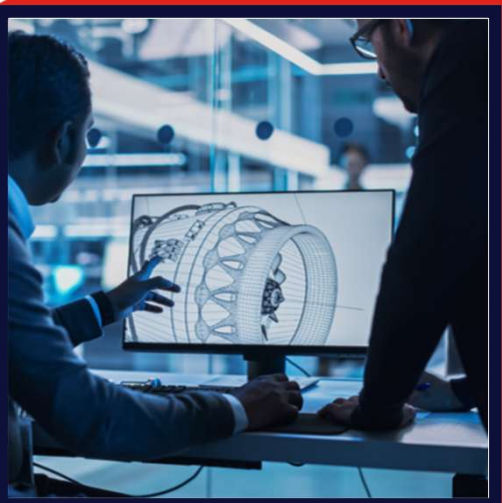
National Supercomputing Centre

ASPIRE2A Advanced Job Management Training



Manjunath Doddam, Director - HPC & Cloud Technology, Altair
Ayush Mathur, Sr. HPC Systems Engineer, Altair

V2.0 - OCT 2026

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

National Supercomputing Centre
SINGAPORE

Table of Contents

1. Introduction to HPC and HPC Works 03

- Products, features, and benefits
- What is PBS Professional?
- Components and Roles
- PBS Job lifecycle
- How PBS uses cgroups
- Practice session details
- Projects

2. Job Management 19

- PBS job types
- Submitting jobs
- Managing jobs
- Job attributes
- Requesting job resources
- Using MPI with PBS
- Multithreaded applications
- Default resources
- Exercise

3. Advanced Job Management 80

- Job exit codes
- Managing Jobs
- Specifying Email Notifications
- File Processing
- PBS File Staging
- Job Submission Dependencies
- Signaling
- Exercise

4. Job Arrays & Reservations 115

- Concept of job array
- Example scripts
- Job array environment variables
- Querying a job array
- Job array and subjob states
- Job array terminology
- Advance Reservations
- Exercise

5. Using GPUs with PBS 149

- Using GPU with PBS
- Requesting GPU Job Resource
- How to Access assigned GPUs
- ASPIRE2A AI Cluster
- Use case of local NVMe Storage and using them on AI Cluster
- Exercise

6. Troubleshooting 166

- Why Jobs May be in this State
- Why Jobs can't be submitted
- Documentations and Altair Community

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

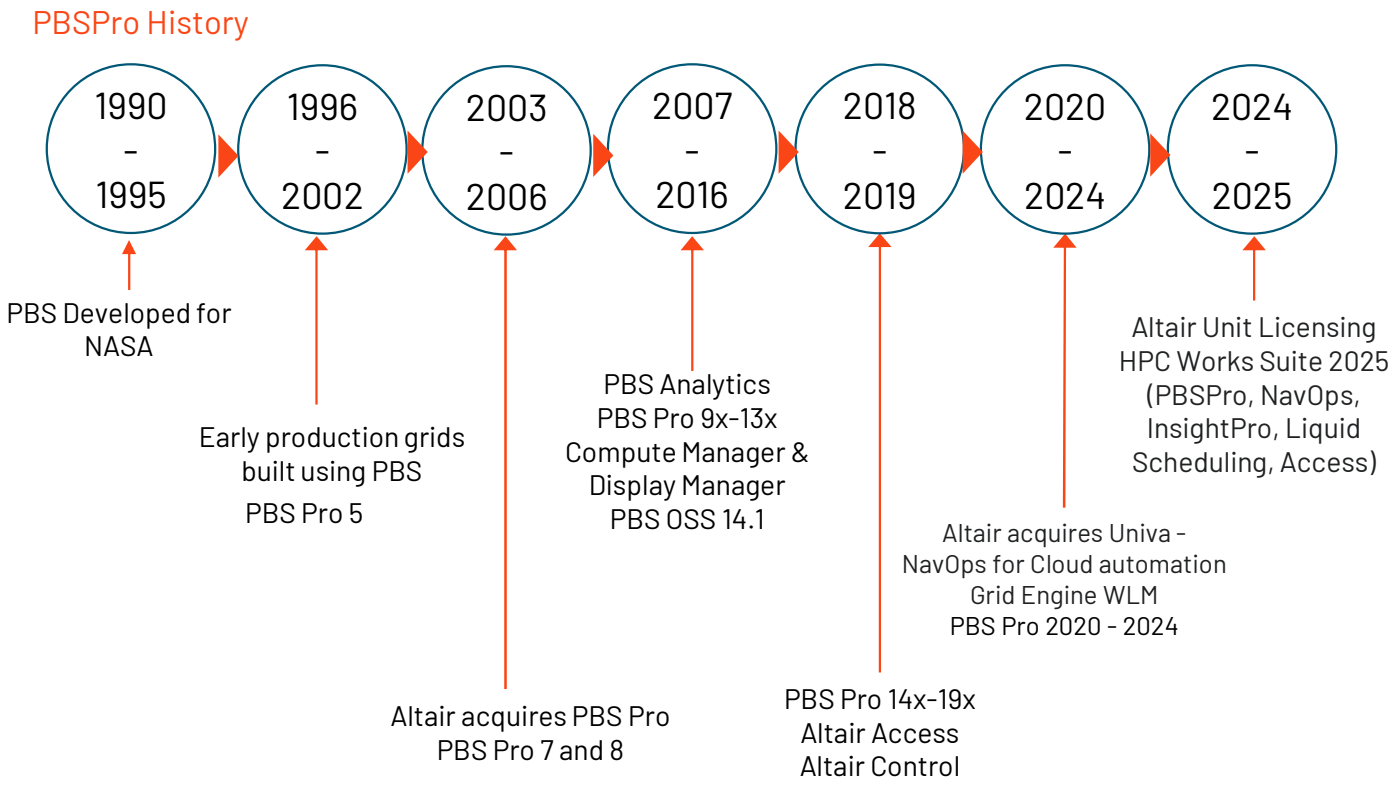
National Supercomputing Centre

Introduction to HPC and PBS Works



© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

HPC Works Suite



© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

HPC

Overview

HPC most generally refers to the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer to solve large problems in science, engineering, or business.

Need of a job scheduler:

A job scheduler is a computer application for controlling unattended background program execution (commonly called batch processing).

Today's job schedulers, often termed workload automation, typically provide a graphical user interface and a single point of control for definition and monitoring of background executions in a distributed network of computers.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Altair HPC Works

Altair HPCWorks enables efficient workload management and user productivity across environments—from small clusters to large-scale data centers and cloud platforms.

- **PBS Professional**: High-performance workload manager and job scheduler for HPC and high-throughput computing. Optimizes resource utilization, throughput, and job prioritization.
- **Altair Access**: Unified job submission and monitoring portal for engineers and researchers. Simplifies remote job execution, data handling, and integrates remote 3D visualization for advanced collaboration.
- **Altair Control**: Web-based interface for HPC system administration. Provides real-time monitoring, analytics, reporting, and management of jobs, queues and nodes.
- **Altair Budgets**: Comprehensive usage tracking and cost management tool. Enforces project- or user-level compute budgets to ensure fair usage and control expenses.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

What is PBS Professional?

Altair PBS Professional is a high-performance workload management system engineered to optimize resource utilization, streamline job scheduling, and simplify administration across HPC clusters, cloud environments, and supercomputing platforms. Designed for scalability and robustness, it supports a wide range of workloads—from massive, tightly coupled HPC applications to large volumes of high-throughput, single-core jobs.

Key Features:

- Policy-driven Scheduling : PBS Professional accelerates job execution and selects optimal job placement across diverse, broadly distributed resources. The scheduler is topology-aware and supports GPU scheduling.
- Allocation and Budget Management : Manage budgets across clusters by allocating users credits for PBS Professional workloads. Credits can be provided in one or more customizable currencies.
- Enterprise Resilience : Startup is fast and reliable, even for huge MPI jobs. PBS Professional is tested at tens of thousands of MPI ranks and minimizes delays caused by faulty nodes.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

PBS Professional

Components and Roles

Server

- Manages jobs, server queues and nodal attributes and settings
- Processes user and admin commands
- Daemon executes as **pbs_server.bin**

Scheduler

- Enforces scheduling policies
- Determines when and where jobs run
- Daemon executes as **pbs_sched**

MoM

- Executes the job scripts on behalf of the user
- Manages job processes
- Daemon executes as **pbs_mom**

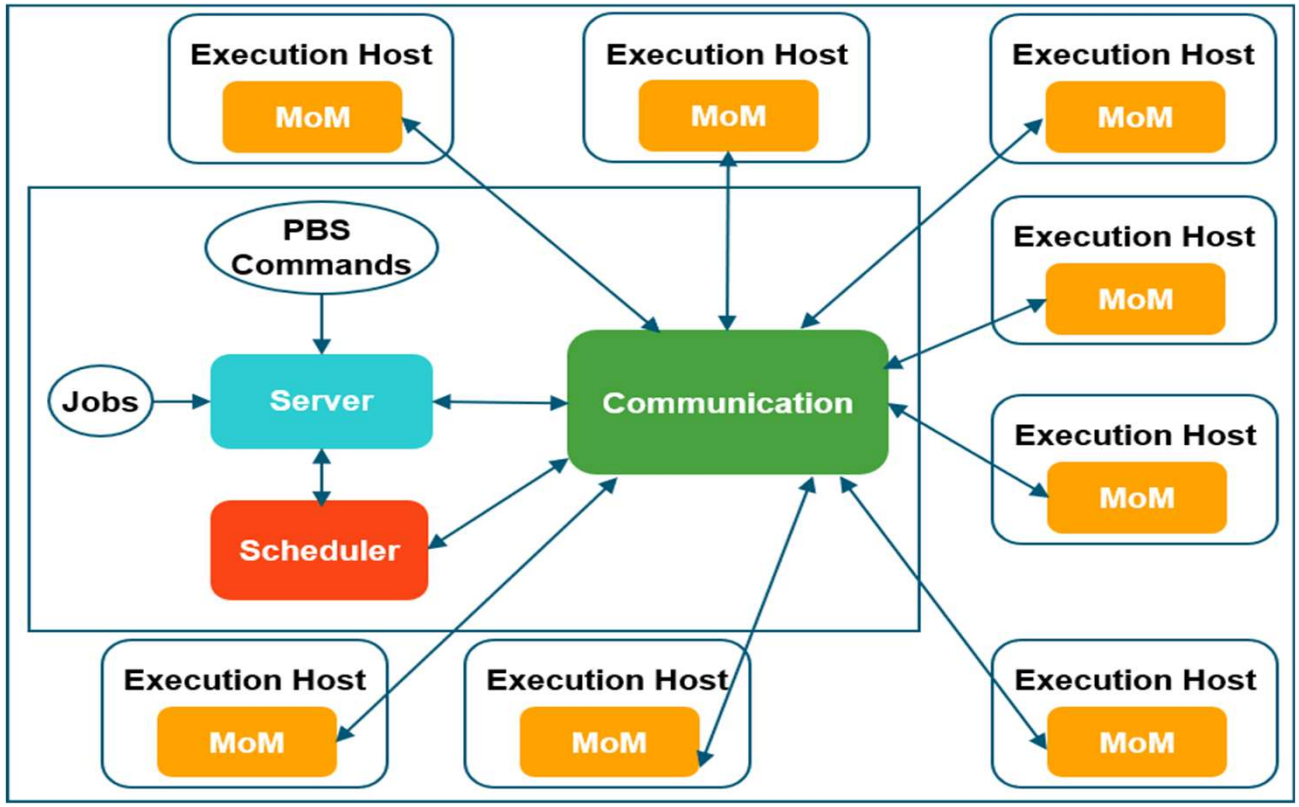
Communication

- Inter-daemon communication
- Larger systems can use multiple instances
- Daemon executes as **pbs_comm**

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

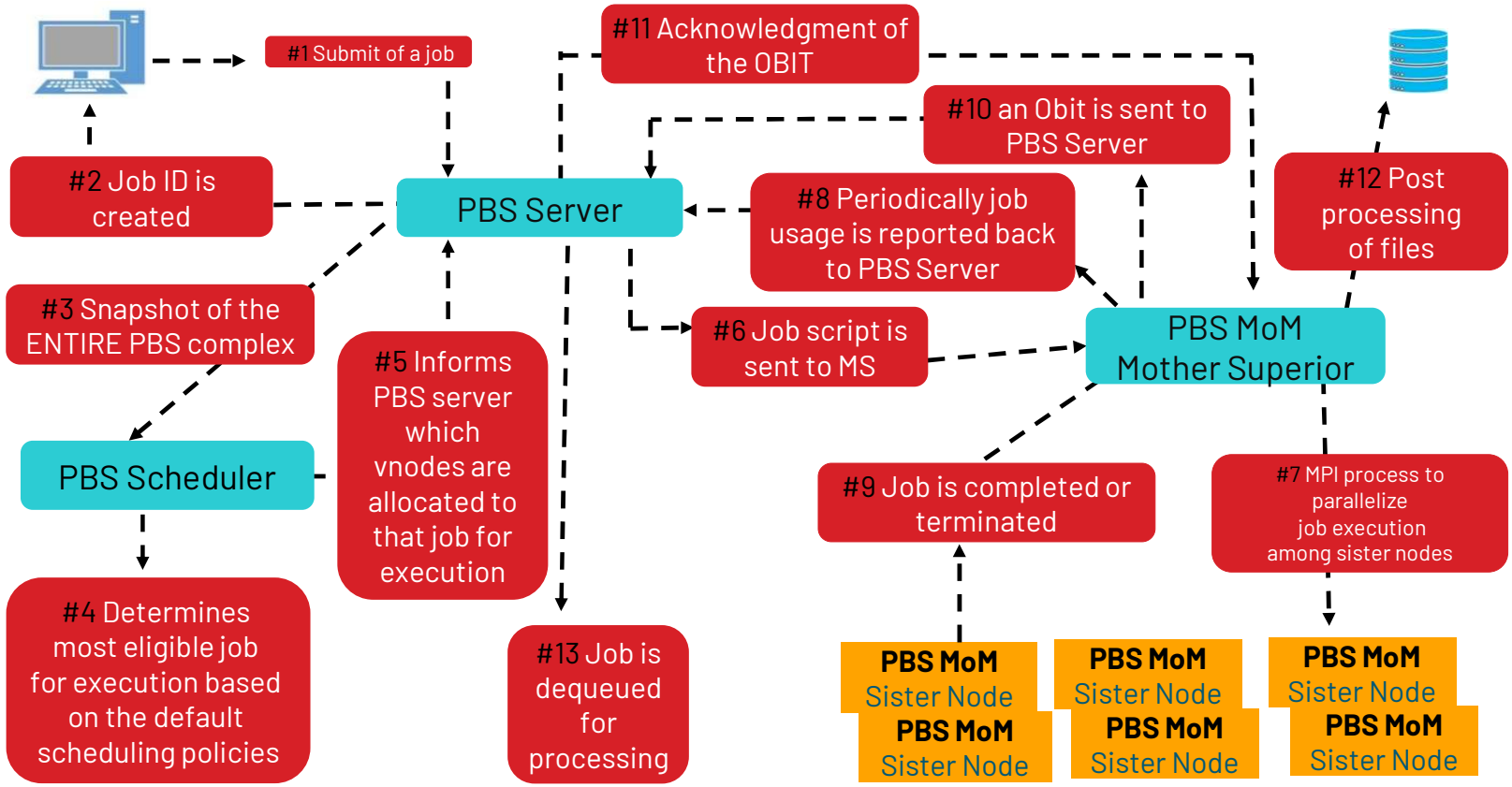
PBS Professional

Components and Roles



© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Lifecycle of a Parallel PBS Job



© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Resource usage monitoring and Isolation using cgroups

Why cgroups?

- Mechanism to identify collection of processes as a group
- Cannot be defeated by user job processes if set up by root
- Purposes:
  - Monitoring (what did my job really use?)
  - Resource usage limit enforcement (prevent jobs from using more than requested)
  - Job isolation (carve out private resources of system for each job)
- Example:
  - Ensure the jobs cannot impair the host OS or swap out its services
  - Ensure that if two MPI jobs share a host, they do not pin processes to the same CPUs
  - Limit access to GPU devices to only those assigned to the job

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

How PBS uses cgroups in job lifecycle

Overview

- Creates shepherds **per-job cgroups** through the job lifecycle
  - At execjob_begin event, creates cgroup for each job and sets resource limits
  - At execjob_launch (Task Management created job task) attaches job task starter child to cgroup
  - At execjob_attach (task created outside MoM to be attached to the job) also attaches designated processes to job cgroups – attaches all processes of designated SID
  - At execjob_epilogue, execjob_abort and execjob_end, cleans up processes and cgroups; marks a node offline if stuck processes are discovered
- Use of **pbs_attach**
  - **Example:** sshrc file to force compute node processes to attach to job
    - Prevents users from accessing nodes where they do not have a job. PBS_JOBID is required to access running job's node via SSH. Example: export PBS_JOBID=123456.pbs101
    - SSH sessions are attached automatically using TM API to its owner's job. This prevents any bypass methods to use unassigned resources
    - 'Automagical' MPI integration of PBS-unaware libraries

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

GPU isolation

- Fence off devices not assigned to job from job
  - Device isolation enforces job isolation even for ‘rogue’ jobs that do not listen to CUDE_VISIBLE_DEVICES
  - A Job only see the device ID which is assigned to it and usage of other IDs forcefully is prevented.
  - Hook will set CUDA_VISIBLE_DEVICES for all processes spawned through TM API
    - If not set automatically, need to read env file $PBS_NODEFILE.env

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Scheduling policy on ASPIRE2A

Fairshare

- **Concept**
  - A "fair" method for ordering the start times of jobs, using resource usage history
  - A **scheduling tool** which allocates certain percentages of the system to specified users
  - Ensures that jobs are run in the order of how deserving they are
- **Fairshare policy**
  - The default fairshare behavior gives all users with queued jobs an equal share of CPU time
  - An entity with a lot of current or recent usage has lower priority for starting jobs, but if the entity cuts resource usage, its priority goes back up after a few decay cycles
- **Basic outline of how fairshare works**
  - Scheduler collects usage from the server at every scheduling cycle
  - Default resource whose usage is tracked for fairshare is **ncpus*walltime**
  - Scheduler chooses which fairshare entity is most deserving
  - The job to be run next is selected from the set of jobs belonging to the most deserving entity, and then the next most deserving entity, and so on
    - On ASPIRE2A, **by_queue** is turned on, so that fairshare works queue by queue instead of on all jobs at once

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Scheduling policy on ASPIRE2A

- **Queue Sorting and Prioritization :** In ASPIRE2A, the scheduler evaluates jobs from the highest priority queue first, then moves on to the next highest priority queue.
- **Sorting Jobs According to a Formula :** Jobs are sorted on ASPIRE2A with a finer granularity by using a formula expression. Formula is applied for each job at the beginning of each scheduling cycle, ranking them internally from HIGH to LOW priority. Formula on ASPIRE2A: fairshare_factor+(1*jobprio)*0.99

Terms		Allowable Values			
Resources		ncpus	walltime	mem	cput
Custom numeric resources		Requested value of resources. Valid types: long | float | size			
Constants		NUM or NUM.NUM			
Attribute Values	**queue_priority**	Value of priority attribute for the queue in which the job resides			
	job_priority	Value of the job's priority attribute			
	eligible_time	Amount of wait time job has accrued while waiting for resources			
	fairshare_perc	Percentage of fairshare tree allotted to this job's entity			
	fairshare_tree_usage	The effective usage by the entity			
	fairshare_factor	Value allowing direct comparison between entities			

- **Dedicated Time Slots :** The dedicated times are set by admins for reserving the PBS Complex to do System hardware and software upgrades. The scheduler does not allow any jobs to be executed in that time frame for whatever other reasons, including the standing reservation jobs.
- **Preemptive Scheduling :** Preemption is disabled on ASPIRE2A

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Practice session details

- Please note that you must be connected to NSCC system (via VPN etc.,)
- Reservation queue : R123456
- Submitting to reservation queue
  - qsub –q R123456 -P resv
- Note: You may face issues in submitting jobs to reservation queue if your user ID is not added correctly. We can help to correct it.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Projects in ASPIRE2A

Personal projects

- Personal project naming convention: "personal-<userid>"
- Personal project quota is 100K / 10K SU life time.
- To submit jobs under personal project, user can use keyword "personal" and correct project name will be automatically added:
  - qsub -P personal -l walltime=10:00:00 myjob.sh

Registered projects

- Projects registered via NSCC project portal and approved by project admins. Project names are usually 8-digit numbers.
- The SU quota and duration (validity) is based on the request and approvals.
- To submit jobs under registered project:
  - qsub -P <project-name> -l walltime=10:00:00 myjob.sh

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

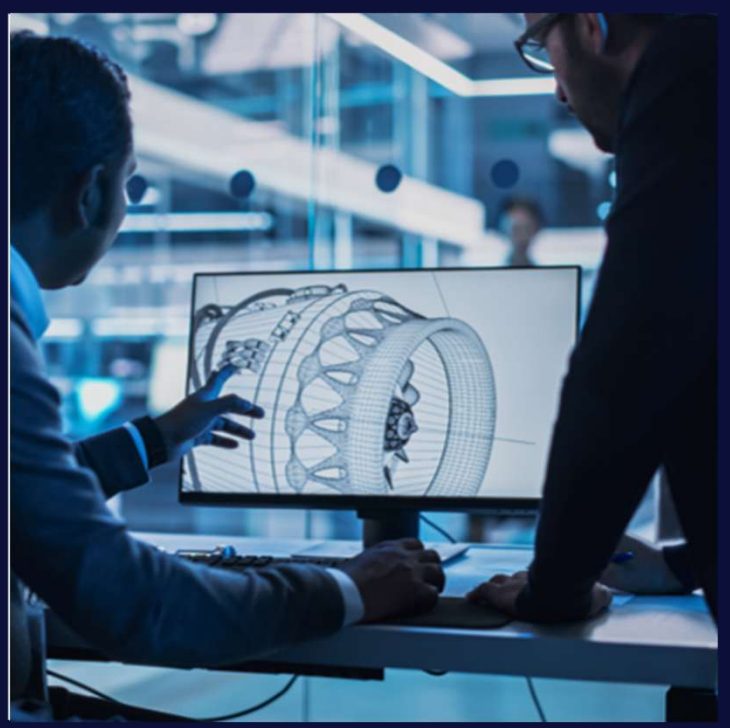
National Supercomputing Centre

Job Management



© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

National Supercomputing Centre



PBS job types

Submitting jobs

Understanding PBS Queues

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

PBS Job types

Batch Job

- Set of commands and/or applications you want to run on one or more execution host.
  - **Examples**: Python, Perl, Linux shell script
  - Maximum allowable job script is 100MB

```
#!/bin/bash
sleep 5
/home/altair/scripts/optistruct –cpu 2 handlebar.fem
```

Interactive Job

- The user's terminal input and output are connected to the execution host, like a login session
- Allows users to debug a job script
- Can be used to verify that a new application runs properly
  - **Usage: qsub –I <job attributes/resources>**

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Submitting PBS Jobs

Batch job script - qsub command

- Using the **qsub** command:
  - **Usage**: qsub <job attributes/resources> <job script>
  - **Example**: qsub my_script.sh

```
[user01@pbsworks ~]$ qsub my_script.sh
20.pbsworks
```

- If the job is accepted by PBS, a job identifier is returned
  - Comprised of the job number and the PBS server's host name



Note: If a job is rejected by the PBS Server, it **WILL NOT** return a job identifier, however internally the JOB SEQUENCE ID is incremented.
Largest possible job sequence ID is 12 digits: 999, 999,999,999; site administrators can limit the ID by setting the server level attribute **'max_job_sequence_id'**.
Resets to zero once reached

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Understanding PBS Queues

- PBS uses a resource-based scheduling system, where submitted jobs are held in a container waiting for execution.
  - This container is known as a **"queue"**
- There are two types of queues:
  - **Execution queue:** jobs in this queue are eligible for execution
    - Jobs waiting for execution
    - Jobs that are currently running
  - **Route queue:** Jobs in this queue are not eligible for execution
    - Routes jobs either to an execution queue or to another routing queue
    - Routes jobs to another PBS complex
- Queue attributes are set, during queue creation or later, such as:
  - Maximum number of jobs that can be running
  - Maximum number of queued jobs
  - Resources available at queue level
  - Which users/groups/hosts allowed to enqueue jobs (ACLs)

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Understanding PBS Queues

Queue Status

- To obtain status of all the queues in a PBS complex: **qstat –Q[f]**

Output of: **qstat –Q**

```
Queue              Max   Tot Ena Str   Que   Run   Hld   Wat   Trn   Ext Type
---------------- ----- ----- --- --- ----- ----- ----- ----- ----- ----- ----
workq                0     0 yes yes     0     0     0     0     0     0 Exec
```

Output of: **qstat –Qf**

```
Queue: workq
    queue_type = Execution
    total_jobs = 0
    state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0 Begun:0
    resources_assigned.ncpus = 0
    resources_assigned.nodect = 0
    enabled = True
    started = True
```



Note: Attributes that start with "resources_assigned.<res>" indicate how much of those resources are being consumed in that queue

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Queues in ASPIRE2A

- Users are allowed to submit their jobs to “normal” and “ai” routing queues or their reservation queues only. All other queues are for operations purpose. Users can't submit jobs to these queues directly.

Server	Routing queue Name	Exec Queue Name	Condition of routing	Restrictions (subject to change by admin as per need)	Priority	Walltime
pbs101 (Cray EX)	normal	q1	-eq 1 core	Max running jobs per user = 20	10	between 2hrs and 24hrs
		q2	-ge 2 cores, -le 64 cores	Max running jobs per user = 15	10	
		q3	-ge 65 cores, -le 127 cores (less than 1 nodes)		10	
		q4	128 cores (1 node)		40	
		q5	-ge 129 cores, -le 2048 cores (16 nodes)		60	
		q6	-ge 2049, -le 98304 cores (-gt 16 nodes, -lt 768 nodes)		80	
		qlong	max 128 cores (max 1 node)	Max running jobs per user = 5 Max running ncpus per user = 128	10	between 24hrs and 120hrs
		qdev	max 128 cores and 440gb memory	Max running jobs per user = 2 Max running ncpus overall in queue = 256	100	less than or equal to 2hrs
		g1	-eq 1 GPU	Max running jobs per user = 1	10	between 2hrs and 24hrs
		g2	-ge 2 GPUs, -le 3 GPUs	Max running jobs per user = 1	10	
		g3	-eq 4 GPUs (1 node)	Max running jobs per user = 2	20	
		g4	-gt 4 GPUs	Max running jobs per user = 1	30	
		glong	-le 64 GPUs cards (-le 16 nodes)	Max running jobs per user = 1	10	between 24hrs and 120hrs
		gdev	-ge 1 GPUs, le 4 GPUs and 440gb memory	Max running jobs per user = 2 Max running ngpus overall in queue = 16	100	less than or equal to 2hrs
	None (Managed by large memory routing hook)	l1	-gt 440GB -le 1TB mem in a chunk, -le128 cores total	Max running jobs per user = 2	10	less than or equal to 24hrs
		l2	-gt 1TB , -le 2TB mem in a chunk, -le128 cores total		20	
		l3	-gt 2TB , -le 4TB mem in a chunk, -le128 cores total		30	
pbs102 (AI)	ai	aiq1	-eq 1 GPU	Max running jobs per user = 2	10	between 2hrs and 24hrs
		aiq2	-ge 2 GPUs, -le 3 GPUs	Max running jobs per user = 2	10	
		aiq3	-eq 4 GPUs (1 node)	Max running jobs per user = 2	20	
		aiq4	-gt 4 GPUs	Max running jobs per user = 2	30	
		aiq5	-gt 4 GPUs (in a single chunk)	Max running jobs per user = 2	30	
		aidev	-ge 1 GPUs, -le 4 GPUs	Max running jobs per user = 2	100	less than or equal to 2hrs
		ailong	-ge 1 GPUs, -le 4 GPUs	Max running jobs per user = 2	10	between 24hrs and 120hrs

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

National Supercomputing Centre
SINGAPORE

Managing Jobs



USING QSTAT
JOB STATES
FINISHED JOB HISTORY
JOB COMMENTS

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

National Supercomputing Centre

Using QSTAT



© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Managing Jobs

Querying Jobs using qstat command

- To show status of list of current PBS jobs:
  - Using **qstat** command
  - **Usage: qstat <-a, -n, -s, ,-f , -x -1, -w, -r, -l ..>**
  - **Example:**

```
[use01@pbsworks ~]# qstat
Job id            Name             User         Time Use S Queue
----------------- ---------------- -----------  -------- - -----
6.pbsworks        test_script      user01       00:00:00 R workq
7.pbsworks        jobA             user02       00:00:00 R workq
8.pbsworks        test_2           user04              0 Q workq
9.pbsworks        test_script      user01       00:00:00 R workq
```



Note: In ASPIRE2A, the job history is set to 12 hours. If a job was deleted or completed in the past 12 hours, it can be listed via **qstat.** All previous jobs can't be listed.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Managing Jobs

qstat Output Format

```
[use01@pbsworks ~]# qstat
Job id            Name             User         Time Use S Queue
----------------- ---------------- -----------  -------- - -----
6.pbsworks        test_script      user01       00:00:00 R workq
7.pbsworks        jobA             user02       00:00:00 R workq
8.pbsworks        test_2           user04              0 Q workq
9.pbsworks        test_script      user01       00:00:00 R workq
```

Column	Description
Job id	Job id assigned by PBS
Name	Job name; default is job script name
User	Username of job owner
Time Use	CPU time used by the job
S	Job's state
Queue	Queue in which the job resides

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Managing Jobs

Commonly used qstat Options

- **'qstat -a'** Job name, session ID, # nodes req'd, #ncpus req'd, req'd mem, req'd time, elapsed time

```
pbsworks:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
20.pbsworks     user01   workq    STDIN        2428   1   1    --     --  R 00:00
```

- **'qstat -s'** Same as option **-a**, but with comments

Most informative option

```
pbsworks:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
20.pbsworks     user01   workq    STDIN        2428   1   1    --     --  R 00:00
   Job run at Sat Jul 15 at 10:53 on (node01:ncpus=1)
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Managing Jobs

Commonly used qstat Options

- **'qstat -n'** Same as option **–a**, but indicates which vnode(s) the job is running on

```
pbsworks:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
20.pbsworks     user01   workq    STDIN        2428   1   1    --     --  R 00:00
   node01/0
```

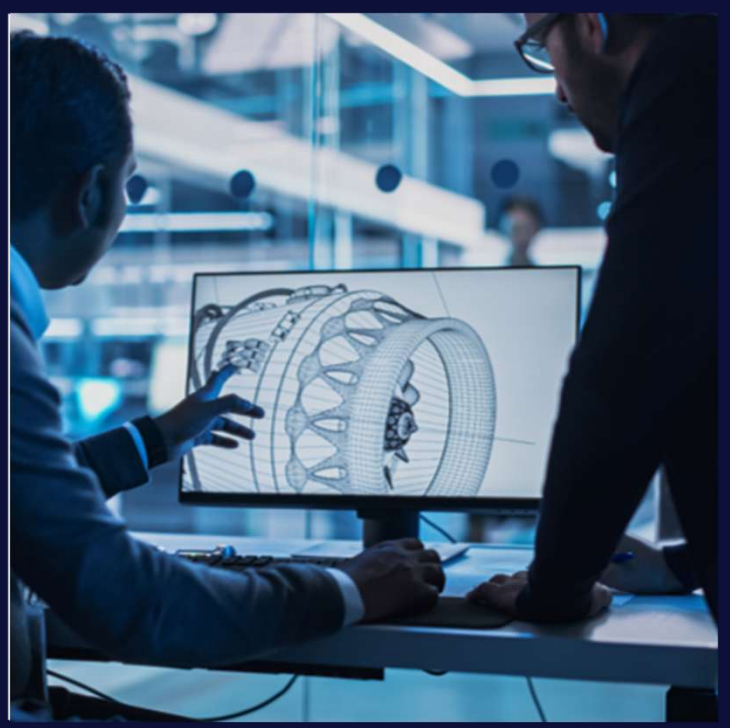


Note: Using the '**-1**' option outputs each entry on a single line instead of wrapping around

Using the '**-w**' displays the full output of individual fields

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

National Supercomputing Centre



JOB STATES

FINISHED JOB HISTORY

JOB COMMENTS

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Managing Jobs

Job States

States	Description
E	Job is exiting after having run
F	Job is finished. Job has completed execution, job failed during execution, or job was deleted
H	Job is held
M	Job was moved to another PBS complex
Q	Job is queued, eligible to run or be routed
R	Job is running
S	Job is suspended by PBS
T	Job is in transition
W	Job is waiting for its requested execution time to be reached, or job specified a stage-in request which failed for some reason

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Managing Jobs

Finished Job History

- To view only jobs that have been deleted, moved, or finished: **'qstat –H'**

```
                                                                  Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
10.pbsworks     user01   workq    STDIN        1451   1   1    --     --  F 00:00
11.pbsworks     user01   workq    STDIN        1465   1   1    --     --  F 00:00
12.pbsworks     user01   workq    STDIN        1762   1   1    --     --  F 00:32
```

- To view all jobs, regardless of state: **'qstat –x'**

```
Job id            Name             User              Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
9.pbsworks        STDIN            user01            00:00:00 F workq
10.pbsworks       STDIN            user01            00:00:00 F workq
11.pbsworks       STDIN            user01            00:00:00 F workq
12.pbsworks       STDIN            user01            00:00:00 F workq
```



Note: On ASPIRE2A, when a job has been finished, deleted, or moved, it is only shown for the next 12 hours in the output of **qstat.**

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Managing Jobs

Job Comments

- **Job 1**: Before the server sends the job to an execution host
- **Job 2**: After the server gets confirmation from MoM
- **Job 3**: Job is rejected by the MoM
- **Job 4**: Job is finished
- **Job 5**: Job is in queue due to insufficient amount of resources on compute nodes

```
[user01@pbsworks]$ qstat -x -s

pbsworks:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
1.pbsworks      user01.  workq    STDIN        2320   2   2    --     --  R 00:00
   Job was sent for execution at Mon Jul 10 at 12:45 on (node01:ncpus=1)
2.pbsworks      user01   workq    STDIN        1438   1   1    --     --  R 00:00
   Job run at Sat Jul 15 at 08:03 on (node01:ncpus=1)
3.pbsworks      user01   workq    STDIN        1465   1   1    --     --  F 00:00
   Not Running: PBS Error: Execution server rejected request
4.pbsworks      user01   workq    STDIN        1762   1   1    --     --  F 00:00
   Job run at Sat Jul 15 at 08:03 on (node01:ncpus=1) and finished
5.pbsworks      manish   workq    STDIN         --   15  30    --     --  Q   --
   Can Never Run: Insufficient amount of resource: ncpus (R: 30 A: 2 T: 2)
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

National Supercomputing Centre
SINGAPORE

Job Attributes



SETTING ATTRIBUTES
VIEWING ATTRIBUTES
USING PBS DIRECTIVES
JOB ENVIRONMENT VARIABLES

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Setting Job Attributes

Job Submission (qsub) Options

Options	Value/Format	Description
-A	<alphanumeric>	Specifying accounting string
-a	[[[[YYYY]MM]DD]hhmm[.SS]]	Deferring execution
-C	"<string>"	Changing the job's PBS directive prefix
-c	[c | c=<minutes> | w |w=<minutes> | n | s | u>	Specifying job checkpoint interval
-e	</path/filename>	Specifying path for output and error files
-f		Prevents qsub from spawning a background process
-G	[<path to GUI application or script>]	Starts a GUI session (Interactive jobs only)
-h		Holding a job (delaying execution)
-I | -X		Interactive jobs with X11 forwarding enabled
-J	<X-Y[:z]>	Defining job array
-j	[o | e]	Merging output and error files
-k	[o | e]	Retaining output and error files on execution host
-l	<resource_list>	Requesting job resources
-M	<id@domain.xxx>	Setting email recipient list

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Setting Job Attributes

Job Submission (qsub) Options

Options	Value/Format	Description
-m	<string>	Specifying email notification
-N	<alphanumeric>	Specifying a job name
-o	</path/filename>	Specifying path for output and error files
-p	<numeric>	Setting your job's priority
-P	<alphanumeric>	Specifying the project name
-q	<queue_name>	Specifying the queue or server name
-r	[Y | N]	Marking a job as rerunnable or not
-R	e | o	Avoiding creation of STDOUT and/or STDERR
-S	</path/shell_name>	Specifying scripting language to use
-u	<user_name>	Specifying job's user ID
-V		Exporting ALL environment variables
-v	<alphanumeric>	Expanding specific environment variables

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Setting Job Attributes

Job Submission (qsub) Options

Options	Value/Format	Description
-W block=	<Boolean>	Requesting qsub wait for job completion
-W create_resv_from_job=	<Boolean>	Create job-specific reservation queue on job start
-W depend=	<dependency type>	Specifying job dependencies
-W group_list=	<group@host...>	Specifying job owner's group ID
-W pwd=	"<alphanumeric>"	Per-job password method
-W release_nodes_on_stageout=	<Boolean>	Requesting to release vnodes during file stage-out
-W run_count=	<numeric>	Controlling number of times a job is re-ran
-W sandbox=	[HOME | PRIVATE]	Staging and execution directory: user's home vs. job-specific
-W stagein=	<list>	Input file staging
-W stageout=	<list>	Output file staging
-W umask=	<numeric>	Change a Linux job's umask
-X		X output from interactive jobs
-z		Suppressing job identifier

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Attributes

Viewing Job Attributes

- To view **full** job attributes for a particular job, use the **qstat** command
  - **Usage:** qstat –f <job ID> → for finished jobs must add -x option
  - **Example**: **qstat –xf 10.pbsworks**

Expected output will depend on PBS roles

```
Job Id: 10.pbsworks
    Job_Name = my_script.sh
    Job_Owner = user01@pbsworks
    resources_used.cpupercent = 0
    resources_used.cput = 00:00:00
    resources_used.mem = 0kb
    resources_used.ncpus = 1
    resources_used.vmem = 0kb
    resources_used.walltime = 00:00:00
    job_state = F
    queue = workq
    server = pbsworks.local
    Checkpoint = u
    ctime = Thu Mar 19 15:09:46 2018
    Error_Path = pbsworks:/home/user01/my_script.sh.e10
    exec_host = pbsworks/0
    exec_vnode = (pbsworks:ncpus=1)
    Hold_Types = n
    Join_Path = n
    Keep_Files = n
    Mail_Points = a
    mtime = Thu Mar 19 15:09:46 2018
    Output_Path =
pbsworks:/home/user01/my_script.sh.o10
    Priority = 0
    qtime = Thu Mar 19 15:09:46 2018
    Rerunable = True
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Attributes

Viewing Job Attributes

Ordinary users running qstat cannot see custom resources that are invisible to users

```
Resource_List.ncpus = 1
Resource_List.nodect = 1
Resource_List.select = 1:ncpus=1
schedselect = 1:ncpus=1
stime = Thu Mar 19 15:09:46 2018
session_id = 7084
jobdir = /home/user01
substate = 93
Variable_List = PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,
  PBS_O_HOME=/home/user01,PBS_O_LOGNAME=user01,
  PBS_O_WORKDIR=/home/user01,PBS_O_LANG=en_US.UTF-8,
  PBS_O_PATH=/opt/pbs/default/bin:/opt/pbs/default/sbin:/usr/lib64/qt-3.
  3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/hom
  e/user01/bin,PBS_O_MAIL=/var/spool/mail/user01,PBS_O_QUEUE=workq,
  PBS_O_HOST=pbsworks
euser = user01
egroup = user01
hashname = 10.pbsworks.local
queue_rank = 11
queue_type = E
comment = Job run at Thu Mar 19 at 15:09 on (pbsworks:ncpus=1)
etime = Thu Mar 19 15:09:46 2018
run_count = 1
Stageout_status = 1
Exit_status = 127
Submit_arguments = my_script.sh
history_timestamp = 1426802986
project = _pbs_project_default
run_version = 1
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Attributes

qstat JSON Format

```
[user@pbsworks ~]$ qstat -f -F json 13
{
    "timestamp":1523291708,
    "pbs_version":"18.2.0.20180209012644",
    "pbs_server":"pbsworks",
    "Jobs":{
        "13.pbsworks":{
            "Job_Name":"STDIN",
            "Job_Owner":"user01@pbsworks",
            "resources_used":{
                "cpupercent":0,
                "cput":"00:00:00",
                "mem":"348kb",
                "ncpus":1,
                "vmem":"4316kb",
                "walltime":"00:00:49"
            },
        . . .
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Attributes

qstat DSV Format

Separated by '|"

Users may change the delimiter using –F dsv <delimiter> <job sequence id>

```
[user01@pbsworks ~]$ qstat -f -F dsv 13

Job
Id:13.pbsworks|Job_Name=STDIN|Job_Owner=user01@pbsworks|resources_used.cpupercen
t=0|resources_used.cput=00:00:00|resources_used.mem=348kb|resources_used.ncpus=1
|resources_used.vmem=4316kb|resources_used.walltime=00:02:31|job_state=R|queue=w
orkq|server=pbsworks|Checkpoint=u|ctime=Mon Apr  9 09:34:07
2018|Error_Path=pbsworks:/home/user01/STDIN.e13|exec_host=pbsworks/0|exec_vnode=
(pbsworks:ncpus=1)|Hold_Types=n|Join_Path=n|Keep_Files=n|Mail_Points=a|mtime=Mon
Apr  9 09:36:38
2018|Output_Path=pbsworks:/home/user01/STDIN.o13|Priority=0|qtime=Mon Apr  9
09:34:07
2018|Rerunable=True|Resource_List.ncpus=1|Resource_List.nodect=1|Resource_List.s
elect=1:ncpus=1|stime=Mon Apr  9 09:34:07
2018|session_id=8859|jobdir=/home/user01|substate=42|Variable_List=PBS_O_HOME=/h
ome/user01,PBS_O_LANG=en_US.UTF-8,PBS_O_LOGNAME=user01,PBS_O_PATH=/usr/lib64/qt-
3.3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/opt/pbs/bin:/hom
e/user01/.local/bin:/home/user01/bin,PBS_O_MAIL=/var/spool/mail/user01,PBS_O_SHE
LL=/bin/bash,PBS_O_WORKDIR=/home/user01,PBS_O_SYSTEM=Linux,PBS_O_QUEUE=workq,PBS
_O_HOST=pbsworks|comment=Job run at Mon Apr 09 at 09:34 on
(pbsworks:ncpus=1)|etime=Mon Apr  9 09:34:07 2018|run_count=1|Submit_arguments=-
- /bin/sleep 500|executable=<jsdl-hpcpa:Executable>/bin/sleep</jsdl-
hpcpa:Executable>|argument_list=<jsdl-hpcpa:Argument>500</jsdl-
hpcpa:Argument>|project=_pbs_project_default
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Attributes

Using PBS Directives

- Job attributes can be set in two different ways:
  - **Method 1**: that the **qsub** command line
    qsub –N <job_name> <job_script>
  - **Method 2**: within a job script as a PBS directive

```
#!/bin/bash
#PBS –N test_run_01
#PBS –l select=4:ncpus=4:mem=64GB
#PBS -j oe
#PBS –o /home/user01/OUTPUTS
## this is a comment
/bin/sleep 60
```



Note: PBS expects directives before any executable code; comment, blank, **and white lines** acceptable.

Once a line of code is executed, the interpreter stops processing directives.

Command line arguments override PBS directives.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Attributes

Job Environment Variables

- At the time a job is launched into execution, PBS defines multiple environment variables, which can be used from within the submission script to define the correct workflow of the job.

Execute 'env' to view the variables

Variable	Description
PBS_JOBID	The identifier assigned to the job
PBS_JOBNAME	The job name supplied by the user
PBS_NODEFILE	The filename containing a list of vnodes assigned to the job
PBS_O_PATH	Value of the PATH from submission environment
PBS_O_WORKDIR	The absolute path of directory where qsub is to execute
PBS_TMPDIR	The job-specific temporary directory for this job

```
#!/bin/bash
#PBS -l select=8:ncpus=16:mem=32GB
#PBS -l walltime=5:00:00
#PBS -N sim_run_01
#PBS -M user01@altair.com
#PBS -m abe

cd $PBS_TMPDIR
mpirun -np 128 my_script.sh
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

National Supercomputing Centre

Requesting Job Resources



UNDERSTANDING RESOURCES
HOST-BASED RESOURCES
JOB-WIDE RESOURCES
USING MPI WITH PBS
MULTITHREADED APPLICATIONS
RUNNING JOB IN A CONTAINER

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Requesting Job Resources

Understanding Resources

- What are resources?
  - Applications sometimes need certain types and amounts of resources in order to run jobs more efficiently
    - CPUs, memory, architecture, scratch space, GPUs, application licenses, …
- Why is it necessary for PBS to know about these resources?
  - Helps the scheduler to match the resources requested by the job with available resources
  - Ensures the job has enough resources to run properly
  - Properly distributes parallel resources over compute nodes
  - Avoids having jobs attempt to consume the same set of resources

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Requesting Job Resources

Understanding Resources

- PBS defines these resources as "host-level" and "job-wide" resources; host-level resources are specified in "chunks"
  - "host-level" resources : Resources that are associated with hosts, such as ncpus, mem, ngpus. These resources apply to individual job processes
  - "job-wide" resources : Resources that are associated with the entire PBS complex, not a specific host, like project, walltime, etc.. These resources apply to the entire job.

- Built-in resources and custom resources
  - Built in resources are out-of-the box resources like ncpus, mem, etc., and are available in all PBS installations however, custom resources are site-specific and are created by admins of a cluster. Like node_pool, container_engine, etc.. These resources are usually not required by users to be manipulated and are there for operations purposes. Users are strongly discouraged from changing any custom resource of their job which they are not sure of.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Requesting Job Resources

Host-based Resources

- Requesting resources in chunks
  - Smallest set of resources
    - Allocated to part of a job, such as one job process
    - Each chunk is allocated to a single job
  - Chunks are host/vnode level resources
    - Using select statement: qsub –l select=<numerical>:<res1>=<value>:<res2>=<value>…

Colon-delimited

- **Example:**
  - An application requires a total of 32 CPUs and 64GB memory in 4 equal chunks
    - Per chunk: 8 CPUs and 16GB memory

qsub –l select=4:ncpus=8:mem=16GB

This does not guarantee the job will get 4 execution nodes

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Requesting Job Resources

Job-wide Resources

- Requesting **job-wide resources**
  - Requesting resources at server or queue level
  - Resources that are not tied to specific host(s)/vnode(s)
  - Resources that are requested outside a select statement
    - walltime
    - cput
    - Server/queue level resources
    - Custom resources that are not node-based resource

- **Examples**:
  - To request 1 hour of walltime:
    - **qsub –l select=1:ncpus=1:mem=100MB -l walltime=01:00:00**
  - To request 3 hours of CPU time and 2 hours of walltime:
    - **qsub –l select=1:ncpus=1:mem=100MB -l project=personal,walltime=02:00:00**

Comma-delimited

Note: You cannot request a job-wide resource in chunks

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Script Example

Serial Jobs

- Serial jobs are meant to run on a single execution host and single core
- To submit an serial job, simply request a single chunk and single cpu
  - **Example**: **qsub –l select=1:ncpus=1:mem=3GB**

```
#!/bin/bash
#PBS -q normal
#PBS -l select=1:ncpus=1:mem=3gb
#PBS -l walltime=2:00:00
#PBS -P <projectid>
#PBS -j oe
#PBS -N test-serial

cd $PBS_O_WORKDIR || exit $?          #Change current directory to submission directory
echo “hello world”
sleep 10
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Using MPI with PBS

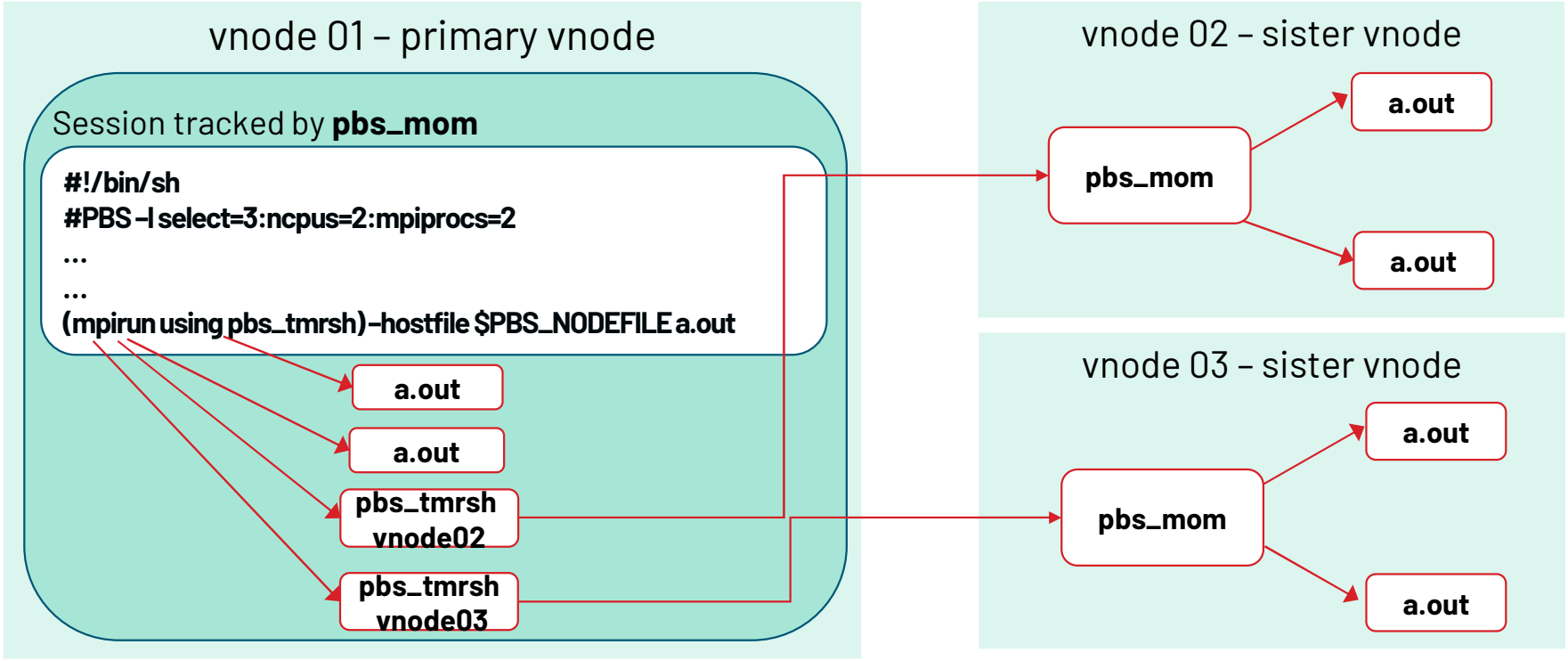
Concept

- Some **MPIs** are **integrated** with PBS
  - When **jobs run under an integrated MPI**, PBS can:
    - Track resource usage
    - Signal job processes
    - Perform accounting for all processes of the job
  - When **jobs run under non-integrated MPI**, PBS doesn't have the same control:
    - Limited to managing the job only on the primary node
    - Resource tracking, job signaling, and accounting happen for the processes on the primary node only
- PBS provides integration with various **MPIs**
  - OpenMPI has "built-in" integration with PBS
  - Intel MPI can be integrated with PBS using environment variables in the job's environment
  - Other MPIs can be integrated with PBS using a wrapper script that calls pbs_attach
    - Check with your PBS administrator about which MPIs are integrated at your PBS site

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Using MPI with PBS

Using pbs_tmrsh for MPI Integration



© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Supported MPI Libraries

MPI	Version	Platform
MPICH (Deprecated)*	1.2.5, 1.2.6, and 1.2.7	Linux
MPICH2	1.0.3, 1.0.5, 1.0.7	Linux
MPICH-GM (Deprecated)*	mpich-1.2.6.14b	Linux
Intel MPI	2.0.022, 3, and 4.0.3	Linux
HP MPI	2.0.0	Linux
Platform MPI	8.0	Linux
HPE SGI MPI (MPT)		Linux on SGI platforms, including over InfiniBand
MVAPICH (Deprecated)*	1.2.7	Linux
MVAPICH2	1.7	Linux
Tightly-/Self-Integrated with PBS		
OpenMPI	1.4.2	

* Wrapper is deprecated as of 14.2.1

Reference: PBS Professional 2022.1 and 2021.1 Release Notes

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Requesting Job Resources

Submitting Multiprocessor Jobs

- **Job node file**
  - Text file containing the name(s) of the host(s) containing the vnode(s) allocated to that job
    - Defaults to 1 line per MPI process for each chunk requesting CPUs, 0 for chunks not requesting CPUs.
    - Primary execution host is the first host listed
  - Created by the MoM on the primary execution host
    - Located in **PBS_HOME/aux/**<job_id>
  - **$PBS_NODEFILE**
    - An environment variable that can be used within the job script

```
node01
node02
node03
node04
```

```
#!/bin/bash
#PBS -N testrun01
#PBS -l select=6:ncpus=32:mem=128gb
#PBS -o /home/user01/my_o_files
#PBS -e /home/user01/my_e_files
#PBS -m be
mpirun -np $MPI_NPROCS -machinefile $PBS_NODEFILE ./my_scripts
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Requesting Job Resources

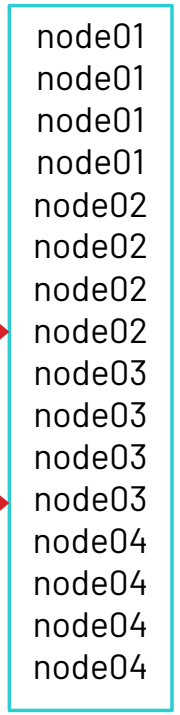
Submitting Multiprocessor Jobs

- Requesting number of MPI processes per chunk
  - **By default:**
    - 1 MPI processes per chunk

- Use the **mpiprocs** resource to set other than the defaults
  - **Example**:
    - **qsub -l select=2:ncpus=2:mpiprocs=2**

      node01
      node01
      node02
      node02

    - **qsub -l select=4:ncpus=2:mpiprocs=4**
    - **qsub -l select=4:mpiprocs=4**



© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Script Example

Multi node MPI Jobs

- MPI jobs are meant to run on multiple execution host
- To submit an MPI job, request more than one chunk of full node
  - **Example**: **qsub –l select=32:ncpus=128:mem=400GB**

```
#!/bin/bash
#PBS -q normal
##Requesting a full node with 128 CPU
#PBS -l select=5:ncpus=128:mpiprocs=128:mem=440gb
#PBS -l walltime=8:00:00
#PBS -P <projectid>
#PBS -j oe
#PBS -N gromacs_test

cd $PBS_O_WORKDIR || exit $?                                  #Change current directory to submission directory
module load gromacs/2023.2-gnu                                #Load correct environment
#export PATH=</absolute/path/to/gmx_mpi/bin>:$PATH            #Add path of executable if using self compiled version
#execute workload
mpirun -np 640 -v gmx_mpi mdrun -s benchMEM.tpr -nsteps -1 -maxh 1.0 -resethway -noconfout
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Requesting Job Resources

Submitting Multithreaded applications

- Requesting **number of threads per process**
  - By default, MoM sets the value of **OMP_NUM_THREADS** based on the ncpus requested in first chunk of the select statement
  - To set a non-default value for **OMP_NUM_THREADS** use the **ompthreads** attribute
  - If you do not request either ncpus or ompthreads for the first chunk of the select statement, then **OMP_NUM_THREADS** is set to 1.
  - You cannot directly set the value of the **OMP_NUM_THREADS** environment variable; MoM will override any setting you attempt.

  - **Example**:
    - To request more threads then ncpus:
      - **qsub -l select=1:ncpus=8:ompthreads=16**
    - To request less threads than ncpus:
      - **qsub -l select=1:ncpus=16:ompthreads=8**

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Script Example

SMP Jobs

- SMP jobs are meant to run on a single execution host
- To submit an SMP job, simply request a single chunk
  - **Example:** **qsub –l select=1:ncpus=8:mem=32GB**

```
#!/bin/bash
#PBS -q normal
## Requesting a full node with 128 CPU. By default ompthreads value is set to ncpus
#PBS -l select=1:ncpus=128:mem=400gb
## ompthreads can be used in select statement to control number of threads
## Ex: select=1:ncpus=128:ompthreads=64:mem=400gb
#PBS -l walltime=14:00:00
#PBS -P <projectid>
#PBS -j oe
#PBS -N test-mthread

cd $PBS_O_WORKDIR || exit $?                #Change current directory to submission directory
echo $OMP_NUM_THREADS                       #checking the value of OMP threads
./mthread_gauss_blur.out 2 20 >& mthread_job_out.txt  #Multithread program that applies Gaussian blur on images
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Script Example

SMP Jobs

- Multiple independent SMP workloads can also be executed in same job instead of requesting separate job for each, to reduce overall wait time and execution time.
  - **Example**:

```
#!/bin/bash
#PBS -q normal
##Requesting a full node with 128 CPU
#PBS -l select=1:ncpus=128:ompthreads=64:mem=400gb
#PBS -l walltime=14:00:00
#PBS -P <projectid>
#PBS -j oe
#PBS -N test-mthread

cd $PBS_O_WORKDIR || exit $?                #Change current directory to submission directory
./mthread_gauss_blur.out 2 20 > mthread_job_out1.txt 2>&1 & PD1=$!  #execute workload 1
./mthread_gauss_blur.out 2 20 > mthread_job_out2.txt 2>&1 & PD2=$!  #execute workload 2

wait $PD1                           #wait for workload 1 to complete
wait $PD2                           #wait for workload 2 to complete
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Running Job in a Container

Singularity

- PBS supports running multi-vnode, multi-host, and interactive jobs in Singularity containers on ASPIRE2A
- If you do not specify a script, for example "qsub -l container_image=hello-world", qsub asks you interactively for a script.
- If you supply a script to qsub, PBS runs the script inside the specified container.
- For a multi-host job, you can use any version of OpenMPI with containers.
- PBS runs an infinite-duration sleep command in the container to keep the container alive.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Running Job in a Container

Singularity

- **Requesting "singularity" Container Engine:**
  - You can specify a container engine by requesting a resource whose value is set to that engine. On ASPIRE2A, the container engine available is "singularity".
  - To request a container engine in your job resource specification, use the host level resource **"container_engine= singularity".** For example:
    - #PBS -l select=1:ncpus=1:mem=10gb:container_engine=singularity
- **Requesting a Container Image:**
  - You request a container image for your job via **-l container_image=<container image>.** For Example:
    - #PBS -l container_image="/home/username/rocker-geospatial.sif"

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Running Job in a Container

Singularity

- **Passing Environment Variables Into Containers:**
  - To pass environment variables directly to PBS, use qsub's option -v <environment variable list>. The –env argument is not supported for passing environment variables into containers. For Example:
    - #PBS -v "USER=username, VAR=Value"
- **Specifying Additional Arguments to Container Engine:**
  - You can specify additional arguments to the container engine via the PBS_CONTAINER_ARGS environment variable, which is a semicolon-separated list. For example, to specify --shm-size to be 1GB and --tmpfs to be "/run:rw,noexec,nosuid,size=65536k":
    - #PBS -v "USER=username, VAR=Value, PBS_CONTAINER_ARGS='--bind=/home:/home;--bind=/scratch:/scratch'"
- **Adding Job Owner to Secondary Groups in Containers**
  - Singularity automatically adds the job owner to all groups.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Running Job in a Container

Running Single-vnode Single-host Jobs in Singularity Containers

- In addition to using PBS to launch your containers, you can always run a single-vnode job in a single Singularity container by prepending your scripts, executables, or commands with the Singularity binary.

```
#!/bin/bash
#PBS -q normal
#PBS -l select=1:ncpus=128:mem=400g
#PBS -l walltime=24:00:00
#PBS -P <projectid>

module load singularity

# Change to directory where job was submitted
cd "$PBS_O_WORKDIR" || exit $?

# Specify which singularity image to use for container
image="/app/apps/containers/ubuntu/ubuntu_v22.04.1_X11.sif"

# Pass the commands that you wish to run inside the container
singularity exec --nv $image python scripts.py <args>
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Distribute tasks to vnodes under PBS

"pbsdsh" command

- The pbsdsh command allows you to distribute and execute a task on each of the vnodes assigned to your job by executing (spawning) the application on each vnode. The pbsdsh command uses the PBS Task Manager, or TM, to distribute the program on the allocated vnodes.
- pbsdsh will spawn the program on all vnodes allocated to the PBS job (one task for each entry in the job's exec_host attribute). The spawns take place concurrently; all execute at (about) the same time.
- Options:
  -c <copies> : The program is spawned <*copies*> times on the vnodes allocated, one per vnode. (unless *copies* is greater than the number of vnodes.).
  -n <vnode index> : The program is spawned only on a single vnode, which is the vnode index-th vnode allocated.
  -o : No obit request is made for spawned tasks. The program does not wait for the tasks to finish.
  -s : The program is run in turn on each vnode, one after the other.
  -v : Produces verbose output about error conditions and task exit status.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Distribute tasks to vnodes under PBS

"pbsdsh" command

- Operands:
  program : The first operand, program, is the program to execute. The double dash must precede program under Linux.
  program args : Additional operands, program args, are passed as arguments to the program.

```
#!/bin/bash
#PBS -q normal
#PBS -l select=30:ncpus=128:mem=400g
#PBS -l walltime=24:00:00
#PBS -P <projectid>

# Change to directory where job was submitted
cd "$PBS_O_WORKDIR" || exit $?

pbsdsh -- ./myapp app-arg1
```



Note: In this example, PBS will spawn "myapp" binary on 30 nodes. "myapp" can also be replaced with a shell / python script containing personalized sequence of commands to run multi-node singularity container workload, Distributed PyTorch Script, Job farming, etc..

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Transfer or download data using PBS Job

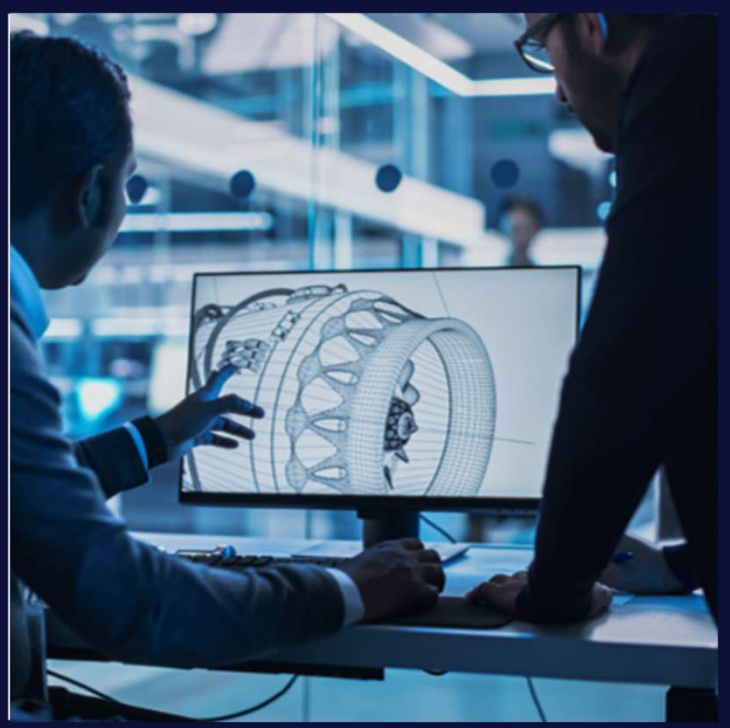
Sample Job Script

- Users can use PBS jobs (Batch or Interactive) to carry out heavy data movement tasks. This is helpful since login nodes are not designed to carry out such tasks and impacts end user usability.
- Users can use tools like wget, cp, rsync, etc. or any application installed in user's home for transfer purpose.

```
#!/bin/bash
#PBS -q normal
#request enough resources to carry out the task
#PBS -l select=1:ncpus=16:mem=55gb
#request enough time to finish the task
#PBS -l walltime=14:00:00
#PBS -P <projectid>
#PBS -j oe
#PBS -N transfer_data
cd $PBS_O_WORKDIR || exit $?                #Change current directory to submission directory
# Execute transfer task
#wget https://webaddress.com/path/to/file.zip
#cp -a /data/project/123456/file.zip /scratch/user/abcdef/file.zip
#rsync -P –a /data/project/123456/file.zip /scratch/user/abcdef/file.zip
#$HOME/my_data_transfer_executable
# wait until transfer is finished (if the transfer is run in background)
wait
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

National Supercomputing Centre



Default job resources

Nodes in PBS

Verify job resources

Backward Compatibility

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Default Job Resources

Values

- PBS assigns **default values for certain resources** that the user doesn't specify during job submission
- The following are resource defaults assigned to a job, if not requested :
  - **default_queue = normal**
  - **default_chunk.ncpus=1**
  - **resources_default.ncpus=1**
  - **resources_default.mem=10mb**
  - Default 'walltime' per job is 5 min



Note: Walltime and Project ID are still required to be defined during job submission to successfully submit a job on ASPIRE2A.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Nodes in PBS

Understanding PBS Vnodes

- What is a host?
  - A machine on which a **PBS** daemon runs
  - A machine from which users can submit jobs
- What is a PBS MoM (Machine-oriented Mini-server)?
  - Executes the job script
  - Reports back to the server when the job is completed
  - Enforces some job resource limits
  - Can manage **multiple vnodes**
  - Tracks job resource usage
- What are vnodes?
  - An abstract object representing a set of resources which form a usable part of a machine
  - Can be one of the following: **host, board (node/memory), or blade**
  - A single host can be made up of multiple vnodes
  - Each **vnode** can be managed and scheduled **independently**

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

PBS Vnodes

Using the pbsnodes Command

- Use **pbsnodes** to get a detailed listing of all hosts or vnodes in a PBS complex

  - **Usage:** pbsnodes [-{o|r}][-C comment][-s server] host host ...
    - pbsnodes -l [-s server]
    - pbsnodes [-s server] -v vnode vnode ...
    - pbsnodes -a[v][S[j][L]][-F format][-D delim][-s server]
    - pbsnodes -[H][S[j][L]][-F format][-D delim] host host ...

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

PBS Vnodes

Using the pbsnodes Command

Options	Description
-a	Lists all hosts and their attributes
-v <vnode list>	Lists all attributes for specified vnodes
-l	Lists all hosts or vnodes that are marked as DOWN or OFFLINE
-C "<comment>" <host_list>	Add comment string to listed hosts. To remove comment use "". Can be used with '-o | r' option
-o <host_list>	Marks listed hosts as OFFLINE
-r <host_list>	Clears OFFLINE from listed hosts
-S <host_list>	Displays specialized vnode information, including certain custom resources
-j <host_list>	Displays job-related information from listed hosts
-F <json | dsv> <host_list>	Sets summary output to json or dsv format
-D "<delimiter>	Sets delimiter for -F dsv option. Default is "|"
-L	Displays each column with unlimited length
-H <host list>	Prints all non-default attributes for specified host(s) and all vnodes on specified host(s)
-s <server>	Specifies PBS server to which to connect

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

PBS Vnodes

Using the pbsnodes Command

pbsnodes –aS

```
vnode    state     OS    hardware    host     queue    mem    ncpus    nmics    ngpus   comment
--------------- --------------- -------- -------- -------------- ---------- --------
node01   free      --      --        node01     --      3gb      2        0        0        --
node02   free      --      --        node02     --      3gb      2        0        0        --
node03   free      --      --        node03     --      3gb      2        0        0        --
node04   free      --      --        node04     --      3gb      2        0        0        --
```

pbsnodes -aSj

```
                                                                                  mem       ncpus
nmics    ngpus
vnode            state            njobs    run    susp      f/t         f/t      f/t      f/t    jobs
--------------- --------------- ------ ----- ------ ------------ ------- ------- ------- -------
node01           free                1      1       0       3gb/3gb      1/2      0/0      0/0 17950
node02           free                0      0       0       3gb/3gb      2/2      0/0      0/0 --
node03           free                0      0       0       3gb/3gb      2/2      0/0      0/0 --
node04           free                0      0       0       3gb/3gb      2/2      0/0      0/0 --
```

Column	Description
mem	Amount of memory in GB
ncpus	Total amount of available NCPUS
nmics	Total amount of available many integrated cores - Intel
ngpus	Total amount of available GPUS
f/t	f=free and t=total

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

PBS Vnodes

Viewing Existing Vnodes

- Following is the way to view the vnodes and it attributes in a PBS complex
- Using **pbsnodes –av** at command line

```
[root@pbsworks ~]# pbsnodes node01
node01
     Mom = node01
     Port = 15002
     pbs_version = 2020.1.0.20200711141213
     ntype = PBS
     state = free
     pcpus = 2
     resources_available.arch = linux
     resources_available.host = node01
     resources_available.mem = 994564kb
     resources_available.ncpus = 2
     resources_available.vnode = node01
     resources_assigned.accelerator_memory = 0kb
     resources_assigned.hbmem = 0kb
     resources_assigned.mem = 0kb
     resources_assigned.naccelerators = 0
     resources_assigned.ncpus = 0
     resources_assigned.vmem = 0kb
     resv_enable = True
     sharing = default_shared
     last_state_change_time = Tue May  1 16:33:01 2018
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Verify job resources

Using the vnodes and its attributes to verify job resources

- There are 4 types of nodes on ASPIRE2A
    1. Cray EX nodes (CPU only)
    2. Cray GPU nodes
    3. Large Memory nodes
    4. AI system nodes with GPU
- Each type of node has a specific set of resources available, which can be determined using the “pbsnodes -av” command. A job can run on a specific node if the node can satisfy the resources requested in a chunk.
- If the requested resource in a chunk exceeds what the node has available, the job will be considered to run on a different node. If no node can satisfy the chunk, the job will never be able to run. Therefore, it's important for users to ensure that the requested resources in a chunk are below the available resources of the node on which they intend to run the job.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Verify job resources

Using the vnodes and its attributes to verify job resources

- For example, the available memory in the nodes on ASPIRE2A are as follows:
  - Cray EX and GPU nodes = 440gb (450867mb)
  - 2tb large memory nodes = 1952gb (1999151mb)
  - 4tb large memory nodes = 3968gb (4063411mb)

To run a job on 2tb large memory node, the requested memory in the chunk must be greater than 440gb and less than 1952 gb

#PBS -l select=1:ncpus=128:mem=1952gb

If the requested memory is :
#PBS -l select=1:ncpus=128:mem=2tb
Although the job was intended for a 2tb node, the job will run on a 4tb node if the requested memory is “2tb” (2048gb), which is more than what is available on a 2tb node (1952GB).

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Backward Compatibility

Conversion of Old Style to New

- Old versions of PBS allowed job submitters to use old style specifications to request resources like "-l nodes=..." which was called a "node specification" to specify where the job should run and "-lresource=value" which was called a "resource specification", to ask for resources outside of a select statement.
- Resource specification and node specification syntax are **deprecated**.
- For backward compatibility, a legal node specification or resource specification is converted into select directive. However, **it is highly recommended to submit the job resource specification in new "select" statement only to avoid any uncertainty in resource request after conversion.**
- For example, a script requesting following resources in old syntax :
  #PBS -lnodes=14:ncpus=1
  #PBS -l mem=280mb
  is converted to :
  #PBS -l select=14:ncpus=1:mem=20mb

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Main Concepts - Summary

Batch Job: Set of commands and/or applications you want to run on one or more execution host.

SMP jobs are meant to run on a single execution host

Check the status of list of current PBS jobs using qstat

Usage: qstat <-a, -n, -s, ,-f , -x -1, -w, -r, -l ..>





Example: `qsub -l select=1:ncpus=8:mem=32GB`



© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Exercise 1 - Job Management

Objective: Submit, check status of jobs and view the status of a PBS MoM

Prerequisites: None

PBS User Tasks:

1. qsub a job
2. qstat the job to see the job status
3. Run pbsnodes to see the node status

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

National Supercomputing Centre

Advanced Job Management



JOB EXIT CODES
MANAGING JOBS
FILE PROCESSING
JOB SUBMISSION DEPENDENCIES
RUNNING JOB MANAGEMENT

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Exit Codes

- After each job is finished or deleted, an exit status code is set for the job
- The code is reported in the **Exit_status** job attribute, and visible using the **tracejob** command

```
03/19/2018 15:09:16  S  Exit_status=271 resources_used.cpupercent=0 resources_used.cput=00:00:00
                        resources_used.mem=0kb resources_used.ncpus=1resources_used.vmem=0kb
                                                resources_used.walltime=00:00:00
```

- The exit status code is recorded in the PBS server logs and the accounting logs

Exit Code Range	Reason	Description
X< 0	The job could not be executed	This is a special PBS return value indicating that the job could not be executed. See chart on next slide.
0 <= X < 128	Exit value of shell	This is the exit value of the top process in the job, typically the shell.
X >= 128	Job was killed with a signal	The job was killed with a signal. The signal is X modulo 128 or 256. Example: exit value 137 modulo 128 is signal 9 (137%128) which is SIGKILL

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Exit Codes, cont.

PBS Code	Name	Description
0	JOB_EXEC_OK	Job execution was successful
-1	JOB_EXEC_FAIL1	Job execution failed, before files, no retry
-2	JOB_EXEC_FAIL2	Job execution failed, after files, no retry
-3	JOB_EXEC_RETRY	Job execution failed, do retry
-4	JOB_EXEC_INITABT	Job aborted on MoM initialization
-5	JOB_EXEC_INITRST	Job aborted on MoM init, checkpoint, no migrate
-6	JOB_EXEC_INITRMG	Job aborted on MoM init, checkpoint, ok migrate
-7	JOB_EXEC_BADRESRT	Job restart failed
-10	JOB_EXEC_FAILUID	Invalid UID/GID for job
-11	JOB_EXEC_RERUN	Job was rerun
-12	JOB_EXEC_CHKP	Job was checkpointed and killed
-13	JOB_EXEC_FAIL_PASSWORD	Job failed due to a bad password
-14	JOB_EXEC_RERUN_ON_SIS_FAIL	Job was requeued or deleted due to communication failure between mother superior and a sister
-15	JOB_EXEC_QUERST	Requeue job for restart from checkpoint

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Exit Codes, cont.

PBS Code	Name	Description
-16	JOB_EXEC_FAILHOOK_RERUN	Job execution failed due to hook rejection; re-queue for later retry
-17	JOB_EXEC_FAILHOOK_DELETE	Job execution failed due to hook rejection; delete the job at end
-18	JOB_EXEC_HOOK_RERUN	A hook requested for job to be requeued
-19	JOB_EXEC_HOOK_DELETE	A hook requested for job to be deleted
-20	JOB_EXEC_RERUN_MS_FAIL	Mother superior connection failed
-21	JOB_EXEC_FAIL_SECURITY	Security breach in PBS directory
-22	JOB_EXEC_HOOKERROR	Job failed due to unexpected exception or hook execution timeout
-24	JOB_EXEC_KILL_NCPUS_BURST	Job execution failed due to exceeding ncpus (burst)
-25	JOB_EXEC_KILL_NCPUS_SUM	Job execution failed due to exceeding ncpus (sum)
-26	JOB_EXEC_KILL_VMEM	Job execution failed due to exceeding vmem
-27	JOB_EXEC_KILL_MEM	Job execution failed due to exceeding mem
-29	JOB_EXEC_KILL_WALLTIME	Job execution failed due to exceeding walltime
-30	JOB_EXEC_JOINJOB	Job execution failed due to join job error
-32	JOB_EXEC_RERUN_ZOMBIE_JOB	Job execution hung due to lost job info from MoM

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Managing Jobs

Deleting Jobs

- To delete a job
  - Using **qdel** command:
    - **Usage:** qdel <job id>
    - **Example:** qdel 2.pbsworks
- Normal job termination sequence
  - **Single-vnode job**
    - PBS sends the job a **SIGTERM**
    - PBS waits for the amount of time specified in the **kill_delay** queue attribute (default=10 seconds)
    - PBS sends the job a **SIGKILL**
  - **Multi-vnode job**
    - Mother superior sends **SIGTERM** to all processes on the primary execution host
    - If any of the processes of the top task of the job are still running, PBS waits a minimum time specified in the **kill_delay** queue attribute
    - Mother superior sends a **SIGKILL** to all remaining job processes on the primary execution host
    - The subordinate MoM sends a **SIGKILL** to all the processes belonging to that job



Note: A batch job may be deleted by its owner, a PBS operator, or a PBS manager

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Managing Jobs

Deleting Jobs

- To forcefully delete a job from the PBS server
  - Job may or may not continue to run on the execution host(s) and SU may not be refunded.
  - **Usage:** qdel -W force <job id>

Note !!: NSCC strictly recommends NOT using forceful deletion of job from the PBS server. If a job does not get deleted by using graceful deletion, it may hint for any infrastructure issue at the node where job is currently running and may require to be operated by system administrators. Please write to help@nscc.sg if such a case occurs

- To delete a job and its preserved history from the PBS server
  - The job will not appear if qstat -H or qstat -x is used.
  - **Usage:** qdel -x <job id>
- To remove all jobs of the given user
  - All jobs (including those which are running) from the user who is running the command will be deleted. Be very cautious when using this command. Jobs once deleted cannot be recovered.
  - **Example:** qdel `qselect`

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Managing Jobs

Requeuing Jobs

- Allowing running job to be terminated and requeued in the execution queue from which it was run
  - Jobs are only requeuable if the **qsub** option '**-r**' is set to '**y**' (default=**y**)
  - Requeued jobs .OU and .ER are copied to the PBS server's **PBS_HOME/spool** directory to be appended for the next run

- Using **qrerun** command
  - **Usage:** qrerun <job id>
  - **Example:** qrerun 0.pbsworks



Note: Only root, PBS Managers, or Operators can perform this operation

- To requeue a job, even if that job's execution host is not reachable:
  - **Usage:** qrerun –W force <job id>
  - **Example:** qrerun –W force 2.pbsworks

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Managing Jobs

Holding Jobs

- Making job (not in RUN state) ineligible for execution so that it remains queued
- During job submission use '**qsub –h**'
- After job submission use '**qhold |-h**'
  - **Usage:** qhold <-h hold_list> <job_id>
  - **Example:** qhold 134.pbsworks

```
$ qsub -h my_script.sh
$ qstat

Job id            Name             User              Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
134.pbsworks      my_script.sh     user01                   0 H workq
```

Job state is H'old

hold_list	Description
n	None: no hold type specified
u	User: the user may set and release this hold type
p	Password: set if job fails due to a bad password; can be unset by the user
o	Operator: requires operator privilege to unset
s	System: requires root or manager privilege to unset

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Managing Jobs

Releasing Jobs

- Allowing job(s) that are held to be eligible for execution
  - Using qrls command
  - **Usage:** qrls <-h hold_list> <job_id>
  - **Example:** qrls 134.pbsworks

hold_list	Description
n	None: no hold type specified
u	User: the user may set and release this hold type
p	Password: set if job fails due to a bad password; can be unset by the user
o	Operator: requires operator privilege to unset
s	System: requires root or manager privilege to unset

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Managing Jobs

Deferring Job Execution

- Specifying the time, a job is eligible for execution
  - **Usage:** qsub –a [[[[CC]YY]MM]DD]hhmm[.SS
  - **Example:** qsub –a 201812311230 my_script.sh

- Output from qstat –f :

```
Execution_Time = Mon Dec 31 12:30:00 2018
job_state = W
```

'W'ait state

Type	Description
CC	2 first digits of the century
YY	2 digits of the year
MM	2 digits of the month
DD	Day of the month
hh	Hour
mm	Minutes
SS	Seconds

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Managing Jobs

Altering Requested Job Resources

- Job's requested resources and most attributes can be modified before or during execution
- Requested amounts cannot exceed server or queue limits
  - Cannot alter the resources of a running job array job
- Using qalter command:
  - **Usage:** qalter –l select=<numerical>:<res1>=<value>:<res2>=<valve>
  - **Example:**

```
[user01@pbsworks ~]$ qsub -h -l select=1:ncpus=2 my_script.sh
144.pbsworks.local
[user01@pbsworks ~]$ qstat -f 144 | grep Resource_List.ncpus
Resource_List.ncpus = 2
[user01@pbsworks ~]$ qalter -l select=1:ncpus=4 144
[user01@pbsworks ~]$ qstat -f 144 | grep Resource_List.ncpus
Resource_List.ncpus = 4
```

- Resources that can be modified before and during execution

Resource	Before Execution	During Execution
cputime	Yes	Yes
walltime	Yes	Yes
ncpus	Yes	No
memory	Yes	No

Only root, PBS Managers, and Operators can set additional amount.

Users can only reduce the amount.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Managing Jobs

Selective Job Querying - Using qselect

- A method to list the job identifiers of those jobs which meet criteria specified by various options
  - Each option acts as a filter restricting the number of jobs which might be listed
  - To list all job IDs, use no options
- To use the qselect command, specify criteria for selectable attributes:
  - Account string
  - Hold types
  - Finished or moved jobs
  - Job resources (-l select=...)
  - Job name
  - Project name
  - Destination queue
  - Job state
  - Job time attributes: start, end, queued, etc...
  - Username

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Managing Jobs

Selective Job Querying - Using qselect

resource_list	Description
.eq.	Equal to
.ne.	Not equal to
.ge.	Greater than or equal to
.gt.	Greater than
.le.	Less than or equal to
.lt.	Less than

sub_option	time_attribute	Description
a	Execution_Time	Time job began execution
c	ctime	Job creation time
e	etime	Job end time
g	eligible_time	Accrued eligible time
m	mtime	Modification time
q	qtime	Job queued time
s	stime	Job start time
t	estimated.start_time	Estimated start time

Option	Value	Description
-A	<account_string>	Matches Account_Name
-c	<interval>	Checkpoint interval attribute
-h	<hold_list>	Specifies Hold_Types
-H		Finished or moved jobs
-J		Job arrays
-l	<resource_list>	Specified resource amounts
-N	<job_name>	Specifies job Job_Name
-p	<job_priority>	Specifies job Priority
-P	<project_name>	Specifies job project
-q	<queue_name>	Specifies job queue
-r	<y|n>	Specifies job is Rerunnable
-s	<states>	Specifies job job_state
-t	<sub_option>	Specifies job time attribute values
-T		Job and subjob identifiers
-u	<user_name>	Specifies job Job_Owner
-x		Finished or moved jobs
-X		Completed or deleted subjobs

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Managing Jobs

Selective Job Querying - Using qselect

- **Examples:**
  - To find job IDs of jobs belonging to user user01, regardless of state:
    qselect –x –u user01

```
143.pbsworks.local
144.pbsworks.local
145.pbsworks.local
```

  - To find job IDs of running jobs that have requested more than 128 CPUS:
    qselect –s R –l ncpus.gt.128
  - To list jobs with start time greater than 2018/09/17 @0800 and end time less than 2018/09/21 @0800:
    qselect -x -ts.gt.1809170800 -te.lt.1809210800

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Managing Jobs

Specifying Email Notifications

- Users can specify whether they want email notification for each job state
- Using **qsub** command with the following options
- Users can set their own notifications:

  Usage: qsub –m <options>

  Example: qsub –m abe my_script.sh

Options	Description
a	Job is aborted (default)
b	Job has begun
e	Job has finished execution
n	Do not send any email

```
From adm@pbsworks.local  Mon Mar 30 11:27:55 2018
Return-Path: <adm@pbsworks.local>
X-Original-To: user01@pbsworks
Delivered-To: user01@pbsworks.local
Received: by pbsworks.local (Postfix, from userid 0)
        id 7CD43A58AC; Mon, 30 Mar 2018 11:27:55 -0700 (PDT)
To: user01@pbsworks.local
Subject: PBS JOB 143.pbsworks.local
Message-Id: <20180330182755.7CD43A58AC@pbsworks.local>
Date: Mon, 30 Mar 2018 11:27:55 -0700 (PDT)
From: adm@pbsworks.local (root)

PBS Job Id: 143.pbsworks.local
Job Name:   STDIN
Post job file processing error; job 143.pbsworks.local on host node01
```

Email content cannot be customized

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

File Processing

Standard Output and Standard Error Files

- PBS creates separate standard output and standard error files for each job
- For a multi-vnode job, only one **.o** and **.e** are created
  - Any standard output/error files from the sister vnodes are sent to the primary execution host
- Users can specify the path and name of the output and error files for each job at submission ti

Usage:

qsub –o <hostname>:<path>/<filename>

qsub –e <hostname>:<path>/<filename>

PBS directives

#PBS -o pbsworks:/home/user01/OUT

#PBS –e pbsworks:/home/user01/ERR

Examples:

qsub –o pbsworks:/home/user01/OUT

qsub –e pbsworks:/home/user01/ERR

Viewable in

'qstat –f' output

- To avoid the creation of STDOUT and/or STDERR files, use the 'qsub –R' command option:

Examples:

qsub –R -e my_script.sh
qsub –R –o my_script.sh
qsub –R –oe my_script.sh

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

File Processing

Merging Output and Error Files

- Users can choose to merge the output and error files into one file
- This cuts down on the number of files produced
  - **Usage:** qsub –j <joining option>
  - **Examples:** qsub –j oe

    #PBS –j oe

Option	Description
oe	Both files are merged; standard output
eo	Both files are merged; standard error
n	Both files not merged (default)

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

File Processing

Returning Output and Error Files

- After a job finishes, by default PBS copies the standard output and standard error files back to **$PBS_O_WORKDIR** on the submission host
  - Output file format:<job_name>.o<jobid>
  - Error file format:<job_name>.e<jobid>
- If PBS MoM is unable to copy the .OU and .ER from the primary execution host's **PBS_HOME/spool** directory
  - It is placed in the **PBS_HOME/undelivered** directory on the primary execution host

```
[user01@node01 undelivered]$ pwd
/var/spool/pbs/undelivered

[user01@node01 undelivered]$ ll
total 0
-rw-------. 1 user01 user01 0 Jul 15 08:49 13.pbsworks.ER
-rw-------. 1 user01 user01 0 Jul 15 08:49 13.pbsworks.OU
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

File Processing

Keeping Output and Error Files on Execution Host

- PBS copies STOUT and STDERR to **PBS_O_WORKDIR** directory or to the specified directory

- Users can specify whether to keep either or both files in the execution host's job staging and execution directory on the mother superior
  - **Usage:** qsub –k <options>
  - **Example:** qsub –k oe

#PBS –k oe

Using –k overrides –o and –e options

OPTION	Description
o	Keep only STDOUT
e	Keep only STDERR
oe	Keep both files
n	Keep neither - default
d	Write directly to destination directory

Note: Users are suggested **NOT to use this option on ASPIRE2A**. Since the files will not be copied back, and users are not allowed to access the host without a running job, recovering files will not be possible.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

File Processing

Job's Staging and Execution Directory

- Default job staging and execution directory is the user's home directory or a job-specific directory
- Users can specify whether each job uses a unique job-specific staging and execution directory
- This is done by setting the sandbox attribute at job submission
  - **Usage:** qsub –W sandbox = <HOME | PRIVATE>
  - **Where:** HOME user's home directory; default
    PRIVATE PBS creates a job-specific directory
    - If MoM's **$jobdir_root** parameter is set, it is created under **$jobdir_root (/scratch on ASPIRE2A)**
    - If Mom's **$jobdir_root** parameter is unset, it is created under user's home directory
- Once the job is completed, the **ENTIRE** directory and its contents are **REMOVED**

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

File Processing

Job's Staging and Execution Directory

- PBS sets job's **jobdir** attribute to path of staging and execution directory

- If using '-W sandbox=PRIVATE':
  - **jobdir = /scratch/pbs.21.pbsworks.x8z**
  - .OU and .ER are created in the **$jobdir** directory
  - **.o** and **.e** are copied from **$jobdir** to location of **qsub** command or specified location
  - After the job is completed the job-specific **jobdir** directory is deleted

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

File Processing

Job's Staging and Execution Directory

- If MoM **$jobdir_root** parameter is set
  - Directory will be created where the parameter is set (/scratch on ASPIRE2A)

- **Example:** $jobdir_root /scratch
  - **jobdir = /scratch/pbs.9.pbsworks.x8z**
  - OU and .ER are created in the **$jobdir** directory
  - .o and .e are copied from **$jobdir** to location of **qsub** command or specified location
  - After the job is completed the job-specific **jobdir** directory is deleted

Only if sandbox=PRIVATE

```
[root@pbsworks scratch]# ls -ls
total 0
0 drwx------. 2 manish manish 48 Jun  3 19:48 pbs.9.pbsworks.x8z
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

File Processing

PBS File Staging

- Input/output file staging
  - **STAGE IN :** Users can specify which files/directories are copied onto the execution host before their job executes.
  - **STAGE OUT :** Users can specify which files/directories are returned to the submission host or specified directory after the job completes.
  - After a job has been completed, all files and sub directories in the execution directory are **REMOVED.**



Note: Admin can configure remote file transfer mechanism.

Walltime is **NOT** charged during staging in and out of files

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

File Processing

PBS File Staging

- Using in **qsub** command line
  - **qsub –W stagein =** <execution_path>@<hostname>:<storage_path>
  - **qsub –W stageout =** <execution_path>@<hostname>:<storage_path>

- Using as PBS directive
  - **#PBS -W stagein =** <execution_path>@<hostname>:<storage_path>
  - **#PBS -W stageout =** <execution_path>@<hostname>:<storage_path>

    - **stagein:** where input files come from (are stored)
    - **stageout:** where output files end up when job is done
    - execution_path: filename in job's staging & execution directory on execution host
      Do not specify a hostname
    - storage_path: filename on host *hostname*

- The **'@'** character separates execution path specification from storage path specification

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Script Example

PBS File Staging and Private Execution Directory

```
#!/bin/bash
#PBS -q normal
##Requesting a full node with 128 CPU
#PBS -l select=5:ncpus=128:mpiprocs=128:mem=400gb
#PBS -l walltime=8:00:00
#PBS -P <projectid>
#PBS -j oe
#PBS -N gromacs_test
#PBS -W sandbox=PRIVATE
#PBS -W stagein = benchMEM.tpr@asp2a-login-nscc01:/home/users/org/username/input_folder/benchMEM.tpr
#PBS -W stageout = result_file@asp2a-login-nscc01:/home/users/org/username/input_folder/result_file

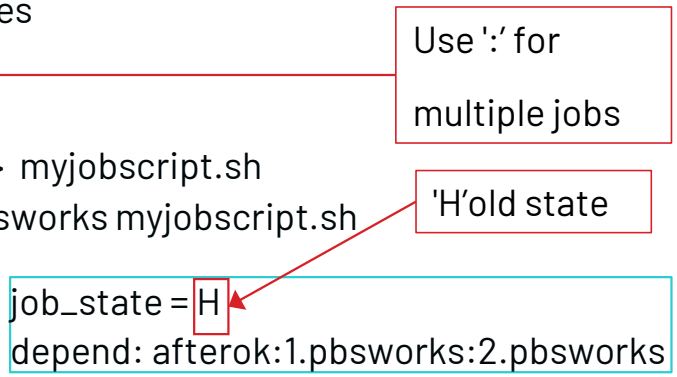
module load gromacs/2023.2-gnu                    #Load correct environment
export PATH=</absolute/path/to/vasp/bin>:$PATH      #Add path of executable
#execute workload
mpirun -np 640 -v gmx_mpi mdrun -s benchMEM.tpr -nsteps -1 -maxh 1.0 -resethway -noconfout
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Submission Dependencies

Concept and Usage

- Users can specify dependencies between their jobs, such as:
  - Specify order of execution
  - Execute the next job only if previous job finished
  - Place jobs on hold until a particular job starts or finishes
- Using '–W depend' attribute:
  - **Usage:** qsub –W depend=<type>:<arg_lists> <job ID> myjobscript.sh
  - **Example:** qsub -W depend=afterok:1.pbsworks:2.pbsworks myjobscript.sh
- To view a job's dependencies, use **qstat –f** <job ID>
- Enabling job history changes the behavior of dependent jobs
  - If job history is not enabled, and job j1 depends on a finished job j2 that has been purged, PBS rejects job j1 as if j2 no longer exists



© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Submission Dependencies

Dependency Types

Dependency Type	Description
after:<arg_list>	Job may be scheduled for execution after all jobs in <arg_list> have started execution
afterok:<arg_list>	Job may be scheduled for execution only after all in <arg_list> have terminated with no errors.
afternotok:<arg_list>	Job may be scheduled for execution only after all jobs in <arg_list> have terminated with errors.
afterany:<arg_list>	Job may be scheduled for execution after all jobs in <arg_list> have terminated with or without errors.
before:<arg_list>	Jobs in <arg_list> may begin execution once this job has begun execution
beforeok:<arg_list>	Jobs in <arg_list> may begin execution once this job terminates without errors
beforenotok:<arg_list>	Jobs in <arg_list> may begin execution once job terminates execution with errors
beforeany:<arg_list>	Jobs in <arg_list> may begin execution once this job terminates execution, with or without errors
on:<count>	Job may be scheduled for execution after count dependencies on other jobs have been satisfied

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Submission Dependencies

Runone Dependency

- Users may want to start a **flexible job** as soon as possible on a smaller/alternate set of resources rather than waiting longer for a larger/specific set of resources
- Users can submit **a set of jobs** where each job has a "runone" dependency on the others, and PBS will run only one of the jobs in the "runone set".
  - When any of the jobs in the set starts, PBS applies a system hold to the others.
  - The other jobs in the set are deleted:
    - When a job ends, regardless of its exit status
    - When the running job is deleted

- **Example:** only one of the jobs in this set will get run:
  - **qsub -lselect=200:ncpus=16 -lwalltime=1:00:00 myscript.sh**

    10.myserver
  - **qsub -lselect=100:ncpus=16 -lwalltime=2:00:00 -Wdepend=runone:10 myscript.sh**

    11.myserver
  - **qsub -lselect=50:ncpus=16 -lwalltime=4:00:00 -Wdepend=runone:10 myscript.sh**

    12.myserver

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Running Job Management

Sending Messages to PBS Job's Output and Error Files

- String messages can be sent to a job's output (.o) or error (.e) file
- Reasons why to record a message to a job's **.o** and **.e** file
  - To have external events recorded for the job
  - Useful for administrators to notify a job's owner that system events occurred where that job was running

- Using qmsg command:
  - **Output** file: qmsg –O "<msg>" <job_id>
  - **Error** file: qmsg –E "<msg>" <job_id>

- While job is running, any messages are sent to that job's **.OU** and **.ER** files located in PBS_HOME/spool directory on the primary e.xecution host



Note: If flag **"-O"** or **"-E"** are not specified, the message is sent to the error file

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Running Job Management

Sending Signals to PBS Jobs

- Why send a signal?
  - To force a program to take a specific action

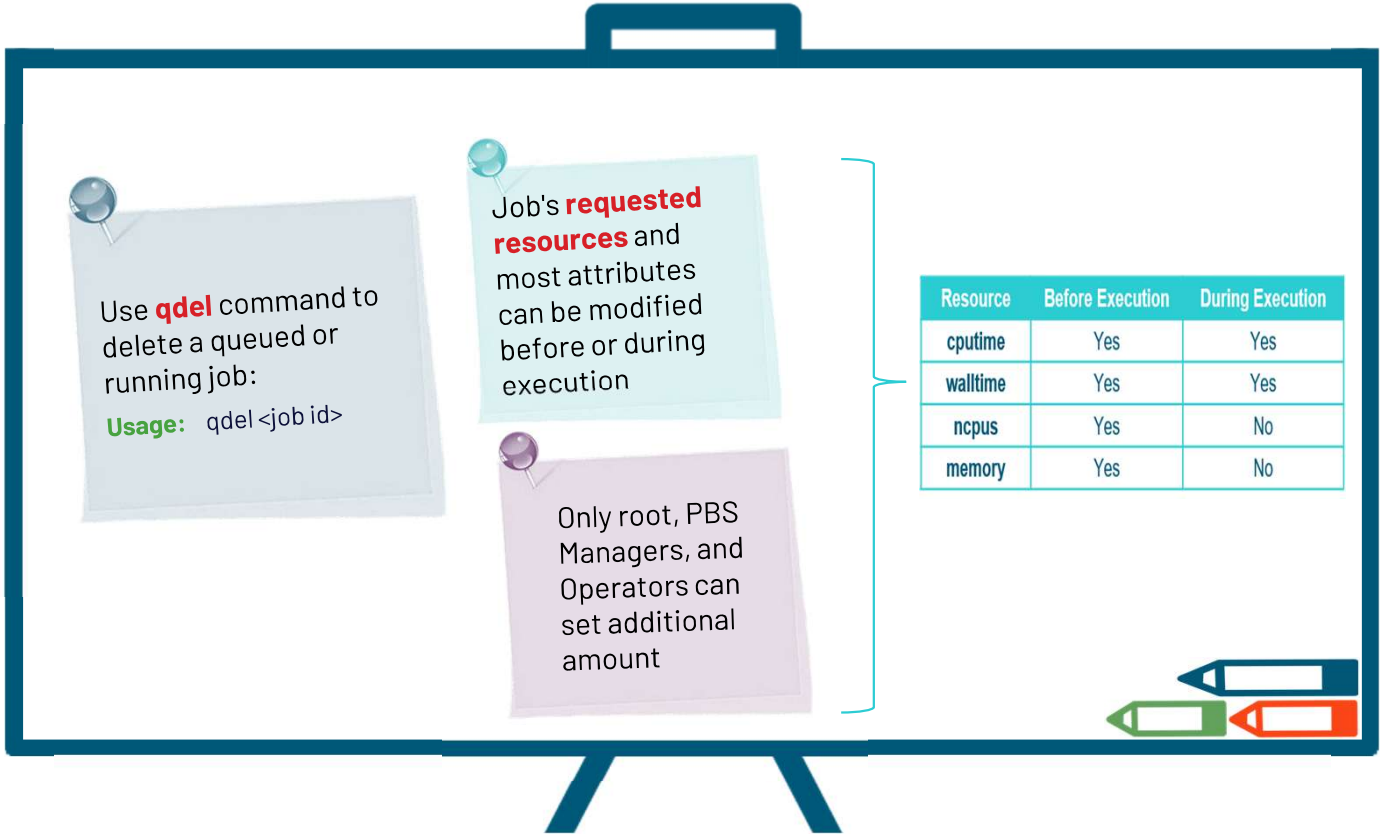
- Using qsig command
  - **Usage:** qsig –s <signal> <job_id>
  - **Example:** qsig –s suspend 0.pbsworks
    qsig –s resume 0.pbsworks

- Most commonly used signals:

Signal	Description
SIGHUP	Hangs up the program process
SIGTERM	Terminates the program process
SIGINT	Interrupts the program process
SIGKILL	Kills now regardless of the state of the program
suspend	Suspends a job process
resume	Resumes a job process

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Main Concepts - Summary



Resource	Before Execution	During Execution
cputime	Yes	Yes
walltime	Yes	Yes
ncpus	Yes	No
memory	Yes	No

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Exercise 2 – Submitting jobs, Specifying Dependencies

Objective: Understand how to submit a job that is dependent on a previously submitted job's status

Prerequisites: none

PBS User Tasks:

1. Create a job script that will return a negative value so it will fail, also make sure that this job will run for a while.
2. Submit this job where it will go into execution right away.
3. Using the job script, submit a second job with a dependency on the first job, such that if the first one failed then the second job will not execute.
   - Using qsub, set up a dependency based on a job name

Observation:

1. Observe whether the 2nd job was allowed to run after the 1st job completed.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

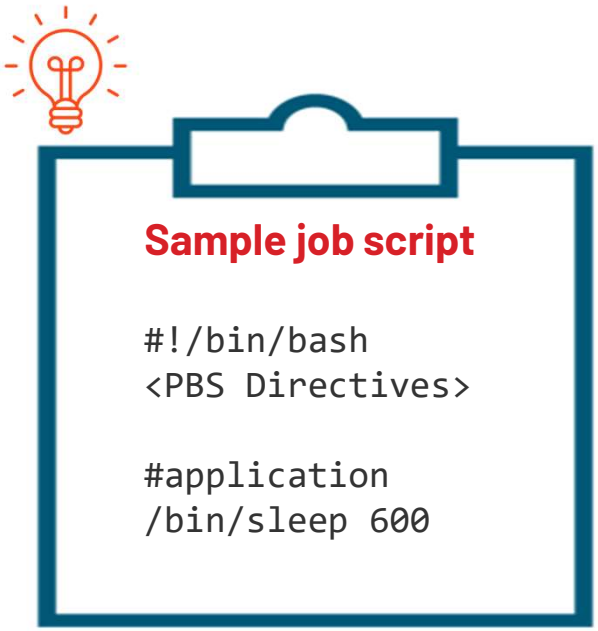
Exercise 3 - Using PBS Directives

Objective: Using PBS directives in a job script instead of specifying on **qsub** command line

Prerequisites: none

PBS User Tasks:

1. Create a job script with the following job options as PBS directives
   a. Job name = "**my_test_run_01**"
   b. Submit to "**normal" queue or reservation queue**
   c. Account name = "**pbs_train**"
   d. Error path = **"/tmp**"
   e. Output path = **"/tmp**"
   f. Job is not rerunnable
   g. Requesting **2** units of **6** NCPUS with **32GB** per chunk
   h. Requesting one chunk per vnode/host
   i. Requesting walltime of **5**hrs **30** minutes
2. Submit the job script



Observation:

- Using the output of qstat –f <job_id> , see whether any of the directives in the job script set job attributes.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Exercise 4 – Specifying Mail Notifications

Objective: Using the various mail notifications, observe information sent to the user's email address

Prerequisites: none

PBS User Tasks:

1. Create a job script
2. Either at **qsub** command line or as a PBS directive, set the following email notification conditions:
   - Job is aborted
   - Job begins execution
   - Job ends execution

Observation:

- When the job is completed (no longer showing in **qstat**), check user's email in **/var/mail/**<user>.
- What information, if any, did PBS send to that address?

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

National Supercomputing Centre
SINGAPORE

Job Arrays



CONCEPT
ENVIRONMENT VARIABLES
ARRAY JOB MANAGEMENT
IN MORE DEPT
EXAMPLES
EXIT CODES

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Arrays

Concept

- What is a job array?
  - Collection of subjobs differing by a single parameter

- Why use job arrays?
  - Allows users to group similar jobs and submit as one job
  - Allows users to query, modify, and display the set as a single unit

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Arrays

Concept

- Multiple uses:
  - **Job arrays** are a great way to organize the execution of multiple short jobs
    - Jobs with similar properties
    - Jobs using similar data with different algorithms
    - Jobs using a serial input file numbering system, e.g. file01, file02, file03
  - **Programming tool**
    - Can be used anywhere the facility to creatively program the launching of multiple jobs as a group is useful
- **Organizational tool**
  - Use job arrays to run and return together a related group of tasks
- **Scientific applications**
  - BLAST, Molecular modeling, drug screening, and discrete optimization
- **Film industry**
  - Animation rendering
- **Monte Carlo**

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Arrays

Submit **10 jobs** with consecutive index numbers

```
#!/bin/sh Examples Using a Job Script
#PBS -N Simn1010Jobs
#PBS -J 1-10
echo "Main script: index " $PBS_ARRAY_INDEX
/opt/AppA –input /home/user01/runcase1/scriptlet_$PBS_ARRAY_INDEX
```

Submit **5 jobs** with odd indices (1,3,5,7,9)

```
#!/bin/sh
#PBS -N SimOddJobs
#PBS -J 1-10:2
echo "Main script: index " $PBS_ARRAY_INDEX
/opt/AppA  –input /home/user01/odd/scriptlet_$PBS_ARRAY_INDEX
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Arrays

Submit **5 jobs** with even indices

```
#!/bin/sh
#PBS -N SimEvenJobs
#PBS -J 2-10:2
echo "Main script: index " $PBS_ARRAY_INDEX
/opt/AppA  -input /home/user01/even/scriptlet_$PBS_ARRAY_INDEX
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Arrays

Environmental Variables

Environment Variables	Used For	Description
$PBS_ARRAY_INDEX	subjobs	Subjob index in array job, e.g. 7
$PBS_ARRAY_ID	subjobs	Identifier for a job array. Sequence number of job array, e.g. 1234
$PBS_JOBID	Jobs, subjobs	Identifier for a job or a subjob. For subjob, sequence number and subjob index in brackets, e.g. 1234[7]

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Arrays

Querying Job Arrays

- Shows state of job and time used by job array and subjobs: 'qstat –t'

```
Job id            Name             User              Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
0[].pbsworks      test             user01                   0 B workq
0[1].pbsworks     test             user01            00:00:00 R workq
0[2].pbsworks     test             user01                   0 Q workq
```

- Shows state only: 'qstat –J'

```
Job id            Name             User              Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
0[].pbsworks      test             user01                   0 B workq
```

- Shows the % completed (completed / total): 'qstat –p'

```
Job id            Name             User               % done  S Queue
----------------  ---------------- ----------------  -------- - -----
0[].pbsworks      test             user01                  0  B workq
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Arrays

Job Array and Subjob States

- Job array states

State	Description
B	Job array has started
W	Job array has a wait time in the future
H	Job array is being held
T	Job array is in transit
Q	Job array is queued or has been qrerun
E	All subjobs are finished; server is cleaning up the array job

- Subjob states

State	Description
Q	Subjob is queued
R	Subjob is running
E	Subjob is ending
X	Subjob has expired/completed or been deleted
S	Subjob is suspended
U	Subjob is suspended by keyboard activity

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Arrays

In More Depth

- A **job array** is a compact representation of one or more jobs
  - Have the same job script
  - Have the same values for all attributes and resources, with the following exceptions
    - Each subjob has a unique index
    - Job identifiers of subjobs differ only by their indices
    - The state of subjobs can differ
    - For both stagein and stageout, the file list has the form:

      <execution path>^array_index^@<storage host>:<storage path>^array_index^[,...]

- **All subjobs** in a job array have the same scheduling priority
- A **job array** is submitted through a single command which returns, on success, a "job array identifier" with a server-unique sequence number

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Arrays

Terminology

Term	Description
Subjob	Individual entity in a job array
Sequence number	The numeric part of a job or job array identifier
Subjob index	The unique index which differentiates one subjob from another
Job array identifier	The identifier returned upon success when submitting a job array
Job array range	A set of subjobs in a job array

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Arrays

Job Attributes for Job Arrays

Name	Type	Applies To	Description
array	Boolean	Job array	True if item is a job array
array_id	String	Subjob	Subjob's job array identifier
array_index	String	Subjob	Subjob's index number
array_state_count	String	Job array	Similar to **state_count** attribute for server and queue objects. Lists number of subjobs in each state.
array_indices_remaining	String	Job array	List of indices of subjobs still queued. Range or list of ranges, e.g. 500, 552, 596-1000
array_indices_submitted	String	Job array	Complete list of indices of subjobs given at submission time. Given as range, e.g. 1-100.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Arrays

Job Identifier

- Subjob indices are specified at submission time
- There are two types of indices
  - Contiguous range
    - e.g., 1 through 100
  - Range with a stepping factor
    - e.g., every second entry in 1 through 100 (1, 3, 5, ... 99)
- A job array identifier can be used ...
  - By itself to represent the set of all subjobs of the job array
  - With a single index (a "job array identifier") to represent a single subjob
  - With a range (a "job array range") to represent the subjobs designated by the range

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Arrays

Identifier Syntax

- Job arrays have **3 identifier syntaxes**:
  - The job array object itself :
    1234[ ].server or 1234[ ]
  - A single subjob of a job array with index M:
    1234[M].server or 1234[M]
  - A range of subjobs of a job array:
    1234[X-Y:Z].server or 1234[X-Y:Z]

  - **Example:**
    - Full job array identifier: **1234[ ].server.domain.com**
    - Short job array identifier: **1234[ ]**
    - Subjob identifier of the 73rd index of job array:
      1234[73]
    - Error, if 1234[ ] is a job array: **1234**
    - Error, if 1234[ ].server.domain.com is a job array:
      1234.server.domain.com

- The sequence number (1234 in 1234[ ].server) is **unique**, so that jobs and job arrays cannot share a sequence number

  - On some OSes, the "[" and "]" can be interpreted as shell meta-characters, so any array/subjob identifiers must be enclosed in double quotes

  - **Example:** qdel "1234[ ].pbsworks"
    qdel "1234[73].pbsworks"

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Arrays

Submitting a Job Array

- Use 'qsub with –J <range>' option to submit a job array

- Where range is of the form X-Y[:Z]
  - **X** is the starting index
  - **Y** is the ending index
  - **Z** is the optional stepping factor
  - **X** and **Y** must be whole numbers, and **Z** must be a positive integer
  - **Y** must be greater than **X**
  - If **Y** is not a multiple of the stepping factor above **X**, (i.e. it won't be used as an index value) the highest index used is the next below **Y**
  - **Example:** qsub –J 1-10:2 results in odd numbers between 1-10
    qsub –J 2-20:2 results in even numbers between 1-20

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Arrays

Examples of using qsub –J

- Submitting a job array at the command line
  - A job array of 10,000 subjobs, with indices 1, 2, 3, ... 10000

'qsub -J 1-10000 jobarray_script'

  - A job array of 100 subjobs, with indices 500, 501, 502, 503, ... 599

'qsub -J 500-599 jobarray_script'

  - A job array with indices 1, 3, 5 ... 999

'qsub -J 1-1000:2 jobarray_script'



Note: Interactive submission of job arrays is not supported. No **'qsub –I –J'**

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Arrays

Limiting number of simultaneously running subjobs

- By default PBS simultaneously runs as many subjobs from a job array as possible.

- Use the **max_run_subjobs** job attribute to limit the number of subjobs running at the same time
  - This is helpful if for example every subjob needs access to the same shared data file and you want to prevent slowdowns due to an access bottleneck.

- You can set the limit at submission by appending %<max subjobs> to your -J option
  - **Example:** **qsub -J 1-20000 %500 myscript.sh**

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

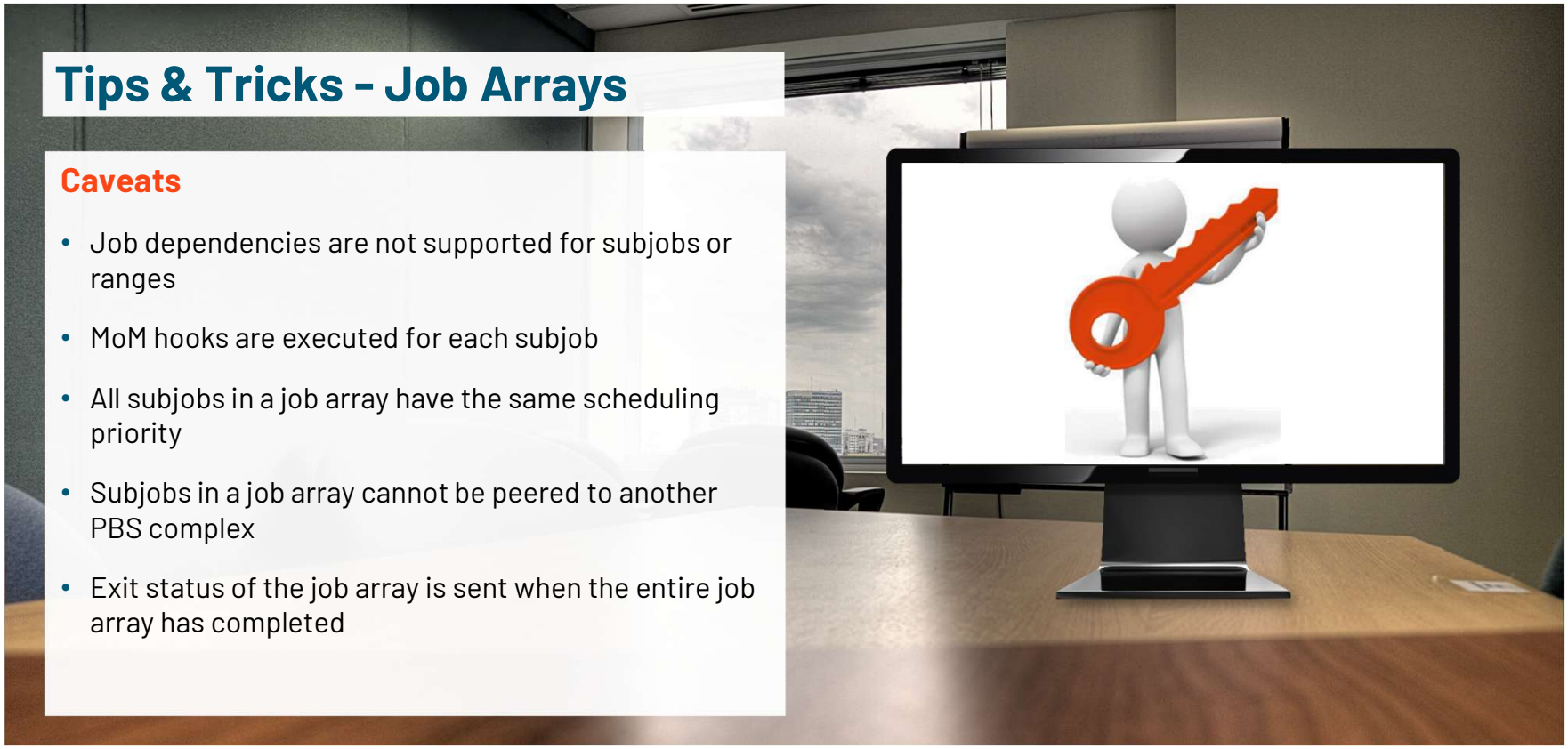
Job Arrays

Exit Codes

Exit Status	Meaning
0	No PBS error occurred. All subjobs of the job array returned an exit status of 0
1	At least 1 subjob returned a non-zero exit status
2	A PBS error occurred

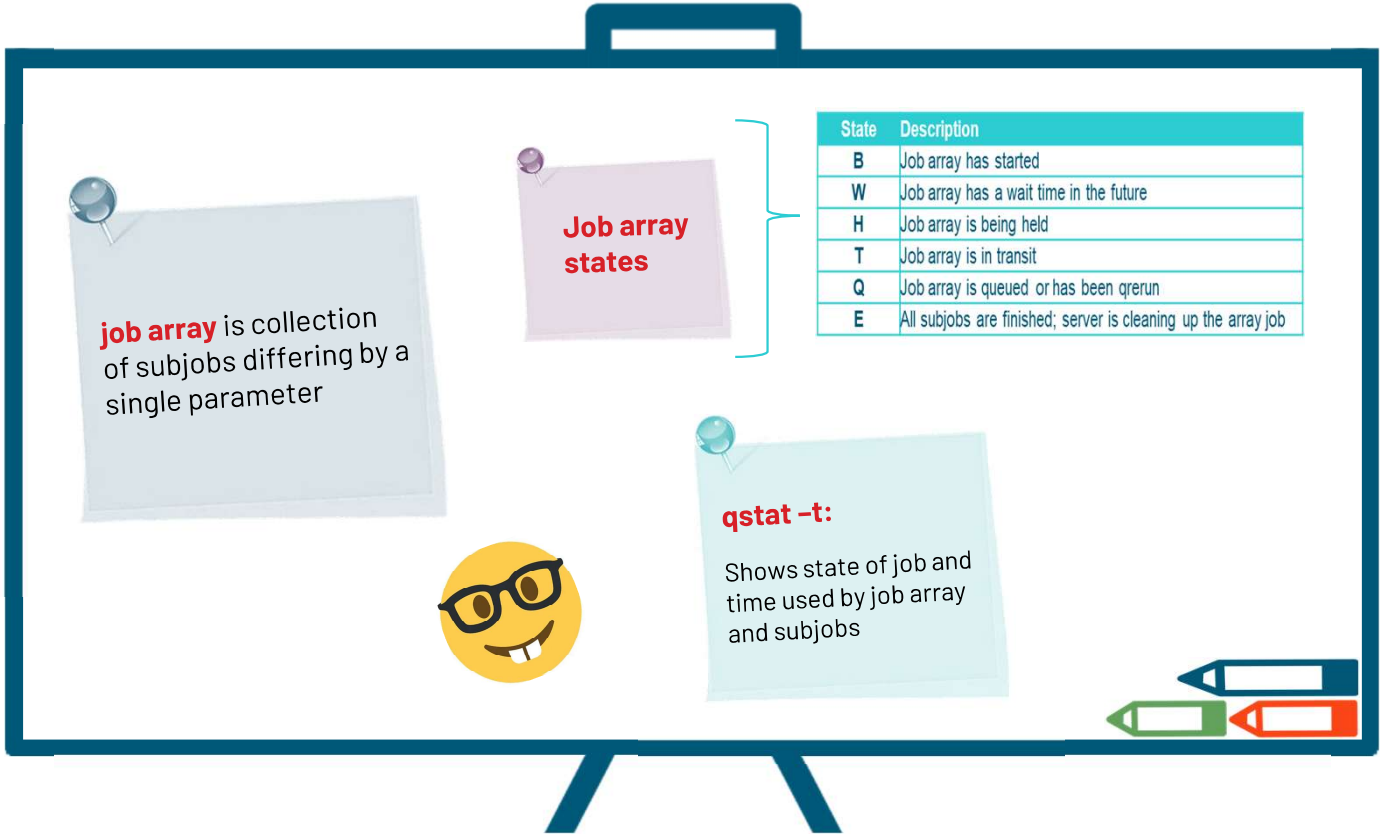
© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Tips & Tricks - Job Arrays



© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Main Concepts - Summary



© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

National Supercomputing Centre
SINGAPORE

Reservations



ADVANCED RESERVATIONS
STANDING RESERVATIONS

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Advance Reservations

Concept

- **What is an advance reservation?**

  - Requesting dedicated amount of resources at a given time
  - Allowing jobs to be executed within a granted timeslot with guaranteed reserved resources
  - One time requested job reservation
  - Jobs in an advance reservation have the highest priority, and cannot be preempted

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Advance Reservations

Concept

- Internal process for requesting a reservation
  1. Users need to request a reservation from admins. They can't create the reservations on ASPIRE2A by themselves.
     - A reservation number starting with "R" +"<number>" is informed to the users.
  2. If the reservation request is accepted:
     a. A queue is created for that reservation
     b. Vnodes assigned to that reservation can be identified using **'pbsnodes –a'** output, which appear in the reservation **resv_node** attribute: **resv = R3.pbsworks**
     c. Users who are allowed to submit jobs into that reservation can do so using: **qsub –q <reservation_id>**
     d. Jobs in the reservation queue will not run until the time specified
     e. Once a reservation reaches its end time, it is deleted, including any jobs in that reservation

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Advance Reservations

Status of a Reservation

- Use pbs_rstat to obtain status of all PBS reservations
  - Short status
    - Execute: pbs_rstat

```
Resv ID    Queue    User     State              Start / Duration / End
--------------------------------------------------------------------
R169.pbswo R169     user01@p CO             Wed 17:00 / 1800 / Wed 17:30
R170.pbswo R170     user01@p UN             Tue 15:00 / 3600 / Tue 16:00
R178.pbswo R178     user01@p RN              Mon 22:00 /  600 / Mon 22:10
```

- CO Reservation confirmed
- UN Reservation not confirmed
- RN Reservation running

  - Reservation identifier only
    - Execute: pbs_rstat -B

```
Name: R169.pbsworks
```

  - Comprehensive information about a reservation
    - Execute: qstat -Q <reservation_id>

```
Queue              Max   Tot Ena Str   Que   Run   Hld   Wat   Trn   Ext Type
---------------- ----- ----- --- --- ----- ----- ----- ----- ----- ----- ----
R169                 0     0 yes  no     0     0     0     0     0     0 Exec
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Advance Reservations

Status of a Reservation

- **pbs_rstat –f R169.pbsworks** detailed status

```
Resv ID: R169.pbsworks.local
Reserve_Name = NULL
Reserve_Owner = user01@pbsworks
reserve_state = RESV_CONFIRMED
reserve_substate = 2
reserve_start = Sat Dec 12 17:00:00 2018
reserve_end = Sat Dec 12 17:30:00 2018
reserve_duration = 1800
queue = R7Resource_List.mem = 1gb
Resource_List.ncpus = 1
Resource_List.walltime = 00:30:00
Resource_List.nodect = 4
Resource_List.select = 4:ncpus=4:mem=8GB
resv_nodes = (pbsworks:ncpus=1:mem=1048576kb)
Authorized_Users = user01@pbsworks
server = pbsworks.local
ctime = Wed Oct  7 16:25:20 2018
mtime = Wed Oct  7 16:25:28 2018
Variable_List = PBS_O_LOGNAME=user01,PBS_O_HOST=pbsworks,PBS_O_MAIL=/var/spool/mail/user01
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Advance Reservations

Submitting a Job to a Reservation

- Once a reservation has been confirmed, users can submit jobs to the reservation queue or move jobs into that reservation
  - Using **qsub –q** command to submit a job within a reservation:
  - **Example**: qsub –q R169 my_script.sh
- If reservations are no longer required, it can be requested to be deleted.
- Only the Admins can delete a reservation
- Once a reservation has been deleted, any jobs residing in that reservation are automatically deleted regardless of whether a job is running

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Standing Reservations

Concept

- What is a standing reservation?
  - A recurring advance reservation
- What is the difference?
  - Standing reservation jobs are submitted to the whole reservation, not to an occurrence
  - When an occurrence of a standing reservation has ended, only running jobs are deleted
- How to request a standing reservation?
  - Must specify a start time and end time, plus reservation rules
  - Recurrence rule must be on one unbroken line, in double quotes
  - Specify frequency, occurrence count or end time, and occurrence interval
  - PBS user must set the PBS_TZID environment variable. **Example:** /home/user01/.bashrc

```
# .bashrc
# User specific aliases and functions
source /etc/pbs.conf
PATH=$PBS_EXEC/bin:$PBS_EXEC/sbin:$PATH:/opt/openmpi/bin
export PATH
export mpirun=/opt/openmpi/bin/mpirun
export PBS_TZID=America/Los_Angeles
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Standing Reservations

Getting Status

- Use pbs_rstat to obtain the status of a standing reservation
  - Short status 'pbs_rstat S208'

```
Name        Queue    User     State             Start / Duration / End
--------------------------------------------------------------------
S208.pbswo S208      user0 CO            Fri 18:00 / 3600 / Fri 19:00
```

  - Reservation identifier only 'pbs_rstat -B'

```
Name: S208.pbsworks
```

  - Deletes a standing reservation 'pbs_rdel S208'
  - Comprehensive information about that reservation 'qstat –Q S208'

```
Queue              Max   Tot Ena Str   Que   Run   Hld   Wat   Trn   Ext Type
---------------- ----- ----- --- --- ----- ----- ----- ----- ----- ----- ----
S208                 0     0 yes  no     0     0     0     0     0     0 Exec
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Standing Reservations

Getting Status

- Full status:

pbs_rstat –f S208

```
Resv ID: S208.pbsworks.local
Reserve_Name = NULL
Reserve_Owner = user01@pbsworks
reserve_state = RESV_CONFIRMED
reserve_substate = 2
reserve_start = Fri Apr  3 18:00:00 2018
reserve_end = Fri Apr  3 19:00:00 2018
reserve_duration = 3600
queue = S208
Resource_List.ncpus = 1
Resource_List.walltime = 01:00:00
Resource_List.nodect = 1
Resource_List.select = 1:ncpus=1
resv_nodes = (node01:ncpus=1)
Authorized_Users = user01@pbsworks
server = pbsworks.local
ctime = Tue Mar 31 12:39:34 2018
mtime = Tue Mar 31 12:39:42 2018
Variable_List =
PBS_O_LOGNAME=user01,PBS_O_HOST=pbsworks,PBS_O_MAIL=/var/spool/
mail/user01,PBS_TZID=America/Los_Angeles
reserve_rrule = FREQ=WEEKLY;COUNT=4;BYDAY=FR
reserve_index = 1
reserve_count = 4
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Reservations

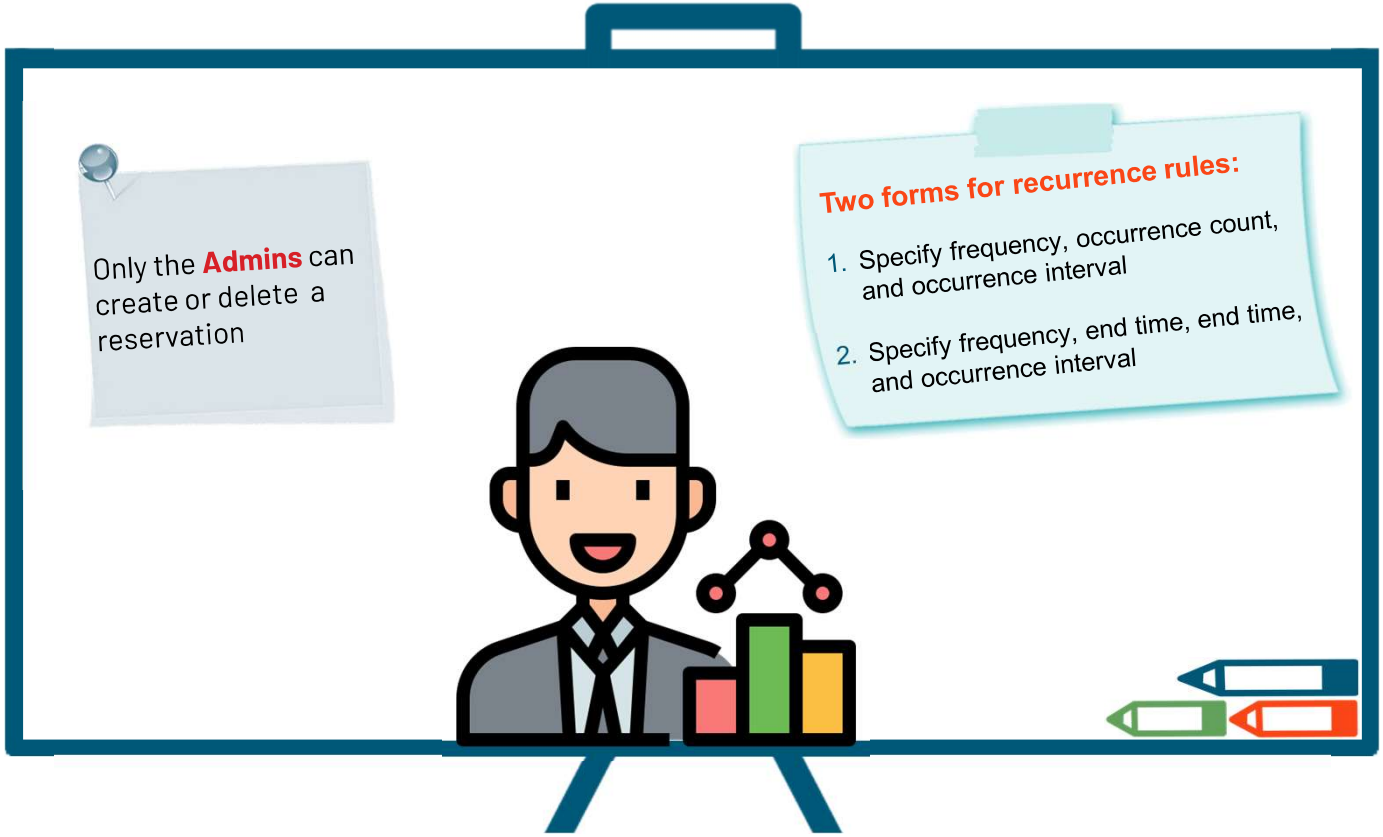
Altering Reservations Requests

- A standing and advance reservation can be modified after its has been confirmed or running

- The following **parameters** that are allowed to be modified:
  - Start time
  - End time – including when an reservation has already started
  - Mail points and mail list
  - Reservation name
- What **cannot** be modified:
  - Resources for which the reservation was confirmed on
    - Nodal resources using the **–l** option
    - Job-wide resources
    - **-r** option for standing reservations

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Main Concepts - Summary



© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Exercise 5 - Submitting jobs using job arrays

Objective: Practice submitting array jobs

Prerequisites: none

PBS User Tasks:

1. Submit an array job that will run 10 subjobs
2. Submit an array job that will run at most 2 subjobs at a time

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Exercise 6 - Submitting PBS Jobs and using NODEFILE

Objective: Practice submitting jobs and observe behavior

Prerequisites: none

PBS User Tasks:

1. **qsub -l select=2:ncpus=1 job.script**
   - Each "chunk" describes 1 process of an MPI job equals 2 chunks of 1 ncpus by default NODEFILE will have 1 line/chunk, in this example 2 lines, each a hostname of where a chunk will run
2. **qsub -l select=1:ncpus=2:mpiprocs=2 job.script**
   - mpiprocs can be used to describe a custom NODEFILE, you can specify the number of processes/chunk in this case you again get 2 lines in the NODEFILE even though you requested 1 chunk

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Exercise 7 - Submitting Jobs to a Reservation Queue

Objective: Submitting a job to the reservation

Prerequisites: An approved reservation

PBS Administrator Tasks: Provide reservation queue

PBS User Tasks:

1. Submit job using **-q <reservation_queue>** flag
2. When the job is running, check **qstat** to see if it's within a reservation

Observation:

1. Verify that the job is in the reservation queue and running on the nodes reserved
2. If job isn't running or isn't accepted in reservation:
   - Was **walltime** within reservation scope?
   - Is the resource requested in job is within limits of resources reserved?

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

National Supercomputing Centre

Using GPU with PBS



REQUESTING GPU JOB RESOURCE
USING CUDA_VISIBLE_DEVICES TO ACCESS ASSIGNED GPUS

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Requesting GPU Job Resource

Host-based Resource

- GPUs can be requested as a host-level resource in chunks
- The resource name to be requested is : “ngpus”
- On ASPIRE2A, CPU and Memory value in a resource specification is automatically applied / modified when GPU resource is requested based on following ratio - 1 ngpus:16 ncpus:110GB mem
  - **Example:**
    - To request 1 GPU on 1 node, the resource specification to be defined in a PBS job script :
      #PBS –l select=1**:ngpus**=1**:**ncpus=16:mem=110GB:mpiprocs=16
      or
      #PBS –l select=1**:ngpus**=1
      (ncpus and mem will be automatically added. mpiprocs need to be manually defined)
    - To request 4 nodes with 4 GPU each, the resource specification to be defined in a PBS job script :
      #PBS –l select=4**:ngpus**=4

NOTE: if “ncpus” and “mem” values are defined in a job request, but does not follow as per the ratio, it will be replaced as per the ratio.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Accessing NVIDIA GPUs using assigned IDs

CUDA_VISIBLE_DEVICES

- A host can have either 4 or 8 GPUs on ASPIRE2A Cray and AI systems.
- GPUs are assigned to each job based on the requested value and are fenced, ensuring that only the assigned GPUs can be used. This allows other jobs to utilize other GPUs on the same host.
- Each GPU device on a node has a unique ID, which is used to specify which GPU an application should use. The $CUDA_VISIBLE_DEVICES variable is used for this purpose. In a multi-node job, the IDs are automatically assigned on the Mother Superior (Primary execution host). On Sister nodes, the IDs are stored in an environment file, which can be accessed using the $PBS_NODEFILE.env variable.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Accessing NVIDIA GPUs using assigned IDs

CUDA_VISIBLE_DEVICES

- To load the environment file with CUDA_VISIBLE_DEVICES values, users can run this command from inside a PBS Job:
  - $ source $PBS_NODEFILE.env
- To check the IDs of the GPUs assigned on a node, users can run this command:
  - $ echo $CUDA_VISIBLE_DEVICES
- Generic IDs (0, 1, 2, 3, etc.) are mapped to the unique IDs of the allocated GPUs for each job on a node. This mapping ensures that applications can refer to GPUs using consistent IDs across different nodes and jobs.
- Using unique IDs to assign to CUDA_VISIBLE_DEVICES is indeed recommended to avoid potential conflicts or issues. If more generic IDs are assigned to CUDA_VISIBLE_DEVICES than the number of GPUs allocated, the job may fail to run due to the mismatch between the expected and available GPU resources. **It is recommended not to change CUDA_VISIBLE_DEVICES value**.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Script Example

Single node, Single GPU job

```
#!/bin/bash
#PBS -q normal
#PBS -l select=1:ngpus=1:mpiprocs=1
#PBS -l walltime=8:00:00
#PBS -P <projectid>
#PBS -j oe
#PBS -N lammps_gpu_test

module load lammps/23Jun2022_update1-gpu          #Load correct environment
cd $PBS_O_WORKDIR || exit $?                      #Change current directory to submission
directory
#execute workload
mpirun -np 1 -d 1 --cpu-bind depth lmp -k on g 0 -sf kk -in in.lammps
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Script Example

Single node, Multiple GPU job

```
#!/bin/bash
#PBS -q normal
#PBS -l select=1:ngpus=1:mpiprocs=1
#PBS -l walltime=8:00:00
#PBS -P <projectid>
#PBS -j oe
#PBS -N lammps_gpu_test

module load lammps/23Jun2022_update1-gpu          #Load correct environment
cd $PBS_O_WORKDIR || exit $?                      #Change current directory to submission
directory
#execute workload
mpirun -np 1 -d 1 --cpu-bind depth lmp -k on g 0 -sf kk -in in.lammps
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Script Example

Single node, Multiple GPU job, Multiple tasks

```
#!/bin/bash
#PBS -q normal
##Requesting a full node with 4 GPU
#PBS -l select=1:ngpus=4:mpiprocs=64:ompthreads=1
#PBS -l walltime=8:00:00
#PBS -P <projectid>
#PBS -j oe
#PBS -N My_gpu_tasks

cd $PBS_O_WORKDIR || exit $?                #Change current directory to submission directory
#execute workloads
CUDA_VISIABLE_DEVICES=0 ./my_executable argument1 > result1 & PD1=$!
CUDA_VISIABLE_DEVICES=1 ./my_executable argument2 > result2 & PD2=$!
CUDA_VISIABLE_DEVICES=2 ./my_executable argument3 > result3 & PD3=$!
CUDA_VISIABLE_DEVICES=3 ./my_executable argument4 > result4 & PD4=$!
wait $PD1
wait $PD2
wait $PD3
wait $PD4
#The CUDA_VISIBLE_DEVICES variable defines on which GPU the task will run. Please ensure that tasks are run only on as many GPUs as requested.
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Job Script Example

Multiple node, Multiple GPU job

```
#!/bin/bash
#PBS -q normal
##Requesting 4 full node with 4 GPUs each
#PBS -l select=4:ngpus=4:mpiprocs=64:ompthreads=1
#PBS -l walltime=8:00:00
#PBS -P <projectid>
#PBS -j oe
#PBS -N My_gpu_tasks

cd $PBS_O_WORKDIR || exit $?                    #Change current directory to submission
directory
#execute multi-node workload
./my_multinode_workload
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

National Supercomputing Centre
SINGAPORE

ASPIRE2A AI Cluster



OVERVIEW
USE CASE OF LOCAL NVME STORAGE
PBS TEMP DIRECTORY
SAMPLE JOB SCRIPT FOR AI CLUSTER
QUERYING PBS COMMANDS FROM AI CLUSTER

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

ASPIRE2A AI Cluster

Overview

- ASPIRE2A system has 4 PBS clusters:
  - Pbs101 – Cray cluster which is has a total of 848 nodes (768 standard nodes, 64 GPU nodes and 16 large memory nodes)
  - Pbs102 – AI system which has a total of 18 nodes with local NVMe storage for high IOPs
  - Pbs103 – High throughput cluster with 16 High Frequency Nodes (Currently not open for users)
  - Pbs104 – Visualization cluster with 8 NVIDIA A40 GPU nodes (accessible via Visualization portal)
- ASPIRE2A AI system (pbs102) has following features:
  - Nodes have local NVMe storage for AI training that requires lower latency and greater bandwidth to process massive amount of data
  - 6 nodes with 8 NVIDIA A100 40GB GPUs,14TB local NVME scratch storage, 128 Cores and 944GB memory
  - 12 nodes with 4 NVIDIA A100 40GB GPUs,12TB local NVME scratch storage, 64 Cores and 440GB memory
  - All the nodes and storage are connected with the Slingshot high speed network.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

ASPIRE2A AI Cluster

Use case of local NVMe Storage

- While parallel file systems like Lustre excel at handling massive parallel I/O operations common in multi-node environments, local NVMe storage offers significant advantages for specific Machine Learning (ML) and Large Language Model (LLM) workloads, particularly on single nodes. Here's why:

  - **Reduced I/O Latency**: Local NVMe storage boasts significantly lower latency compared to traditional storage options like HDDs or even network-attached storage (NAS). This translates to faster data access times, crucial for ML and LLM tasks that involve frequent data reads and rereads during training and inference. Faster I/O translates to quicker training times and lower inference latency.
  - **Improved Performance for Single-Node Jobs**: Many ML and LLM workflows, especially in the research and development phase, involve running jobs on a single node. In such scenarios, local NVMe can significantly outperform Lustre by eliminating the network overhead associated with accessing data on a shared file system. This translates to faster model training and experimentation cycles.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

ASPIRE2A AI Cluster

Use case of local NVMe Storage

- **Optimized for Data Rereading**: LLMs, in particular, are known for rereading data multiple times during training and inference. Local NVMe's high bandwidth and low latency make it ideal for these workloads, as it can quickly retrieve the required data without introducing bottlenecks.
- **Hybrid Approach for Multi-Node Workloads**: For multi-node ML and LLM applications that involve both local and parallel I/O operations, a hybrid approach can be beneficial.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

PBS Temp Directory

For accessing local NVMe storage on ASPIRE2A AI cluster

- Users can utilize the local storage available on the compute nodes of the ASPIRE2A AI cluster by leveraging the PBS Temp Directory ($TMPDIR).
- The temporary directory is created for each job on each of its allocated nodes, allowing access from inside the job. After the job completes, this temporary directory is removed, clearing the local storage for the next job.
- There is no quota per user for this temporary storage. If users are not sharing the node with other jobs, they can utilize the entire 12/14TB available on the node for their job. However, if the node is running multiple jobs, users may not be able to use the entire available storage.
- As this is local storage, users must first copy the data they intend to use onto every node of their job individually.
- The variable which can be used to access the local temporary storage on a node is: $TMPDIR
- Temp Directory path and name format: /raid/pbs.<jobid>.pbs101

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Sample Job Script for AI Cluster

```
#!/bin/bash
#PBS -q ai
##Requesting a full node with 4 GPU
#PBS -l select=1:ngpus=4
#PBS -l walltime=24:00:00
#PBS -P <projectid>
#PBS -N My-AI-Job
#PBS -j oe

# Change directory to where job was submitted
cd $PBS_O_WORKDIR || exit $?

# Copy the dataset to TMPDIR
cp $HOME/mydataset/node1.tar $TMPDIR

# Untar the dataset
tar -xf $TMPDIR/node1.tar -C $TMPDIR

# Run the AI workload (The command here is indicative only)
$HOME/myscript --dataset-path $TMPDIR
```

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Querying PBS commands from AI cluster

- Submit a job to AI cluster, use “ai” queue :

  # PBS -q ai

- Query job state using “qstat” command, specify the AI cluster as follows:

  $ qstat -a @pbs102

  $ qstat -ns <job_ID> @pbs102

- Delete jobs using “qdel” command, specify the AI cluster as follows:

  $ qdel <job_ID>

- Check the nodes specification and status using “pbsnodes” command, specify server using -s flag:

  $ pbsnodes -aSj -s pbs102

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Exercise 8 – Submitting GPU jobs

Objective: Understand how to submit a job that is requesting GPU resource and resource fencing

Prerequisites: none

PBS User Tasks:

1. Create a job script that requests GPU resource in an interactive job.
2. Submit this job where it will go into execution right away.

Observation:

1. Verify that the requested number of GPUs are visible using “nvidia-smi” command

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Why Jobs May be in this State

- Jobs in "Q" state
  - Requested resources are **not available** or free
    - Not enough free ncpus, mem, ngpus, etc
  - No enough custom resources
  - **Queue** is not started
  - User has exceeded its resource limits
  - Scheduling parameter is **not enabled**
  - Check: server, scheduler, and mom logs
- Jobs in "W" state
  - Issues with SCP/password-less SSH between the PBS Server and PBS Mom
    - Job will be waiting for the input files to be copied to the job execution directory
  - User may not have proper permissions to copy the input files to the execution job directory

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Why Jobs May be in this State

Jobs in "H" state

- Issues with user's authentication on compute nodes
  - User's home directory not mounted or not available on the compute nodes
- Manually held jobs using **qsub** or **qhold** PBS
- Job is dependent on other job(s)
- Job failed 21 times and held due to multiple failures. Most likely the job was sent to a faulty node and it failed to get resources.

Jobs in "E" state

- Issues with password-less ssh/scp from the compute node to the PBS server
- Job submission directory doesn't have proper read/write permissions for that user



Note: Users can check the comments on job using command “qstat -anstw”

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Why Jobs can't be submitted

- Required SU balance may not be available in the project to submit the jobs.
- Passed incorrect project ID in the job script.
- Project is expired.
- Requested combination of resources which are not as per the allowed limits.
- Requested a resource which does not exist.
- Sent the job to an incorrect queue or a queue that only accept jobs from a routing queue.
- Submitted job script with no executable command.
- Syntax errors (like using "node" instead of "select")
- Invalid or no select statement in resource request.
- Job rejected by a hook.



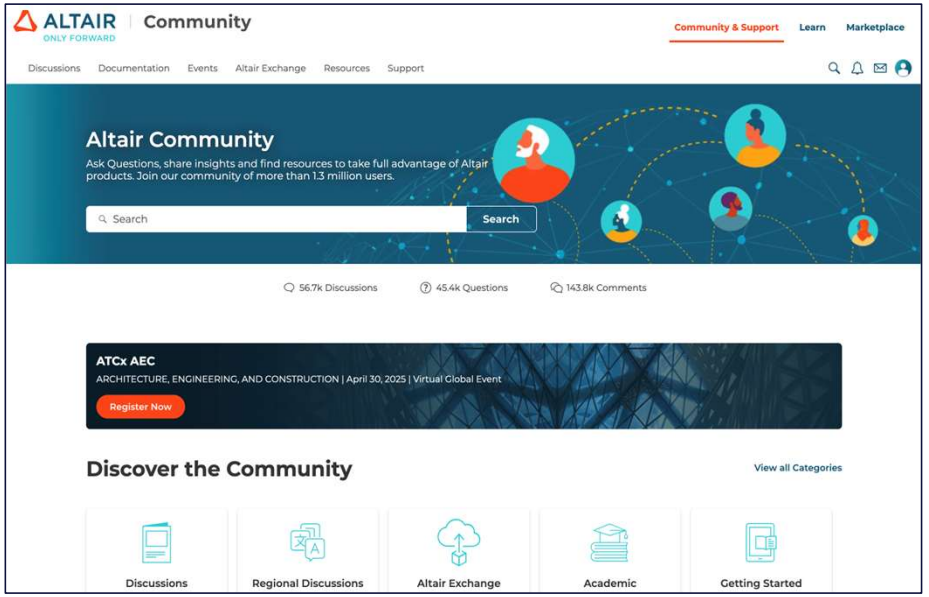
Note: Users get a statement if the job fails to get submitted explaining the reason of rejection.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Documentations and Altair Community

Website

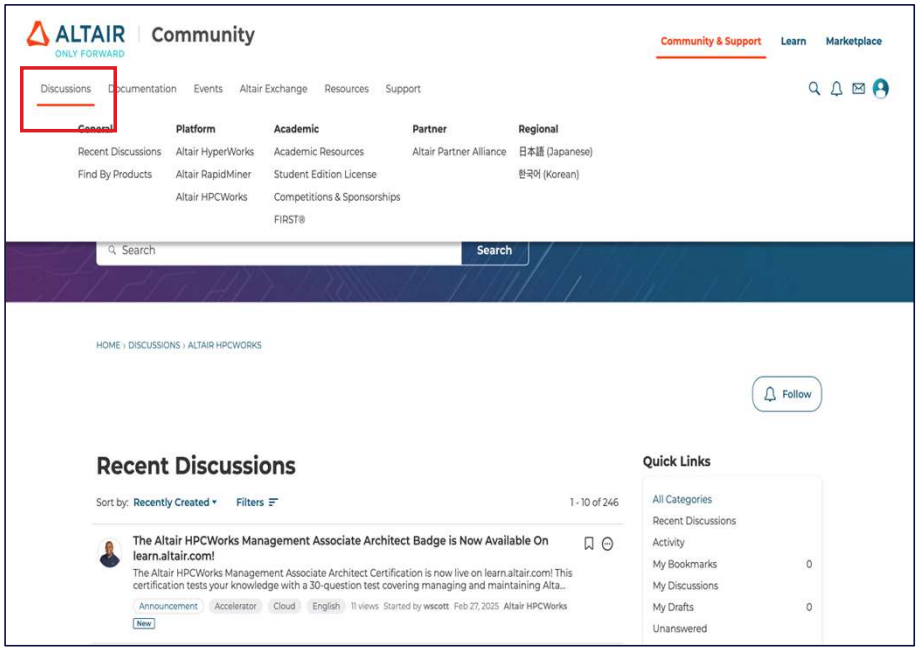
- https://community.altair.com/community?id=altair_product_documentation
  - PBS Works Documentation
    - Download the latest supported docs
    - Support
      - Phone and Email
- **https://https://keris.service-now.com/fapl_csm**
  - NSCC User Ticket Management
    - Create / comment / close support tickets
  - **Knowledge Base**
    - Find answers to questions



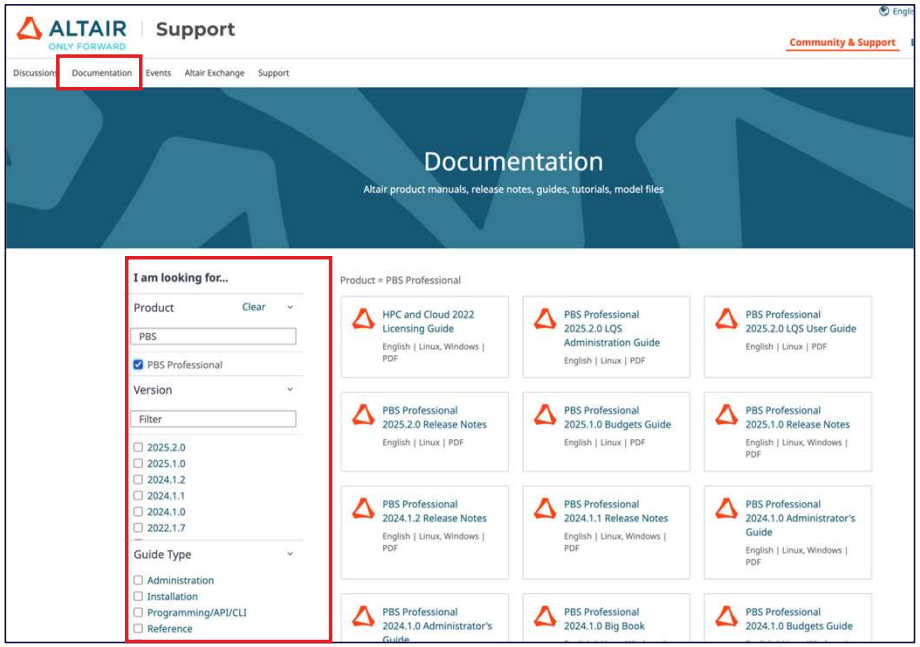
© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Welcome to our Community, Support, Services & Resources

Join our online community of experts
https://community.altair.com/



Download Tutorial, Installation and Documentation for Altair solutions:
https://support.altair.com



© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

National Supercomputing Centre

Thank you.

Email: contact@nscc.sg
Website: www.nscc.sg



NSCC.SG

National Supercomputing Centre

Discover Continuously.
Advance Infinitely.

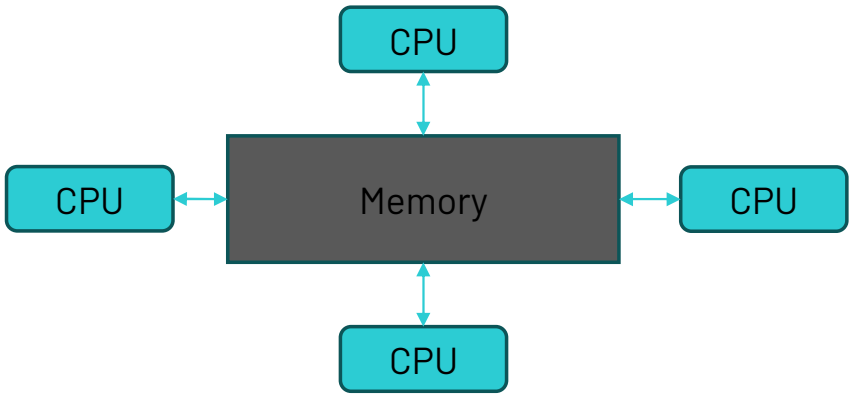


Visit altair.com to learn more.
#ONLYFORWARD

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Shared Memory

- Shared memory is the memory which all the processors can access.
- In hardware point of view it means all the processors have direct access to the common physical memory through bus based (usually using wires) access. These processors can work independently while they all access the same memory.
- Any change in the variables stored in the memory is visible by all processors because at any given moment all they see is a copy or picture of entire variables stored in the memory and they can directly address and access the same logical memory locations regardless of where the physical memory actually exists.



© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

SMP Programming

```
#include <stdio.h>
#include <omp.h>

#define NUM_THREADS 4

int main() {
  int sum = 0;

  omp_set_num_threads(NUM_THREADS);

  // Parallel region begins
  #pragma omp parallel shared(sum)
  {
    int thread_id = omp_get_thread_num();

    // Critical section to prevent race conditions
    #pragma omp critical
    {
      printf("Thread %d: Current sum is %d, incrementing by 1.\n", thread_id, sum);
      sum++;
      printf("Thread %d: New sum is %d.\n", thread_id, sum);
    }
  } // Parallel region ends

  printf("Final sum: %d\n", sum);

  return 0;
}
```

SMP (Symmetric Multiprocessing) refers to a type of multiprocessing where multiple processors (or cores) share a common memory space and are capable of accessing the same physical memory. In SMP systems, all processors have equal access to memory and can execute processes or threads concurrently, sharing resources and data.

1. **OpenMP Parallel Region**:

   The #pragma omp parallel shared(sum) directive creates a parallel region where all threads execute the code within the block. The variable sum is shared among all threads.

2. **Thread ID**:

   Each thread obtains its unique thread ID using omp_get_thread_num(). This ID is used to distinguish the actions of different threads.

3. **Critical Section**:

   The #pragma omp critical directive ensures that the block of code inside it is executed by only one thread at a time, preventing race conditions when multiple threads attempt to update the shared variable sum.

4. **Final Sum**:

   After all threads have finished executing the parallel region, the program prints the final value of sum.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

RDMA Vs SMP

RDMA (Remote Direct Memory Access)

1. **Definition**:

   RDMA allows for direct memory access from the memory of one computer into that of another without involving the operating system or CPU of the receiving computer. This is commonly used in distributed computing environments.

2. **Characteristics**:

   Remote Access: Facilitates communication between different machines over a network.

   Zero-Copy: Data transfers bypass the CPU, reducing latency and overhead.

   Low Latency and High Throughput: Provides high performance for data transfer compared to traditional network communication methods.

3. **Use Case**:

   RDMA is used in distributed systems where multiple machines need to access and exchange data efficiently, such as in large-scale HPC clusters, cloud computing, and data centers.

Can SMP be Achieved with RDMA?

- **SMP Context**: SMP refers to the use of multiple processors or cores sharing a common memory space within a single machine. RDMA is not typically used for SMP because it operates over a network and involves remote memory access.
- **Distributed Memory Systems**: RDMA is more suited to distributed memory systems where each node has its own memory and RDMA facilitates efficient inter-node communication.
- **Local Memory Access**: For local memory access within a single machine, shared memory (as used in SMP) is more appropriate. RDMA is designed for scenarios where you need to access memory across different machines, which goes beyond traditional SMP.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Types of Parallel Computing

	Single Instruction	Multiple Instructions	Single Program	Multiple Programs
Single Data	SISD	MISD		
Multiple Data	SIMD	MIMD	SPMD	MPMD

- **MIMD (Multiple Instruction, Multiple Data):**
  - **MIMD** is the parallelism model that MPI employs. In MIMD, multiple processors execute different instructions on different data simultaneously. Each process runs independently, and they can perform different tasks, making MIMD highly flexible for distributed computing.
  - MPI allows processes to communicate with each other using message passing, enabling coordination in a distributed memory environment.
  - Example: Different processes can be working on different parts of a simulation, and they exchange information as needed.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

MPI

```
#include <mpi.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char** argv){
  char hostname[256];
  gethostname(hostname, sizeof(hostname));

  printf("This runs before MPI_Init on processor %s - executed by the single process
before parallel region.\n", hostname);

  MPI_Init(&argc, &argv);

  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

  printf("Hello from processor %s, rank %d out of %d processors\n",
      hostname, world_rank, world_size);

  MPI_Barrier(MPI_COMM_WORLD);

  MPI_Finalize();

  printf("This runs after MPI_Finalize on processor %s - executed by the single process
after parallel region.\n", hostname);

  return 0;
}
```

MPI_COMM_WORLD:
This is the default communicator provided by MPI, which includes all the processes launched by mpirun or mpiexec. It's a handle that allows communication among all the processes in the group.

Rank (MPI_Comm_rank):
The rank is a unique identifier assigned to each process within a communicator. It starts from 0 and goes up to size - 1 (where size is the total number of processes). The rank is used to distinguish processes and allows them to perform different tasks or manage specific data.

Size (MPI_Comm_size):
This function returns the total number of processes in the communicator (like MPI_COMM_WORLD). It's used to determine how many processes are participating in the communication.

© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.

Solving system of linear equations

Gaussian elimination using the **upper triangular form**.

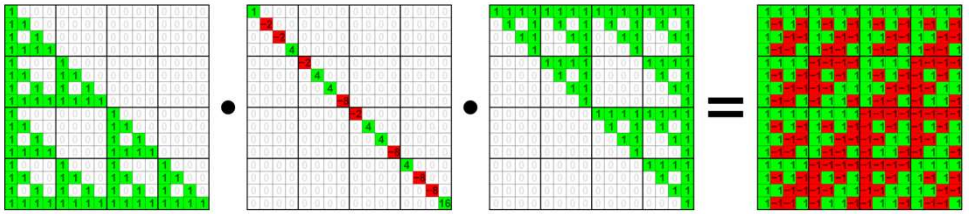
Algorithm: Parallel Gaussian Elimination (Upper Triangular Form)

Input: A system of linear equations represented by the matrix A and vector b, where A is an $n \times n$ matrix and b is an n-dimensional vector.

Output: Solution vector x such that $A \cdot x = b$.

Steps:

1. **Initialization**:
   - Let A be an $n \times n$ matrix.
   - Let b be an n-dimensional vector.
   - Convert the system into an augmented matrix $[A|b]$.
2. **Forward Elimination (Parallelized)**:
   - **For** $k = 0$ to $n - 1$ (pivot row):
     - **Parallelize**:
       - **For** each row i where $i > k$ (below the pivot row):
         - Calculate the multiplier: $m_{ik} = \frac{A[i][k]}{A[k][k]}$.
         - **Parallelize** the following operations for each element in row i:
           - **For** each column j from k to $n - 1$:
             - $A[i][j] = A[i][j] - m_{ik} \cdot A[k][j]$.
           - Update the corresponding element in vector b:
             - $b[i] = b[i] - m_{ik} \cdot b[k]$.
3. **Back Substitution**:
   - **Initialize** an empty vector x of size n.
   - **For** $i = n - 1$ to 0 (from the last row to the first row):
     - Compute $x[i]$ using:
       - $x[i] = \frac{b[i] - \sum_{j=i+1}^{n-1} A[i][j] \cdot x[j]}{A[i][i]}$.
4. **Return** the solution vector x.



© Altair Engineering, Inc. Proprietary and Confidential. All rights reserved.