CRONO: A Configurable and Easy to Maintain Resource Manager Optimized for Small and Mid-Size GNU/Linux Cluster⋆

Marco Aurélio Stelmar Netto
Research Center in High Performance Computing
CPAD-PUCRS/HP, Brazil
stelmar@cpad.pucrs.br

César A. F. De Rose
Catholic University of Rio Grande do Sul
Computer Science Department, Brazil
derose@inf.pucrs.br

Abstract

This paper presents the design and implementation of a new management system called CRONO aimed at small and mid-size GNU/Linux cluster installations owned by non-specialized users. CRONO implements only the basic management services needed to share a cluster among several users and is optimized for machines with up to 64 nodes, being therefore easy to install, maintain and use, while still being highly configurable. We also show how to configure CRONO for an environment with one and other with three clusters, as well as some maintenance procedures to give an idea of the simplicity of both tasks.

1. Introduction

Cluster architectures [5] are becoming a very attractive alternative when high performance is needed. With a very good cost/performance relation cluster systems are becoming more popular in universities, research labs and industries in different sizes and configurations. GNU/Linux is usually the operating system used in these systems because it's free, efficient and very stable.

But despite the cluster's number of nodes and their processing power one issue remains a problem for non-specialized users, the management of the machine. To get most of their investment it is common to connect the cluster to a local network and share the processing power among several users. This is accomplished by a tool called *resource manager* [14]. Among other tasks it has to deal with issues like access rights, node allocation and reservation, and queuing of requests.

There are already several resource managers available, like Portable Batch System (PBS) [2], Computing Center Software (CCS) [12, 13], Condor [11] and SLURM [15]. These systems support clusters with thousand of nodes and

⋆Research done in cooperation with HP Brazil.

implement a wide range of services, being therefore reasonably complex to install and configure, especially for non-specialized users.

In this context we decided to implement a new management system called CRONO aimed at non-specialized users with small cluster installations. CRONO implements only the basic management services and is optimized for installations up to 64 nodes, being therefore easy to install, maintain and use.

CRONO was introduced in a short paper [1] which presented its main functionalities, a brief description of the system architecture and its configuration files and a detailed description of the installation process. This paper is focused on how CRONO was implemented and what is done to simplify the system configuration and maintenance. It also presents some enhancements and optimizations since the last paper. The system now supports batch jobs and has an optimized scheduler and an increased flexibility in the pre and post processing scripts.

This paper is organized as follows: Section 2 presents some of the well-known resource managers and their main characteristics; Section 3 describes the CRONO functionalities and its architecture; Section 4 presents CRONO's configuration files demonstrating how easy is to install and maintain the system. Our conclusions and ongoing work are presented in Section 5.

2. Related Work

There are already several resource managers available [14]. The main purpose of these systems is to provide high throughput of the user requisitions on the system they manage. Furthermore, the resource managers address some problems like scalability, portability, fault tolerance, scheduling of the user requisitions, load balancing, security and many others. This of course has a big impact in their complexity. Among the resource managers available are PBS, CCS, Condor and SLURM.

The Portable Batch System (PBS) [2] is a flexible batch queuing system originally developed at NASA Ames Research Center at Moffett Field, California from 1993 to 1997. In late 1997, Veridian-MRJ began full PBS development, support and distribution. More recently, PBS became an enabling technology of NASA's Information Power Grid (IPG). PBS operates on networked, multi-platform UNIX environments, and uses the concept of multiple queues, where the jobs are queued on a server with one or more queues with different priorities and properties. PBS provides an API in the C, Tcl and BaSL programming languages and support for external scheduler, like Maui Scheduler [4]. For each node in PBS is possible to define a fixed or variable number of jobs that can share it, indicating the processor utilization that jobs can use. PBS has around 150 thousand lines of code spread over more than 430 files.

Computing Center Software (CCS) [12, 13] has been developed in the Paderborn Center For Parallel Computing (PC2), Germany, since 1992. CCS provides a hardware-independent scheduling of interactive and batch jobs through a Resource Description Language (RDL) for specifying resources, requests, and system components. Using this facility, the administrator describes the topology of the managed machine making possible mapping the request on the specified topology. Besides it, CCS has a high degree of reliability (*e.g.* automatic restart of crashed daemons) and fault tolerance in the case of network breakdowns. CCS has more than 100 thousand lines of code.

Condor [11] is a project that has been focusing on customers with large computing needs and environments with heterogeneous distributed resources. It has been developed at the University of Wisconsin-Madison (UW-Madison) since 1988. Condor explores pools of dynamic resources composed for machines that become idle at a given time. Due to the tendency of the large, distributed and dynamic environments, an important functionality is check pointing of the application executions. Condor has this mechanism to take a snapshot of the current state of a executing program which can be used to restart the program from that state at a later time. This is very useful, for example, when a node fails, the program running on that node can be restarted from its most recent checkpoint on another node and is also useful for giving a scheduler the freedom to reconsider scheduling decisions through preemptive-resume scheduling.

Simple Linux Utility for Resource Management (SLURM) [15] is a highly scalable cluster management tool that has been developed by Lawrence Livermore National Laboratory (LLNL) and Linux NetworX, USA. SLURM is an open source system aimed to manage Linux clusters of thousands of nodes. Besides the scalability, SLURM provides some simple fault tolerance and security mechanisms, partition and job management, framework for starting, executing and monitoring work. Its default scheduler implements First In First Out and it provides an API for adaptation of external schedulers. Although the SLURM's authors consider SLURM simple enough to allow end users to add functionalities, its source code has more than 50 thousand lines and it uses threads and sockets (concepts that may not be easy for end users to understand).

3. CRONO

Managing clusters is not an easy process, especially when it is done by non-specialized users. Installation involves the creation of many configuration files and the definition of several parameters depending on the execution environment. Any changes on the machine configuration, utilized tools or the inclusion or removal of end users will reflect in updates on the configurations files. The frequency of this system maintenance will depend on the number of shared machines, utilized tools and end users, but it's not seldom that updates become necessary at a daily basis.

The complexity of the management is of course directly related to the number of functionalities of the used resource manager. When complex management systems are used, the number of configuration files and parameters to be set increase and consequently the effort to install and maintain the system.

The most important design goals for CRONO were (1) to provide an open source manager system aimed at non-specialized users with small cluster(s) installations; (2) to provide a good level of configurability to manage different user profiles on an environment a small number of clusters; and (3) to provide a system with a reduced number of lines of source code which could be easily modified by a specialized user (a programmer) to solve some particular management problems. To address these design goals we reduced the complexity of the management avoiding the implementation of a resource manager with many services. Therefore we defined a set of basic functionalities, most of them based on the available resource managers [14].

To give an idea on how this simplifications affect the complexity of the system, CRONO has about 9000 lines of code divided in 30 files. There are 7 simple configuration files being 3 of them optional. It's interesting to highlight that as CRONO has a source code with a few number of lines (in relation to the available systems), a specialized user can easily add functionalities to solve some peculiar problem on the managed environment.

This section presents an overview of the functionalities provided by CRONO and a detailed description of all modules present in its architecture.

IEEE
COMPUTER
SOCIETY

3.1. Overview of the Functionalities

The scheduler is essential to achieve high throughput on a computational environment. It organizes the access of multiple requests to a given resource ensuring that all requests will be satisfied in the shortest possible time slot. The first versions of CRONO we provided a simple First In First Out (FIFO) scheduler. We identified, however, the need to improve the scheduler to increase resource utilization. There was the possibility to use the Maui Scheduler [7], but because its complexity we preferred to implement a simple backfilling algorithm, that will be explained in Section 3.5.2. Furthermore, CRONO doesn't provide a qualitative allocation yet, that is, the users don't have the possibility to choose which nodes of a cluster they want to use but just the number of nodes. Therefore the users with an heterogeneous cluster may not make the best use of their resources using CRONO.

Some common functionalities of a resource manager on an environment with thousands of nodes like load balancing, fault tolerance, scalability and monitoring were not implemented because these have little impact on small clusters. Other functionalities, like mechanisms to improve the job launching were also not considered. For example, STORM [8] implements mechanisms to speed up the launching of the jobs, instead of simply suppose the users will distribute their applications from the frontend nodes to the clusters nodes on an environment with a shared file system (like the Network File System - NFS).

The use of MPI [9] on parallel environments is very common. There are many MPI implementations for different network technologies. Therefore it is important to provide mechanisms to provide an easy way to allow the users to execute their applications on the cluster(s). To configure the execution environment, CRONO supplies scripts for pre and post processing of requisitions. When the user time initiates, CRONO will use two scripts: one of them controlled by the administrator, and the other by the user itself. This mechanism can be used, for example, to automatically generate MPI machine files. When the time of an user is over, two post processing scripts will be used in the same way. CRONO also provides scripts for compiling and executing the user applications. These scripts are useful on environments with, for example, several MPI implementations.

CRONO provides two basic allocation modes, space-sharing and time-sharing. The first one is used when the user needs exclusive access to allocated nodes, for example when application performance is being measured. The second one is used in situations where the users are only testing their programs and, therefore, do not care about performance. Time-sharing is a very interesting alternative in teaching environments, allowing large groups of students to use the same cluster partition at the same time.

Another main feature of CRONO is its flexibility to define access rights. Through configuration files the system administrator can create user profiles and associates them to user groups or to individual users. These access rights are defined by the maximum time and maximum amount of nodes used in allocations and reservations. There is also the possibility to define restrictions based on periods of the day, day of the week and target machine.

3.2. System Architecture

CRONO has been coded for the GNU/Linux operating system using the C language and rely on system commands and some optional scripts for compilation and execution of user applications. CRONO has an architecture composed by four modules, being the Node Manager module optional:

- The User Interface (UI) is composed by several system commands responsible for the user access and utilization of the cluster resources;

- The Access Manager (AM) is a daemon responsible for the authentication and the verification of the access rights;

- The Request Manager (RM) daemon does the scheduling of the user requests, the execution of batch jobs and the preparation of the execution environment;

- The Node Manager (NM) is the daemon running on each cluster node and its main function is to control the user access to the nodes.

The communication between the modules is done through the TCP sockets, therefore allowing modules to be in different machines. Moreover, the modules are organized like a chain, that is, if the User Interface needs to send a message to the Node Manager, the message will pass through the Access Manager and the Request Manager. The AM, RM and NM daemons may be executed asynchronously.

3.3. User Interface

The User Interface is the module responsible for providing interaction to the system, through the UNIX shell environment. It is composed by six system commands to access the other CRONO modules, two shell scripts to facilitate the use of programming environments (like MPI) and one simple program to set up the parameters of these system commands and shell scripts. The system commands and their respective functions are listed below:

- `crqview` to display information about the requisitions queue, like user names, starting and finishing

time, cluster name, number of nodes available and allocation modes (space-sharing or time-sharing);

- **cralloc** to allocate nodes, in the case the user wants the resources as soon as possible, or to submit batch jobs;

- **crrls** to release nodes or to cancel an user request;

- **crnodes** to obtain a list of nodes which the user has access;

- **crinfo** to display information about the clusters which the user has access like access rights, number of cluster nodes, special periods of use, maximum values for allocation and reservation, etc;

- **crnmc** to execute operations directly on the nodes that are allocated. The available operations are provided by the administrator, and the users may obtain which commands are available also with this command. An example of user operation could be **killp** to kill all processes running on the nodes;

- **crrun** and **crcomp** are scripts to help users in compiling and running programs, for example, in environments in which have more than one MPI implementation. The administrator can define, the path directories, machine files and other information and the users can choose the programming environment;

- **crsetdef** to define and set default parameters for system commands. If the user omit some parameters the system will look for default values in these variables.

The first six commands send data to the next module of the architecture, the Access Manager. These commands access a file, specified by the user or by the administrator, which has the hostname and the port of the Access Manager. The **crsetdef** system command just modifies the content of the configuration file at the user home directory.

3.4. Access Manager

The Access Manager is the module responsible for receiving the user requisitions from the User Interface and validate them, before forwarding them to the Request Manager. The Access Manager daemon can manage many clusters and may use distinct policies for each of them. Replication of the Access Manager is supported and may be interesting if fault tolerance is needed.

CRONO allows the system administrator to attribute access rights to individual users and groups of users. These user groups are not the same groups used by GNU/Linux. For request validation, the Access Manager uses three

files: the groups file, the users file and the access rights file. The first step to validate an user request is to check if the user belongs to a group. If this is the case, the group name is used to get the access rights. For users that do not belong to any group the user name is used. The `accessrights.users` file contains the relation of the access rights identifications with their respective user and user groups. If an user or a group is not specified in this file, the default access right is used by the system. With the access right identification, the Access Manager verifies whether the requisition is possible through the `accessrights.defs` file. In this file the policies for the cluster access control are defined. The administrator can define the maximum time and number of nodes for allocations and reservations. Furthermore, it's possible to define special periods for using the cluster, hence each access right can have two definitions, one for normal periods and the other one for special periods. For example, it's common to extend the time and number of nodes limits at weekend and at night, when there are fewer user requests.

After the Access Manager daemon checks these files, it forwards the request to the Request Manager, if necessary, or sends a message to the user informing that the user doesn't have access to the requested cluster. The connection to the solicited Request Manager is also checked.

3.5. Request Manager

The Request Manager is composed by two sub-modules: the execution manager, responsible for executing the batch jobs and preparing the execution environments, and the scheduler, responsible for scheduling the requests authorized by the Access Manager. The next subsections describe each one of the sub-modules in detail.

3.5.1. Execution Manager The Execution Manager is responsible for executing the batch jobs, the pre- and post-processing scripts and for advising the users when their time starts and finishes.

Four scripts are used to allow the execution of some tasks when the user time starts and finishes:

- The master pre-processing script (MPREPS) is used by the administrator and defines the operations that are executed when the user time starts;

- The master post-processing script (MPOSTPS) is used by the administrator and defines the operations that are executed when the user time finishes;

- The user pre-processing script (UPREPS) is used by the user and defines the operations that are executed when the user time starts;

- The user post-processing script (UPOSTPS) is used by the user and defines the operations are executed when the user time finishes.

Both the user and administration scripts are defined for each cluster managed by the system.

CRONO now supports batch jobs. With this mechanism is possible to submit user jobs automatically when the requested time starts. The resources will be released when the job or the user time finishes. A simple example of a batch jobs script is showed below:

```
#!/bin/sh
# Multiply matrix
#################################################
/usr/local/bin/crrun -c $CR_CLUSTER -np 16 mult_matrix
/usr/local/bin/crrun -c $CR_CLUSTER -np 32 mult_matrix
#################################################
```

Figure 1. Example of a batch job script.

Some important considerations about the UPREPS and batch job scripts are listed below:

- Both of them are implemented with a script that is executed when the user time starts;

- The user can use a UPREPS and a batch job script together;

- When the execution of a batch job script finishes the user time finishes too;

- When the execution of a UPREPS finishes the user time doesn't finish. For example, if an user requested 5 minutes and the UPREPS is executed in 1 minute, the user still has 4 minutes (in interactive mode).

To simplify the usage of the pre- and pos-processing scripts and the batch job scripts, CRONO provides variables that may be used to discover for example the cluster name, user id, user name, requisition identifier and list of user nodes.

Besides these variables, the administrator can specify a file to be evaluated before the execution of the pre- and pos-processing scripts and batch job scripts. This file can contain, for example, some environment variables which are important for the correct execution of these scripts (like libraries and programs path directories).

After the execution of the MPREPS at the starting time of an allocation request, the Request Manager sends a message to the users through the `tty` terminal when the resources become available. Because users are usually accessing with more than one terminal, CRONO uses the `utmp` file (operating system file to discover users currently using the system) to discover the terminal with the least idle time, and sends the message to that terminal. This is done because there is a greater probability the user is reading that terminal.

3.5.2. Scheduler A scheduler is essential to achieve high throughput on a computational environment. It organizes the access of multiple requests to a given resource ensuring that all requests will be satisfied in the shortest possible time slot.

The CRONO scheduler attempts to make good use of available resources that would be wasted using the First In First Out algorithm, but without penalizing the users that are already waiting for resources. If an user expects to be attended at a specific time, this user will be attended in the worst case at that time. The scheduler is implemented using an enhanced First in First Out algorithm. Expected attendance times are minimized by first checking whether a request fits into a gap of the current state of the scheduler using a back-filling method. The Figure 2 illustrates this method, considering an user U4 requests 2 nodes for 20 minutes at 8:05. The scheduler checks that U4 will not interfere at the expected starting time of the user U2. Therefore, it allows the user U4 to be attended before U2. When an user releases the resources before the expected finishing time, all the requests are rescheduled.
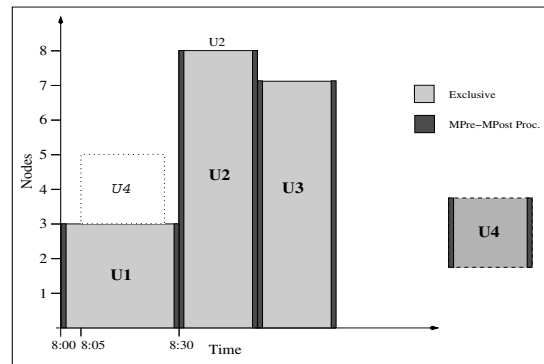

Figure 2. Back-filling method.

The block to be schedule is composed by the sum of the execution time of the MPREPS and MPOSTPS and the user time (Figure 3). The master scripts have timeouts to guarantee that a script defined by the administrator doesn't interfere on the expected user starting time.
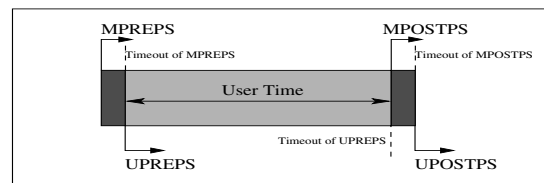

Figure 3. Block to be scheduled.

Each cluster must have its own Request Manager daemon. When the queue state changes (user time starts or

finishes, when a new user has requested nodes, for example), it is stored on a file specified by the `crono.conf` file (Section 4). This mechanism is useful when it becomes necessary to restart the daemon with the last queue state.

When the user time finishes all the processes on each user node are killed. This guarantees that the nodes are always ready for next utilization.

The scheduler supports space-sharing and time-sharing requests. Because CRONO doesn't support heterogeneous clusters yet, that is, all nodes are treated the same way, there is no sense to distinguish whether a node can execute more processes then other. For this reason, CRONO allows the specification of an unique number of users that can share a node at the same time, and this number is used by all the nodes of a cluster. PBS uses the concept of *virtual processors*, in which it's possible to define for each node the number of users that can share the same node at same time. SLURM also has support for exclusive and non-exclusive requests. However, it's not possible to define a limit of jobs that can be executed on a node, therefore, a high number of users on a shared node may negatively impact the performance of the jobs.

Another import feature of the scheduler is that it tries to satisfy a new time-sharing request overlapping it with other already allocated time-sharing requests, if the maximum number of requests per node is not exceeded. This procedure is done to increase the number of exclusive nodes that could be allocated by other users (Figure 4). The state 3.1 is the case when the scheduler uses this optimization, and in state 3.2 this optimization was not used.
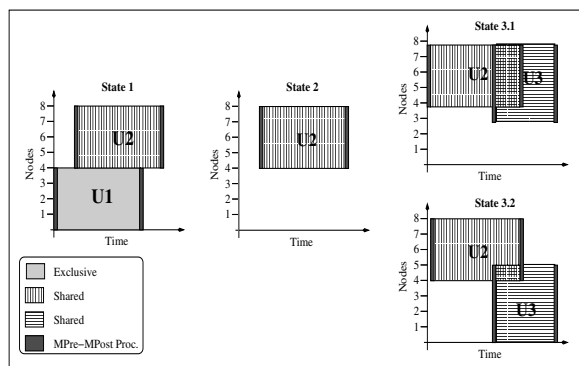

Figure 4. Scheduler - time-sharing requests placed together.

To make allocations the users have to provide: the cluster name, number of nodes, time of the allocations, shared or exclusive access, and a batch job script if this is the case. Because there are many parameters to allocate cluster nodes and to execute other CRONO commands, the `crsetdef`

command was implemented. With this command the user may define a set of default values to be used when command parameters are omitted. This is handy when several commands are issued in a row.

CRONO allows reservation nodes for a given time in the future. This is very useful when planning interactive sessions. When an user or administrator makes a reservation to a given time the scheduler checks immediately if the time slot is free and queuing is not possible. This doesn't happen when an allocation is requested, which the worst case is the request be placed in the last position of the queue.

3.6. Node Manager

Node Manager is the daemon executed on each node of the managed cluster and it is responsible for modifying the node operating system files used to control the user access and the execution of some operations on the node.

To guarantee that only allowed user can access a node, the Node Manager modifies GNU/Linux configuration files to accept or refuse the login of an user. These files are `login.access` (or `access.conf`) and `hosts.equiv`. Because some GNU/Linux distributions support PAM (Pluggable Authentication Modules) and some not, CRONO has an option to specify if PAM is available. If the PAM option is turned on, only the `login.access` is modified.

The administrator can define a set of operations which can be executed by the users through the `crnmc` command. The user can execute a operation on all allowed nodes or in a group of nodes. An example of such operation could be killing processes on the nodes, installing a network kernel module, or some other operation only allowed to the administrator.

3.6.1. Taking off the Node Manager The Node Manager has just two simple responsibilities. It was implemented to simplify the access control and the execution of operations on nodes.

Consider an environment in which a cluster has nodes with other operating system than GNU/Linux. Instead of porting the Node Manager and some procedures of the Request Manager, a simple solution is to run an Access Manager and a Request Manager on a machine with GNU/Linux (the frontend), turning on the option which specify that the Request Manager should not communicate with the Node Manager daemons, and create scripts the replace the Node Manager functionalities. For example, using the MPREPS and MPOSTPS, it's possible to run a remote shell to modify the access configuration files.

IEEE
COMPUTER
SOCIETY

4. System Configuration and Maintenance

In this section we show how simple it is to configure CRONO and to execute some maintenance procedures, like add new nodes, users, groups and user profiles. First we start with a simple configuration of a cluster, following by the configuration on an environment with multiple clusters and finally we show the execution of some maintenance procedures.

4.1. Configuring CRONO on an Environment with only one Cluster

A possible module disposition for managing only one cluster could be an Access Manager and a Request Manager on the frontend node and a Node Manager in each cluster node (Figure 5).
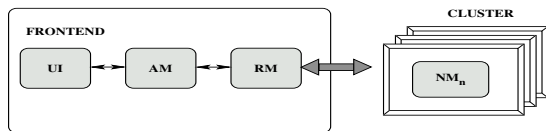

Figure 5. CRONO modules managing a cluster.

It is necessary to set up four files: `crono.conf`, `accessrights.defs`, `accessrights.users` and `nodes`.

- `crono.conf`: the main configuration file of the cluster. This file is used to specify the hostname where each daemon is executed, the listen port, the paths of the log, queue file, the number of users that share a node at the same time, PAM support and the timeout of connections between the modules. Besides, it's possible to define whether the Request Manager should ignore the function calls that communicate with the Node Manager. This is useful when the user wants to test CRONO without a real cluster or if the Node Manager daemons are not necessary;

- `accessrights.defs`: the access rights (profiles) definitions. The administrator can define the maximum time and number of nodes for allocations and reservations for special and normal periods;

- `accessrights.users`: the access rights to user and user groups;

- `nodes`: list of the node hostnames of the cluster.

Optionally, the `groups` file can be used to define the user groups:

There are other two files that can be configured, the `amconf` and `commands` files. The `amconf` contains the default hostname and port for the Access Manager and the `commands` contains the commands that the users can execute on the nodes through the `crnmc` command (see Section 3.3).

4.2. Using CRONO on an Environment with Several Clusters

CRONO can easily works on an environment with several clusters. A typical environment with several clusters is composed by a frontend node which the users can compile and run their applications and the clusters connected to this frontend. In this case, it is preferred to use only one Access Manager daemon, because the users don't have to worry about which Access Manager they should send the requisitions. For each one of the managed cluster it is necessary to have a Request Manager, and for each node of a cluster is optional the use of a Node Manager (as showed in Section 3.6.1). An example of an environment like this is our research lab (CPAD) [6] (Figure 6). Each cluster has its own peculiarities, having a different number of nodes (4, 16, and 32) and different interconnection networks (combinations of SCI [10], Fast-Ethernet and Myrinet [3]).
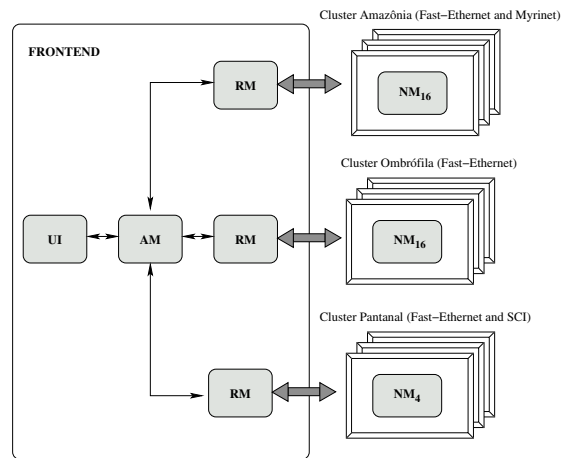

Figure 6. CRONO modules on an environment with several clusters.

The difference between configuring one cluster and several ones is that for each managed cluster should have a configuration directory with its own files. Some configuration files of the clusters can have the same content, therefore the administrator can use symbolic links to facilitate the actualization of the system, for example, to include a new user group.

4.3. Maintenance Procedures

Some maintenance procedures may be needed very often. Therefore, they should be very easy to execute. Listed below are some of these procedures with an explanation on how they are execute in CRONO:

- Add an execution environment: to add a new execution environment, like MPI, it is necessary to modify the `crrun` and `crcomp` scripts. In these scripts is possible to set up directories, environment variables, path of machine files and so on;

- Add an user: CRONO uses the user names of the operating system, therefore it is not necessary to create user names specially for CRONO. However, it is interesting to give an access right to the user or include the user in a group;

- Create a group: the groups are created by editing the `groups` file. It is not necessary to restart the Access Manager daemon;

- Create an user profile: the profiles (access rights) are created by editing the `accessrights.defs` file. It is not necessary to restart the Access Manager daemon;

- Add a node on a cluster: this can be done by editing the `nodes` file and restarting the Request Manager daemon responsible for that node.

5. Conclusion and Future Work

In this paper we have presented the design and implementation of the CRONO resource manager for cluster architectures. Since its beginning, CRONO has been aimed at small cluster installations (up to 64 nodes) and should be easy to install, use and maintain. To achieve this goal CRONO implements only the basic management services keeping a simple allocation interface while still being highly configurable.

We are using CRONO in production mode in our lab for the past year managing three small clusters with different configurations and the system is already very stable. Our environment has about 30 users and most of them are students. The students are allowed to make requests to use 4 nodes during 15 minutes at day and 8 nodes during 30 minutes at night and at weekends on the amazonia cluster. We also have special users with long jobs. They are allowed to make requests to the ombrofila cluster for during 2 weeks at any time. Other installations in partner research groups are also using the system and their feedback is being useful to update the system constantly with patches and new features. Future work includes some graphical interfaces for users and administrator, the possibility to include a cluster node

without restart the Request Manager daemon and support for 64 bit platforms (IPF). CRONO is open source (GNU license) and can be downloaded at www.sourceforge.net.

CRONO should not be seen as an replacement for the more powerful systems like CCS and PBS but as an alternative to them when smaller systems need to be shared and specialized human resources are not available to maintain the system up and running.

References

[1] M. A. Stelmar Netto and C. De Rose. CRONO: A Configurable Management System for Linux Clusters . *The Third LCI International Conference on Linux Clusters: The HPC Revolution 2002*, October 2002.

[2] A. Bayucan. Portable Batch System Administration Guide. *Veridian System*, August 2000.

[3] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 15(1):29–36, 1995.

[4] Brett Bode, David M. Halstead, Ricky Kendall, Zhou Lei and David Jackson. The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters. *Usenix Conference*, 2000.

[5] R. Buyya. *High Performance Cluster Computing*. Prentice-Hall, 1999.

[6] CPAD. Research Center in High Performance Computing. http://www.cpad.pucrs.br, 2001.

[7] David B. Jackson, Quinn Snell and Mark J. Clement. Core Algorithms of the Maui Scheduler. *7th International Workshop, JSSPP 2001*, 2001.

[8] Eitan Frachtenberg, Fabrizio Petrini, Juan Fernandez, Scott Pakin and Salvador Coll. STORM: Lightning-Fast Resource Management. *In Proceedings of the IEEE/ACM Conference on Supercomputing SC'02*, 2002.

[9] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, sep 1996.

[10] IEEE standart 1596-1992, New York. *IEEE: IEEE Standart for Scalable Coherent Interface (SCI)*, 1993.

[11] M. L. J. Basney and T. Tannenbaum. High throughput computing with Condor . *HPCU News*, June 1997.

[12] A. Keller and A. Reinefeld. CCS Resource Management in Networked HPC Systems . *IEEE Comp. Society Press*, pages 44–56, 1998.

[13] A. Keller and A. Reinefeld. Anatomy of a Resource Management System for HPC Clusters. *Annual Review of Scalable Computing*, 3, 2001.

[14] M. Baker and G. Fox and H. Yau. *Cluster Computing Review*. Northeast Parallel Architectures Center, Syracuse University, USA, 16 November,1995.

[15] M. Jette and M. Grondona. SLURM: Simple Linux Utility for Resource Management. *The Fourth LCI International Conference on Linux Clusters: The HPC Revolution 2003*, 2003.