Porting and Optimization Workshop

Hawk Batch System

(PBSpro)

Dr.-Ing. Martin Bernreuther

Overview

- PBS overview
- some PBSpro
  - commands
  - environment variables
  - concepts

Batch processing systems

- Batch systems administrate ressources:
  - compute nodes (potentially different types), exclusively
  - pre/postprocessing nodes
  - parts of (large memory) nodes
    - Cores
    - Memory
  - other ressources (e.g. licenses?)
- a user job will ask for ressources for a requested walltime and the batch system will allocate free ressouces to the jobs for that time.
- a user job might execute commands through a batch script (batch job) or through a shell (interactive job)

PBS History

- 1991: NASA startet to develop the "**P**ortable **B**atch **S**ystem" with contractor MRJ Technology Solutions
- 1998: MRJ released a open source „OpenPBS" version
- 2003: Altair Engineering acquired PBS technology (from Veridian, which acquired MRJ before). → PBS professional
- on the other hand, Adaptive Computing Enterprises maintains/commercializes a fork of OpenPBS → TORQUE (with MOAB)
- 2016: release of a PBSpro Open Source version (s. https://www.pbspro.org/)
- PBSpro and TORQUE have the same roots (and still similarities)
- at HLRS moving from Hazel Hen to Hawk, TORQUE/Moab (and apshed) was replaced by PBSpro (some HLRS systems use other Batch systems like SLURM)

PBS components

- Server (pbs_server.bin daemon)
  - manages nodes, queues, jobs
- Scheduler (pbs_shed daemon)
  - determine when and where to run jobs
  - enforce scheduling policies
- Machine **o**riented **M**ini-server (pbs_mom daemon)
  - execute user job scripts
  - manage job processes
  - job scripts running on first job node (Hawk)

    or dedicated MoM nodes (Hazelhen)

pbsnodes

- `pbsnodes` lists the known ressources, e.g.:

```
$ pbsnodes -aSj
                                                              mem       ncpus   nmics   ngpus
vnode           state           njobs   run   susp      f/t        f/t     f/t     f/t   jobs
--------------- --------------- ------ ----- ------ ------------ ------- ------- -------
-------
r1c1n1          job-exclusive        1     1      0  252gb/252gb 127/128     0/0     0/0 6351
r1c1n2          job-exclusive        1     1      0  252gb/252gb 127/128     0/0     0/0 6408
r1c1n3          job-exclusive        1     1      0  252gb/252gb 127/128     0/0     0/0 6409
r1c1n4          free                 0     0      0  252gb/252gb 128/128     0/0     0/0 --
r1c2n1          free                 0     0      0  252gb/252gb 128/128     0/0     0/0 --
r1c2n2          free                 0     0      0  252gb/252gb 128/128     0/0     0/0 --
r1c2n3          free                 0     0      0  252gb/252gb 128/128     0/0     0/0 --
r1c2n4          free                 0     0      0  252gb/252gb 128/128     0/0     0/0 --
r2c1n1          free                 0     0      0  252gb/252gb 128/128     0/0     0/0 --
r2c1n2          free                 0     0      0  252gb/252gb 128/128     0/0     0/0 --
r2c1n3          free                 0     0      0  252gb/252gb 128/128     0/0     0/0 --
r2c1n4          free                 0     0      0  252gb/252gb 128/128     0/0     0/0 --
r2c2n1          free                 0     0      0  252gb/252gb 128/128     0/0     0/0 --
r2c2n2          free                 0     0      0  252gb/252gb 128/128     0/0     0/0 --
r2c2n3          free                 0     0      0  252gb/252gb 128/128     0/0     0/0 --
r2c2n4          free                 0     0      0  252gb/252gb 128/128     0/0     0/0 --
rome1           job-exclusive        1     1      0  503gb/503gb 255/256     0/0     0/0 6413
```

Submitting a Job

- `qsub` submits a job (and will return a Job-ID on success)
- job types:
  - interactive job:

    `qsub -I` option (add `-X` to redirect X11 output)
    → user gets a shell prompt (similar to login shell)
    input/output through shell
  - batch job:
    - if filename given, input will be read from a job-script
      (usually a bash-script, but also e.g. python… can be used)
    - if no filename is given, `qsub` will read commands from stdin
      (input can be piped into `qsub`)

    output (stdout,stderr) by default will be redirected to files (*.[oe][0-9]*.*)
    qsub options can also be defined at the beginning of a job-script
    within `#PBS` comments before the first command
- within a job, a new job can be submitted

qsub ressource definition

- `qsub -l <ressouce>=<value>` option with
  - select=<NNODES>[:node_type=<NODETYPE>]
    [:ncpus=<NCORES>][:mem=<MEM>]...[+...]
    to ask for chunks with NCORES cores on each of NNODES nodes of type NODETYPE (NNODES default=1)
  - walltime=<WALLTIME>
    for a given WALLTIME in sec or HH:MM:SS
- examples:

```
$ qsub -I -l walltime=300 -l select=16:node_type=naples
  # interactive job with 16 naples nodes for 5 min
$ qsub -q workday -IXl select=1:node_type=rome:ncpus=128,walltime=1800
 # use workday queue for a X11 interact. job on 128 rome cores for 30min
$ qsub -l select=1:node_type=naples,walltime=0:30:00 testjob.pbs
 # 30min batch job on a naples node
$ echo "date; hostname" | qsub select=1:node_type=rome,walltime=0:01:00
 # pipe in commands to be executed on a rome node for max. 1 min
```

more qsub options (1)

- `qsub -q <queue>`
  - choose a queue (without, a default routing queue is used)
- `qsub -N <JobName>`
  - define job name
- `qsub -m [n|a|b|e|j] -M <Emailaddresses>`
  - Email notification at **a**bortion,**b**egin,**e**nd (for sub**j**ob)
- `qsub -a [[[[YYYY]MM]DD]hhmm[.SS]]`
  - deferring job execution
- `qsub -h`
  - hold a job

more qsub options (2)

- `qsub -o <path> -e <path>`
  - specify path for output and error files
- `qsub -j [n|oe|eo]`
  - join/merge output and error file
- `qsub -W <attr=value>`
  - set job attributes, like e.g. dependencies (cmp. next slide) or accounting group (`-W group_list=abc12345`)
- `qsub -V|-v <variable>=<value>[,<variable>=<value>…]`
  - export all (DANGEROUS) | set specific environment variables

Job dependencies

- `qsub -W depend=<DEPDEF>` to ensure a certain order for the job execution
- the dependencies might be depending on the job exit returncode (0 corresponds to „OK“, ≠0 to „NOTOK“; „ANY“ for both)
- execute the job „after“ another job started/terminated or another job „before“ the submitted job
  - e.g. `afterok:<JOBIDs>`
    schedule for execution after jobs with JOBIDs terminated without errors (exit with 0)
    (before the job is in **H**old state)
- example:

```
$ qsub testjob1.pbs
1233.hawk-tds-pbs1
$ qsub testjob2.pbs
1234.hawk-tds-pbs1
$ qsub -W depend=after:1233:1234 testjob3.pbs
  # schedule for execution after job 1233 and 1234 started
```

Job environment variables

- PBSpro environment variables set on MoM node:

PBS_JOBID	job identifier
PBS_JOBNAME	job name
PBS_O_WORKDIR	absolute path of current directory, qsub was executed
PBS_O_PATH	$PATH value, when qsub was executed
PBS_NODEFILE	name of file containing a list of the nodes
PBS_ENVIRONMENT	PBS_BATCH or PBS_INTERACTIVE
PBS_O_QUEUE, PBS_QUEUE	submission, execution queue
PBS_O_HOST, PBS_O_LOGNAME	submission host, username

(hybrid) MPI jobs

- `mpiprocs` ressource is used to duplicate the node entries in the `PBS_NODEFILE` (default: 1)
- `ompthreads` ressource to set environment variable `OMP_NUM_THREADS` on all nodes (default: ncpus or 1)
- examples:

```
#testjob_mpi.pbs
#PBS -l select=10:node_type=rome:ncpus=128:mpiprocs=128
#PBS -l walltime=0:30:00
cd $PBS_O_WORKDIR # default PWD: $HOME
mpirun -np 1280 ./mpihello

#testjob_mpiomp.pbs
#PBS -l select=2:node_type=rome:ncpus=128:mpiprocs=32:ompthreads=4
#PBS -l walltime=0:30:00
mpirun -np 64 omplace -nt 2 -vv ./mpihello
```

PBS remote shell replacement

- `ssh` (or formerly `rsh`) can be used to execute a command on a remote host (in the past also MPI startup used this)
- but processes started with ssh are not „under control“ of PBS (but eventually also killed by the epilogue script)
- `pbs_tmrsh` is a `rsh` replacement, which will execute commands remotely as child processes of the pbs_mom daemon. (e.g. signals can be forwarded to the processes…)
- alternatively `pbs_attach` initiates the pbs_mom daemon to „adopt“ a process.

Job-Arrays

- a job array is a collection of batch subjobs with an unique array index („rank“ parameter) assigned to each
- `qsub -J <i0>-<i1>:<step>` option to define the range
- the `PBS_JOBID` (e.g. 1234[1]) is a composition of the `PBS_ARRAY_ID` (e.g. 1234[]) and a `PBS_ARRAY_INDEX` (e.g. 1)
  - `qdel <JOBID>; qdel <ARRAYID>`
  - stdout/stderr output: `jobname.[oe]<JOBID>.[0-9]*`
- example:

```
#PBS -l select=1,walltime=0:30:00
#PBS -J 1-10
#PBS -r y
echo "PBS_JOBID:${PBS_JOBID}"
echo "PBS_ARRAY_ID:${PBS_ARRAY_ID}, PBS_ARRAY_INDEX:${PBS_ARRAY_INDEX}"
# do something dependend on $PBS_ARRAY_INDEX
```

job properties

- `qstat` is the utility to list job attributes (of jobs the user is authorized to)
- job listings:
  - `qstat` without options
    JobID, Jobname, User, used cputime, State, Queue
  - `qstat -a`
    JobID, User, Queue, Jobname, SessionID, #Nodes #Tasks, Req.Memory, Req.Time, State, Elapsed Walltime
  - `qstat -s`
    `-a` output with comments
    - `-w` option for untruncated wide fields
    - `-1` option: print data entries in a single line
  - `qstat -T`
    display estimated start time
  - `qstat -H; qstat -x`
    list finished jobs; list all jobs
  - `qstat -f <JobIDs>`
    show attributes of specific Jobs
    - `-w` option to omit line breaks, e.g. `qstat -fw 1233 1234`
    - `-F json` (or `-F dsv`) to produce other formats, like JSON

queue properties

- `qstat` also lists queue attributes
- queue listings:
  - `qstat -q`
    Queue, Memory, CPU Time, Walltime, Node, #Running, #Queued, Lm, State
  - `qstat -Q`
    Queue, #MaxAllowed, #Total, Enabled?, Started?, #Queued, #Running, #Held, #Waiting, #Transiting(Moved), #Exiting, Type(exec/routing)
  - `qstat -Qfw <Queuename>`
    show attributes for a specific queue
- server listings:
  - `qstat -B`

deleting/modifying jobs

- `qdel <JobIDs>`
  delete a job
- `qhold <JobIDs>`
  set a user hold state for job JobID
- `qmove <newqueue> <JobIDs>`
  move a job to another queue
- `qalter <newoptions> <JobIDs>`
  alter the qsub options after submission. ncpus and memory cannot be changed when the job is already running (and users are not allowed to increase the walltime of a running job)

sending a signal

- `qsig -s <SIGNAL> <JobID>`
  send a signal to the job-script of a job
- SIGNALs:
  - suspend
  - resume
  - SIGTERM terminate
  - SIGHUP hang up
  - SIGINT interrupt
  - SIGKILL kill
- might be processed by a signal handler within the job-script

selecting jobs

- `qselect` without any arguments „lists all jobs at the server which the user is authorized to list“

  (due to the restrictions equivalent to `qselect -u $USER`)
- options act as filters
  - `qselect -N <JobName>`
    select job with given name
  - `qselect -s R`
    select **R**unning jobs (analog, e.g. **Q**ueued, **H**eld, Finished,…)
  - qselect -l walltime.le.0:30:0
    select jobs with a requested walltime≤30min
- particularly useful in combination with other commands, e.g.

  `qdel ``qselect -u $USER``` to delete all jobs of the user

HLRS HAWK configuration

- common routing queue,
  but special queues e.g. for XXL jobs
- due to privacy and security issues, users only have the authorization to see their own job properties

  (e.g. `finger` is also disabled)
- large jobs might have to use chunks of 64 nodes
  - improving topology awareness
  - reducing offcut

HLRS add-ons

- `batchstat`
  show (reduced information of) queued jobs of *all* users
- `nstat` only on Vulcan

- `qwtime.sh` (after `module load cae`)
  show the (elapsed and remaining) walltime of a running job (e.g. for a quick check within an interactive job)

more information

- man pages

  ```
  man pbs_professional
  ```

- https://www.altair.com/pbs-works-documentation/
  - PBS Professional User‘s guide

    https://www.altair.com/pdfs/pbsworks/PBSUserGuide19.2.3.pdf
  - PBS Professional Reference Guide

    https://www.altair.com/pdfs/pbsworks/PBSReferenceGuide19.2.3.pdf