

Experience with PBSPro on a 4 x 512p Altix SuperCluster

Dale Talcott

NASA Advanced Supercomputing Division (NAS)
e-mail: dtalcott@mail.arc.nasa.gov

(Rec. 30 May 2006)

Abstract: The Columbia project at NASA Ames includes twenty 512 processor Altix systems. Four of those systems are joined together into a Numalink based globally accessible non-coherent memory Altix SuperCluster. PBSPro 7.0 and 7.1 have been used to schedule and run work on this system. In this presentation, we will go over features of PBSPro especially applicable to such systems, some problems we have encountered and how they have been addressed, and remaining, unresolved issues.

Key words: PBS, PBSPro, Linux batch processing system, Altix, mpiexec, Columbia

1. INTRODUCTION

PBSPro historically does a good job of managing batch jobs on small to large clusters of 1- to 2-processor hosts. It also does well on single hosts with many processors. The Altix SuperCluster of four hosts, each with 512 processors, part of the NASA Columbia system provided a challenge, though. The job mix includes both smaller jobs, several of which should run at the same time on the same host, as well as larger jobs that span multiple hosts, but might not use all the processors on any one of the hosts.

This presentation is about our experience with PBSPro on the NAS supercluster, also called “the 2048”. The version of PBS reviewed is primarily PBSPro7SP1. The version of the O/S is SLES 9, with SGI's ProPack 4.

2. PBS OVERVIEW

PBSPro (Portable Batch System) provides batch job services for Unix, Linux, and Windows-based computing systems. On Unix and Linux, it is comprised of a “Server” that maintains the database of queues, jobs, users, reservations, *etc.* Various client commands allow users to create, status, and delete jobs through interactions with the server. Similarly, there are administrative commands that talk to the server to modify PBS configuration and settings. A scheduler queries the server about the state of the system, then decides when and where to run user jobs. The server forwards the jobs to MOMs (Machine Oriented Monitors) on the execution hosts. The jobs are scripts run by the shell of the user's choice.

There are other functions that I won't go into here, such as staging-in files before job execution and staging-out after script completion, as well as returning normal job output.

2. NEW FEATURES IN 7.1

Cpusets

Earlier versions of PBS have issues that make their use in this environment difficult. In particular, 7.1 includes several improvements related to cpuset handling.

If you want to run multiple jobs at the same time on the same Altix box, it is a good idea to run each job in its own “cpuset”, so that the job's use of processors and memory does not interfere with other jobs' uses.

So, PBS creates a cpuset for each job. In 7.1, these cpusets are created before the prologue is run, which means the site can use the prologue to customize the cpuset. We take advantage of this feature to create a sub-cpuset that is writable by the user. This allows operations on the sub-cpuset (*e.g.*, *bcfree*) that are not permitted on the base cpuset.

For multi-host jobs, a cpuset of the correct size is created on each host involved in the job. We'll see how these are used in a minute.

Another significant improvement with 7.1 is that each MOM works hard to clean up the cpuset when the job terminates. Under 7.0, PBS relied on the “notify_on_release” cpuset flag to clean up cpusets. This worked most of the time, but could fail if the epilogue failed to remove the child cpuset created by the prologue.

Under 7.1 the MOMs recursively clean up the job cpusets. If there are processes still running in the cpuset, the MOM first tries to kill them. If they don't go away, the MOM moves them to the */PBSPro/suspended* cpuset, then removes the job cpuset. Notice that this can leave processes occupying memories that PBS now considers available for reassignment.

Select and Place Fixed (mostly)

Prior to version 7.1, the use of the select and place resource specifications was problematic. Under 7.1, both work most of the time. There is still an issue with a complex select with multiple “host=” specifications. Namely, the scheduler will often decide it can never satisfy such specifications. This can usually be fixed by rewriting the specification so the “host=” attribute is the first element of each chunk.

Mpiexec

The mother superior starts the job shell within the cpuset created for the job on its first host. However, it took a little jiggery-pokery to get MPI processes on the remote hosts started in their cpusets. PBSPro 7.1 handles this very nicely with a script called mpiexec. The SGI version of MPI (mpt) includes a command “mpirun” for starting an MPI program. In turn, mpt uses another SGI product, “array services”, when it needs to start MPI processes on remote hosts. As of ProPack 4 SP3, array services can be replaced with “secure array services”, which uses SSH to establish the underlying communication with a remote host.

The net result of this layering is a good example of the Unix philosophy of tool reuse. On the first host, mpiexec examines the -n argument and the PBS_NODEFILE to create an equivalent set of hosts and number of processes for mpirun. It also rewrites the user command to prefix a pbs_attach command, as described later. Mpirun starts the proxy process and uses ssh to build connections to the execution hosts. A copy of /usr/etc/sarrayd is also started on each host *via* a separate mechanism.

On the execution hosts, sshd starts a shell to execute the command requested by mpirun, which is /usr/lib/array/bin/sasremex with magic cookie arguments it uses to talk back to the originating host. From there sasremex learns how many MPI processes to start, which ranks they have, etc, and what command to start and what environment to set for it.

The command sasremex starts is the pbs_attach program, inserted by mpiexec, with the -j PBS_JOBID argument identifying which job the process wants to attach to, and thus, which cpuset. Pbs_attach attaches itself to the correct cpuset, discards its first three arguments, and execs the rest. This is the user program. In this case, it is an MPI program, so it forks one time for each rank on that host, and hangs around as a shepherd process. Each child finally runs the user code. Whew!

4. PROLOGUE ISSUES

As mentioned above, we run a custom prologue at the start of a job. This prologue performs several functions.

Bcfree. User data pages are placed in memory following a “first-touch” protocol by default. Node memory is also

used for file system buffer cache. One result of this is that a previous job’s I/O can affect the amount of available memory local to the CPUs assigned to a new job. To help with this, the prologue runs a bcfree across all memories belonging to the job to flush out cache pages from previous jobs.

Initially, this had its own issue because a kernel bug sometimes caused bcfree to essentially hang for up to several days. This is fixed in the newest ProPack4 kernels.

Child Cpuset. As mentioned above, the prologue creates a user-writable sub-cpuset of the job cpuset. This allows the user to run bcfree themselves or to change the buffer cache management within the cpuset (*i.e.* to select between local memory or round-robin among memories in the cpuset).

File System Checks. The hosts involved mount a clustered filesystem (CXFS). Unfortunately, CXFS sometimes spontaneously unmounts. So, the prologue includes tests that certain file systems are mounted. If not, the prologue indicates to the MOM that the job can not start now and should be retried later.

SSH Key. Normally, the MOM runs the prologue and epilogue only on the first host in a multi-host job. We wanted prologue operations performed on all hosts. So, the first prologue determines which other hosts to run on and uses ssh to invoke itself on these other hosts. Now, the prologue runs as root, but we don’t allow unrestricted ssh access via root. Instead, the prologue uses a special ssh key and matching entry in the destination authorized_keys file. This, in turn, requires a lot of paranoia in the prologue to prevent abuse should the key be compromised.

In addition, some of the arguments passed to the prologue (and epilogue, which uses the same mechanism), include special characters that would not survive the trip through ssh. So, the prologue essentially MIME-encodes such arguments on the way to ssh and decodes them on the other end.

5. OUTSTANDING ISSUES

Backfill

There appear to be cases where PBS runs a backfill job even though that delays the start time of the starving job PBS is trying to backfill around. We have seen at least one instance where multiple such backfill jobs delayed a starving job by days.

E-mail conversations with Altair suggest that whatever bug causes this, might also explain cases where jobs have been started even though they conflict with an advance reservation.

A possible clue: Each time we have observed these problems, it involved a job requesting a specific host, and we have seen starving and backfill work correctly when the starving job did not request a specific host.

Partial System Dedicated Time

PBSPro supports dedicated time, but only for the whole system at once. We run regular hardware maintenance on hosts, one at a time. Our solution is to create per-host advance reservations spanning the dedicated time. However, the current PBS has an issue that prevents this from working cleanly.

The hosts normally have 512 processors, with 4 set aside for system processes and the remaining 508 available to PBS. So, the straightforward solution is to create a reservation with `ncpus=508`. Suppose, though, that some CPU develops a problem before the scheduled maintenance window. The usual procedure is to power that CPU (and its neighbors) down and bring the remaining system back up minus a few CPUs. Now, PBS believes the reservation cannot be satisfied (even though it is confirmed) and ignores it!

Another possibility is to reserve only a minimum number of CPUs, but with `place=excl` to get exclusive access to the whole system. This appears to work and would handle the hardware maintenance needs. However, we also sometimes use dedicated time for user block time. The reservation would be too small for this use.

Our solution for now is to monitor the size of the system from a cron job and recreate the reservations at the correct size whenever the number of CPUs changes.

Stray Cpusets

Even though PBSPro 7.1 does a better job of cleaning up cpusets at job termination than 7.0, it still misses some. Especially in the case of multi-host jobs where one host involved crashes. The other MOMs don't always clean up. The result is that MOM's left hand thinks more CPUs are available than her right hand can find when it is time to create a cpuset for a new job.

The MOM should watch for such stuck cpusets and adjust its reported inuse CPU count accordingly. It also needs to report it somewhere so operators can be alerted to the situation.

Node_fail_requeue Problem

By default, if a MOM doesn't checkin with the server for 310 seconds, the server reruns jobs assigned to that MOM. The thinking is that the MOM host has crashed. Rather than wait for the MOM to requeue the job when the host comes back up, the server takes the job away and puts it back in the queue. Just in case, though, the server incre-

ments the run count for the job before running it again. The server then ignores job-end ("obit") information from MOMs reporting on the job, but with a lower run count.

What we observed was that it was not uncommon for a host to go incommunicado for more than five minutes, but then come back with running jobs still intact. Meanwhile the server reran the jobs. The other hosts were busy, so the jobs stayed in the queue until the original host started talking again. The scheduler then asked the server to run the job on that host. When the server sent a run job request to the MOM, the MOM noticed that it was already running the job and replied to the server with an error. Which the server ignored. Eventually, the job finished, but when the MOM reported this to the server, the run count was wrong, so the server ignored the obit. The result is the server thinks the job is running, but the MOM no longer knows anything about it.

Our solution was to disable the feature by setting `node_fail_requeue` to 0.

Hybrid MPI + OpenMP

There is a basic flaw in PBS 7.1's GRUNT (Grand Unified Node Theory). It does not let you specify how many threads you want per process. That is, at one time, there were three pieces of information you could specify for a job: how many hosts, how many processes per host, and how many CPUs per process. The 7.1 implementation squashes the last two quantities into one. If you are running a pure MPI code, it gives the number of MPI processes on the host, *via* the contents of the `PBS_NODEFILE`. For pure OpenMP, it gives the number of threads (*via* `OMP_NUM_THREADS`). However, a mixed MPI+OpenMP program run with `ncpus=N` would try to start N MPI processes, each with N OpenMP threads. A recipe for system meltdown. So, our local MOM forces `OMP_NUM_THREADS` to 1 and leaves it up to OpenMP or mixed-mode users to set it to the desired value.

There is a more subtle problem. Suppose you are running scaling studies of a code within a single, multi-host PBS job. Suppose you are varying `OMP_NUM_THREADS`. When the Altair `mpiexec` constructs placement information for `mpirun`, it ignores `OMP_NUM_THREADS` and is perfectly happy to place half of one process's threads on one host and the other half on another host. Things don't work so well.

To fix this, we modified the `sgiMPI.awk` script used by `mpiexec` to take `OMP_NUM_THREADS` into account when placing CPUs.

Array Services and \$SPATH

While it was pretty clever of PBS to leverage the existing `mpirun` plus array services facilities, it also means PBS MPI

jobs suffer from any deficiencies in these facilities. And, there is a big one. When array services starts a process on a remote host, it passes all the original environment except LANG, PATH, SHELL, USER, and TZ. These get set to values specific to the remote host. Generally, this is correct for USER and harmless for LANG, SHELL, and TZ, but modifying PATH frequently breaks the user codes.

The hack that NAS uses to ameliorate the problem is to put a wrapper around mpirun. This wrapper copies these environment variables (if set) to new variables with “__” reserved names. Array services copies these new variables across without issue. On the remote hosts, the system-wide shell startup files (/etc/csh.cshrc, /etc/bash.bashrc) check for the __ variables and copy their values back to the original variables. This handles users with csh, tesh, and bash as their shells, but not those with sh or ksh. Fortunately, there are few of those.

Red/Blue/Purple problem

When we started running PBS on the SuperCluster, the users were divided into three groups, that I'll call “red”, “blue”, and “purple”. For administrative reasons, the red users were supposed to run on only the first two hosts. The blue users were to run on only the last two hosts. The purple users could run anywhere.

The obvious solution was to create three queues, “red”, “blue”, and “purple” with appropriate ACLs, plus some other characteristics that would force jobs from those queue to run on the correct hosts. We could never discover the appropriate “other characteristics” to get this to work. We tried assigning a red boolean property to the first two hosts and a blue property to the last two hosts. We then set these as default and minimum attributes for their respective queues. A job with a single chunk is placed correctly. However, something like “... -q blue -l select=2:ncpus=128 -l place=scatter” gets placed anywhere, ignoring the blue requirement. We tried several variations, but found nothing that worked in all cases. Talking with Altair, this might be doable in version 8 when it comes out.



Before joining the NASA Advanced Supercomputing (NAS) Division, **DALE TALCOTT** worked at Purdue University for 20 plus years as a systems analyst, supporting HPC systems such as CDC 6600, Cyber 205, ETA 10P*, Intel Paragon, and IBM SP. He was also involved in large development projects such as checkpoint/restart and batch job scheduling. Apart from HPC, he created tools to help administer Macintosh systems in student labs. Dale holds a Bachelor's Degree in Mathematics from the Ohio State University.

6. QSTAT ENHANCEMENTS

It seems that no one is satisfied with the default qstat output – sites with source usually modify it in some way. NAS is no exception.

A little-known feature of qstat is that you can compile it so that it delegates output to TCL scripts when you specify the “-f” option. Using TCL has a couple advantages: It is easier to produce nicely-formatted output in TCL and the TCL can be set up so users can substitute their own formatting scripts.

That is just what NAS has done. However, we moved the TCL flag from -f to -e, because our users were accustomed to using -f to get the detailed display for a job or queue. Next, we made -e the default. So, if someone wants the Altair output, they need to specify “-ee” to turn TCL off.

Our TCL code itself provides auto-adjusting column widths, so information is not truncated (although there is a way to set min and max columns widths on a field-by-field basis). The TCL code sorts queued jobs using (almost) the same criteria as the scheduler. The standard -n option provides a way to see which hosts each running job is on. We added the complementary option to show which jobs are running on a particular host or hosts.

A future enhancement will allow users to specify just which fields they want, a la the ps “-o” option. This will permit adding more fields, that are not normally displayed because the display would not fit in 80 columns. The users can add or remove fields to fit in whatever widths they want.

7. SUMMARY

It works well, for the most part. We would like to see the backfill and stuck cpuset issues fixed by Altair, and SGI should fix Array Services so it does not clobber PATH, *etc.*