An Initial Foray Into Configuring Resources and Reporting Utilization on the Cray X1

Liam Forbes
ARSC/UAF
PO Box 756020
909 N. Koyukuk Drive
Fairbanks, AK 99775
907-450-8618 Fax: 907-450-8605
`lforbes@arsc.edu`
`http://www.arsc.edu/~lforbes`

Jan Julian
ARSC/UAF
PO Box 756020
909 N. Koyukuk Drive
Fairbanks, AK 99775
907-450-8641 Fax: 907-450-8605
`julian@arsc.edu`

Click here for this paper in PDF format.

ABSTRACT:
This paper and presentation review ARSC's initial (and ongoing) experiences configuring the Cray X1 resource managers, PBS and psched. It also shares our efforts to report resource utilization based upon the accounting mechanisms used by those managers and the system accounting software. Our goal is to share what we've already learned and solicit ideas & information from other X1 sites.
KEYWORDS:
X1 UNICOS/mp Accounting Resource Utilization Management PBS psched

I. Introduction

At the Arctic Region Supercomputing Center (ARSC), three generations of Crays have crossed the floor. Each succeeding system has provided greater resources for users. Faster CPUs, larger memory, and better features have provided increasing code speed ups.

Unfortunately, the requirements for managing those resources have not changed. User jobs have to be queued, dispatched, and executed on improved but still finite resource pools. As the systems have changed, so have the tools to manage them. Process accounting has evolved (and sometimes devolved), work load management has grown more complicated in response to varying situations, and job execution has developed to fit systems that are now both parallel, and vectorized.

This paper will share some of the modifications and strategies developed at ARSC to manage our 32-node X1. Installed autumn 2003, klondike is the latest ARSC Cray to go into production. Along with it, PBS Pro has replaced NQS, more features of psched are being employed, and process accounting has reverted back to standard UNIX accounting instead of Cray System Accounting (CSA).

ARSC's resource utilization reporting requirements have not changed though. So scripts are being redesigned to incorporate input from these three loosely coupled sources (PBS accounting, psched logging, and process accounting records). Hopefully, other sites will be able to take advantage of some of the lessons we have learned, and maybe help us and Cray to improve X1 resource management.

Klondike.arsc.edu System Version Levels:

1. UNICOS/mp 2.4.14
2. PBSPro 5.3.4c
3. Programming Environment 5.2

II. Resource Management Configuration

Limits

One of the first tools used to manage resources are system and user limits. The X1 has four limit scopes[1] of which we use three: interactive support, interactive application, and batch application. Currently, klondike is not configured to support executing applications on the support node, so the batch support scope is not applicable.

Interactive support limits are those applied to user login sessions and commands executed on the support node. Interactive application limits are those applied to aprun or mpirun commands executed from the command line. The limits database (`limit_mkdb(8)`, /etc/acct/limits.db) contains the resource bounds for both of these scopes. We use the limits database primarily to set the interactive support limits. PBS is configured to not include 4 MSPs, thus guaranteeing that they will always be available for interactive jobs. We also limit those jobs to 15 minutes of CPU time. By using the limits database only for the interactive scopes, we have avoided conflicts with the psched and PBS configurations (queue limits for example).

Initially, we tried to identify the batch limit(s) that matched our configuration from the T3E and/or SV1. The primary limit used on the T3E is mppt. However, as Table 1 shows, mppt is the one limit not available on the X1.

Resource	Scope	Enforcer	Operating System
cput	Job	MOM	All
mem	Job	MOM	All
vmem	Job	MOM	All
ncpus	Job	MOM	All
walltime	Job	MOM	All
file	Process	Kernel	All
pcput	Process	Kernel	All
pmem	Process	Kernel	All
pvmem	Process	Kernel	All
mppe	Job	MOM,Psched	UNICOS/mk,UNICOS/mp
mppssp	Job	MOM,Psched	UNICOS/mp
mppt	Job	MOM,Psched	UNICOS/mk
mppfile	Process	Kernel	UNICOS/mp
pmppt	Process	Kernel	UNICOS/mp
pmppmem	Process	Kernel	UNICOS/mp
pmppvmem	Process	Kernel	UNICOS/mp

Table 1: Available Limits and the Component that Enforces the Limit[1]

Instead, we chose to use walltime to control how long interactive jobs may execute. This configuration is only partially satisfying because a job may or may not spend all or even a majority of that time using the PEs, which is what we prefer to measure. However, until mppt time is supported, walltime is simpler to enforce compared to trying to calculate time based on individual PEs, ie pmppt.

Limiting memory and the number of PEs for the batch application scope is handled by the PBS queue definitions.

Overall, limits are still in development. As new releases of Unicos/mp, PBS, and psched are made available, the definitions and enforcement mechanisms are updated. This moving target requires re-reviewing configurations with each system and PBS upgrade. Since this is still a developing system, this is acceptable as long as it is taken into account.

Resource Management

The ARSC X1 is never oversubscribed (as long as the workload management configuration is working), nor do we use PRIME scheduling (at this time), nor do we gang schedule the resources, or time share them. Given these conditions, it is possible to function as if PBS has a complete and correct view of the available resources (primarily the number of available PEs). This simplification makes resource management easier, but because psched is the true execution scheduler and places the jobs on the nodes (making it the final arbiter of which resources are scheduled and which are not), PBS will not be 100% accurate at all times. Also because PBS only periodically surveys the system (every two minutes by default) its enforcement has a granularity that needs to be adjusted based on individual site workload mixes (long running jobs could have a longer polling period, short running jobs might need a shorter polling period).

PBS Queue Configuration

Here at ARSC, configuring PBS focuses on configuring the queues. Like the interactive limits, the queues are configured to limit the number of processing elements (PEs), MSPs or SSPs (referred to in PBS terms as mppe and mppssp) a job can use, and the length of walltime for which it can execute. A pipe queue is used to direct jobs into the execution queues based upon these two resources.

As mentioned previously, the "server resources_available.mppe" global setting is set to 120 (124 application MSPs - 4) to ensure that 4 MSPs are always available for interactive work. This allows users to continue debugging or executing small post-processing interactive jobs, even when multiple large jobs are running, thus providing better throughput for both types of jobs. For a while, we also had "server resources_available.mppssp" set to 480, but we discovered that this allows the system to be oversubscribed with SSP jobs. The explanation for why is later, but the solution is to unset "resources_available.mppssp".

Each queue has a resources_max.mppe and resources_max.walltime set to some value depending on the size of jobs expected to be run through that queue. Each queue also has resources_max.mppssp set to 0. All three are enforced by PBS as it periodically scans the executing jobs. The PBS directives in the job submission file determines the queue chosen. However, a job may try to use more PEs on the aprun or mpirun command. When PBS detects a job using more PEs then it was allocated, it will terminate the job. Currently our queues execute both types of job from the same queue.

An interesting effect occurs when trying to configure a queue for both MSP and SSP jobs. Suppose a user wants to run a single MSP job. Initially, our single queue contained the following configuration settings:

- set queue single resources_max.mppe = 1
- set queue single resources_max.mppssp = 4
- set queue single resources_max.walltime = 96:00:00
- set queue single resources_default.mppe = 1
- set queue single resources_default.mppssp = 4

Our assumption was that the MSP and SSP maximums and defaults would be applied to the appropriate type of job. That was incorrect. Instead, the SSP settings are assigned a value based on the following formula:

mppssp + (4 * mppe)

So when a user executed a job, if they did not include mppe directives, they would be allocated 8 SSPs (or 2 MSPs), 4 SSPs + 4*1 MSPs, rather than a single MSP. If the users did include a directive, they would still be allocated more PEs than desired. This excess allocation is caused by setting the resources_default.mppe and mppssp. To resolve it, both are now set to 0. As a side effect, the user is required to include directives specifying the PE resource(s) required to execute the job.

Next we discovered that an SSP job executed through the single queue could use 8 SSPs instead of only 4. The resources_max.mppssp is set according to the formula specified above. The condition was much more visible when SSP jobs, which should have been quantified as large, were running out of the medium queue. The solution, also based on the above formula, is to set resources_max.mppssp to 0 in every queue.

Lastly, as the system finally loaded up, we discovered that it could be oversubscribed with SSP jobs. It turns out that the SSP formula also applies to the global server limits, resources_available.mppssp. When the system is full of work, and more SSP jobs are submitted, the single interactive node was not left untouched by PBS. The solution was to unset resources_available.mppssp thus defaulting that setting to 4 times the resources_available.mpp setting.

In both the global and queue configurations, the mppssp settings are now calculated from the mppe settings. However, there is no official documentation on this interaction, which lead to lots of trial and error as well as support from Cray.

After discovering the magic of the SSP formula, the single queue is now configured as follows:

- set queue single resources_max.mppe = 1
- set queue single resources_max.mppssp = 0
- set queue single resources_max.walltime = 96:00:00
- set queue single resources_default.mppe = 0
- set queue single resources_default.mppssp = 0

User jobs must have one of the following PBS directives to be submitted to the single queue:

- #PBS -l mppe=1
  OR
- #PBS -l mppssp=4 #or less

PBS Fairshare

In an attempt to make access to the resources more fair by manipulating the queue ordering, fairshare has been enabled. Our desire is to not allow a single user to "hog" the system for days by loading in many jobs at once, but at the same time not have to restrict the number of jobs a user may have in the queue(s) at one time. This desire is based upon years of manually managing the T3E by hand to reorder the queues in attempts to more politically schedule the system and better manage system throughput.

Fairshare decisions can be very complicated and may produce unanticipated results. Rather than try to build a hierarchical structure that encompassed all the projects and users, we allocated 100 shares to the unknown group. Since no users are defined, all users are in the unknown group and received an equal percentage of the entire group's shares.

By lumping all the users into the group unknown, a flat structure is created that requires no intervention to maintain. These are strong positives from an administrative point of view. The usage is measured based on walltime accumulated so that a user who has not been executing jobs will have priority over a user who has been, since both have the same share value. Thus when the queue is evaluated to choose the next job for dispatch, the user who has not been running will be chosen.

Since fairshare only manipulates the job order in a queue, the order that the queues are evaluated (queue priority in our case) overrides the internal queue ordering. Thus, a user in the larger queues who has been running jobs may still have higher prority over users in the smaller queues who have not been running. Because this fits into our priority scheme, it is not a problem here (currently), but other sites may also need to consider using the job starvation settings to enforce their own policies.

Psched Configuration (or lack thereof)

Once a job works through the PBS queues, psched determines where in the system it will execute, allocates the resources, and starts it running. There are many possible configurations with psched, but we have chosen to leave it relatively untouched. By allowing PBS to be the primary resource manager and leaving job placement to psched, the interaction between the two is relatively straightforward, and trouble free. During initial system install, job migration and the limits maintained by psched were verified to function as expected. Beyond that, no changes have been made to the psched configuration.

Configuring PBS and and Psched to Work Together

There are some options that allow the resource manager, PBS Pro, and the scheduler, psched, to work together or independently. Depending on the configuration, psched will transfer either more or less information about the resources in use back to PBS. Our approach has been to tie the two packages together as tightly as possible to minimize the possible unexpected interactions and to maximize PBS's ability to be the overall workload manager. The primary option to couple PBS and psched is the "psched_fit" option.

By enabling psched_fit, PBS uses information from psched to help determine what jobs may be dispatched next. Thus, PBS has a more accurate view of the available PEs. "Pbs_sched uses information collected by pbs_mom from PScheD to determine whether sufficient MSP and SSP resources to initiate a job exist [4]." This ties right back into the goal of making PBS the arbiter of resources, even though psched actually allocates them.

III. Accounting and Reporting

Once the resources are being properly allocated, it has to be possible to account for their utilization, and even to report it to financially responsible parties. The X1 does not use Cray System Accounting (CSA), a familiar tool from previous generations. However, it appears that most of the information collected under CSA is still available in Unicos/mp, just in different locations. Table 2 shows the three locations of resource utilization records (accounting information), which daemon generates the records, and the identifier field that denotes a unique job.

Record	Generating Daemon	Identifier Field
/var/adm/pacct#	process accounting, kernel	process ID (PID)
/var/log/psched/PsLog*YY.JJJ*	political scheduling daemon, /usr/libexec/psched	application ID (APID)
/var/spool/PBS/server_priv/accounting/ *CCYYMMDD*	PBS server daemon, /opt/pbs/sbin/pbs_server	job ID (req)

Table 2. Sources of X1 accounting data.
YY.JJJ == two digit year followed by the modified Julian day.
CCYYMMDD == century, year, month, day.

From these three sources, we have been able to draw most of the information necessary to report utilization to the agency that allocates X1 hours to ARSC users[5][6].

Process Accounting

Process accounting is a basic UNIX operating system feature. The X1 utilizes standard SysV accounting, with a few additions. Process accounting serves as a great comparison to verify that the PBS and psched utilization records are accurate. It is also easily available to users and system administrators via the `acctcom(1)` command as well as the standard accounting API defined in `/usr/include/sys/acct.h`.

Unfortunately, unlike CSA, process accounting on the X1 does not span the entire life of a job. Within a user job, processes only exist when the application is executing on the node(s). One process for each PE is created. Trying to accumulate all this information and develop a full utilization report from just the process accounting records would require a lot of programming and calculation. Even then information such as the time spent queued could not be generated. So it is necessary to move up one more level to see what information is available in psched.

Process Accounting Records and Available Information

Before looking at psched, it is worthwhile to review what information is available in the process accounting, especially if it is used to verify other information sources. Figure 1 is an example of the output of acctcom(1) focusing on a single 4 MSP job.

```
klondike> acctcom -prtu lforbes | head -2
COMMAND                         START    END              REAL          CPU       (SECS)    CPU
NAME       USER      TTYNAME    TIME     TIME            (SECS)         SYS         USER  FACTOR      SID       PID      APID   ACCT ID
klondike> acctcom -prtu lforbes | grep 298073 | grep pxfd
pxfd       lforbes   ?          16:10:46 16:11:42         56.21         2.87       13.32    0.82   298073    298086    298086         0
pxfd       lforbes   ?          16:10:50 16:11:42         52.29         0.62       12.91    0.95   298073    262932    298086         0
pxfd       lforbes   ?          16:10:50 16:11:42         52.30         0.65       13.25    0.95   298073    298108    298086         0
pxfd       lforbes   ?          16:10:50 16:11:42         52.42         0.68       13.32    0.95   298073    298097    298086         0
pxfd       lforbes   ?          16:12:32 16:13:23         51.63         2.65       41.10    0.94   298073    298086    298086         0
pxfd       lforbes   ?          16:12:32 16:13:23         51.50         2.66       41.16    0.94   298073    298108    298086         0
pxfd       lforbes   ?          16:12:32 16:13:23         51.54         2.66       41.16    0.94   298073    298097    298086         0
pxfd       lforbes   ?          16:12:32 16:13:23         51.45         2.82       40.96    0.94   298073    262932    298086         0
```

Figure 1. acctcom(1) output for a 4 MSP job.

Because we are concerned about the length of time PEs are used, and the number that are used, it should be easy to identify from the above information that four PEs were in use by counting the number of processes associated with the same APID. On closer inspection, we see there are eight processes. The job was checkpointed during operation. When it resumed, a new set of processes was generated. The start and end times show the disconnect between the two sets of process. There is nothing else in the process accouting that indicates a checkpoint occurred.

Note that that job's execution time was in the range of 103 to 107 seconds (add together the first and fifth process' CPU seconds). Further, it is possible to tell that the job used most of that time to do actual calculation (by looking at the number of CPU seconds or the CPU factor). This gives an indication of the job's execution time, and demonstrates how well it is taking advantage of the number of PEs allocated to it. Thus accounting can be used to determine system performance.

These calculated times will be equal to, or less than the numbers reported by either psched or PBS. They represent the finest possible utilization granularity short of hardware counters. It is also the most accurate measure of utilization. As such, they make great verification statistics, but they do not tell the whole tale of a user's job. Specifically, they cannot provide an idea of how long a user job has existed, i.e. the qsub time to when results are returned. In addition, they do not provide an idea of the complete execution time of a batch job containing multiple commands, possibly including multiple aprun or mpirun commands.

We have also observed that the memory statistics (not shown in Figure 1) are not always correct. Since memory is not a resource that we use for allocations, scheduling, or report, we have not yet had time to track down the problems. Currently, our evidence is anecdotal at best, but it does appear that each revision of the kernel and operating system reduces the inaccuracies.

Tying Process Accounting to the PBS and Psched Job

The "-p" option is very important when using `acctcom(1)` to review applications. Only processes with an associated APID are application processes. Everything else is just a process running on the support node, and not of interest when calculating resource utilization. If "-p" is not specified, then the identifiers (SID, PID, and APID) are not available, and it is these identifiers that allow process accounting records to be tied to the other accounting sources.

The session ID (SID) field (added to the process accounting structure in Unicos/mp 2.4) is an indirect connection between the process accounting, and PBS accounting. The PBS job ID is not, and perhaps cannot be, stored with the process records. The SID, on the other hand, is recorded with every process. It is almost the parent PID; almost because the SID is the PID of the login shell from which the process is initiated. For interactive jobs, this would be the user's login shell from which the aprun or mpirun command is initiated. For batch jobs, this would be the shell launched by PBS from which it executes the job script. The SID is stored in the PBS exit records for each batch job. Using the SID, one can work back to the PBS job ID via the exit record (more on PBS records later).

The application ID (APID) field is a direct connection between the process accounting and psched accounting. Once the process record is obtained, the APID is available to search out any related psched records.

Psched Logging

Accounting information from psched comes out of the log file generated by the daemon. One log file is maintained for each day of the year and nightly a new file is started. Within the log files are records for each scheduling event that occurs during a job's life cycle. There are also other log entries containing information that may be useful for debugging either the psched daemon or the PBS mom daemon when they or jobs are not running correctly. Also, in the same directory as the log files, core files are stored if ever psched crashes and generates one. These files can be sent back to Cray to determine what occurred during catastrophic types of errors.

Records for each psched event include posting (putting the job into psched's queue), placing (allocating PEs and memory to a job), launching (starting job execution, deleting (stopping a job for checkpointing), restarting (resuming a checkpointed job), and deleting (stopping a job upon completion). Every job record starts with a timestamp, and since each log file is a single day, a complete history is accumulated. From these records, a history of the job, from psched's perspective, can be generated.

Psched Logging Records and Available Information

Some of the psched records contain resource utilization information. The number and type of PEs are stored with each posted record, as well as the command name and some basic limit information. The connect time is stored with each deletion record, whether it was a checkpoint or job exit that caused the delete. Figure 2 shows the record contents for a job that was dispatched from PBS to psched, checkpointed during execution, released, and completed successfully.

```
klondike> grep 298086 /var/log/psched/PsLog04.120
29.16:10:48 Posted apid 298086 uid 929 flags bMNAX w:d:N 4:1:0 time UNLIMITED memory UNLIMITED cmd ./pxfd
29.16:10:48 Place apid 298086 in/Domain/app gasid none modules: 0x9
29.16:10:48 Launched apid 298086
29.16:11:44 Deleted apid 298086 Connect time 00:00:00:55 dd:hh:mm:ss
29.16:12:32 Restarting apid 298086 uid 929 flags bMNFUX w:d:N 4:1:0 time UNLIMITED memory UNLIMITED cmd ./pxfd
29.16:12:32 Placed apid 298086 in /Domain/app gasid none modules: 0x8
29.16:13:25 Deleted apid 298086 Connect time 00:00:00:52 dd:hh:mm:ss
```

Figure 2. psched accounting records for a 4 MSP job.

The job in Figure 2 is the same job shown in Figure 1. The APID demonstrates the connection. Again, because the job was checkpointed, there are two start (launch and restart) and end (delete) records generated. The connect time can be added together to see that the job executed for 107 seconds (the greater of the values calculated by adding process accouting statistics together).

Using the record timestamps, it is possible to calculate some time statistics as well. The job was posted at 16:10:48, launched at the same time, and completed at 16:13:25. Therefore it spent no time queued waiting for psched to allocate resources and ran for 157 seconds. It checkpointed at 16:11:44 and was restarted at 16:12:32. The amount of time checkpointed was 48 seconds. Subtracting the checkpoint time from the run time is a wall execution time of 109 seconds. The difference between that and the 107 seconds connect time appears to be the slack time needed for psched to do its work moving the job on and off the PEs. Depending on the reporting requirements, the site has to choose which time (connect time or walltime) to report.

We use psched records to report interactive job resource utilization and we report the calculated wall execution time. Even though the job is not truly executing when checkpointing, those resources are still allocated to it and cannot be used by another job. Therefore the user has to be "charged" for them (the term charged is used loosely since ARSC does not receive money from users, but rather deducts time from SSP hours allocated to the user by the DOD HPC Modernization Program). As long as another user cannot access the resource, it has to be counted as utilized.

There is still information missing that psched cannot provide. In the case of batch user jobs, psched does not record when a user submits a job via qsub. Nor does it have the full history of the user's job beginning with the qsub, and ending with the returned results. Once again, another set of records must be referenced and the only source left is PBS. First, a connection between the psched records and the PBS records must be established.

Tying Psched Logs to the Process Accounting Records and the PBS Job

As Figure 1 and Figure 2 demonstrate, the APID generated for each psched job directly connects the psched log records to the process accounting records. No further effort is required.

Unfortunately, the SID is not stored in the psched log records, nor is the PBS job ID (in fact they probably are not available to psched without rewriting some code). The only way to tie the psched log entries to PBS accounting entries is through process accounting. Using the APID, a single process accounting entry is needed in order to establish the SID. From there, the exit record in the PBS accounting can be found, which also contains the PBS job ID. Again, the SID is useful, this time as the indirect connection between psched and PBS.

PBS Accounting

PBS provides a single accounting file for each day on which jobs are executed or executing. These files are generally off limits to users and provide a lot of information about each user job that goes through the system.

Because of the conditions enumerated previously, ARSC is able to treat the PBS accounting records as complete and accurate for all batch jobs executed on the X1. We do not have to worry about calculating statistics for jobs that are suspended or displaced by other jobs; nor do we have to attempt to calculate the results of jobs sharing the same PEs. Further, because memory and disk space are not resources that we currently use to calculate allocations, we do not have to calculate the utilization either (but the data is available).

PBS Accounting Records and Available Information

The key PBS accounting record is the exit record, denoted with an "E" in the second field (fields are separated by semi-colons). This record contains information about the PE usage, the memory usage, the cpu time and factor, the connect and walltimes, as well as information about the original resources requested. The exit records also contains times, in epoch seconds, of some of the other events occurring during the job's life such as the enqueue time, the start time, what queue it was dispatched from, and the job name.

The start time can be corrupted by checkpoint though. Since the restart code in PBS is the same as the original start execution code, the start time information is overwritten with the time of the last restart. The loss of this information makes calculating a job's wait time a little more difficult, but not much. Figure 3 is an example of the records written by the user job that contained the psched job and processes shown in Figures 1 and 2.

```
klondike> grep 8835 /var/spool/PBS/server_priv/accounting/20040429
04/29/2004 16:10:36;Q;8835.klondike;queue=default
04/29/2004 16:10:36;Q;8835.klondike;queue=Qsmall
04/29/2004 16:10:38;S;8835.klondike;user=lforbes group=staff jobname=pxfd.q queue=Qsmall ctime=1083283836 qtime=1083283836 etime=1083283836
start=1083283838 exec_host=klondike/0 Resource_List.mppe=4 Resource_List.mppssp=0 Resource_List.walltime=00:30:00
04/29/2004 16:11:46;C;8835.klondike;
04/29/2004 16:12:31;T;8835.klondike;
04/29/2004 16:15:04;E;8835.klondike;user=lforbes group=staff jobname=pxfd.q queue=Qsmall ctime=1083283836 qtime=1083283836 etime=1083283950
start=1083283951 exec_host=klondike/0 Resource_List.mppe=4 Resource_List.mppssp=0 Resource_List.walltime=00:30:00 session=298073 end=1083284104
Exit_status=0 resources_used.cpupercent=77 resources_used.cput=00:13:18 resources_used.mem=2165440kb resources_used.mppe=4 resources_used.mppssp=0
resources_used.ncpus=1 resources_used.vmem=439229376kb resources_used.walltime=00:02:50
```

Figure 3. PBS accounting records for a 4 MSP job.

Given the contents of the exit record, we calculate the following usage statistics for batch jobs:

- the overall lapsed time = end - qtime
- the job's execution time = resources_used.walltime (1)
- the job's wait time = lapsed time - execution time (2)
- the number of PEs used in SSPs = (4 * resources_used.mppe) + resources_used.mppssp
- PE reserved time = execution time * number PEs (3)
- PE used time = resources_used.cput (3)

(1): Walltime is of course not really the execution time, but again, because the resources are unavailable to another job, the user is "charged" for those resources as if their job were executing 100% the whole time on the resource.
(2): Wait time incorporates both the time in the PBS queues, as well as any time the job might have been checkpointed.
(3): The reserved and used times give an good idea of how efficiently a job is using the resources it has been allocated.

Looking at the exit record in Figure 3, the lapsed time is 268 seconds, the execution time is 170 seconds, and the wait time is 98 seconds. This leads one to wonder why there is such a large difference between the PBS statistics and the psched statistics. It is important to remember that the PBS job includes more than the application. There may be commands before and after the aprun or mpirun which are also accumulating in those time periods. Thus it is important to compare those statistics with the reserved and used times.

In this case, the reserved time is 2720 seconds and the used time is 798 seconds. The used time divided by the number of PEs (in SSPs) is 49 seconds per SSP, approximately the same as the CPU seconds per process and the connect time reported by psched. This double checking indicates that the calculations are based in reality and can be legitimately used in reporting. Of course, the remaining question is how to tie the PBS records to the process accounting and the psched logs.

Tying PBS Accounting to the Process Accounting Records and Psched Jobs

As before the SID can be used to tie the PBS record to the process accounting. This time though it is a direct connection since the SID is in both the PBS record and the process record. Once the SID is known, all of the pertinent process records are available. If only the application process records are desired, make sure to look for only process records that also have APIDs.

Once a process record with the APID is found, then the psched records are available as well. The APID becomes the indirect connection between PBS and psched by using the process accounting records.

IV. Conclusion

Finally, from process accounting through PBS accounting, one has a full picture of a user's job's resource utilization. From the time the user submits their job, to the time that they receive the output, there is a record of the changes that the job goes through. From that record utilization can be calculated, reported, and used to improve the system efficiency. It is important to configure those three systems not only to enforce the site policies on resource allocation, but also to communicate enough to generate the resource utilization statistics.

By relying on three different accounting sources, there will be some inevitable "slop". The three systems do not account for the utilization in the same way, and they each have a slightly different view of the system based upon their functionality. For ARSC though, by configuring PBS to only release an amount of work less than or equal to the available physical resources, the resulting accounting information is accurate enough for us to use. By using process accounting to double check that user jobs are clicking off CPU cycles approximately equal to the values recorded by PBS, the system has enough correctness to be manageable and fair.

Of course every site's usage calculations will be a little different, depending on what matters to their local policies and what resources are most important. However, by utilizing the operating system, the batch system and the scheduling system's accounting mechanisms, every job's resource utilization can be tracked. From these statistics, we are able to generate utilization reports that allow us to "charge" our user's allocations for the resources they use, and monitor the efficiency of the resource usage. By reviewing these statistics we can identify

Learning a new system like the Cray X1 can be an enjoyable challenge. Porting existing tools and requirements to the new architecture provide a method of learning the new system unavailable in other forums. However, it is necessary to have a roadmap of where everything is or the challenge just becomes a major frustration.

This paper attempts to provide the start of a roadmap for administrators who need to learn the resource management features available on the X1 and/or the resource utilization accounting features. At ARSC, we provide daily and monthly utilization information to our funding agency, which is used to verify that users are making use of their allocations. By ensuring that the resources are made available and reporting the utilization, we are ensuring that our user population is maintained, and hopefully even grows as we grow our resources.

The unique challenges of the X1 lie in the way that three different resource management and accounting products, the operating system, PBS, and psched, have to be configured to work together to create an efficient system. There is documentation available for each of the individual products, but only time and experience allows administrators to really make everything work together to create a usable platform. Hopefully our experience will help other X1 sites reduce their time required to achieve that goal.

V. References

[1] **PBS Pro(tm) Release Overview, Installation Guide, and Administration Addendum for Cray Systems (S-2345-533).** Cray Inc. 2003.
[2] **Unicos/mp v2.4 Manpages.** Cray Inc. 2004.
[3] **Unicos/mp v2.4.** Cray Inc. 2004.
[4] **PBS Pro v5.0.** Altair Engineering, 2004.
[5] **HPC System Utilization Metrics.** DOD HPC Modernization Program, 2002.
[6] Carlson, Kurt. **Reporting Unicos and Unicos/mk System Utilization.** CUG SUMMIT Proceedings, 2000.

VI. Author Biographies

Liam Forbes was co-Chair of the CUG Security SIG. In 1996 he started working at the Arctic Region Supercomputing Center as a High Performance Computing Systems Analyst. In 1998, he received a Masters of Science in Computer Science from the University of Alaska, Fairbanks. Recently Liam was co-project lead in installing and verifying the Cray X1, klondike, at ARSC. Currently a member of CUG, Usenix, and SAGE, Liam's interests include information security, system administration, stained glass, Highland bagpiping, and completing the house projects thought up by his wife.

Jan H. Julian is a Technical Services System Analyst at the Arctic Region Supercomputing Center. He has been working with the Cray X1 for the last 9 months. Prior experience includes 12 years as a Systems Engineer with IBM Corporation and 7 years of experience with the National Weather Service as a system administator.