pbsacct: A Workload Analysis System for PBS-Based HPC Systems

Troy Baer
Senior HPC System Administrator
National Institute for Computational Sciences
University of Tennessee

Doug Johnson
Chief Systems Architect
Ohio Supercomputer Center
Overview

• Introduction to pbsacct

• Technical Overview
  – Database Structure
  – Data Ingestion
  – User Interfaces

• Example Deployments

• Workload Analysis
  – NICS Kraken historical retrospective
  – OSC Oakley

• Conclusions and Future Work
Introduction to pbsacct

- pbsacct started at Ohio Supercomputer Center in 2005:
  - Grew from need to do workload analysis from PBS/TORQUE accounting logs.
  - Stores job scripts as well as accounting log data.
  - Ability to do on-demand queries on jobs across multiple systems and arbitrary date ranges.
  - Despite the name, not an allocation/charging system!
  - Open source (GPLv2)

- Structure:
  - Data sources
  - Database (MySQL)
  - User interfaces

- Development moved to NICS in 2008.
  - Available at http://www.nics.tennessee.edu/~troy/pbstopools/
pbsacct Architecture
Database Structure

• Accounting data and scripts are stored in a MySQL database

• Two tables:
  – Jobs
    • Job accounting data and scripts
    • Used by just about everything
    • Indexed by system, username, groupname, account, queue, submit_date, start_date, and end_date to accelerate queries
  – Config
    • Used to track system changes WRT core count
    • Mainly used by web interface to compute utilization
Data Ingestion

• Accounting data comes in from hosts that run `pbs_server`:
  – A Perl script called `job-db-update` parses the accounting logs in `$PBS_HOME/server_priv/accounting` and inserts the results into the database.
  – Typically run out of a `cron` job (hourly, daily, etc.).

• Job scripts can also be captured on hosts that run `pbs_server`:
  – `dnotify`- or `inotify`-based daemon watches for new files created in `$PBS_HOME/server_priv/jobs`.
  – When new `.SC` files are created in the `jobs` directory, daemon launches a Perl script called `spool-jobscripts`.
  – `spool-jobscripts` copies the `.SC` files to a temp directory and launches another Perl script called `jobscript-to-db`, which inserts the scripts into the database.
  – This is done to be able to keep up with high throughput situations where there may be thousands of short-running jobs in flight and the database might not be able to keep up.
User Interfaces

• Command line
  – js – Look up job script by jobid.
  – Want to develop more, but need to figure out a workable security model.

• Web
  – PHP based, using several add-ons
    • PEAR DB
    • PEAR Excel
    • OpenOffice spreadsheet writer
    • jQuery
  – Lots of premade reports
    • Individual jobs, software usage, utilization summaries...
    • Site-specific rules to map job script patterns to applications
  – Meant to be put behind HTTPS
Web Interface Example
Example Deployments

• **OSC**
  - ~14.9M job records (~13.4M with job scripts)
  - ~30GB database size
  - Web interface accessed over HTTPS with HTTP Basic authentication against LDAP

• **NICS**
  - ~5.4M job records (~5.0M with job scripts)
  - ~13.1GB database size, growth rate of ~600MB/month
  - Web interface accessed over HTTPS with RSA Securid one-time password authentication
Workload Analysis: NICS Kraken
Historical Retrospective

• NICS Kraken
  – Cray XT5 system with 9,408 dual-Opteron compute nodes
  – Operated in production for NSF from February 4, 2008, to April 30, 2014
  – Batch environment is TORQUE, Cray ALPS, and Moab
  – Queue structure:
    • batch (routing queue)
      – small (0-512 cores, up to 24 hours)
      – longsmall (0-256 cores, up to 60 hours)
      – medium (513-8192 cores, up to 24 hours)
      – large (8193-49536 cores, up to 24 hours)
      – capability (49537-98352 cores, up to 48 hours)
      – dedicated (98353-112896 cores, up to 48 hours)
    • hpss (0 cores, up to 24 hours)
## Kraken Workload Analysis
### 2009-02-04 to 2014-04-30

<table>
<thead>
<tr>
<th>Overall</th>
<th>NSF Teragrid/XSEDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>• 4.14M jobs</td>
<td>• 3.84M jobs</td>
</tr>
<tr>
<td>• 4.08B core-hours</td>
<td>• 3.85B core-hours</td>
</tr>
<tr>
<td>• 2,657 users</td>
<td>• 2,252 users</td>
</tr>
<tr>
<td>• 1,119 projects</td>
<td>• 793 projects</td>
</tr>
</tbody>
</table>

85.6% average utilization (not compensated for downtime)
### Kraken Workload Analysis by Queue
#### 2009-02-04 to 2014-04-30

<table>
<thead>
<tr>
<th>QUEUE</th>
<th>JOBS</th>
<th>CORE HOURS</th>
<th>USERS</th>
<th>PROJECTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>small</td>
<td>3,576,368</td>
<td>768,687,441</td>
<td>2,602</td>
<td>1,090</td>
</tr>
<tr>
<td>longsmall</td>
<td>3,570</td>
<td>2,782,681</td>
<td>169</td>
<td>122</td>
</tr>
<tr>
<td>medium</td>
<td>488,006</td>
<td>2,003,837,680</td>
<td>1,447</td>
<td>718</td>
</tr>
<tr>
<td>large</td>
<td>27,908</td>
<td>983,795,230</td>
<td>521</td>
<td>301</td>
</tr>
<tr>
<td>capability</td>
<td>2,807</td>
<td>306,724,698</td>
<td>117</td>
<td>73</td>
</tr>
<tr>
<td>dedicated</td>
<td>338</td>
<td>11,765,421</td>
<td>17</td>
<td>7</td>
</tr>
<tr>
<td>hpss</td>
<td>36,462</td>
<td>53,285</td>
<td>184</td>
<td>123</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td>4,136,759</td>
<td>4,077,647,799</td>
<td>2,657</td>
<td>1,119</td>
</tr>
</tbody>
</table>
Kraken Workload Analysis by Queue
2009-02-04 to 2014-04-30

Kraken Job Count By Queue

Kraken Core-Hours By Queue

- small
- longsmall
- medium
- large
- capability
- dedicated
- hpss
## Kraken Top 10 Applications by Core Hours
### 2009-02-04 to 2014-04-30

<table>
<thead>
<tr>
<th>APP</th>
<th>JOBS</th>
<th>CORE HOURS</th>
<th>USERS</th>
<th>PROJECTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>namd</td>
<td>347,535</td>
<td>421,255,609</td>
<td>358</td>
<td>164</td>
</tr>
<tr>
<td>chroma</td>
<td>38,872</td>
<td>178,790,933</td>
<td>17</td>
<td>10</td>
</tr>
<tr>
<td>res</td>
<td>58,630</td>
<td>161,570,056</td>
<td>268</td>
<td>190</td>
</tr>
<tr>
<td>milc</td>
<td>22,079</td>
<td>146,442,361</td>
<td>37</td>
<td>21</td>
</tr>
<tr>
<td>gadget</td>
<td>6,572</td>
<td>131,818,157</td>
<td>29</td>
<td>21</td>
</tr>
<tr>
<td>cam</td>
<td>66,267</td>
<td>124,427,700</td>
<td>88</td>
<td>68</td>
</tr>
<tr>
<td>enzo</td>
<td>15,077</td>
<td>112,704,917</td>
<td>54</td>
<td>37</td>
</tr>
<tr>
<td>amber</td>
<td>103,710</td>
<td>110,938,365</td>
<td>208</td>
<td>120</td>
</tr>
<tr>
<td>vasp</td>
<td>148,686</td>
<td>94,872,455</td>
<td>147</td>
<td>85</td>
</tr>
<tr>
<td>lammps</td>
<td>137,048</td>
<td>94,398,544</td>
<td>187</td>
<td>127</td>
</tr>
</tbody>
</table>
Workload Analysis: OSC Oakley

- **OSC Oakley**
  - HP Xeon cluster with 693 compute nodes
    - Most nodes are dual-Xeon with 12 cores
    - One node is quad-Xeon with 32 cores and 1TB RAM
    - 64 nodes have 2 Nvidia M2070 GPUs each
  - Operated in production since March 19, 2012
  - Batch environment is TORQUE and Moab
  - Queue structure:
    - batch (routing queue)
      - serial (1-12 cores, up to 168 hours)
      - parallel (13-2040 cores, up to 96 hours)
    - longserial (1-12 cores, up to 336 hours)
    - longparallel (13-2040 cores, up to 250 hours)
    - dedicated (2041-8336 cores, up to 48 hours)
    - hugemem (32 cores, up to 1 TB mem, up to 48 hours)
Oakley Workload Analysis
2012-03-19 to 2014-03-14

Overall

• 2.12M jobs
• 112M core-hours
• 1,147 users
• 403 projects

77.6% average utilization (not compensated for downtime)
### Oakley Workload Analysis by Queue
#### 2012-03-19 to 2014-03-14

<table>
<thead>
<tr>
<th>QUEUE</th>
<th>JOBS</th>
<th>CORE HOURS</th>
<th>USERS</th>
<th>PROJECTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>serial</td>
<td>1,799,890</td>
<td>32,938,880</td>
<td>1,088</td>
<td>387</td>
</tr>
<tr>
<td>parallel</td>
<td>324,848</td>
<td>77,614,464</td>
<td>595</td>
<td>256</td>
</tr>
<tr>
<td>longserial</td>
<td>36</td>
<td>58,456</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>longparallel</td>
<td>158</td>
<td>1,574,567</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>hugemem</td>
<td>299</td>
<td>54,466</td>
<td>28</td>
<td>23</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>2,125,231</strong></td>
<td><strong>112,240,833</strong></td>
<td><strong>1,147</strong></td>
<td><strong>403</strong></td>
</tr>
</tbody>
</table>
Conclusions and Future Work

• **pbsacct** is feature rich and extensible
  – Written in Perl and PHP
  – Support for site-specific code
  – Scales to millions of jobs across tens of machines

• Future work
  – Better packaging to ease installation – RPMs?
  – Port to another DBMS (e.g. PostGreSQL)?
  – Speed up full text job script searches with external indices (e.g. Apache Lucene Solr)?
  – Interface with other RM (Grid Engine, SLURM)?