Combining Quick-turnaround and Batch Workloads at Scale

Greg Matthews – Deputy PBS Administrator
NASA Advanced Supercomputing (NAS) Division
NASA Ames Research Center, Moffett Field, CA
Gregory.Matthews@nasa.gov
Pleiades on an Average Day

• Nearly 12,000 compute nodes
  • Each node runs pbs_mom
    • sharing=default_excl

• Job mix
  • 200-500 running jobs
  • 100-400 queued jobs

• Scheduling cycle
  • Max length 10-15 minutes
  • Average 2-5 minutes
  • Bulk of time is spent putting 6-10 top jobs on the calendar (backfill_depth)
  • No job preemption
Different Workloads

• Batch
  • Submit it and forget it (for a while)
  • Vast majority of our CPU-hours

• Quick-turnaround (aka “devel queue”)
  • Enabling jobs
    • Code development
    • Setup jobs for large batch runs
  • “Warm-body” jobs, need a shell prompt
    • Limit 1 per user
  • Threshold-to-coffee-break
    • Shrinks every iPhone release
Earlier Devel Queue Approaches

• Standing reservation
  • Modeled on warm-body availability

• Queue with assigned nodes (24x7 availability)
  • Trade decrease in utilization for:
    • Predictability (fewer agitated users)
    • Decrease in scheduling complexity

• Both the above approaches resulted in more coffee breaks, user agitation

• Separate PBS server
  • Users happy
  • Admins agitated
  • Management somewhere in between

  • Loss of flexibility to resize quick-turnaround resources
A Better Way

• Get back to 1 PBS server/scheduler
  • Primary problem is the scheduling cycle length
    • So why not just interrupt the scheduler?
  • Devel queue jobs start (more) quickly
    • Batch jobs pay the penalty, but they won’t notice

• The One Big Assumption
  • Devel queue has nodes assigned to it

• We modified the scheduler
  • Given the appropriate signal the scheduler will stop the current cycle and start a new one (within 1-5 seconds)
  • Issue the signal from a server hook
  • Guard against scheduler DOS:
    • Add a tunable parameter for minimum cycle length before interruption
    • Add another parameter for max consecutive interruptions
Job Priority Game

- Our server hook implements local policy as to which jobs qualify
  - Hook is limited to 1-bit signal
  - Scheduler has no clue which job(s) triggered a cycle interruption

- New cycle needs to look at devel queue jobs immediately
  - Sorting devel queue jobs to the top may consume top job slots

- One mod begets another: created a separate pool of top job slots
  - Used only for devel-like queues via setting a queue-level resource flag
  - Reserves backfill_depth for batch workload
  - Partly controls the additional scheduling cost paid for quick-turnaround

- One more mod for good measure
  - Our local job sorting uses “node count” as a last sort key
  - Favors larger jobs in the batch workload
  - Doesn’t make sense for devel
  - We ignore “node count” sorting for queues with a new queue-level resource flag
Example

Cycle A
...
Job cannot run
Cycle interrupted!

Cycle B
Job D1 run
Job D2 will run at time X
...
Job Dj run
Job B1 will run at time Y
...
Job Bk will run at time Z
...
...
Job Bn run

← separate_pool = j
← backfill_depth = k

Cycle C
...

← separate_pool = j
← backfill_depth = k
A Bright Future

• Why all the fuss about using just 1 PBS server?
  • Now the real fun begins: resizing devel queue as needed
    • Moving nodes around now as “simple” as assigning/unassigning queue
    • Return to modeling node availability on warm-body availability

• Our proposed approach
  • Set basic target numbers
    • E.g. X nodes at 4AM (Pacific), 2X nodes at 9AM, 0.1X at 6PM
  • Enforce an absolute maximum
  • Set targets for number of *free* nodes
    • Critical to achieve quick-turnaround

• Interesting questions yet to be answered
  • Can we maintain some semblance of node continuity?
  • What is the best mechanism for moving nodes around?
    • High priority 1-node jobs?
  • What type of node movement rate can we expect on average?
Questions?