellen O. Pe-Aguirre[1] Jessamine Y. So[1] Mara Ailiez C. Sy[1]
[1]College of Computer Studies, De La Salle University
2401 Taft Avenue, Manila 1004 Philippines
chuae@dlsu.edu.ph, {fleas2016,francesellen8,jazzy_ph,ailiez}@yahoo.com
[2]School of Computer Science and Software Engineering
Monash University, Caulfield, VIC Australia
pchan@csse.monash.edu.au

ABSTRACT

Handling common tasks, such as listing of files or processes, and manually setting up consistent software installation on all nodes of a cluster will require a lot of time and effort and may affect productivity of the cluster. Thus, interactive monitoring and management functions in a cluster environment are becoming a necessity. CluMSy is a cluster middleware that provides support for parallel programming through interactive monitoring and management functions on a cluster of independent nodes running Linux. These are classified into three (3) categories: cluster-level file management, cluster-level process management, and cluster-level account management commands. Aside from monitoring and management functions that incorporate classical UNIX commands, CluMSy also assists users in the preparation and termination of parallel program executions.

1. INTRODUCTION

Clusters of workstations (COWs) are fast becoming successful alternatives to more expensive specialized parallel computers. A cluster is typically a collection of interconnected standalone computers, or computing nodes, and is used as a single, unified computing resource [4]. Because of their loosely coupled architecture, clusters can be built with a large number of nodes. Management of such huge systems is a tedious as it is often necessary to individually install/configure software on each node. This can be eased by software systems, which allow monitoring and managing of the cluster of workstations. Such systems are known as cluster management systems.

Cluster management systems aid users in monitoring and managing COWs. These systems manage the utilization of nodes in a cluster and maintain up-to-date information about the load of each node, the memory utilization and the processes running on the nodes. Currently, almost all cluster management systems deal solely with either monitoring of processes of the cluster or the management of processes of the cluster. In addition, there exist cluster management systems for load balancing but they focus mainly on high-performance computing functionality to provide a stable service cluster configuration, but status monitoring was not considered [3].

One common pitfall of neophyte cluster administrators is that these administrators would have to maintain an entire cluster by performing the administrative same task on each of the nodes. At first glance it appears manageable, especially for small clusters, but as clusters scale and as time passes, small differences in each node configuration negatively affects system stability.

1.1 Related Works

There are some systems that are similar to CluMSy.

Gropp, Ong, and Lusk have implemented *scalable UNIX tools for parallel processor*s [2]. It makes use of UNIX commands modified for parallel environments, which were utilized because the usefulness of these commands *(*`ls`, `ps`, `find`, `grep`, etc.) has endured the age of GUI not only because of their simple, straightforward design but also because of the way they work together.

In specifying hosts, all the commands use the same method for specifying the collection of hosts on which the given commands is to run. A host list can be given either explicitly, as in the blank-separated list *'donner dasher blitzen*', or implicitly in the form of a pattern like *ccn%d@1-32,42,65-9*6, which represents the list, *ccn1,…,ccn32,ccn42, ccn65,…, ccn9*6. All of the commands that will be described below have host arguments as a (optional) first argument [2].

The Parallel Unix Commands fall into three types: straightforward parallel versions of traditional commands with little or no output (parallel `shgrp`, parallel `chmod`, parallel `cp`, etc), parallel versions of traditional commands with specially formatted output (parallel `ls`, parallel `find`, parallel `cat`), and new commands in the spirit of the traditional commands but particularly inspired by the parallel environment (parallel `exec`, parallel `pred`, parallel `distrib`).

In February 2000, Alta Technologies developed *ClusterWorX* [5], a commercially-available solution to control the cluster as one single system and provides remote monitoring and management capabilities [5]. It is primarily designed to monitor and manage Linux-based clusters. Its notable features include increased system uptime, improved cluster efficiency, cluster performance tracking, and cluster installation and configuration.

ClusterWorX provides monitoring of the system properties, among them are CPU usage, memory usage, disk I/O and network bandwidth [5]. It also allows administrators to remotely monitor and manage Linux NetworX cluster systems from an onsite or offsite location, with any Java enhanced browser. These functions are all accessible through their own Graphical User Interface. ClusterWorX is designed to work with Linux NetworX hardware

solution to Linux-based clusters, ICE Box™ to provide power management and serial console access to each node in the cluster.

The *Cluster Command Control* (C3) [1] is a suite of cluster tools developed at Oak Ridge National Laboratory. The tools cover cluster-wide command execution, file distribution and gathering, process termination, remote shutdown and restart, and system image updates [1]. Similar to the study of Gropp el. al. [2], the C3 tool suite makes use of native UNIX commands modified to serve as parallel commands.

Windows Network Neighborhood [10] allows users to remotely look at the files that other PCs have, provided that they are connected through a network and the files are shared. The user is able to manage files or folders on a remote node just as he does on his own local node. The user can view shared files and folders in My Network Places or Network Neighborhood, or, if users have mapped the drives or folders, in My Node or Windows Explorer. Depending on the share access level of the files and folders, users can select files and delete them, or move them to another folder on the remote node. The user can also create, rename, move, or delete folders on the remote node.

Table 1. File Management Commands

Command	Description
`clu_ls`	Performs `ls` on target nodes. It allows the user to view available files and directories on each node of the cluster where the user has an account. Other than listing the names of the available files, the user also can view other information about the file like size of the file, owner and the current user's permissions on the file.
`clu_find`	Performs `find` on target nodes. It allows the user to search for a specific file or directory using a given expression on the current directory hierarchy.
`clu_locate`	Performs `locate` on target nodes. It allows the user to search for a specific file or directory with a name satisfying the given regular expression on the entire file system.
`clu_cp`	An extension of the `cp` utility. It copies files or directories to a specific location on target nodes. It uses the remote copy DPI of the CCM.
`clu_rm`	Performs `rm -vf` on target nodes. The function will allow the user to delete for a specific file or directory. This utility will delete for a specific file or directory indicated by the user on nodes where the user has an account.
`clu_mv`	Similar to the `mv` command. It copies files or directories to a specific location on target nodes. It uses the remote copy DPI of the CCM. After copying, it deletes the source files; hence the old file is moved to the new destination.
`clu_cat`	Performs `cat` on target nodes. The utility will allow the user to view a specific file. It does not support `stdin` (standard input) redirection.
`clu_exec`	Allows users to execute an arbitrary command on target nodes. It uses the remote execution DPI of the CCM.
`clu_chmod`	Performs `chmod` on target nodes. It allows the user to change the file permissions of a given file.
`clu_stat`	Performs `stat` on target nodes. It allows the user to view statistics of a single file or directory.
`clu_mkdir`	Performs `mkdir` on target nodes. It allows he users to create directories.

1.2 Motivation

The short survey above reveals that there is substantial interest in developing software for cluster management. We found that none of these systems provide services for account management and software component deployment. Account management functions include cluster-wide user account creation and deletion as well as handling inconsistencies between nodes during downtimes. As our base operating system is Linux, we also needed a facility to manage Red-Hat Packages at the level of the cluster. We need to install software packages consistently on selected or all nodes.

The goal of the paper is to present a familiar and complete cluster management system running on top of the Linux Operating System [7]. This system called CluMSy is aimed not only for cluster management but also to provide as a support tool for high-performance computing on top of clusters. Parallel execution of applications will require some preparations (e.g., deploying executables, moving data files between nodes, etc.). CluMSy aims to provide this support, rather than focus on being a parallel programming environment like MPI [6] implementations or PVM [8][9].

2. CLUMSY COMMAND INTERFACE

CluMSy provides the sets of commands at the command-line. Table 1 is the set of commands that handle file-related activities; Table 2 is the set of commands that handle process-related activites; and Table 3 is the set of commands that handle user-related and group-related activities.

CluMSy also provides a package manager utility that runs rpm on target nodes. It allows the user to view the information regarding a specific package. It can show details of the specific package such as the following: name, version, release, install date, group, size, summary, description, distribution, vendor, build date, build host and source RPM. It also allows the user to install, uninstall and update a specific package on the different nodes in the cluster.

Table 2. Process Management Commands

Command	Description
`clu_ps`	Performs `ps` on target nodes. It allows the user to monitor the processes running on the different nodes of the cluster. It can show the current processes running as well as properties like the process name, process ID, RSS value, nice value, priority, arguments, CPU usage and memory usage of the process.
`clu_top`	Performs `top n 1 b` on target nodes. It shows the current CPU usage, memory usage, and SWAP usage. The CPU usage will show the system, user and idle CPU space. The memory and SWAP usage will show the total memory, used and free memory.
`clu_kill`	Performs `kill` on target nodes. It allows the user to terminate a process with the specified process ID.
`clu_killall`	Performs `killall` on target nodes. It allows the user to terminate a series of processes.
`clu_iostat`	Performs `iostat` on target nodes. It allows the user to view the current CPU Usage of the user, system, nice as well as the remaining idle CPU space.
`clu_free`	Performs the `free` command on target nodes. It allows the user to view the current swap and memory usage. It displays both swap and memory total used and free space.

Table 3. Account Management Commands

Command	Description
`clu_useradd`	Allows the superuser to add another user.
`clu_userdel`	Allows the superuser to delete another user.
`clu_usermod`	Performs `usermod` on target nodes. It allows the user to modify information about his account.
`clu_userlist`	Performs `userlist` on target nodes. It allows the user to list all user accounts present in the node. The `userlist` utility is not built-in and it is installed with the system.
`clu_users`	Performs `users` on target nodes. It allows the user to list logged user accounts in the node.
`clu_groupadd`	Performs `groupadd` on target nodes. It allows the superuser to add a group.
`clu_groupdel`	This utility runs `groupdel` on target nodes. It allows the superuser to delete a group.
`clu_groupmod`	This utility runs `groupmod` on target nodes. It allows the superuser to modify information about groups.
`clu_grouplist`	This utility runs `grouplist` on target nodes. It allows the user to list all groups present in the node. The `grouplist` utility is not built-in and it is installed with the system.
`clu_groups`	This utility runs `groups` on target nodes. It allows the user to list the groups the user is a member of.

3. SYSTEM ARCHITECTURE

Figure 1 shows the architecture of CluMSy. The cluster consists of a number of independent homogenous Linux nodes interconnected by a network. These nodes can be logically considered as a single-unified resource due to the CluMSy middleware.

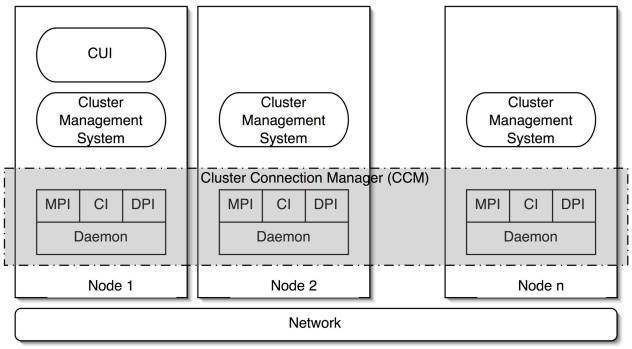

Figure 1. System Architecture

The Cluster Connection Manager (CCM) is the base layer that provides the parallel programming environment used by the upper layers. The Cluster Management System (CMS) is the set of cluster-enabled utilities that, in turn, provides the file, process, accounts, and package management and monitoring. The CluMSy User Interface (CUI) provides the user with a point-and-click interface and issues corresponding commands to the CMS.

Every node connected in the cluster or unified resource will run the CluMSy daemon which performs the monitoring and management functions in each node.

3.1 Cluster Connection Manager (CCM)

The Cluster Connection Manager (CCM) is all the necessary low-level facilities like communication and node membership management. The CCM makes use of XML messages as communication protocols. It is comprised of three (3) Application Programming Interfaces (APIs) and a daemon.

XML. Extensible Markup Language (XML) messages are plain-text documents that can be converted into XML tree. XML was used because of the provided extensibility of XML. As long as the XML document follows the basic XML message formats needed by the CCM, it is possible to add additional information without major modifications to internal data structures. Also, the ability to convert the XML tree back to XML documents makes it easier to use TCP to send XML messages over the network. It also makes it easier to make those connections secure.

Secured Communication. The CCM uses TCP sockets in order to communicate and ensure reliability of connections. The CCM also encrypts messages before sending it over the network as a security measure. This is to discourage users with malicious intent from easily obtaining any information from the CCM messages. It is also a necessity since the user password is included in the distributed processing messages.

Functional Requirements. Similar to MPI and PVM, the CCM requires the needed libraries and programs and the CCM daemon to be running in order to function. It does not require users to create a specific account for use of the parallel programming environment. Developers that aspire to use CCM only need to include the CCM header file *ccm.h* and link the static CCM library *libccm.a*. After linking, developers can now use any of the API (Common Interface, Message Passing Interface, and Distributed Processing Interface) that the CCM provides. Each of these interfaces contains functions that instruct the CCM daemon to do specific tasks and return the result of the task.

3.1.1 CCM Daemon

This provides the core CCM services such as node initialization, handling of XML messages, handling broadcasts, remote execution, resource tracking, multicasting, and other utility functions.

3.1.2 CCM Common Interface

The Common Interface (CI) is a set of functions that creates the appropriate CCM message for manipulating the cluster groups, node list and client list on the local daemon. More specifically, the common interface functions allows users to clear, set or retrieve a cluster group; to clear or retrieve the current online node list; and to retrieve the client list.

3.1.3 CCM Message Passing Interface

This covers a set of functions for node registration (subscription) and message passing. It uses the multicast facility provided by the daemon for sending messages to a collection of receivers.

3.1.4 CCM Distributed Programming Interface

The Distributed Processing Interface (DPI) deals with remote host operations, providing functions for remote execution or remote copying. It communicates with the daemon via port 2016 for issuing commands like `remove_execute`, `local_execute`, etc.

3.2 Cluster Management System (CMS)

The Cluster Management System (CMS) is a set of cluster-enabled extensions of the Linux built-in utilities. The CMS uses the remote execution DPI of CCM to allow the Linux built-in commands to be executed in parallel. Afterwards, the consolidated XML output generated by the DPI is processed to be similar as much as possible to the output of the Linux built-in commands to facilitate familiarity with Linux users.

Each utility that makes up the CMS includes an option parser to determine the cluster-enabled utilities options from the built-in UNIX commands. The option parser stores the targets specified by `-target` in a string called `sTargets`. The optional username specified by `-user` is stored in a string called `sUsername`. The option `-prefix` sets an integer `isPrefix` to 1. The optional `-dir` option for file utilities is stored in the string called `sCWD`. All other options are appended to the `sOptions` string and non-options are appended to the `sArguments` string. The variables used to store the different values of global variables of the specific utility. These variables are later used in the general implementation of the CMS.

General Implementation. The CMS is implemented using the Distributed Processing Interface (DPI) of the Cluster Connection Manager (CCM). The parameters of the remote execute DPI is supplied in the `main()` function of each utility. The password parameter is retrieved using the `clu_getpassword()` function provided by the `clu_login.h` header file. The command to be executed is the built-in LINUX command with `sOptions` and `sArguments` appended to it. The array of hosts is tokenized from the string `sTargets`. Once the parameters are fixed, the remote execute DPI is called and returns an XML reply. Inside the XML reply is the names of the output files stored in the `/tmp` directory. These temporary files contain the consolidated output of the different nodes and should be removed after use. Each file is opened and printed to `stdout` to display the output of the command. If `isPrefix` is equal to 1, the alias of the current node is displayed with prefixed before each line of its output.

Exceptions. There are exceptions to the general implementation stated above.

The cluster login utility is not implemented using Distributed Processing Interface (DPI). It creates a CluMSy session file in the specified user's home directory. This session file contains the encrypted password to be used by the different cluster-enabled utilities.

The cluster file utilities have an extra command before the actual built-in UNIX command. The first command is a `cd` command changing the current working directory to the directory specified by `sCWD`. This is added so that CluMSy users can perform built-in UNIX file commands on other directories besides the user's home directory without specifying the full path to the file.

The cluster `rm` has an extra fixed option attached to it. It is fixed with the `-vf` option to make its output verbose and to prevent interactive input from the user.

The cluster `top` also has an extra fixed option attached to it. It is fixed with the `n 1 b` option to make it a non-interactive command.

3.3 CluMSy User Interface (CUI)

The CluMsy User Interface (CUI) is a sample GUI which uses the output of CluMSy utilities for input. Its design is based on the client-server model similar to the X-Windowing system in Linux. The CUI provides abstraction and configuration only on selected CluMSy utilities.

Unlike the other layers in the architecture, the CUI is implemented in Java instead of C. This is to demonstrate the possibility of creating a Graphical User Interface (GUI) based on the standard I/O streams of the Cluster Management System (CluMSy). The CUI was constructed mainly to abstract the users from the lengthy utility calls that need to be called from the backend and at the same time format and display the output generated by these calls.

When the CUI is opened, the CluMSy login screen would be displayed initially. After the user has entered his username and password, the `clu_login` command is called to check the user name and passwored keyed in. If the user has entered the correct user name and password the CluMSy main screen is displayed. Otherwise, and error message is displayed indicating that the username and password entered is incorrect.

Upon login, a tree that displays the current online nodes is shown. This tree is called the node explorer. To facilitate easier monitoring and management operations performed on these nodes, groups that consist of particular nodes can be assigned instead. This is helpful more specifically if the number of online nodes in the system is too large to handle. A file containing a listing of group names and nodes is stored and is parsed every time the user opens the CluMSy GUI.

When issuing management commands, a tree view containing the list of online nodes where the user has an account in the cluster is shown. This is quite similar to the node explorer mentioned above. The only difference is that from this tree, the users are allowed to choose the targets from this tree and CUI will produce the proper CluMSy utility will be called. Almost all of the outputs of management commands that are implemented in the CUI are displayed through dialog boxes that contain the result of the management utility calls.

The CUI uses the `Runtime.exec()` command in Java to access the CluMSy utilities. Once `Runtime.exec()` is executed, the system waits for the process to be completed. If the result of the command is successful, two situations may arise. First, a message may be displayed to convey that the operation has been successful in such cases where the verbose (`--verbose`) option is allowed; utilities such as `clu_cp` and `clu_kill` are examples of this format. If not, the resulting output of the query from the input stream will be parsed line by line. The parsed values will then be stored into hash tables and classes before being displayed in the CUI. If the operation generates and exit status thus indicating that

an error occurred, the error message from the input stream will be displayed in an alert / dialog box. The following basic algorithm demonstrates how the CluMSy utilities are called and how the I/O stream data is handled, stored and formatted. The file, process, account, and package utilities make use of this procedure in invoking its respective tasks.

We present some representative screen captures of the CUI below: First is the process explorer shown in Figure 2. It lists processes executing on the nodes, similar to the `top` command, except that it displays processes running on nodes of the clusters.

Next, we show the package explorer in Figure 3. It allows the user to ensure that each node has the same set of packages installed.

In Figure 4, we show the account explorer allowing a superuser to manage user accounts on each node.

4. PROTOTYPE EVALUATION

The prototype implementation of CluMSy was built and tested on a cluster of four (4) nodes connected via a local area network (LAN) - one node runs Red Hat Linux 8.0 and the others run Red Hat Linux 9.0. The specifications of the different nodes are listed in Table 4.

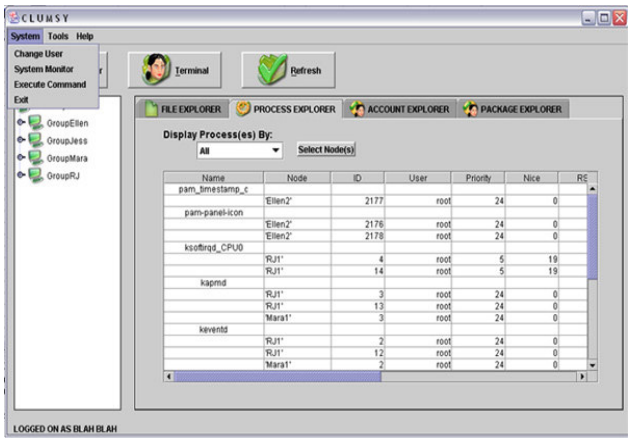

Figure 2. Process Explorer

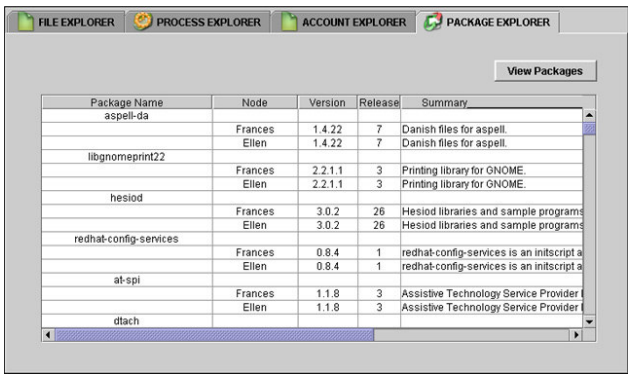

Figure 3. Package Explorer

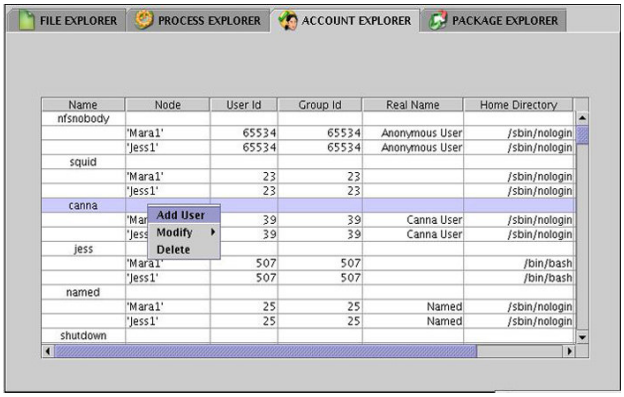

Figure 4. Account Explorer

First, we compared the elapsed times for running commands using CluMSy in parallel mode against the sequential mode. In the sequential mode of operation, we used the DPI execute command locally, thus, the result will reflect the same delay experienced by the DPI remote execute due to the login process. The results of the sequential execution will provide a greater reference point for measuring the scalability of the performance of the parallel commands.

In Figure 5, we see a difference between running the short commands using the parallel mode versus sequential mode. Short output commands include `clu_kill`, `clu_mkdir`, etc.

Table 4. Test Nodes Specification

Processor	Memory
Intel Pentium IV - 1.8 Ghz	256 DDRAM
Intel Pentium IV - 1.6 Ghz	512 DDRAM
Intel Pentium IV - 1.5 Ghz	512 DDRAM
Intel Pentium III - 1.0 Ghz	128 SDRAM

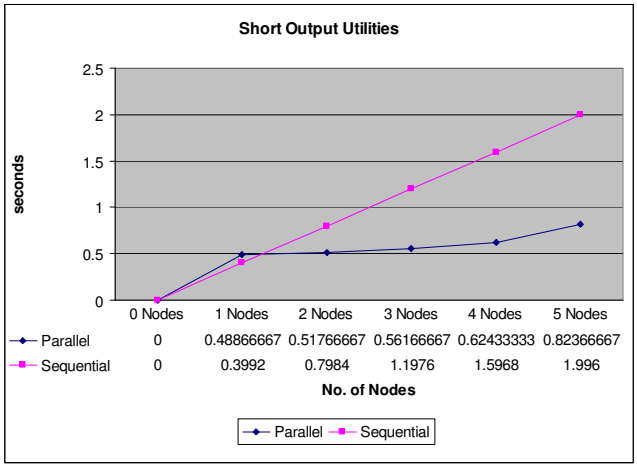

Figure 5. Short Output Commands

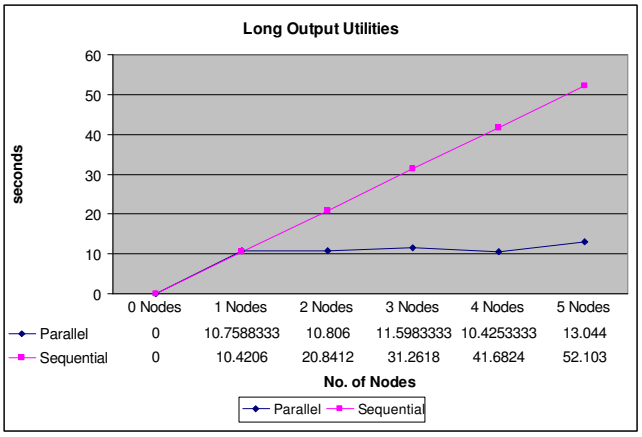

Figure 6. Long Output Commands

Finally, we present the results for remote copying a 300 MB file to all other nodes in Figure 7. In one case, we measured the times for copying a 300MB file from 1 source to all other target nodes. In the second case, we considered copying two 0MB files (one from a different source node) instead of one source for the file. Because of the extra traffic on the network, the 2-source case took additional time to complete.

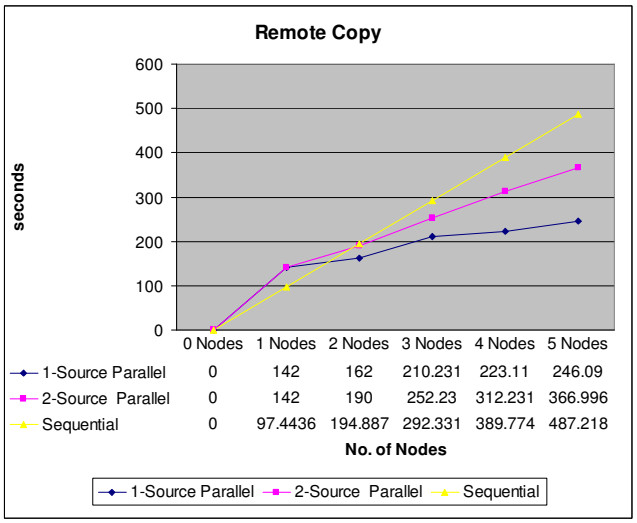

Figure 7. Remote Copy

5. Conclusions and Further Work

We have presented our prototype cluster management system called CluMSy. This system is aimed to provide a complete set of commands that operate at the level of a cluster. It is hoped that this will be a useful facility for management and monitoring of a cluster. We have also described the architecture and some design and implementation issues. Finally, we present some timing results obtained from performing commands on an experimental cluster.

Our goal is to continue developing CluMSy so that it will become a production environment facility. By providing the following features and functionalities: job monitoring, job termination, job execution, and resource monitoring and statistics through the CPU and memory monitoring, file and account monitoring and management, and package manager, we believe that CluMSy already satisfies the requirements of a cluster management tool.

Nevertheless, several aspects of CluMSy may be improved upon, namely: the GUI may be improved to provide a more user-friendly environment; optimizations may be used to improve the performance of global operations like remote copy; and considering the possibility of interoperability with MPI implementations and PVM.

References

[1] Brim, M., et. al. (2000). Cluster Command and Control (C3) Tool Suite [online]. Available: http://www.csm.ornl.gov/torc/C3/Papers/pdcp-v2.0.pdf. (February 9, 2003).

[2] Gropp, W. & Lusk, E. (1994). Scalable Unix Tools for Parallel Processors: A High-Performance Implementation. In Proceedings of the Scalable High Performance Computing Conference [online]. Available: www-fp.mcs.anl.gov/~lusk/papers/scalable/paper.html. (February 7, 2003).

[3] Kim, M., Choi, M. & Hong, J. (2002). A Load Cluster Management System (LCMS) using SNMP and Web.International Journal of Network Management, vol. 12 pp.367 - 378.

[4] Leopold, Claudia. Parallel and Distributed Computing A Survey of Models, Paradigms and Approaches. John Wiley & Sons, Inc., Canada, USA, 2001.

[5] Linux Networx ClusterWorx Management [online]. Available: http://www.linuxnetworx.com/products/clusterworx.ph. (July 2, 2003)

[6] Message Passing Interface Forum (1994). The MPI Standard.

[7] Papadopoulos, P, Katz, M. & Bruno, G. (2001). NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux. Available: http://www.rocksclusters.org/rocks-documentation/2.3/papers/clusters2001-rocks.pdf. (February 15, 2003).

[8] PVM [online]. Available: http://www.netlib.org/pvm3/. (July 5, 2003).

[9] PVM Tutorial [online]. Available: http://csep1.phy.ornl.gov/CSEP/PVM/NODE1.html#SECTION00010000000000000000. (July 5, 2003).

[10] Supercomputing: From Classics to Clusters [online]. Available: http://www.microsoft.com/windows2000/hpc/. (July 2, 2003)