WHITE PAPER

Production and Operational Use of the Cray X1 System

Technical Description

April 2004

TABLE OF CONTENTS

Introduction

A supercomputer requires a significant customer investment over the life of the system. It is critical to customers that system utilization is planned and managed so that they may get their best possible value. System utilization is measured not only on the percentage of time that a system is idle, but also on the extent to which a customer's internal organizational and financial goals are met. As a result, most customers manage a complex set of requirements to ensure that their systems are effectively used. Cray Inc. provides a number of features for managing Cray X1 system resources. This paper provides an overview of those features.

While most Cray customers have resource management requirements, operational numerical weather prediction (NWP) customers face a particularly demanding resource management challenge. These customers need systems that meet stringent requirements to maintain an operational schedule for the generation of NWP products, while also supporting a secondary research load. The combination results in the need for an environment that delivers high system functionality and reliability. Cray systems are designed to meet these sorts of requirements.

This document assumes the reader is familiar with Cray X1 hardware and software at a fairly high level. The Cray X1 System Overview (S-2346), a standard Cray publication that can be found through the Cray corporate web site (www.cray.com), provides a thorough overview of the Cray X1 system.

Overview

Resource management covers a wide range of issues and features. This document covers the following major topics:

- Primary features and concepts
- `Psched`, the Cray X1 placement scheduler
- PBS Pro, workload management system
- Checkpoint/restart (CPR)
- Partitioning
- Resource accounting

These features are delivered with UNICOS/mp, the Cray X1 operating system, and PBS Pro, a workload management system from Altair Grid Technologies. Cray Inc. has worked closely with the PBS Pro developers to integrate PBS Pro with UNICOS/mp, and in turn to meet the resource management requirements of Cray X1 customers. In addition, Cray provides first-line technical support for PBS Pro for Cray X1 systems, allowing for effective and rapid response to customer problems.

Primary Features and Concepts

The following are major definitions and concepts used in this document:

Term	Definition
accelerated mode	The preferred mode for production runs where performance variation between runs is small. Accelerated mode applications are assigned a GASID and use the Remote Translation Table (RTT) for remote memory access. Accelerated applications must be allocated on logically contiguous processors. This mode of execution provides maximum performance for the application. (See "flexible mode").
application overflow node	A node with least one other flavor assigned to it. `psched` may be configured to use the multiple-flavored nodes as Application nodes only if an application is too wide to fit into the domain's Application-only nodes. In most cases, nodes of this type would also be declared as Support nodes so they would not remain idle except for wide application use.
application team (`apteam`)	The collection of all the processes that constitute an application; the collective identity of an application.
batch job	A batch job is a shell script that contains the set of commands you want run on your system resources. It also contains directives that specify the characteristics (attributes) of the job, and resource requirements (for example memory or CPU time) that your job needs. Once you create your batch job, you can reuse it or modify it for subsequent runs. PBS Pro also provides a special kind of batch job called *interactive batch*. An interactive batch job is treated just like a regular batch job (in that it is queued up, and has to wait for resources to become available before it can run). Once it is started, terminal input and output are connected to the job in what appears to be an `rlogin` session. Resources requested by the job are reserved for its duration. Many users find this useful for debugging or for computational steering.
Consumer	Threads consume CPU cycles. The CPU is the hardware that delivers CPU cycles, which are the resource.
Domain	See "scheduling domain".
domain attributes	Attributes that manage the behavior of the collection of regions assigned to the domain.

Flavor	One of three, flavor specifies how a node is to be used:: **Application node**: Runs user applications. The user declares an executable object to be an application by launching it with `aprun` or `mpirun`, placing it under the control of `psched` **Support node**: Run single-SSP commands such as daemons, shells, editors, and other user-level commands **OS node**: Provides kernel level services to application and support nodes In the default configuration, one node is assigned to serve as both an OS and support node, and the remaining nodes are assigned to serve as application nodes.
flexible mode	Execution mode that may improve system utilization. No GASID is assigned because the RTT is not used for remote memory access in this mode. Because flexible mode applications do not require a contiguous range of processors, it is possible that flexible applications will be launched ahead of accelerated applications if the current layout of running applications has left "holes" in the domain. This is the preferred mode for test runs and background runs. (See "accelerated mode").
gang	A set of processes making up an application that will be synchronously scheduled.
Global Address Space ID (GASID)	Used by applications executed in accelerated mode. Translates remote virtual addresses to physical addresses at the destination node. When accelerated mode is requested by an application, `psched` places the application in a span of logically contiguous processing elements where one common GASID is available. There are four GASIDs per module.
global attributes	Attributes that manage the behavior of characteristics that span domains.
multiple program, multiple data	A software feature under development for the Cray X1 system which will allow multiple, coupled applications to be launched together in a way that allows them to communicate using MPI, shmem, or Co-Array Fortran.

node	On the Cray X1 system, a node is a set of 4 MSPs that share memory on a single module, one node per module. On the Cray X1E, a node is a set of 4 MSPs that share memory on a single module, two nodes per module. With PBS Pro, a node is a set of resources managed by a single `pbs_mom` daemon (that is, a single system image).
Partition	A partition is, virtually, a separate system. Partitions are separately bootable, they can run different versions of the operating system, and they cannot communicate with each other.
Party	A set of gangs that allow for concurrent execution due to their non-overlapping allocation.
processing element (PE)	A software concept pertaining to a process within an application.
provider	A physical node that makes resources available.
region	A collection of one or more nodes administered identically. The nodes that constitute a region need not be contiguous nor they need to be physically identical. A node may be assigned to only one region; a region may be assigned to only one domain. Limits and gates are described at the region level but are enforced on a per-node basis.
region attributes	Attributes that describe the list of nodes assigned to the region plus region gates, limits and labels. Each node within a region has its own set of physical attributes that `psched` reads from the kernel. The physical attributes consist of flavor, memory size, number of SSPs, number of MSPs, and other characteristics known to the kernel.
resource	A *resource* is made up of memory and physical CPUs.
scheduling domain	A set of one or more regions collected into a named group for scheduling purposes. Each scheduling domain is controlled by an instantiation of placement, load balancing, and gang scheduling components. Nodes of application flavor must be included in a region assigned to one of the scheduling domains if they are to be utilized by `psched`. A region may not be included in more than one domain.
service provider	An identification of the portal through which the application was introduced into UNICOS/mp. Batch and interactive service providers are defined.

`Psched`, the Cray X1 Placement Scheduler

Applications require three things before they can run on the Cray X1 system: the application startup code, the kernel, and system placement by psched, the placement scheduler daemon. psched determines on which nodes an application should run and returns this placement information to the kernel. When the application is then initiated with an exec() system call, the placement information guides the startup process to organize the components of the application into an optimized configuration on the system.

This section describes placement scheduling of Cray X1 system resources using the `psched` daemon. This daemon assigns system CPU and memory resources to applications, according to hardware and administrative rules, as they are posted for execution.

Placement Scheduling

Cray has ported the `psched` placement scheduler from the UNICOS/mk operating system to the UNICOS/mp operating system. The `psched` daemon controls the placement of applications on application nodes. It enforces the requirements of the hardware and operating system, manages the use of Global Address Space IDs (GASIDs), processors and memory on the nodes, and implements the oversubscription policy established by the administrator.

`Psched` Overview

The placement scheduling daemon, `psched`, is the software mechanism that schedules and places all applications launched with the `aprun/mpirun` command. (The `aprun/mpirun` command is the only way for an unprivileged user to launch an application on UNICOS/mp.)

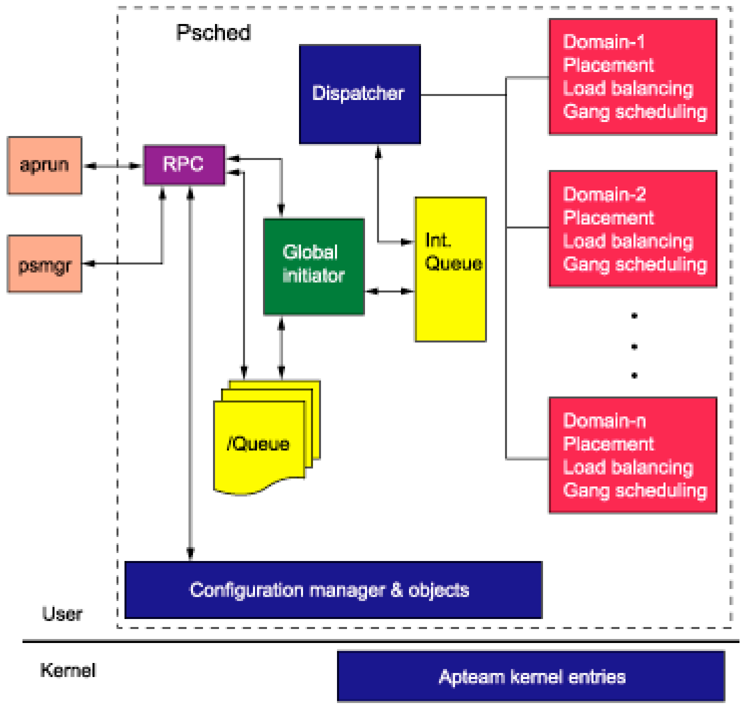
Some important features of `psched` are:

- It accepts requests for application placement from `aprun` and `mpirun`; and it locates sets of nodes that satisfy the requirements of the application, the hardware, and operating system.
- It evaluates machine and node usage to best manage resources in consideration of site environment and resource management policy.
- Time slice, load balancing, and a number of other controls and limits may be changed during operation without restarting `psched`.
- It schedules processors as a group (that is, “gang scheduling”) to maximize application performance and minimize interference among applications.

- It maintains sufficient information to recover all work should `psched` fail or be terminated. Typically `psched` needs only to be restarted to recover all running applications.
- It assists checkpoint/restart in placing and starting an application.
- It generates current process information and displays it via `psview`.
- It provides support for user limits such as application size and time limit. There is also an extensive set of node access controls that may be used for special needs. Support of *overflow nodes* is included in these access controls.
- It provides load and configuration information to PBS Pro, and enforces the PBS Pro queue limits for applications.

An administrative interface using the `psmgr` command (that is, placement scheduler manager) is used to configure `psched` functions. A set of displays is provided for both administrators and users by using the `psview`(1) command.

Each node runs a copy the UNICOS/mp kernel. This in turn allocates node processors and memory. The `psched` daemon provides information to the kernel through the `apteamctl(2)` interface. Using this information, the kernel allocates the required resources to serve the applications.


Psched
Dispatcher
Domain-1 Placement Load balancing Gang scheduling
aprun
RPC
psmgr
Global initiator
Int. Queue
Domain-2 Placement Load balancing Gang scheduling
/Queue
Domain-n Placement Load balancing Gang scheduling
Configuration manager & objects
User
Kernel
Apteam kernel entries

Scheduling Components

This figure shows the major application scheduling components. Components inside the dashed-line rectangle comprise the daemon. Requests are posted by the `aprun/mpirun` command to the queue. This is a FIFO ranked list of requests to allocate applications. The global initiator accepts placement requests, creates internal representations, and presents them to the domain placement process for placement and launching. Entities in this list are scanned by the dispatcher and offered to each domain placement function. When a domain accepts an application, the request is moved to a list of placed applications. The application waiting to start is signaled to continue. Startup code in the application, along with placement information provided to the kernel by the daemon, brings the application into execution.

Rules for Domains

The following rules apply to domains running under `psched`:

- A domain is made up of one or more application nodes.
- A node can belong to only one domain.
- More than one domain may be configured in `psched`, each with its own set of configuration options, if desired.
- Domain qualification gates may be established to direct applications to specific domains, based on their attributes.
- An application is restricted to a single domain. It cannot span domains.
- The node assignment to domains may not be reconfigured while `psched` is running.

Note that although multiple domains may be useful under some circumstances, machine utilization can suffer if not enough work is supplied to fully occupy each domain. For this and other reasons, a single domain divided into *regions* is more flexible than multiple domains, if properly configured. (See `Psched` Scheduling Hierarchy, next page, for more information.)

Application Placement

The following steps initiate placement of the application using the `aprun` or `mpirun` command:

1) The `aprun` / `mpirun` command is started with the necessary options and the name of the application file (`a.out`).
2) The application id (`apid`) becomes the process id (`pid`) of the `aprun` / `mpirun` command used to launch the application. A set of directives is composed that creates a `/Queue/apid` entry in the list of posted applications, binds it for placement, and sends it to `psched` using remote procedure call (RPC). The `aprun` / `mpirun` command specifies a signal object in the directive set (`SIGUSR1`). This signal is sent to `aprun` / `mpirun` to indicate that the application has been placed.
3) Errors detected by `psched` during posting are reported to `aprun` / `mpirun` which writes error messages to `stderr` and then exits.
4) The `aprun` / `mpirun` command receives a launch signal. (This signal indicates that the application has been given a place to run and the apteam kernel structure has been set up properly.) `aprun` / `mpirun` then `exec`()'s the `a.out` – this initiates startup for PE 0 of the application. Thus, the `aprun` / `mpirun` from a command on a support node becomes the PE 0 of the application.

`Psched` Daemon Elements

The `psched` daemon includes the following components:

- an *rpc* (remote procedure call) *interface* used by the `psmgr` and `psview` commands (and PBS Pro) to view/manipulate data
- an *object manager* provides an information repository for configuration data objects … here is where communication between `psched` and the rest of the system takes place
- a *feature manager* implements the internal execution control functions of the daemon, such as bind, verify, action, and exception.

`Psched` Scheduling Hierarchy

A region is a collection of one or more identically administered nodes, nodes that have identical `psched` scheduling attributes. The nodes making up a region need not be contiguous nor do they need to be physically identical. Any application node may be assigned to any region, but only one region. The hierarchy of resource elements managed by `psched` is as follows:

1) *Domain*: scheduling by `psched` is at the level of the *domain*.

2) *Region*: a domain is a set of one or more *regions*, grouped for scheduling purposes.

3) *Node*: a region, assigned to a single domain, is made up of one or more hardware *nodes* and managed by a common set of scheduling rules.

Scheduling Elements

Depending on the number of applications domains that you configure into your system, from 1 to *n* of the following scheduling elements are likewise configured into your `psched` environment to schedule domains:

load balancer	assesses the load on each node in the domain. Moves applications as needed to preserve balance (based on your configuration rules).
gang scheduler	synchronizes the scheduling of all member processes (gang) of an application. This guarantees that all are used by all processors simultaneously.
gate/limit manager	enforces access policies. Access to a domain is managed through gates and limits set up when domain is configured.
global dispatcher	handles application launch/termination on demand in light of your load balance policy.

These are each described in more detail in the following sections.

Load Balancer

The load balancer maximizes resources within a scheduling domain under a number of configurable rules that accomplish the following:

- filter applications into groups that are eligible or ineligible for migration.
- evaluate alternative positioning scenarios.
- migrate applications from one set of nodes to another to consolidate free space into contiguous areas.

The following load balancing rules can be defined:

prime	minimizes the number of prime applications that overlap with other prime applications.
swap	minimizes the amount of swapping needed to do a gang scheduling context switch.
parties	minimizes the number of parties in the domain.

fragmentation	minimizes fragmentation to maximize contiguous free space.
utilization	maximizes node utilization by comparing the number of nodes, idle SSPs, and parties within the domain.
idle	maximizes the number of idle SSPs in the domain. This rule is not typically configured but might be needed in some cases.

Gang Scheduler

The gang scheduler enables you to schedule all members of an application (the gang) so they are synchronized across all processors allocated to the application. This feature also allows the allocation of more than one application per processor.

Distributed memory applications typically have some degree of inter-PE synchronization to maintain orderly progress among the parallel portions of the solution. Implicit in synchronization design is that parallel execution will proceed at the same time. The system guarantees parallel execution by scheduling at once (that is, in a *gang*) all of the processors and memory that belong to an application, without exception. `psched` will gang schedule an application when there is competition for that application's resources. If many applications are present, one or more may run at the same time, provided that they do not compete with each other. All of the gangs eligible to run at once are named a *party*. The set of gangs that constitute a party execute at the same time. An application may be promoted from one party to another and so get more processor cycles if it is located such that it does not compete with the other gangs in the time slice.

`psched` time slices typically range from a few seconds to many minutes. The overhead cost of a context switch between parties primarily involves loading memory for applications about to execute. Distributed memory applications that span more than one node run with their memory locked in place. This is required to satisfy off-node memory references that cannot tolerate page faults.

Context switches are accomplished in two phases:

1) Sequentially disconnect all gangs of the running party to stop execution and unlock memory pages.
2) Connect each gang of the next party in parallel. This makes it possible for each node to participate in parallel with the work of loading memory and preparing the gangs to execute.

Unlocked memory pages are only written to disk to make space for an application beginning to execute. Context switching among applications whose memory usage does not exceed the physical capacity of the node does not require any memory loading. The duration of a time slice is partly determined by the time needed to swap memory for a new application. If much memory loading is needed, the time to do this should be factored into the time slice duration to amortize the overhead for the context switch. Durations can be changed at any time to accommodate changes in system load or to experiment with different scheduling behavior. Gang scheduling also allows the allocation of more than one application per processor, using the oversubscription configuration variables for processors and memory.

A domain's gang scheduler time-slices processes based on configuration settings.

Note: `psched` does not assign processors to an application. Instead, the kernel chooses which processors will be used when the context switch is made. The memory scheduler then assures that enough memory and processors are present on a given node to satisfy the resource needs that will be demanded of it.

The placement information that `psched` places in the apteam structure tells the processor and memory scheduler on each node which resources an application requires. Psched makes sure not to ask for more resources than can be delivered.

Gang Scheduling within Domains

Processes are gang scheduled at the level of the domain under the following rules:

1) A domain is made up of one or more application nodes.
2) A domain may be divided into one or more regions.
3) A node can be a member of only one domain.
4) More than one domain may be configured in `psched`, each with its own set of configuration options.
5) Domain qualification gates may be established to direct applications to specific domains, based on their attributes. Although multiple domains may be useful within a given system, machine utilization can suffer if not enough work is supplied to fully occupy each domain.
6) A node cannot switch domains while `psched` is running.

Note: Without exception, all processes within an application are gang scheduled. Gang scheduling is independent in each domain.

Gate / Limit Manager

The gate/limit manager enforces the access policies configured for your site. Access to a domain or portions of a domain may be managed by creating gate and limit controls at the time the domain is configured.

- A *gate* is a boolean predicate that tests an attribute and returns a pass or fail.
- A *limit* is a cumulative function which answers the question "If *nnn* units of resource *R* is added to the amount in use, is the sum still less than the limit?" If Yes, the limit is passed.

Region gates and limits are configured at the region level but are enforced at the node level. In effect, a region administers multiple nodes at once.

Domain gates and limits are configured and enforced at the domain level. These control aggregate domain resources, so per-node processor and memory usage controls do not work at this level.

Global Dispatcher

The global dispatcher is a separate execution thread handling application launch and termination requests. Although application placement and load balancing are performed by the same feature, load balancing cycles may be quite lengthy while application initiation and termination need to be done on demand. The global dispatcher makes this possible without disrupting the load balancing policy.

`Psched` Configuration File

The default `psched` configuration file, `/etc/psched.conf`, is installed with `psched` if a file by that name does not already exist. You can create alternative `psched` configuration files if you choose, depending on your needs. Then specify the file you want to use in the `psched` command line, or make a symbolic link (named `/etc/psched.conf`) that points to the desired configuration file.

PBS Pro Workload Management System

UNICOS/mp supports the execution of batch jobs. Cray Inc. has selected PBS Pro, a leading high-performance computing workload manager from Altair Grid Technologies, to provide batch queuing services for the Cray X1 system. PBS Pro provides a number of tools for monitoring operation of the batch subsystem. Scriptable command line interfaces are also available for customized system monitoring functions.

The following are high-level benefits of PBS Pro:

- solid migration path from NQE/NQS, provided with legacy Cray platforms
- first level technical support from Cray Inc.
- integrated machine utilization
- PBS Pro interfaces to the Cray `psched` placement scheduler, taking advantage of its resource management and job placement capabilities.
- PBS Pro maintains queues of jobs by resource type and priority, and supports checkpointing and/or killing jobs based on priority, job preëmption, and job dependencies. Both routing and execution queues are supported, however queue complexes are not supported.
- an extensible scheduler tool that allows sites to configure and customize scheduling policies. Basic features include fair share scheduling and job preëmption.
- accounting records for all jobs and sessions PBS Pro manages, including resource utilization outside of `psched`. PBS Pro also tracks accounting information for interactive batch sessions when users log in via an interactive PBS Pro job (otherwise, interactive sessions are recorded/maintained by UNICOS/mp). This data can be used to prepare summary accounting information as required.
- users can change certain job resource characteristics while jobs are queued (for example, CPU time limit or job priority).
- Administrators and users may intervene in job execution including: start/stop queue scheduling, suspend jobs, kill jobs, modify queue priorities, modify scheduling priorities, checkpoint/restart, and so on, with or without root level login.

Dedicated or shared resource allocation can be accomplished through the administration of queues providing dedicated access. Through direct or scripted administrator intervention, the appropriate queues can then be enabled or disabled during the switch between shared and dedicated system time.

PBS Pro Main Components

The primary components of PBS Pro are described in the following sections:

Job Server

The Job Server daemon, `pbs_server` , is the central element of PBS. All commands and daemons communicate with the server via an Internet Protocol (IP) network. It provides the basic batch services such as receiving/creating a batch job, modifying the job, protecting the job against system crashes, and managing queues. The server utilizes a common API and protocol for communication with client commands and other daemons.

Job Executor

The Job Executor is the daemon that actually places the job into execution. This daemon, `pbs_mom`*,* is informally called *MOM* (Machine Oriented Miniserver). MOM places a job into execution when it receives a copy of the job from `pbs_server`. MOM creates a new session as identical to a user login session as is possible. For example, if a user login shell is `csh`, then MOM creates a session in which `.login` is run as well as `.cshrc`. MOM also returns job output to the user (when directed to do so by the job request).

Job Scheduler

The Job Scheduler daemon, `pbs_sched`, implements site policy to control when each job is run and on which resources. The scheduler communicates with the `pbs_server` and select MOMs to query the state of system resources. It communicates with the server to learn about the availability of jobs to execute.

Job Priority and Dependency

Job Priority allows users to specify the priority of their jobs, defaults for which can be provided at both the queue and system level. Note however that user priority is specific to jobs owned by a single user – the scheduler may override priorities in conformance to local scheduling policy.

PBS Pro allows you to specify job dependencies between two or more jobs. Dependencies are useful for a variety of tasks, such as:

1) Specifying the order in which a set of jobs should execute
2) Requesting a job run only if an error occurs in another job
3) Holding jobs until a particular job starts or completes execution

Peer Scheduling

PBS Pro allows you to have different PBS installations automatically run jobs from each other's queues. This provides the ability to dynamically load-balance across multiple peer systems. When this feature is enabled, PBS can map a remote peer queue to a local queue and then move the remote jobs to run locally when resources become available. No job is moved until it can run immediately.

Currently the only restriction on peer scheduling is that it requires a flat user namespace – that is, user `joe` on the remote system(s) is the same as user `joe` on the local system.

Remote PBS Pro Administration

PBS Pro may be administered from any machine configured into a Cray X1 network, however root access to the Cray X1 system is required to install/upgrade PBS Pro, and PBS Pro manager privilege is required thereafter.

For installs and upgrades, PBS Pro uses the Cray Common Installation Tool (CIT), and requires that a GUI be used. Users can `ftp` the CD image to an on-site workstation, but X11 is required to display the GUI back to their local machine.

Note that in choosing between the command-line interface (CLI) and the graphical user interface (GUI), the majority of PBS Pro users and administrators choose to work with the CLI.

Supported Front End Platforms

The following platforms can be used as a front end to PBS Pro:

- Windows operating system
- UNIX platforms including AIX, BSD, HPUX, IRIX, Linux, SCYLD, Solaris, Mac OSX, and UNICOS/mp.

Fair Share Scheduling

PBS Pro includes a fair share capability that schedules submitted jobs mindful of users' entitlement to an equal share of system resources. While a user's job may consume more than its assigned share of CPU usage at a given time, that user's subsequent job(s) will not be scheduled until other users' jobs have been provided their assigned shares.

This implementation is similar to the UNICOS implementation of fair share found on Cray T3E systems, where users are put into a fair share group file. The file is read in and a tree is created. The tree consists of groups (nodes) and entities (leaves). Groups can contain groups. Every node and leaf has a number of shares associated with it. Priorities can be derived from these shares by taking a ratio of them to all the rest of the shares.

Because the PBS Pro fair share configuration is managed as a tree structure, users may only have one parent, be it the root group or a sub-group. However, users can have multiple IDs, one for each project (for example, `jdoe_physics, jdoe_math, jdoe_bio`). In this case, each user ID is associated with the appropriate resource group.

The fairshare capability allows a system administrator to set which PBS resource is collected for fairshare usage. The resource to use is specified in a file (`sched_config`) in which any PBS job attribute can be specified (for example, user, group, account name, and so on). Administrators may view current share usage using the `pbsfs` command.

The PBS Pro hierarchical fairshare tree enforces usage priorities across both individuals and groups of people, based on usage percentages for sorting and running jobs.

Example 1

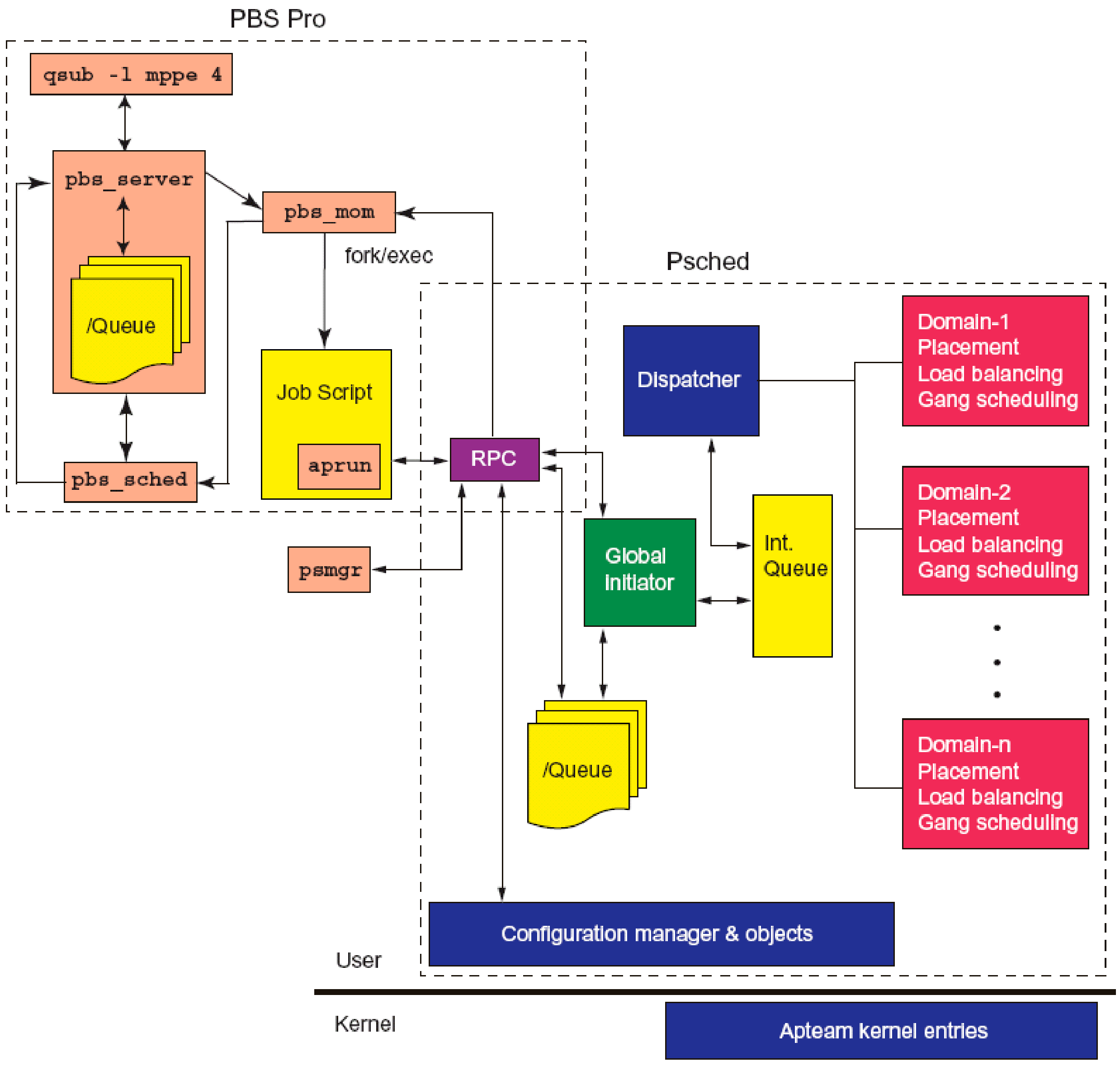
Consider that `user1` and `user2` are in group A, `user3` is in group B, and `user1` uses 100% of the machine for a month. Since group A has all of `user1's` usage, group B has much higher fairshare priority. Thus, `user3's` jobs would run before `user2's` jobs.

Example 2

Consider that there are three nodes/leaves at a given level with shares of 10, 20, and 10. The first entity/group has a priority of 25% or 10/40, the second has 50% or 20/40, and so on. A node with children can establish priorities among them via shares. So, if the second group (50%) is actually a group with 4 users and all the users have equal shares, then each user has 1/4 of 50% or 12.5% of the machine.

`Psched` / PBS Pro Interoperability

`Psched` and PBS Pro work closely together. This figure illustrates the relationship between the two, and is followed by a detailed description (Note here that `qsub -l mppe 4` is a sample command.):



PBS Pro / `Psched` Interoperability

PBS Pro performs two primary functions in its interactions with Psched:

- It collects information from `Psched` that is used to govern scheduling decisions.
- It provides job limit information to `Psched` so that it may set appropriate per-process application domain limits.

PBS Pro ←→ `Psched` interaction is via the `pbs_mom` daemon. The information that `pbs_mom` collects is proxied to the `pbs_server` and `pbs_sched` daemons.

Upon PBS Pro startup, `pbs_mom` informs the `pbs_server` of the number of application MSPs available on the system. This information is used to establish a counter within the `pbs_server` that limits number of MSPs and SSPs that may be used concurrently.

Job Script Considerations

When users construct job scripts, they determine the peak amount of application domain processor (MSP or SSP) resources the job will require. This requirement is expressed to PBS Pro through the `mppe` and `mppssp` resource limits, which may be included in either the job script as `#PBS` directives, or provided as arguments to the `qsub` command, as shown in the illustration.

Job Submission

When a user submits a job, the `pbs_server` notifies `pbs_sched` that a new job has arrived. Then, either of the following takes place:

- If `psched_fit` is enabled (in the scheduler configuration file), `Psched`-related information is requested from `pbs_mom`, including node placement data and lists of the launched and posted applications.
- If `psched_fit` is disabled, the scheduler relies on the available and assigned resource counts maintained by `pbs_server`.

Once `pbs_sched` determines that sufficient resources are available for the job, it tells the `pbs_server` to run the job. The `pbs_server` then assigns the job to `pbs_mom` for execution.

Resource Limits Enforced at Launch

When `pbs_mom` initiates a job, the job script is evaluated within the command domain. When a job makes a call to `aprun/mpirun`, `Psched` contacts `pbs_mom` to collect application domain limits (if the `Psched /Global/UseQueueLimits` parameter is enabled). `Psched` then uses the information provided by `pbs_mom`, together with information from the user limit database (ULDB), to establish the kernel-enforced limits for each user process it invokes.

Configuring `Psched` for PBS Pro

The process that PBS Pro uses to control the amount of work it will allow in execution at one time differs from that used by `psched`. But, PBS Pro and `psched` configurations must be compatible for proper functioning of the scheduling system. A simple recommendation is to configure `psched` without limits or gates and leave control of the workload to PBS Pro. This works well when there is not a significant amount of interactive work that bypasses PBS Pro, or if interactive use is restricted to interactive PBS Pro jobs.

Checkpoint / Restart (CPR)

Checkpoint/restart on the Cray X1 system is provided both with UNICOS/mp and with PBS Pro. When a job is checkpointed, it is written to disk until such time as it is restarted. Note, however, that application placement at restart is not necessarily on the same resources from which it is checkpointed. Placement is instead determined by `psched`, depending on system status and available resources in light of scheduling policy at the time of restart.

CPR and UNICOS/mp

UNICOS/mp provides a full kernel-supported checkpoint/restart capability. A checkpoint of a process or set of processes can be initiated by a program, user command (`cpr`), or direct operator intervention by system administrators, operators, or process owners.

Through the `cpr` command, the system sends a checkpoint signal to each process at checkpoint, and a restart signal to each process upon process restart. MPI jobs can also be checkpointed/restarted for integrated preëmptive scheduling.

Restrictions that can prevent a successful checkpoint include open sockets or special hardware file descriptors.

The following types of processes/jobs can be checkpointed:

- UNIX process and POSIX pthread ID (default)
- UNIX process group ID
- UNIX process session ID
- Process hierarchy (tree) rooted at the PID
- `Apteams` (application teams)

It is important to note that an application accepted into a domain will remain within that domain throughout its lifetime. That said, an application restarted from a checkpoint is considered a *new* application so it may be placed in a domain different from one it had been placed in earlier.

CPR and PBS Pro

Checkpoint/Restart (CPR) functionality is supported in PBS Pro running under UNICOS/mp. There are three kinds of checkpoint available:

manual	Manual checkpoint of PBS Pro jobs is via the `qhold(1)` (that is, *queue hold*) command. These jobs are then released via the `qrls(1)` (that is, *queue release*) command.
periodic	Periodic (automatic) checkpoint may be specified at job submission time through the use of the `-c qsub(1)` argument.
preëmptive	The scheduler can configure jobs for checkpoint/restart under preëmptive scheduling when jobs with higher privilege and/or priority are awaiting execution.

Partitioning

Cray X1 system partitioning allows system administrators to group two or more sets of node modules into separately-bootable systems, each with a completely independent operating system image. These partitions can be separately booted, dumped, halted, and so on with commands issued from the CWS without impacting other running partitions; and halted partitions can be combined into larger or split into smaller partitions. Cray also supports the addition or removal or inactive node modules from running partitions, however two running partitions cannot be combined without halting one of them.

The following is a list of operating conditions under for partitioning on the Cray X1 system:

- The smallest system image allowed runs on two node modules.
- Individual node modules cannot be split between partitions.
- No single job can span partitions.
- Partitions will be separate for normal operations, but there can't be a complete security wall between them.
- A single operator workstation will have access to all partitions – partitioning is facilitated through the System API on the CWS.
- Each partition requires its own network and disk connections. Cray supports a shared file system implemented for a Storage Area Network (SAN) environment that allows each partition to access shared data directly.
- Dynamic partitioning is not allowed. Partitions must be halted before they can be split into smaller partitions.
- There can be no inter-partition communication between Cray X1 system partitions. They are entirely separate system images.

Resource Accounting

UNICOS/mp provides standard UNIX SVR4 accounting evaluation tools such as `acctcom`, `acctcms`, and `acctmerg`. These tools enable the collection of per-process accounting data, with Cray-added functionality that consolidates accounting according to "account ID". In addition to these tools, a per-application record is written upon application termination. This record includes information about resource use, including "requested" versus "consumed" resources.

Note: UNICOS/mp does not use a Cray proprietary `udb` user database to store user characteristics. Instead, it uses two independent databases: one defines membership for account IDs, the other controls per-UID resource limits. The source file for both databases is in ASCII and can be maintained by any standard text editor. The ASCII files are converted into binary using either the `limit_mkdb` or `proj_mkdb` commands. Session initiators then access the binary versions of the databases for lookup use.

Common Accounting Data

The following data is reported by all accounting tools:

- user ID
- login name
- cumulative CPU time
- cumulative kcore-minutes
- cumulative connect time
- cumulative disk usage count of processes
- count of login sessions
- count of disk samples
- fee for special services

System Usage Accounting

UNICOS/mp provides an accounting tool to track the use of the following main system resources on a per-project and per-user basis:

- CPU time
- number and type of processors
- RSS memory
- I/O to secondary storage and over network connections
- amount of secondary storage

Process Accounting

UNICOS/mp supports standard SVR4 process accounting where accounting data is recorded at process (application) termination time. Process accounting data includes the following information:

- exit status
- user ID
- group ID
- process ID
- application ID (command name)
- control typewriter
- beginning time
- user time in clock ticks (a tick in UNICOS/mp is 10ms)
- system CPU time in clock ticks
- elapsed time in clock ticks
- memory usage in clicks
- characters transferred by read/write (I/O transfer counts)
- number of block reads/writes
- start time and elapsed time (wall clock)
- virtual memory integral (over user time) and a high water mark for memory bytes / blocks transferred

UNICOS/mp allows the system administrator to dynamically assign process resource limits on a per-UID basis. Group list modifications for active users do not take effect until the next login, but the modifications may be made at any time.

Note: Process accounting does not allow system administrators to determine the physical location (node/cpu) of where processes are run. The physical location changes over the lifetime of a process.

Application Accounting

The following application information is collected on a per-application basis:

- application flags (such as `MSP`, `SSP`, `FLX`, `ACX` and application id)
- user ID
- start time (that is, when the application is submitted by the developer)
- launch time (that is, when the application was started by `psched`)
- elapsed time
- connect time (different from elapsed time if gang scheduled)
- application name
- application width / depth

The application accounting record can be viewed through the `acctcom` command only.

Accounting Reports

The SVR4 accounting package generates daily accounting reports. Processes accounted are consolidated into a single report line. Consolidation is done either by UID or by account ID through the `acctmerg` utility. The raw output by acctmerg is available for post-processing through `awk` scripts that can be tailored to meet site-specific needs.

In addition to a per-UID / account ID report, a second report is generated by `acctcms`. This report is consolidated by command names. It provides an overview of the commands and applications used on the system.

PBS Pro Accounting

The PBS Pro system tends to produce lots of logfile entries. There are two types of logfiles: the event logs which record events from each PBS Pro daemon (`pbs_server, pbs_mom`, and `pbs_sched`) and the PBS Pro accounting log.

The PBS Pro Event Log

The amount of output in the PBS Pro event log files depends on the specified log filters for each daemon. All three PBS Pro daemons can be directed to record only information pertaining to certain event types. The specified events are logically “or-ed” to produce a mask representing the events the local site wishes to have logged.

The PBS Pro Accounting Log

The PBS Pro accounting log collects information upon job submission, execution, and completion.

Information at Resource Reservation

- The PBS Pro accounting log collects the following information about specified advance resource reservations:
- name of the party who submitted the resources reservation request
- name_string for resource reservation (if supplied)
- account under which reservation was made
- name of the reservation queue
- time at which the resources reservation got created (seconds since the epoch)
- time at which the reservation period is to start (seconds since the epoch)
- time at which the reservation period is to end (seconds since the epoch)
- the duration specified or computed for the resources reservation, in seconds
- allocated set of nodes with specified properties, if so required
- list of `acl_users` on a queue that is instantiated to service a resource reservation
- the list of `acl_groups` on a queue that is instantiated to service a resource reservation
- the list of `acl_hosts` on the queue that is instantiated to service a resource reservation
- list of resources requested by a reservation. Resources are listed individually as, for example: `resource_list.ncpus=16; resource_list.mem=1048676`.

Information at Job Execution

The following information is provided at the time a job is executed:

- the user name under which the job executed
- the group name under which the job executed
- the name of the job
- the name of the queue from which the job is executed
- time in seconds when job was created (first submitted)
- time in seconds when job was queued into current queue
- time in seconds when job became eligible to run (that is, with no holds, etc.)
- time in seconds when job execution started
- name of host on which the job is being executed
- list of the specified resource limits
- session number of job

Information at Job Termination

The PBS Pro accounting log includes the following information at job termination:

- the user name under which the job executed
- the group name under which the job executed
- if job has an "account name" string
- the name of the job
- the name of the queue from which the job is executed
- name if job draws its resources from a resources reservation and that reservation has a name
- if job is as a "reservation-job" (advance reservation of resources)
- time in seconds when job was created (first submitted)
- time in seconds when job was queued into current queue
- time in seconds when job became eligible to run; no holds, and so on
- time in seconds when job execution started
- session number of job
- the exit status of the job
- list of the specified resource limits

Cray Inc.
411 First Avenue South
Suite 600
Seattle, WA

Telephone: 206-701-2000

Fax: 206-701-2500

www.cray.com

WP-0030404