Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

AXEL KELLER AND ALEXANDER REINEFELD

Anatomy of a Resource Management System for HPC Clusters

ZIB-Report 00-38 (November 2000)

Anatomy of a Resource Management System for HPC Clusters*

Axel Keller[1] and Alexander Reinefeld[2]

[1] Paderborn Center for Parallel Computing (PC[2])
[2] Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB)

kel@upb.de, ar@zib.de

Zusammenfassung

Workstation clusters are often not only used for high-throughput computing in time-sharing mode but also for running complex parallel jobs in space-sharing mode. This poses several difficulties to the resource management system, which must be able to reserve computing resources for exclusive use and also to determine an optimal process mapping for a given system topology.

On the basis of our *CCS* software, we describe the anatomy of a modern resource management system. Like Codine, Condor, and LSF, CCS provides mechanisms for the user-friendly system access and management of clusters. But unlike them, CCS is targeted at the effective support of space-sharing parallel computers and even metacomputers. Among other features, CCS provides a versatile resource description facility, topology-based process mapping, pluggable schedulers, and hooks to metacomputer management.

*To appear in: Annual Review of Scalable Computing, Vol. 3, 2001. Author for correspondence: Alexander Reinefeld, ar@zib.de

Inhaltsverzeichnis

1 Introduction **3**
- 1.1 CCS Overview . . . 3
- 1.2 CCS Target Platforms . . . 4
- 1.3 Scope and Organisation of this Paper . . . 4

2 CCS Architecture **5**
- 2.1 User Interface . . . 5
- 2.2 User Access Manager . . . 6
  - 2.2.1 Authentication . . . 6
  - 2.2.2 Authorization . . . 6
  - 2.2.3 Accounting . . . 7
- 2.3 Scheduling and Partitioning . . . 7
  - 2.3.1 Scheduling . . . 8
  - 2.3.2 Partitioning . . . 9
- 2.4 Access and Job Control . . . 10
- 2.5 Performance Issues . . . 12
- 2.6 Fault Tolerance . . . 12
  - 2.6.1 Virtual Terminal Concept . . . 12
  - 2.6.2 Alive Checks . . . 13
  - 2.6.3 Node Checking . . . 13
- 2.7 Modularity . . . 14
  - 2.7.1 Operator Shell . . . 14
  - 2.7.2 Worker Concept . . . 14
  - 2.7.3 Adapting to the Local Environment . . . 15
  - 2.7.4 Monitoring Tools . . . 16

3 Resource and Service Description **17**
- 3.1 Graphical Representation . . . 17
- 3.2 Textual Representation . . . 18
  - 3.2.1 Grammar . . . 18
- 3.3 Dynamic Attributes . . . 20
  - 3.3.1 Example . . . 20
- 3.4 Internal Data Representation . . . 22
- 3.5 RSD Tools in CCS . . . 22

4 Site Management **23**
- 4.1 Center Resource Manager (CRM) . . . 23
- 4.2 Center Information Server (CIS) . . . 25

5 Related Work **25**

6 Summary **26**

1 Introduction

A resource management system is a portal to the underlying computing resources. It allows users and administrators to access and manage various computing resources like processors, memory, network, and permanent storage. With the current trend towards heterogeneous grid computing [17], it is important to separate the resource management software from the concrete underlying hardware by introducing an abstraction layer between the hardware and the system management. This facilitates the management of distributed resources in grid computing environments as well as in local clusters with heterogeneous components.

In Beowulf clusters [34], where multiple sequential jobs are concurrently executed in high-throughput mode, the resource management task may be as simple as distributing the tasks among the compute nodes such that all processors are about equally loaded. When using clusters as dedicated high-performance computers, however, the resource management task is complicated by the fact that parallel applications should be mapped and scheduled according to their communication characteristics. Here, the efficient resource management becomes more important and also more visible to the user than the underlying operating system.

The resource management system is the first access point for launching an application. It is responsible for the management of all resources, including setup and cleanup of processes. The operating system comes only at runtime into play, when processes and communication facilities must be started.

As a consequence, resource management systems have evolved from the early queuing systems towards complex distributed environments for the management of clusters and high-performance computers. Many of them support space-sharing (=exclusive access) and time-sharing mode, some of them additionally provide hooks for WAN metacomputer access, thereby allowing to run distributed applications on a grid computing environment.

1.1 CCS Overview

On the basis of our *Computing Center Software CCS* [24, 30] we describe the anatomy of a modern resource management system. CCS has been designed for the user-friendly access and system administration of parallel high-performance computers and clusters. It supports a large number of hardware and software platforms and provides a homogeneous, vendor-independent user interface. For system administrators, CCS provides mechanisms for specifying, organizing and managing various high-performance systems that are operated in a computing service center.

CCS originates from the transputer world, where massively parallel systems with up to 1024 processors had to be managed [30] by a single resource management software. Later, the design has been changed to also support clusters and grid computing. With the fast innovation rate in hardware technology, we also saw the need to encapsulate the technical aspects and to provide a coherent interface to the user and the system administrator. Robustness, portability, extensibility, and the efficient support of space sharing systems, have been among the most important design criteria.

CCS is based on three elements:

- a hierarchical structure of autonomous “domains”, each of them managed by a dedicated CCS instance,
- a tool for specifying hardware and software components in a (grid-) computing environment,
- a site management layer which coordinates the local CCS domains and supports multi-site applications and grid computing.

Over the years, CCS has been re-implemented several times to improve its structure and the implementation. In its current version V4.03 it comprises about 120.000 lines of code. While this may sound like a lot of code, it is necessary because CCS is in itself a distributed software. Its functional units have been kept modular to allow easy adaptation to future environments.

1.2 CCS Target Platforms

CCS has been designed for a variety of hardware platforms ranging from massively parallel systems up to heterogeneous clusters. CCS runs either on a frontend node or on the target machine itself. The software is distributed in itself. It runs on a variety of UNIX platforms including AIX, IRIX, Linux, and Solaris.

CCS has been in daily use at the Paderborn computing center since almost a decade by now. It provides access to three parallel Parsytec computers, which are all operated in space-sharing mode: a 1024 processor transputer system with a 32x32 grid of T805 links, a 64 processor PowerPC 604 system with a fat grid topology, and a 48 node PowerPC system with a mesh of Clos topology of HIC (IEEE Std. 1355-1995) interconnects.

CCS is also used for the management of two PC clusters. Both have a fast SCI [21] network with a 2D torus topology. The larger of the two clusters is a Siemens *hpcLine* [22] with 192 Pentium II processors. It has all typical features of a dedicated high-performance computer and is therefore operated in multi-user space-sharing mode. The hpcLine is also embedded in the *EGrid* testbed [14] and is accessible via the *Globus* software toolkit [16].

For the purpose of this paper we have taken our SCI clusters just as an example to demonstrate some additional capabilities of CCS. Of course, CCS can also be used for managing any other Beowulf cluster with any kind of network like FE, GbE, Myrinet, or Infiniband.

1.3 Scope and Organisation of this Paper

CCS is used in this paper as an example to describe the concepts of modern resource management systems. The length of the paper reflects the complexity of such a software package: Each module is described from a user’s and an implementator’s point of view.

The content of this paper is as follows: In the remainder of this section, we present the architecture of a local *CCS Domain* and focus on scheduling and partitioning, access and job control, scalability, reliability, and modularity. In Section 3, we introduce

the second key component of CCS, the *Resource and Service Description (RSD)* facility. Section 4 presents site management tools of CCS. We conclude the paper with a review on related work and a brief summary.

2 CCS Architecture

A *CCS Domain* (Fig. 1) has six components, each containing several modules or daemons. They may be executed asynchronously on different hosts to improve the CCS response time.

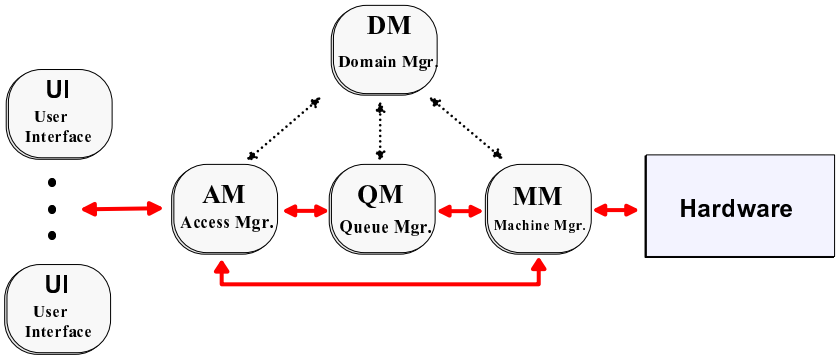

Abbildung 1: Interaction between the CCS components.

- The *User Interface (UI)* provides a single access point to one or more systems via an X-window or ASCII interface.
- The *Access Manager (AM)* manages the user interfaces and is responsible for authentication, authorization, and accounting.
- The *Queue Manager (QM)* schedules the user requests onto the machine.
- The *Machine Manager (MM)* provides an interface to machine specific features like system partitioning, job controlling, etc.
- The *Domain Manager (DM)* provides name services and watchdog facilities to keep the domain in a stable condition.
- The *Operator Shell (OS)* is the main interface for system administrators to control CCS, e.g. by connecting to the system daemons (Figure 2).

2.1 User Interface

CCS has no separate command window. Instead, the user commands are integrated in a standard UNIX shell like tcsh or bash. Hence, all common UNIX mechanisms for I/O re-direction, piping and shell scripts can be used. All job control signals (ctl-z, ctl-c, ...) are supported and forwarded to the application. The CCS user interface supports six commands:

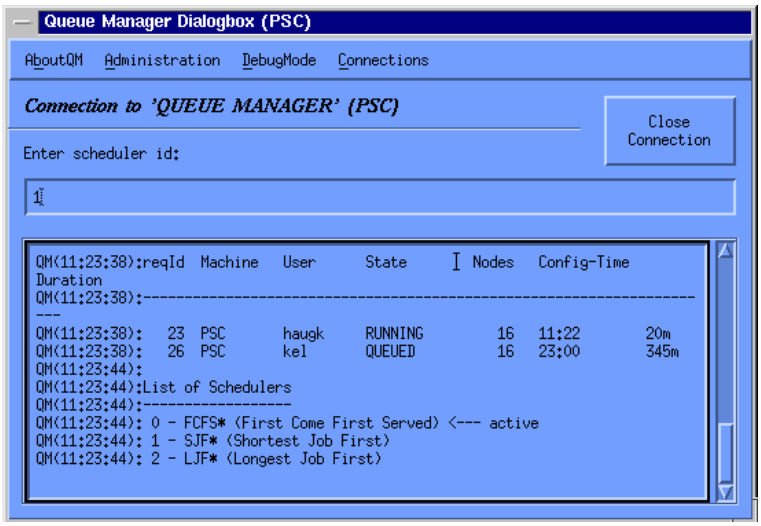

Abbildung 2: The CCS operator shell (OS).

- *ccsalloc* for allocating or reserving resources,
- *ccsrun* for starting jobs on previously reserved resources,
- *ccskill* for killing jobs and for resetting or releasing resources,
- *ccsbind* for re-connecting to a lost interactive application,
- *ccsinfo* for displaying information on the job schedule, users, job status etc.
- *ccschres* for changing resources of already submitted requests.

2.2 User Access Manager

The *Access Manager (AM)* manages the user interfaces, analyzes the user requests, and is responsible for authentication, authorization, and accounting.

2.2.1 Authentication

When first connecting to CCS, users must specify a project name. Users may work in several projects at the same time. In addition to the user input, the UI also determines user specific information (real and effective username, UNIX user-ID) and sends it to the AM. The AM checks its database whether the user is a member of the specified project. If not, the connection request is denied. Otherwise, the AM checks whether the request matches the users' privileges.

2.2.2 Authorization

Privileges can be granted to either a complete project or to specific members, for example:

- access rights (allocate-/reserve/change resource requests),
- maximum number of concurrently used resources,
- allowed time of usage (day, night, weekend, etc.).

Jobs may be manipulated by either the owner or project group members (if not prohibited by a `privacy` flag). This allows to establish working groups.

CCS has three time categories: weekdays, weekend, and special days. The latter can be defined by a calendar file. In each category CCS distinguishes day and night. Three limits can be specified: (1) the number of allocatable nodes in percent of the machine size during day and night time, (2) the maximum duration a partition may be used, and (3) the maximum time extension which can be added after job submission.

These parameters are used to compute an additional (implicit) limit, the "resource integral" (similar to an area in Fig. 3). The resource integral is given by $maxNodes * maxDuration * factor$, where *maxNodes* is the maximum number of nodes over all project limits. Normally, $factor$ is set to 1, but for projects with many members it should be increased.

2.2.3 Accounting

The accounting of the machine usage is done at partition allocation time, at release time and during job runtime. The resulting data can be post-processed by statistical tools. The operator can assign CPU time limits to a project. When the time (wall clock time $\times$ nodes) has been overdrawn, CCS locks the project. All above is specified in an ASCII file which is read at boot time by the AM. The AM periodically checks this file for changes, additionally the system operator may force the AM via the operator shell to read the file.

We currently use an SQL database with a Java interface for administering projects and for creating the ASCII configuration file.

2.3 Scheduling and Partitioning

In the design of CCS, we have tried to compromise between two conflicting goals: First, to create a system that optimally utilizes processors and interconnects, and second, to keep a high degree of system independence for improved portability. These two goals could only be achieved by splitting the scheduling process into two instances, a hardware-dependent and a hardware-independent part.

The *Queue Manager (QM)* is the hardware-independent part. It has no information on any mapping constraints such as the network topology or the amount or location of I/O nodes.

The hardware-dependent tasks are performed by the *Machine Manager (MM)*. It verifies whether a schedule received from the QM can be mapped onto the hardware at the specified time. The MM checks this by mapping the user specification with the static (e.g., topology) and dynamic (e.g., PE availability) information on the system resources. This kind of information is described by means of the Resource and Service Description facility RSD (Sec. 3).

In the following sections we discuss the scheduling model in more detail.

2.3.1 Scheduling

CCS requires the user to specify the expected finishing time of his jobs in the resource requests. This is necessary to compute a fair and deterministic schedule. The CCS schedulers distinguish between fixed and time-variable resource requests. A request that has been reserved for a given time interval is fixed, i.e. it cannot be shifted on the time axis. Time-variable requests, in contrast, can be scheduled earlier but not later than requested. Such a shift on the time axis might occur when other users release their resources before the specified estimated finishing time. Figure 3 shows the scheduler GUI. Note that both, batch and interactive requests are processed in the same scheduler queue.

CCS provides several scheduling strategies: first-come-first-serve, shortest-job-first, or longest-job-first. The integration of new schedulers is easy, because the Queue Manager provides an API to plug in new modules. The system administrator can choose a scheduler at runtime. In addition, the QM may adjust to different request profiles by dynamically switching between the schedulers.

For our request profile, we found that an enhanced first-come-first-serve (FCFS) scheduler fits best. Waiting times are minimized by first checking whether a new request fits into a gap of the current schedule (back-filling).

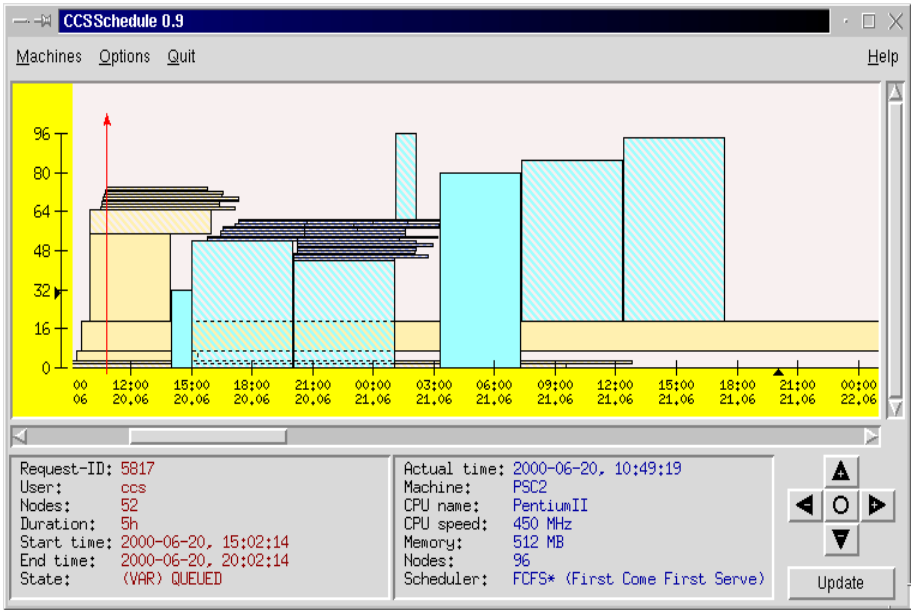

Abbildung 3: Scheduler GUI showing allocated compute nodes over time.

Resource Reservation. With this scheduling model it is also possible to reserve resources for a given time in the future. This is a convenient feature when planning interactive sessions or online-events. As an example, consider a user who wants to run an application on 32 nodes from 9 to 11 am at 13.02.2001. The resource allocation is done with the command: `ccsalloc -n 32 -s 9:13.02.01 -t 2h`.

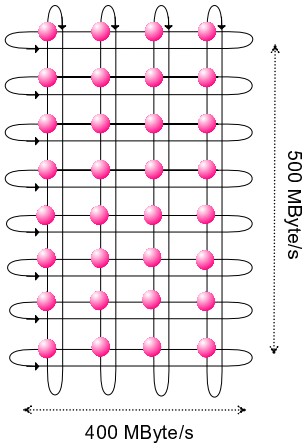

Abbildung 4: Example illustrating the system partitioning scheme on a machine with different link speeds.

Deadline Scheduling. With the deadline scheduling policy, CCS guarantees a batch job to be completed not later than the specified finishing time. A typical scenario for deadline scheduling would be an overnight run that must be finished when the user comes back into his office next morning. Deadline scheduling gives CCS the flexibility to improve the system utilization by scheduling batch jobs at the latest possible time by which the deadline is still met.

More sophisticated policies, like running a batch job as soon as possible but at latest to meet the specified finishing time, can be introduced via the scheduler API.

Balancing Interactive and Batch Jobs. Systems that are primarily used in interactive mode must not be overloaded with batch jobs during daytime. This is taken care of by distinguishing six time slots in CCS: three categories of working days (weekdays, weekend days, exceptional days) and two time categories (day and night time). These six slots are taken into account when scheduling a new request.

With the given project limits (CPU time and nodes in percent of the total machine size) the scheduler is able to schedule the request to the next suitable time slot depending on the request type (interactive, reservation, deadline, etc.). If the machine size or the limits should change later on, the scheduler is notified and it automatically adapts to the new scenario.

With this mechanism, a user may submit an arbitrary number of jobs without blocking the machine. In a certain sense, this mechanism gives each project its own virtual machine of time-dependable size (e.g., 40% during day time and 100% during night time).

2.3.2 Partitioning

The separation between the hardware-independent QM and the system-specific MM allows to encapsulate system-specific mapping heuristics in separate modules. With this approach, system-specific requests for I/O-nodes, specific partition topologies, or memory constraints can be taken into consideration in the verifying process.

The MM verifies whether a schedule received from the QM can be mapped onto the hardware at the specified time. In this check, the MM takes also the concurrent usage of other applications into account. If the schedule cannot be mapped onto the machine, the MM returns an alternative schedule to the QM. The QM either accepts the schedule or it proposes a new one.

Figure 4 is one such example, where system specific characteristics must be taken into consideration in the partitioning process. Here, the 32 compute nodes are interconnected by SCI links of different speeds: the vertical rings have a bandwidth of 500 MByte/s whereas the horizontal rings are a little bit slower with 400 MByte/s due to their longer physical cable lengths. The system specific MM takes such system characteristics into account. In this case, it determines a partition according to the cost functions "minimum network interference by other applications" and "best network bandwidth for the given application". The first function tries to use as few rings as possible (thereby minimizing the number of dimension changes from X- to Y-ringlets), while the second tries to map applications on single rings to ensure maximum bandwidth. Note that the best choice of policy also depends on the network protocol, e.g., in our case SCI [21].

The API of the MM allows to implement mapping modules that are optimally tailored to the specific hardware properties and network topology. As a side effect of this model, it is also possible to *migrate partitions* when they are not active. This feature is used to improve the utilization on partitionable systems. The user does not notice the migration (unless he runs time-critical benchmarks for testing the communication speed of the interconnects—but in this case the automatic migration facility may be switched off).

Another task of the MM is to monitor the utilization of the partitions. If a partition is not used for a certain amount of time, the MM releases the partition and notifies the user via email.

Interface to System Area Networks (SAN). In modern workstation clusters, the SAN administration is often done by an autonomous layer. The Siemens hpcLine provides a SAN administration layer which monitors the SCI network and responds to problems by disabling SCI links or by changing the routing scheme. Ideally, such changes should be reported to the resource management system, because the routing affects the quality of the mapping. However, the hardware vendors do not yet support the interface, so we have not been able to implement it into CCS. Currently, CCS just checks the integrity of the SAN at boot time.

2.4 Access and Job Control

When configuring the compute nodes for the execution of a user application, the QM sends the resource request to the MM. The MM then initializes the compute nodes, loads and starts the application code and releases the resources after the last run.

Because the MM also verifies the schedule, which is a polynomial time problem, a single MM daemon might become a computational bottleneck. We therefore splitted the MM into two parts (Fig. 5), one for the *machine administration* and one for the *job execution.* Each part contains several modules and daemons, which may be executed on different hosts to improve the response time.

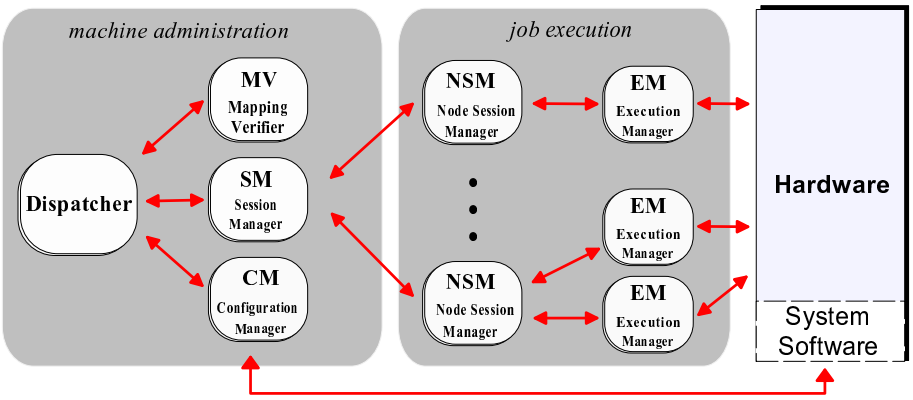

Abbildung 5: Detailed view of the machine manager (MM).

The *machine administration* part consists of three separate daemons (MV, SM, CM) that execute asynchronously. A small *Dispatcher* coordinates them.

The *Mapping Verifier (MV)* checks whether the schedule given by the QM can be realized at the specified time with the specified resources (see 2.3.2).

The *Configuration Manager (CM)* provides the interface to the underlying hardware. It is responsible for booting, partitioning, and shutting down the operating system (resp. middleware) at the target system. Depending on the system's capabilities, the CM may gather consecutive requests and re-organize or combine them for improving the throughput—analogously to the autonomous optimization of hard disk controllers. In addition, the CM provides external tools with information on the allocated partition, like host names or the partition size. With this information, external tools can create host files, for example to start a PVM application (see 2.7.2).

The *Session Manager (SM)* interfaces the job execution level. It sets up the session, including application-specific pre- or post-processing, and it maintains information on the status of the applications.

It also synchronizes the nodes with the help of the *Node Session Manager (NSM)*, that runs on each specified node with root privileges. The NSM is responsible for node access and job controlling. At allocation time, the NSM starts an *Execution Manager (EM)* which establishes the user environment (UID, shell settings, environment variables, etc.) and starts the application.

Before releasing the partition, the NSM cleans up the node. All NSMs are synchronized by the SM. Figure 6 illustrates the control and data flow in a CCS domain.

In clusters, access control is of prime importance, because each node runs a full fledged operating system and therefore could in principle be used as a stand alone computer by its owner. When operated in cluster mode, users must be prohibited to start processes on single nodes, not only because of unpredictable changes in the CPU load, but also because they might create orphan processes which are difficult to clean up by the resource management system.

For this reason, the NSM also takes care about the access- and job control. At allocation time, an NSM modifies several system files (depending on the operating system) to allow exclusive login for the temporary node owner. Before releasing the partition, the NSM removes pending processes and files and locks the node again to

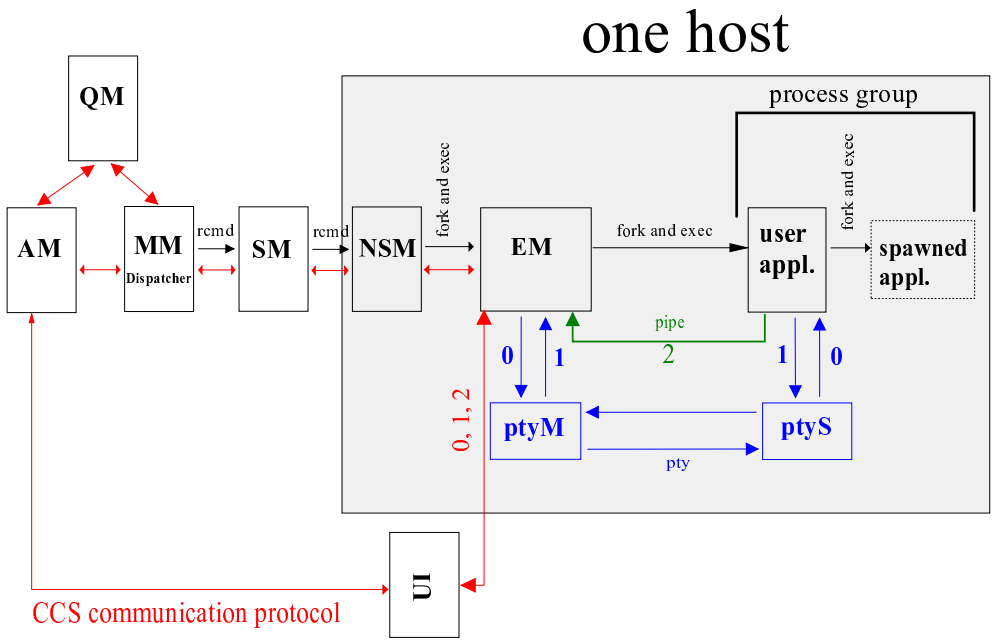

Abbildung 6: Control flow and data flow in a CCS domain.

prohibit direct access by single users.

2.5 Performance Issues

On very large clusters the SM may become a bottleneck, because it has to communicate with all NSMs. Configuring a partition with n nodes needs $2n$ messages to be sent through one communication link. In our simulations, the initialization of a 256 node partition took about five seconds, most of it spent in communication. This is because we use RPCs (for portability) which implicitly serializes the communication. The time can be reduced by implementing a hierarchy of SMs that communicate in parallel (i.e. a broadcast tree). In addition, transient problems (e.g. network congestion or crashed daemons) are handled by the nearest SM without interrogating the MM. The depth and width of the SM tree can be specified with the Resource and Service Description language (Sec. 3).

2.6 Fault Tolerance

2.6.1 Virtual Terminal Concept

With the increasing interactive use via WANs, reliable remote access services become more and more important. Unpredictable network behavior and even temporary breakdowns should ideally be hidden from the user.

When the network breaks down, open standard output streams (`stdout` and `stderr`) are buffered by the EM. The EM then either sends the output via email to the user or it writes it into a file (see Fig. 6). A user can re-bind to a lost session, provided that the application is still running. CCS guarantees that no data is lost in the meantime.

2.6.2 Alive Checks

Today's parallel systems often comprise independent (workstation-like) nodes, which are more vulnerable to breakdowns than traditional HPC systems. System reliability becomes an even more important issue because a node breakdown has an immediate impact on the user's work flow. To ensure a high reliability, erroneous nodes must be detected and disabled.

The resource management system must be able to detect and possibly repair breakdowns at three different levels: the computing nodes, the daemons, and the communication network. Many failures become only apparent when the communication behavior changes over time or when a communication partner does not answer at all.

To cope with this, the *Domain Manager (DM)* maintains a database on the status of all system components in the domain. Each CCS daemon notifies the DM when starting up or closing down. The DM periodically `pings` all connected daemons to check if they are still alive. In addition, when a CCS daemon detects a breakdown while communicating with another daemon (by receiving an error from the CCS communication layer), it closes the connection to this daemon and requests the DM to re-establish the link.

In both cases, the DM first tries to reconnect. If this is not possible, the DM has a number of methods to investigate the problem. Which of them to use depends on the type of the target system. If it is a cluster, the DM tries to `ping` the faulty node. If the node does not answer, it may be in trouble and is marked as faulty in the MM mapping module. The MM then ignores this node in future mappings. The respective jobs are stopped (if specified) and a problem report is sent to the user and the operator.

The DM is authorized to stop erroneous daemons, to restart crashed ones, and to migrate daemons to other hosts in case of system overloads or crashes. For this purpose, the DM maintains an address translation table that matches symbolic names to physical network addresses (e.g. host ID and port number). Symbolic names are given by the triple <`site, domain, process`>. With this feature a cluster may be logically divided into several CCS domains, each of them with a different scheduling or mapping scheme.

For recovery, each CCS daemon periodically saves its state to a disk. At boot time the daemons read their information and synchronize with their communication partners. This allows to shutdown or kill CCS daemons (or even the whole domain) at any given time without the risk to loose scheduled requests.

2.6.3 Node Checking

In space sharing mode users want to access "clean" nodes to achieve a deterministic behavior of their applications. Therefore, both the node and the network have to be cleaned up after terminating a job. In CCS, this is done in two steps.

Before allocating a node, CCS checks its integrity: (1) Is the SCI network interface okay? (2) Is enough memory available? (3) Has the node be cleaned of unauthorized processes? If any of these conditions is not true, CCS tries to fix the problem. If this is not possible, the allocation fails, the user is informed and the operator gets an email describing what went wrong. Additionally, CCS performs a post-processing after each job. For example after running a ScaMPI [31] application on an SCI cluster,

CCS removes remaining shared memory segments. This ensures that all memory is available for the next run.

2.7 Modularity

One of the design goals of CCS is extensibility. We have designed CCS in a modular way, to allow easy adaptation to new system architectures or different execution modes. In the following, we give some examples how this design principle is reflected in CCS.

2.7.1 Operator Shell

The *Operator Shell (OS)* and the other kernel modules (AM, QM, MM) are all independent modules. When the OS contacts a daemon for the first time, the daemon sends its menu items to the OS which automatically generates the corresponding pulldown menues and dialog boxes. This allows to specify any menue items and dialogs on the client side without changing the OS source code.

2.7.2 Worker Concept

Cluster systems comprise nodes with full operating system capabilities and software packages like debuggers, performance analyzers, numerical libraries, and runtime environments. Often these software packages require specific pre- and post-processing. CCS supports this with the so-called *worker concept.* Workers are tools to start jobs under specific runtime environments. They hide specific procedures (e.g. starting of a daemon or setting of environment variables) and provide a convenient way to start and control programs. A worker's behavior is specified by five attributes in a configuration file (Fig. 7)

- the name of the worker,
- the command for CCS to start the job,
- the optional parse command for detecting syntax errors,
- the optional pre-processing command (e.g. initializing a parallel file system), and
- the optional post-processing command (e.g. closing a parallel file system).

Pre- and post-processing can be started with either root or user privileges, controlled by a keyword. The configuration file is parsed by the user interface and can therefore be changed at run time. New workers can be plugged in without the need to change the CCS source code. It is possible to use several configuration files at the same time.

The *pvm-worker* in Fig. 7 may serve as an example to illustrate what can be done with a worker: The pvm-worker creates a PVM host file (the host names are provided by the CM) and starts the master-pvmd. The master-pvmd starts, according to the given host file, all other slave-pvmds via the normal `rsh` or `ssh` mechanism to establish the virtual machine (VM) on the requested partition. This is possible since the NSMs have

```
pvm,                                              #name of the worker
%CCS/bin/start_pvmJob -d -r %reqID -m %domain,    #run command
%CCS/bin/start_pvmJob -q -m %domain,              #parse command
%root %CCS/bin/establishPVM %user,                #pre-processing
%user %CCS/bin/cleanPVM %user                     #post-processing
```

Abbildung 7: Worker definition for job startup in a PVM environment.

granted node access at allocation time. Since the user application cannot be started until all pvmds are running, the pvm-worker starts a special PVM application when the master-pvmd is running. This little program periodically checks how many nodes are connected to the VM until the entire VM is up. Thereafter the user application is started. When the application is done, the worker terminates the master-pvmd which shuts down the VM. In the post processing phase the nodes are cleaned up by removing pending processes and files.

2.7.3 Adapting to the Local Environment

Since CCS is able to manage heterogeneous systems it is possible that the process environment may not always be the same. The automounter tool which automatically mounts file systems (e.g. home directories from remote file serves) may serve as an example to exemplify this problem:

Provided a user has a home directory on host x in directory /home/foo, a pwd would result in /homes/x/foo. However, a pwd submitted on host x itself results in /home/foo. If a user starts a job in /home/foo on host x, the EM is not able to change to this directory. Due to the automounter naming conventions the path should be /homes/x/foo. Therefore applications using this path will not work correctly.

CCS copes with problems like this by modifying the process environment of an application before starting it. Environment variables like `PATH` will be explored and modified with respect to the host name.

```
#EM-Host     SOURCE   DESTINATION
#=======================================
.*           /home1   /homes1/psc-master
.sci-[123]*  /home3   /homes3/psc-master
```

Abbildung 8: CCS configuration file with path mapping.

The operator can specify the mapping in a configuration file as shown in Fig. 8. Each line describes a mapping. The first column (which may be a regular expression) specifies the local host. The second and third column describe what will be mapped. For example, the string “/home1/foo” will be replaced by “homes1/psc-master/foo”.

The configuration will be read by the EM at boot time, hence it can be changed during runtime.

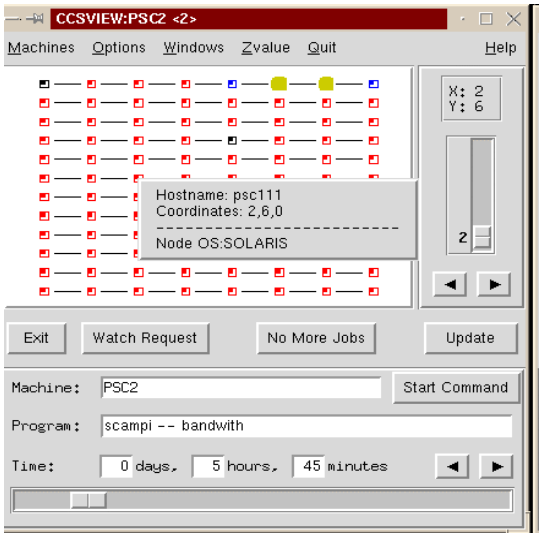

Abbildung 9: Monitoring a partition with ccsView.

2.7.4 Monitoring Tools

A resource management system should provide interfaces to arbitrary external tools to support users with several layers of information. Common users should be given the overall status without burdening them with too many technical details. Application developers, in contrast, need better monitoring tools to be able to see what happens on the nodes and on the network.

CCS offers three monitoring tools. The first one, *ccsView*, provides a logical view of the processor network. It shows the status of the network topology and the nodes (e.g allocated, available, down, etc.). Allocated partitions are highlighted and node specific information is displayed by clicking on a node symbol. Figure 9 depicts the ccsView tool.

The second monitor, *ccsMon* [21] shown in Fig. 10, gives a physical view on the cluster and the status of the nodes. On each node, a local monitor agent samples data and sends it periodically to a central server daemon. The server makes this information available to the GUI. The CCS Configuration Manager provides the ccsMon server with additional CCS specific information on the node status. This information is then used by the ccsMon user interface to highlight the corresponding frame(s) on the display. Clicking on a node opens an additional window which shows more detailed information. ccsMon is useful to inspect the behavior of parallel applications (e.g. communication patterns or hot spots).

The third tool, *SPROF* [32], can be used to locate performance bottlenecks at a very low level. This includes not only run time data, but also cache miss ratios and other events. For this purpose, we have integrated a small monitor in the ccsMon tool to show CPU load, MFlop/s, memory bandwidth, bandwidth of the CPU caches and their misses, and the SCI network bandwidth.

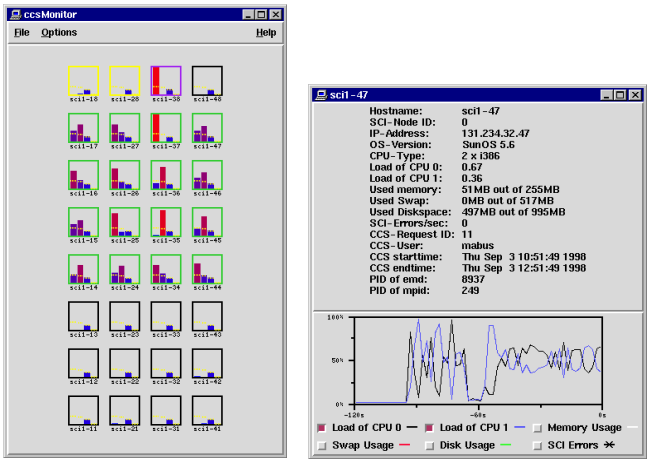

Abbildung 10: Monitoring a node with ccsMon.

3 Resource and Service Description

The *Resource and Service Description RSD* [11] is a versatile tool for specifying any kind of computer resources (computer nodes, interconnects, storage, software services, etc.). It is used at the administrator level for describing the type and topology of the available resources, and at the user level for specifying the required system configuration for a given application.

Figure 11 illustrates the architecture of RSD. There are three basic interfaces: a GUI for specifying simple topologies and attributes, a language interface for specifying more complex and repetitive graphs (mainly intended for system administrators), and an API for access from within an application program. In the following, we describe these components in more detail.

3.1 Graphical Representation

The RsdEditor [3] provides a set of predefined objects (icons) that can be edited and linked together to build a hierarchical graph of the resources in the system environment.

System administrators use RsdEditor to describe the computing and networking components in their computer center. Figure 12 illustrates a typical administrator session. The system components of the site are specified in a top-down manner with the interconnection topology as a starting point. With drag-and-drop, the administrator specifies the available machines, their links and the interconnection to the outside world. New resources can be specified by using predefined objects and attributes via pull down menus, radio buttons, and check boxes.

In the next step, the structure of the systems is successively refined. The GUI offers a set of generic topologies like ring, grid, or torus. The administrator defines the size and the general attributes of the system. When the system has been specified, a window with a graphical representation of the system opens, in which single nodes can be selected. Attributes like network interface cards, main memory, disk capacity, I/O throughput, CPU load, network traffic, disk space, or the automatic start of daemons,

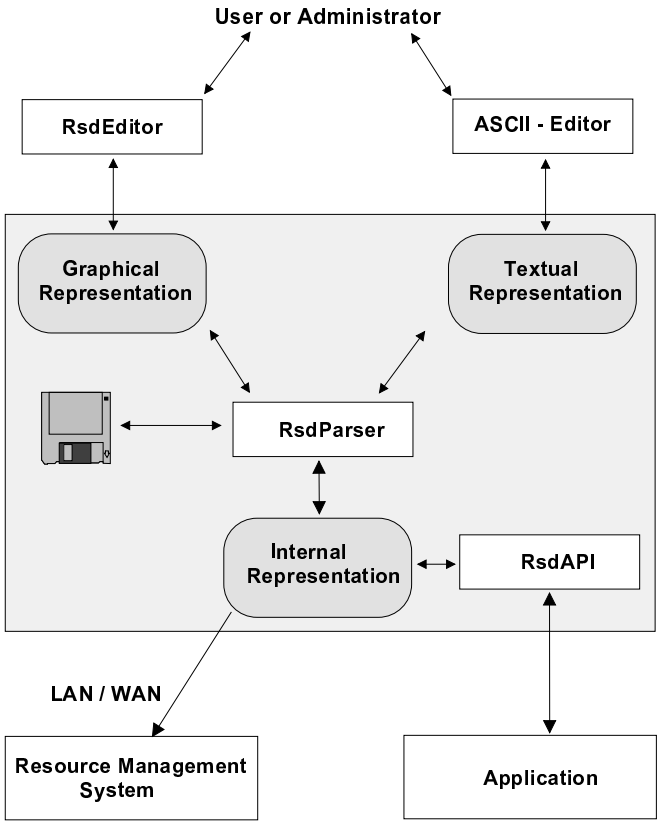

Abbildung 11: RSD architecture.

etc. can be assigned.

At the user level, RsdEditor is used for specifying resource requests. Several predefined system topologies help in specifying commonly used configurations.

3.2 Textual Representation

In some cases, the RsdEditor may not be powerful enough to describe complex computer environments with a large number of services and resources. Hence, we devised a language interface that is used to specify irregularly interconnected, attributed structures. Its hierarchical concept allows different dependency graphs to be grouped for building even more complex nodes, i.e., hypernodes.

3.2.1 Grammar

In RSD resources and services are described by attributed nodes (keyword `NODE`) that are interconnected by edges (keyword `EDGE`) via communication endpoints (keyword `PORT`).

Nodes: Nodes are defined in a hierarchical manner with an arbitrary number of subnodes at the next hierarchy level. Depending on the context a `NODE` is either a “processor” or a “process”. Passive software processes, like CPU performance daemons, can

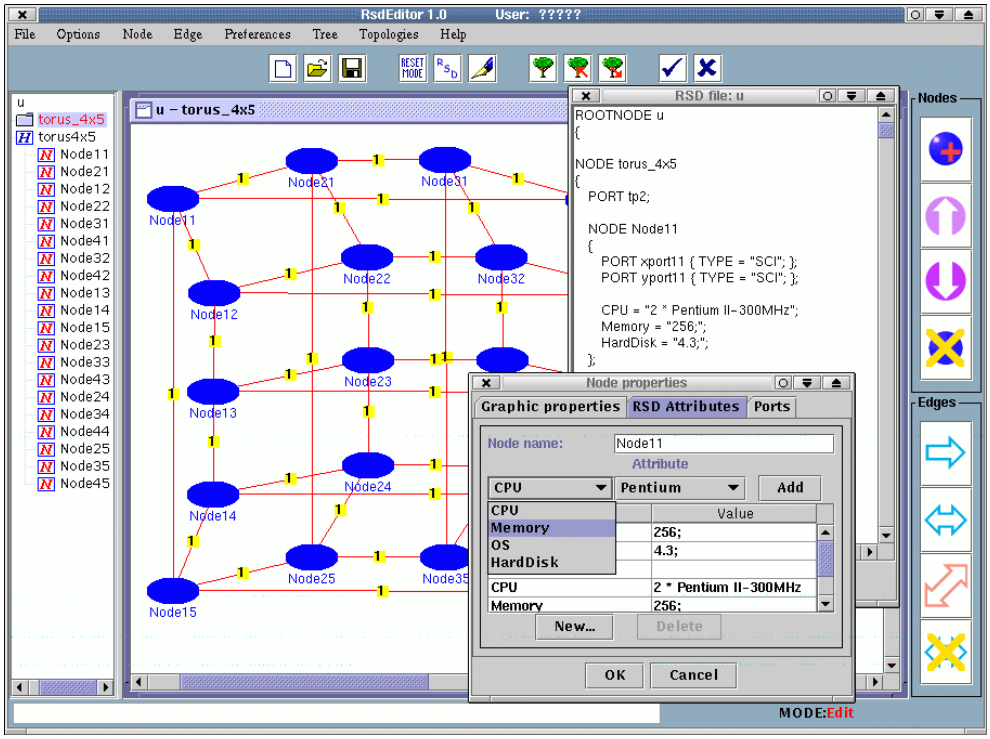

Abbildung 12: The RsdEditor.

be associated with nodes.

Ports: The keyword `PORT` defines a communication endpoint of a node. It acts as a gateway between the node and remote connections. Ports may be sockets, network interface cards or switches.

Edges: An `EDGE` is used to specify a connection between ports. Edges may be directed. Their characteristics (e.g. bandwidth, latency) are specified by the corresponding attributes. Passive software processes, like network daemons, can be associated with edges.

Virtual edges are used to specify links between different levels of the hierarchy in the graph. This allows to establish a link from the described module to the "outside world" by exporting a physical port to the next higher level. These edges are defined by: `ASSIGN` *`NameOfVirtualEdge`* {`NODE` *`w`* `PORT` *`x`* `<=>` `PORT` *`a`*}. Note, that `NODE` *w* and `PORT` *a* are the only entities known to the outside world.

In addition, the RSD grammar provides keywords and statements for including files (`INCLUDE`), sequences, variable declarations (`VAR`), constants (`CONST`), macro definitions (`MACRO`), loops (`FOR`), and control structures (`IF`).

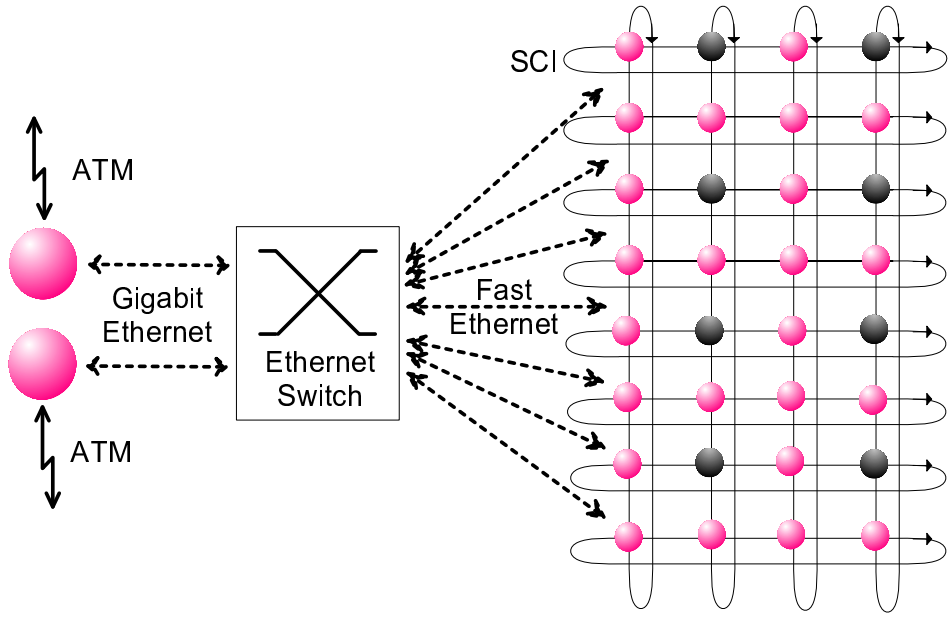

Abbildung 13: Configuration of a 4 x 8 node SCI cluster.

3.3 Dynamic Attributes

In addition to the static attributes of nodes and edges, RSD is also able to handle dynamic attributes. This may be important in heterogeneous environments, where the temporary network load affects the choice of the process mapping. Moreover, dynamic attributes may be used by the application to migrate a job to less loaded nodes during runtime.

Dynamic resource attributes are defined by the keyword `DYNAMIC` with the location of a temporary file that provides up-to-date information at runtime. The RSD parser generates the objects with the appropriate access methods of the dynamical data manager. These methods are used at runtime to retrieve, e.g., the network performance data, which was written asynchronously by a separate daemon (e.g an SNMP agent [33]).

In addition to file access, the data manager of the RSD runtime management also provides methods for retrieving data via UDP or TCP. More information on these services can be found in [12].

3.3.1 Example

Figure 13 shows the configuration of a 4 x 8 node SCI cluster. The corresponding RSD specification is shown in Figure 14.

The cluster consists of two frontend computers, an Ethernet switch and 32 compute nodes. The frontend systems have quad-processors with ATM connections for external communication and Gigabit-Ethernet for controlling the cluster and serving I/O to the compute nodes.

For each compute node, we have specified the following attributes: type of CPU, amount of memory, and the SCI/FE ports. All nodes are interconnected by unidirectional SCI ringlets in a 2D torus topology. The bandwidth of the horizontal rings

```
 NODE PSC                  // This SCI cluster is named PSC
{
   // DEFINITIONS:
   CONST X = 4, Y = 8;      // dimensions of the system
   CONST N = 2;             // number of frontends
   CONST EXCLUSIVE = TRUE;  // resources for exclusive use only

   // DECLARATIONS:
   // We have 2 SMP frontends, each with 4 processors, an ATM-, and a FastEthernet-port.
   FOR i=0 TO N-1 DO
      NODE frontend_$i {
      PORT ATM; PORT ETHERNET; CPU=PentiumII; MEMORY=512 MByte; MULTI_PROC=4;};
   OD
   // The others are dual processor nodes each with an SCI- and a FastEthernet port.
   FOR i=0 TO X-1 DO
      FOR j=0 TO Y-1 DO
         NODE $i$j {
         PORT SCI; PORT ETHERNET; CPU=PentiumII; MEMORY=256 MByte; MULTI_PROC=2;
         IF ((i+1) MOD 2 == 0) && ((j+1) MOD 2 == 0)
            PFSnode = TRUE; DISKsize = 50; // This is an I/O node of the parallel file system
         FI
         };
      OD
   OD
   // All nodes are connected via a FastEthernet switch
   NODE FEswitch {          // The FastEthernet switch
   FOR i=0 TO X*Y+N DO
      PORT port$1;          // Port 0 and 1 are Gigabit uplinks
   OD
   };

   // CONNECTIONS: build the SCI 2D torus
   FOR i=0 TO X-1 DO
      FOR j=0 TO Y-1 DO
         // the horizontal direction
         EDGE edge_$i$j_to_$((i+1) MOD X)$j {
            NODE $i$j PORT SCI => NODE $((i+1) MOD X)$j PORT SCI;
            MAX_BANDWIDTH = 400 MByte/s; DYNAMIC ACT_BANDWIDTH=FILE:/import/SCI_bandwith.txt;};
         // the vertical direction
         EDGE edge_$i$j_to_$i$((j+1) MOD Y) {
            NODE $i$j PORT SCI => NODE $i$((j+1) MOD Y) PORT SCI;
            MAX_BANDWIDTH = 500 MByte/s; DYNAMIC ACT_BANDWIDTH=FILE:/import/SCI_bandwith.txt;};
      OD
   OD

   // CONNECTIONS: build the FastEthernet links between switch and nodes
   port=0;
   FOR i=0 TO N-1 DO
      EDGE FEswitch_$port_to_frontend_$i) {
         NODE FEswitch PORT port_$port <=> NODE frontend_$i PORT ETHERNET;
         BANDWIDTH = 1 Gbps;};
      port=$port+1;
   OD
   FOR i=0 TO X-1 DO
      FOR j=0 TO Y-1 DO
         EDGE FEswitch_$port_to_$i$j) {
            NODE FEswitch PORT port_$port <=> NODE $i$j PORT ETHERNET;
            MAX_BANDWIDTH = 100 Mbps; DYNAMIC ACT_BANDWIDTH=FILE:/import/ether_bandwith.txt;};
         port=$port+1;
      OD
   OD
};
```

Abbildung 14: RSD specification of the SCI cluster in Figure 13.

is 400 MByte/s and 500 MByte/s in vertical direction. Each node is connected by FastEthernet to the Ethernet switch.

Eight of the 32 compute nodes also serve as I/O nodes of a parallel file system. They are accessible via Ethernet as well as via SCI. Therefore we specified two attributes per edge: the maximum bandwidth and the actual bandwidth. The latter one is a dynamical attribute. By means of these two attributes it is possible to determine which network should be used for the parallel file system I/O at runtime.

3.4 Internal Data Representation

The internal data representation [12] establishes the link between the graphical and the textual representation of RSD. It is capable of describing hierarchical graph structures with attributed nodes and edges.

Sending RSD objects between different sites poses another problem: Which protocol should be used? When we developed RSD, we did not want to commit to either of the standards know at that time (e.g. *CORBA*, *Java*, or *COM+*). Therefore we have only defined the interfaces of the RSD object class but not their private implementation. Today XML [36] seems to be a good choice for this purpose. However, since the textual representation of RSD is more readable (for humans) than XML, we will use it further on. To be compatible with other resource description tools we implemented a converter from RSD to XML.

The API layer (written in C++) can be divided into two classes: Methods for parsing and creating resource objects, and methods for navigating through the resource graph. In the following, we list some of the more important navigation methods:

- `RSDnode *parse(char *filename)`
  parses an RSD text file and returns a handle to the resulting objects.
- `RSDnode *GetNeighbours(node)`
  returns a list of nodes that are connected to the given node in the same level.
- `RSDnode *GetSubNodes(node)`
  returns a list of all sub nodes of the given node.
- `RSDattr *GetAttributes(entity)`
  returns the attributes of the given object (node or edge).
- `char *GetValue(attr)`
  returns the value of the given attribute object.

3.5 RSD Tools in CCS

The RSD tools are used in the CCS domain management for describing system resources and user requests. At boot time, all CCS components read the RSD specification created by the administrator. Each module extracts only its relevant information (by use of the *RsdAPI*), e.g., the MM reads the machine topology and attributes, and the QM extracts only the number of processors and the operating system type.

The *User Interface (UI)* generates an RSD description from the user's parameters (or from a given RSD description) and it sends it to the *Access Manager (AM)*. The

AM, checks whether the request matches the administrator given limits and forwards it to the QM. The QM extracts the information, determines a schedule and sends both the schedule and the RSD description to the MM. The MM verifies this schedule by mapping the user request against the static (e.g. topology) and dynamic (e.g. processor availability) information on the system resources. Figure 15 illustrates the RSD data flow in CCS.

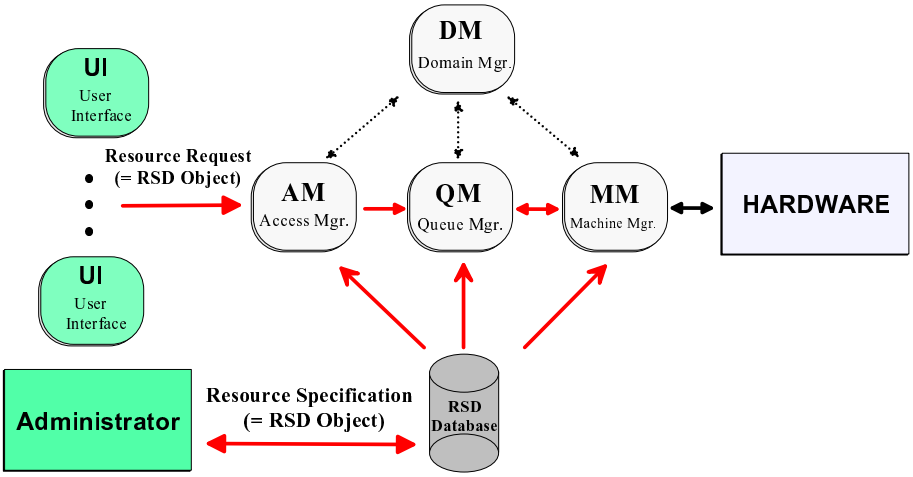

Abbildung 15: RSD data flow in a CCS domain.

Even though all CCS components are based on RSD, we have hidden the RSD mechanisms behind a simple command line interface because in the past, there was no need for a versatile resource description facility. Most systems were homogeneous, their topologies were simple and regular, and nearly all applications ran on only one system.

With the trend towards grid computing environments, resource description becomes more important now. In some metacomputer environments, the system (instead of the user) decides autonomously which of the available resources to use. Hence, the users need a convenient tool to specify their requests, and the applications need an API to negotiate their requirements with the resource management system.

4 Site Management

The local CCS domains described in Section 2 may be coordinated by two higher level tools as shown in Fig. 16: A passive instance named *Center Information Server (CIS)* that maintains up-to-date information on the system structure and state, and an active instance, called *Center Resource Manager (CRM)*, that is responsible for the location and allocation of resources within a center. The CRM also coordinates the concurrent use of several systems, which are administered by different CCS domain instances.

4.1 Center Resource Manager (CRM)

The CRM is a tool on top of the CCS domains. It supports the setup and execution of multi-site applications running concurrently on several CCS domains. Here, the term

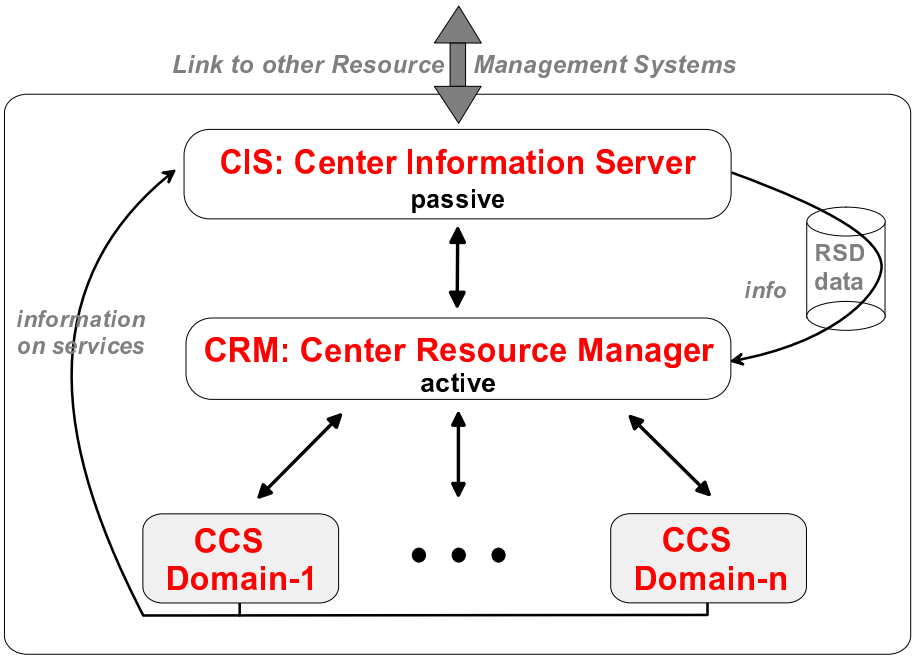

Abbildung 16: A CCS site with a link for metacomputer access.

'multi-site application' can be understood in two ways: It could be just one application that runs on several machines without explicitly being programmed for that execution mode [7], or it could comprise different modules, each of them executing on a machine that is best suited for running that specific piece of code. In the latter case the modules can be implemented in different programming languages using different message passing libraries (e.g., PVM, MPI, or MPL). Multi-communication tools like PLUS [10] are necessary to make this kind of multi-site application possible.

For executing multi-site applications three tasks need to be done:

- locating the necessary resources,
- allocating the resources in a synchronized fashion,
- starting and terminating the modules.

For locating the resources, the CRM maps the user request (given in RSD notation) against the static and dynamic information on the available system resources.

The static information (e.g. topology of a single machine or the site) has been specified by the system administrator, while the dynamic information (e.g. state of an individual machine, network characteristics) is gathered at runtime. All this information is provided by the CCS domains or the Center Information Server CIS (see next section). Since our resource and service description is able to describe dependency graphs, a user may additionally specify the required communication bandwidth for his application.

After the target resources have been determined, they must be allocated. This can be done in analogy to the two-phase-commit protocol in distributed database management systems: The CRM requests the allocation of all required resources at all invol-

ved domains. Since CCS domains allow reservations it is possible to implement a co-scheduling mechanism [19]. If not all resources were available, it either re-schedules the job or it denies the user request. Otherwise the job can now be started in a synchronized way. Here, machine-specific pre-processing tasks or intermachine-specific initializations (e.g. starting of special daemons) must be performed.

Analogously to the domains level, the CRM is able to migrate user resources between machines to achieve better utilization. Accounting and authorization at the site level can also be implemented at this layer.

The CRM does not have to be a single daemon. It could also be implemented as distributed instances like the QM-MM complex at the domain level.

4.2 Center Information Server (CIS)

The CIS is the 'big brother' of the domain manager (DM) at the site level. Like the well-known Network Information Service NIS, CIS provides up-to-date information on the resources in a site. However, compared to the active DM in the domains, CIS is a passive component.

At startup time, or when the configuration has been changed, a domain signs on at the CIS and informs it about the topology of its machines, the available system software, the features of the user interfaces, the communication interfaces and so on. The CIS maintains an RSD based database on the network protocols, the system software (e.g., programming models, and libraries) and the time constraints (e.g. for specific connections). This information is used by the CRM. The CIS also plays the role of a docking station for mobile agents or external users.

For higher level metacomputer components, the CIS data must be compatible or easily convertible to the formats used by other resource management systems (e.g. XML).

5 Related Work

Much work has been done in the field of resource management in order to optimally utilize the costly high-performance computer systems. However, in contrast to the CCS approach, described here, most of today's resource management systems are either vendor-specific or devoted to the management of workstation clusters in throughput mode.

The *Network Queuing System NQS* [25], developed by NASA Ames for the Cray2 and Cray Y-MP, might be regarded as the ancestor of many modern queuing systems like the *Portable Batch System PBS* [6] or the Cray *Network Queuing Environment NQE* [28].

Following another path in the line of ancestors, the *IBM Load Leveler* is a direct descendant of *Condor* [26], whereas *Codine* [13] has its (far away) roots in *Condor* and *DQS*. They have been developed to support high-throughput computing on UNIX clusters. In contrast to high-performance computing, the goal is here to run a large number of (mostly sequential) batch jobs on workstation clusters without affecting interactive use. The *Load Sharing Facility LSF* [27] is another popular software to utilize

LAN-connected workstations for throughput computing. More detailed information on cluster managing software can be found in [2, 23].

Aside from local cluster management, several schemes have been devised for high-throughput computing on a super-national scale. They include the Iowa State University's *Batrun* [35], the Flock of Condors used in the Dutch *Polder* initiative [15], the *Nimrod* project [1], and the object-oriented *Legion* [20] which proved useful in a nation-wide cluster. While these schemes emphasize mostly the application support on homogeneous systems, the *AppLeS* project [8] provides application-level scheduling agents on heterogeneous systems, taking into account their actual resource performance.

Another approach is to include cluster computing capabilities into the operating system itself. MOSIX (*M*ulticomputer *OS* for UN*IX*) [4] is such a software package. It enhances the Linux kernel and allows any size cluster of X86/Pentium based workstations and servers to work cooperatively as if part of a single system. Its main features are preemptive process migration and supervising algorithms for load-balancing and memory ushering. MOSIX operations are transparent to the applications. Sequential and parallel applications can be executed just like on an SMP. MOSIX provides adaptive resource management algorithms that monitor and respond (on-line) to unbalanced work distribution among the nodes in order to improve the overall performance of all the processes. It can be used to define different cluster types and also clusters with different machines or LAN speeds.

6 Summary

With the current trend towards cluster computing with heterogeneous (sometimes even WAN-connected) components the management of computing resources is getting more important and also more complex. Modern resource management systems are regarded as portals to the available computing resources, with the system specific details hidden from the user.

Most often, workstation clusters are not only used for high-throughput computing in time-sharing mode but also for running complex parallel jobs in space-sharing mode. This poses several difficulties to the resource management system. It must be able to reserve computing resources for exclusive use and also to determine an optimal process mapping for a given system topology.

On the basis of our CCS environment, we have presented the anatomy of a modern resource management system. CCS is built on three architectural elements: the concept of autonomous domains, the versatile resource description tool RSD, and the site management tools CIS and CRM. The actual implementation has the following features:

- It is modular and autonomous on each layer with customized control facilities. New machines, networks, protocols, schedulers, system software, and meta-layers can be added at any point—some of them even without the need to re-boot the system.

- It is reliable. Recovery is done at the machine layer. Faulty components are restarted and malfunctioning modules are disabled. The center information manager

(CIS) is passive and can be restarted or mirrored.

- It is scalable. There exists no central instance. The hierarchical approach allows to connect to other centers' resources. This concept has been found useful in several industrial projects.

The resource and service description tool RSD is a generic tool for specifying any kind of computing resource like CPU nodes, networks, storage, software, etc. It has a user-friendly graphical interface for specifying simple requests or environments and an additional textual interface for defining complex system environments in more detail. The hierarchical graph structure of RSD allows to hide complex details from the user by defining them in the next hierarchy level.

The CCS software package is currently being re-engineered so that it can be released under the GNU general public license.

Acknowledgments

Over the years, many people helped in the design, implementation, debugging, and operation of CCS: Bernard Bauer, Mathias Biere, Matthias Brune, Christoph Drube, Harald Dunkel, Jörn Gehring, Oliver Geisser, Christian Hellmann, Achim Koberstein, Rainer Kottenhoff, Karim Kremers, Fru Ndenge, Friedhelm Ramme, Thomas Römke, Helmut Salmen, Dirk Schirmer, Volker Schnecke, Jörg Varnholt, Leonard Voos, Anke Weber.

Also, we would like to thank our colleagues from CNUCE, Pisa, who have designed and implemented the RsdEditor: Domenico Laforenza and Ranieri Baraglia, and their master students Simone Nannetti and Mauro Micheletti.

Biography of Authors

Axel Keller is the principal system engineer and project manager of the CCS project. He is a staff member at Paderborn Center for Parallel Computing (PC^2) since 1994. His interests include parallel system programming, cluster technology, and innovative system area networks.

Alexander Reinefeld has been the managing director of the Paderborn Center for Parallel Computing from 1992 to 1998. He now heads the Computer Science Department at Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) and he holds a professorship at the Humboldt Universität zu Berlin. His research interests include grid computing, cluster computing, and innovative system area networks.

Literatur

[1] D. Abramson, R. Sosic, J. Giddy, and B. Hall. Nimrod: A Tool for Performing Parameterized Simulations using Distributed Workstations. *4th IEEE Symp. High Performance and Distributed Computing*, August 1995.

[2] M. Baker, G. Fox, and H. Yau. Cluster Computing Review. *Northeast Parallel Architectures Center, Syracuse University New York*, November 1995. http://www.npac.syr.edu/techreports/.

[3] R. Baraglia, D. Laforenza, A. Keller, and A. Reinefeld. RsdEditor: A Graphical User Interface for Specifying Metacomputer Components. *Proc. 9th Heterogenous Computing Workshop HCW 2000 at IPDPS*, Cancun, Mexico, 2000, pp. 336-345.

[4] A. Barak, O. La'adan, and A. Shiloh. Scalable Cluster Computing with MOSIX for LINUX. *Proc. Linux Expo '99*, Raleigh, N.C., May 1999, pp. 95-100.

[5] B. Bauer and F. Ramme. A General Purpose Resource Description Language. In: Grebe, Baumann (eds): *Parallele Datenverarbeitung mit dem Transputer*, Springer-Verlag Berlin, 1991, pp. 68–75.

[6] A. Bayucan, R. Henderson, T. Proett, D. Tweten, and B. Kelly. Portable Batch System: External Reference Specification. Release 1.1.7, NASA Ames Research Center, June 1996.

[7] T. Beisel, E. Gabriel, and M. Resch. An Extension to MPI for Distributed Computing on MPPs. In M. Bubak, J. Dongarra, J. Wasniewski (Eds.), *'Recent Advances in Parallel Virtual Machine and Message Passing Interface'*, LNCS, Springer, 1997, pp. 25–33.

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

[9] N. 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* Vol. 15, No 1, Feb. 1995, pp. 29-36.

[10] M. Brune, J. Gehring, and A. Reinefeld. Heterogeneous Message Passing and a Link to Resource Management. *Journal of Supercomputing*, Vol. 11, 1997, pp. 355–369.

[11] M. Brune, J. Gehring, A. Keller, and A. Reinefeld. RSD – Resource and Service Description. *Intl. Symp. on High Performance Computing Systems and Applications HPCS'98*, Edmonton Canada, Kluwer Academic Press, May 1998.

[12] M. Brune, A. Reinefeld, and J. Varnholt. A Resource Description Environment for Distributed Computing Systems. *Proc. 8th Intern. Sympos. High-Performance Distributed Computing HPDC'99*, Redondo Beach, 1999, 279–286.

[13] Codine: Computing in Distributed Networked Environments. http://www.gridware.com, November 2000.

[14] EGrid testbed. http://www.egrid.org/, November 2000.

[15] D. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A Worldwide Flock of Condors: Load Sharing among Workstation Clusters. *FGCS*, Vol. 12, 1996, pp. 53–66.

[16] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit *Intl J. Supercomputer Applications* 11(2), 1997, pp.115–128.

[17] I. Foster, C. Kesselman. The Grid: Blueprint for a New Computing Infrastructure *Morgan Kaufman Publ.* 1999.

[18] J. Gehring and F. Ramme. Architecture-Independent Request-Scheduling with Tight Waiting-Time Estimations. *IPPS'96 Workshop on Scheduling Strategies for Parallel Processing*, Hawaii, Springer LNCS 1162, 1996, pp. 41–54.

[19] J. Gehring and T. Preiss. Scheduling a Metacomputer with Uncooperative Subschedulers. *Proc. IPPS Workshop on Job Scheduling Strategies for Parallel Processing*, Puerto Rico, Springer, LNCS, vol. 1659, April 1999.

[20] A. Grimshaw, J. Weissman, E. West, and E. Loyot. Metasystems: An Approach Combining Parallel Processing and Heterogeneous Distributed Computing Systems. *J. Parallel Distributed Computing*, Vol. 21, 1994, pp. 257–270.

[21] H. Hellwagner, and A. Reinefeld (eds.). *SCI:Scalable Coherent Interface: Architecture and Software for High-Performance Compute Clusters*. Springer Lecture Notes in Computer Science 1734, 1999.

[22] *HpcLine*. http://www.siemens.de/computer/hpc/de/hpcline/, June 2000.

[23] J. Jones and C. Brickell. Second Evaluation of Job Queueing/Scheduling Software: Phase 1 Report. Nasa Ames Research Center, NAS Tech. Rep. NAS-97-013, June 1997.

[24] A. Keller and A. Reinefeld. CCS Resource Management in Networked HPC Systems. *7th Heterogeneous Computing Workshop HCW'98 at IPPS*, Orlando Florida, IEEE Comp. Society Press, 1998, pp. 44–56.

[25] B. A. Kinsbury. The Network Queuing System. Cosmic Software, NASA Ames Research Center, 1986.

[26] M. J. Litzkow and M. Livny. Condor – A Hunter of Idle Workstations. *Procs. 8th IEEE Int. Conference on Distributed Computing Systems*, June 1988, pp. 104–111.

[27] LSF: Product Overview. http://www.platform.com, November 2000.

[28] NQE-Administration. Cray-Soft USA, SG-2150 2.0, May 1995.

[29] *Portable Batch System (PBS)*. http://pbs.mrj.com/, November 2000.

[30] F. Ramme, T. Römke, and K. Kremer. A Distributed Computing Center Software for the Efficient Use of Parallel Computer Systems. *HPCN Europe*, Springer LNCS 797, Vol. 2, 1994, pp. 129–136.

[31] Scali. http://www.scali.no/, November 2000.

[32] J. Simon, R. Weicker, and M. Vieth. Workload Analysis of Computation Intensive Tasks: Case Study pn SPEC CPU95 Benchmarks. *EuroPar97*, Springer LNCS 1300, 1997, pp. 971-983.

[33] Simple Network Management Protocol (SNMP) Specification. http://www.faqs.org/rfcs/rfc1098.html, November 2000.

[34] T. L. Sterling, J. Salmon, D. J. Becker, D. F. Savarese. How to Build a Beowulf. A Guide to the Implementation and Application of PC Clusters. *The MIT Press*, Cambridge, MA (1999).

[35] F. Tandiary, S. C. Kothari, A. Dixit, and E. W. Anderson. Batrun: Utilizing Idle Workstations for Large-Scale Computing. *IEEE Parallel and Distributed Techn.*, 1996, pp. 41–48.

[36] XML 1.0 Recommendation. http://www.w3.org/TR/REC-xml, November 2000.