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

Troy Baer
National Institute for Computational Sciences
University of Tennessee
Oak Ridge, Tennessee, USA
tbaer@utk.edu

Doug Johnson
Ohio Supercomputer Center
Columbus, Ohio, USA
djohnson@osc.edu

ABSTRACT

The PBS family of resource management systems have historically not included workload analysis tools, and the currently available third-party workload analysis packages have often not had a way to identify the applications being run through the batch environment. This paper introduces the `pbsacct` system, which solves the application identification problem by storing job scripts with accounting information and allowing the development of site-specific heuristics to map job script patterns to applications. The system consists of a database, data ingestion tools, and command-line and web-based user interfaces. The paper will discuss the `pbsacct` system and deployments at two sites, the National Institute for Computational Sciences and the Ohio Supercomputer Center. Workload analyses for systems at each site are also discussed.

Categories and Subject Descriptors

D.4.7 [**Organization and Design**]: Batch processing systems; K.6.2 [**Installation Management**]: Performance and usage measurement

Keywords

batch processing, NICS, OSC, PBS, TORQUE, workload analysis

1. INTRODUCTION

Most high performance computing (HPC) systems include a batch processing or resource management system as part of their user environment. One of the most commonly used types of HPC batch system is the Portable Batch System (PBS) family, consisting of three code bases – OpenPBS [12], PBS Pro [1], and TORQUE [3] – derived from the original Portable Batch System developed at NASA Ames Research Center in the 1990s [10]. As the name implies, the PBS family are modular and portable to a wide variety of platforms, including most proprietary and open source UNIX variants as well as Linux. The PBS family of batch systems share a common user interface – the IEEE POSIX.2d batch environment standard [4] – and a largely similar accounting log format. However, the PBS family have historically not included tools for doing statistical analyses or allocation management with the resulting accounting logs, on the theory that this is a site problem and every site would want to do it differently.

XSEDE '14 July 13 - 18 2014, Atlanta, GA, USA

ACM 978-1-4503-2893-7/14/07 ...$15.00.
http://dx.doi.org/10.1145/2616498.2616539

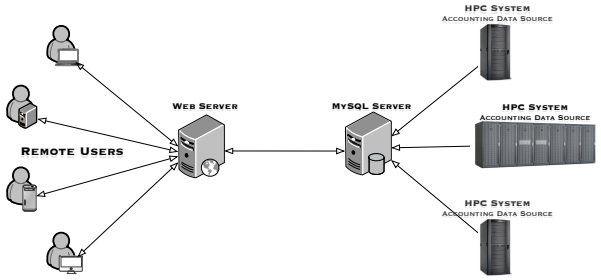

Figure 1: `pbsacct` Components

A number of third-party batch accounting and statistics packages have subsequently appeared, such as Gold [11] and Job Monarch [9]. However, few of them offer insight into what users are doing with the systems in terms of applications being run. Because of this, in 2005 staff at the Ohio Supercomputer Center (OSC) began developing a system called `pbsacct`; in 2008, the project relocated to the National Institute for Computational Sciences (NICS) at the University of Tennessee. Despite the name, `pbsacct` is intended primarily as an analysis tool, with allocation management and charging functions better handled by a separate system such as Gold.

The `pbsacct` system consists of a database, data ingestion tools, and command-line and web-based user interfaces, as shown in Figure 1. The data ingestion tools include both traditional accounting log processing as well as methods to capture users' job scripts as submitted. The command line and web interfaces provide a number of ways to access the resulting data set. The system has been designed to be able to handle multiple systems of various types (including clusters, MPPs, NUMA systems, and vector systems) that come and go over time.

```
CREATE TABLE Jobs (
  jobid        VARCHAR(32) PRIMARY KEY,
  system       VARCHAR(8),
  username     VARCHAR(9),
  groupname    VARCHAR(9),
  account      VARCHAR(32),
  submithost   VARCHAR(32),
  jobname      TINYTEXT,
  nproc        INT UNSIGNED DEFAULT 1,
  mppe         INT UNSIGNED,
  mppssp       INT UNSIGNED,
  nodes        TEXT,
  nodect       INT UNSIGNED DEFAULT 0,
  feature      TINYTEXT,
  gres         TINYTEXT,
  queue        TINYTEXT,
  qos          TINYTEXT,
  submit_ts    INT,
  submit_date  DATE,
  start_ts     INT,
  start_date   DATE,
  end_ts       INT,
  end_date     DATE,
  cput_req     TIME DEFAULT '00:00:00',
  cput         TIME DEFAULT '00:00:00',
  walltime_req TIME DEFAULT '00:00:00',
  walltime     TIME DEFAULT '00:00:00',
  mem_req      TINYTEXT,
  mem_kb       INT UNSIGNED,
  vmem_req     TINYTEXT,
  vmem_kb      INT UNSIGNED,
  software     TINYTEXT,
  hostlist     TEXT,
  exit_status  INT,
  script       MEDIUMTEXT,
  contact      TINYTEXT
);
```

Figure 2: `pbsacct` Jobs Table Schema

2. DATABASE STRUCTURE

The `pbsacct` system stores its information in a MySQL database, also named `pbsacct`. This database consists of two tables, `Jobs` and `Config`. As the name implies, the `Jobs` table stores job information for one or more systems. The current schema for the `Jobs` table is shown in Figure 2.

Most of the columns in the `Jobs` correspond to fields in the PBS accounting logs, with the exceptions of `system` (a mnemonic identifier for the system in case the host running `pbs_server` changes), `script` (for the job script, if available) and `contact` (a site-specific field implemented for OSC). The `Jobs` table is indexed on the `system`, `username`, `groupname`, `account`, `queue`, `submit_date`, `start_date`, and `end_date` columns to speed up query performance.

The `Config` table is used for tracking core-count changes in compute systems over time. This is used primarily by the web interface when calculating utilization. The current schema for the `Config` table is shown in Figure 3.

```
CREATE TABLE Config (
  row_number   SERIAL PRIMARY KEY,
  system       VARCHAR(8),
  nproc        INT UNSIGNED DEFAULT 0,
  start        DATE DEFAULT NULL,
  end          DATE DEFAULT NULL
);
```

Figure 3: `pbsacct` Config Table Schema

3. DATA INGESTION

3.1 Accounting Log Ingestion

Ingestion of accounting log data into `pbsacct` is done by a Perl script called `job-db-update`. This script parses the accounting records found in the `$PBS_HOME/server_priv/accounting` log directory on the host running the `pbs_server` daemon. There are two types of records processed: "S" records, emitted at the starts of jobs; and "E" records, emitted at the ends of jobs. For each record found, the script will parse the record, determine if the jobid already exists in the `Jobs` table or not, and then, depending on the previous existence of the job in the database, either update or insert the corresponding record in the `Jobs` table with the fields from the accounting log record.

The `job-db-update` is typically run at regular intervals, such as hourly or nightly, using the standard `cron` service. Example `cron` job entries can be found in the source code distribution.

3.2 Job Script Capture

Capturing and storing job scripts in `pbsacct` is handled by a daemon that watches the `$PBS_HOME/server_priv/jobs` spool directory where job scripts are stored by the `pbs_server` daemon. On Linux systems, this daemon can be either the now-deprecated `dnotify` or a Perl program called `jobscript_watcher` that uses `inotify_wait` internally. In either case, when a job script file appears in the spool directory, the watcher daemon will launch a script called `spool-jobscripts` that copies the script to a temporary location and then forks a second program called `jobscript-to-db` that inserts the copy into the `pbsacct` database. This arrangement is somewhat complex, but it was selected to be able to handle high throughput situations where the job scripts might only be present on the `pbs_server` host for a matter of seconds.

Like most modern batch systems, the PBS family supports the concept of an interactive job, which is a job that is driven by an interactive shell rather than a shell script. Unfortunately, none of the PBS family record the sequence of commands run by an interactive job, so there is no equivalent to job script to store in the database. As a result, interactive jobs appear in `pbsacct` as jobs with NULL job scripts.

4. USER INTERFACES

4.1 Command Line Interface

The main command line interface for `pbsacct` is a Perl script called `js`, which allows the user to look up job scripts by jobid. The current version of this script will allow the user to look up the job script of any job owned by any user, so it

is strongly encouraged that it be installed with permissions such that it can only be run by admin and support staff.

Work is in progress on another Perl script called `jobinfo`, which would allow full access to the various accounting details of their jobs, as users have often expressed interest in the ability to review job information without the intervention of support staff. However, addressing the issue of how to prevent a user from looking at the jobs of another, completely unrelated user has slowed development.

4.2 Web Interface

Most of the user interface work in `pbsacct` has gone into the web interface. This interface is written in PHP using the PEAR DB and Excel writer modules [5, 7], a PHP class for generating OpenOffice spreadsheets [2], and the jQuery Javascript library [8]. The web interface includes reports for individual jobs, job statistics broken out in a wide variety of ways (e.g. by core count, by job class/queue, by user, by account, etc.), software usage statistics, and numerous others. These reports all support querying jobs on one or more systems over arbitrary date ranges.

The `pbsacct` web interface is designed to be extensible and adaptable to sites' needs. The overall look of the interface is controlled by a master page layout file (`page-layout.php`) and a default CSS file (`default.css`). The majority of the query, metric generation, and output logic is in functions in `metrics.php`. However, site-specific logic for the calculation of CPU hours and charges, sorting criteria, statistical bucketing, and heuristics for identifying software usage from patterns in job scripts are in functions in `site-specific.php`. (In particular, the application identification heuristics can be rather complex and site specific; examples can be found at `http://svn.nics.tennessee.edu/repos/pbstools/trunk/web/site-specific.php` in the `software_match_list` function.) Additionally, reports may implement their own queries and output methods, such as those found in `joblist.php` and `usage-summary.php`.

Software usage reports are involving multiple packages are typcially done as `UNION SELECT` queries using the site-specific application identification heuristics mentioned above. These are full-text searches that cannot be easily indexed a priori, as the heuristics tend to evolve over time to exclude false positives. As a result, the software usage reports tend to be much more time consuming than the other reports available.

5. EXAMPLE DEPLOYMENTS

5.1 Ohio Supercomputer Center

The `pbsacct` deployment at OSC has been in service since the creation of the software in August 2005. It includes job records from a total of ten different HPC systems, some of which were in service before the `pbsacct` software was created. At the time of this writing (March 2014), the database contains 14,865,384 job records, 13,410,843 of which include job scripts. The current database size is approximately 30.0 GB.

The OSC `pbsacct` instance consists of a web server running the web application, a database server running MySQL, and data sources on the batch nodes of each currently operating HPC system. The database server is shared with a number of other services. The web application is accessed solely through HTTPS; authentication is done via HTTP basic authentication against an LDAP database, with authorization by membership in a specific LDAP group.

5.2 National Institute for Computational Sciences

The `pbsacct` deployment at NICS has been in service since September 2008 and includes job records from each of the eleven HPC systems that NICS has fielded. At the time of this writing, the database contains 5,452,959 job records, 5,043,031 of which include job scripts. The current database size is approximately 13.1 GB, with a growth rate of approximately 600 MB/month.

The NICS `pbsacct` instance consists of a web server running the web application, a database server running MySQL, and data sources on the batch nodes of each currently operating HPC system. The database server is shared with a number of other services, including the RT ticket system and multiple wikis. The web application is accessed solely through HTTPS; authentication is done via NICS' RSA SecurId one-time password infrastructure, and authorization by membership in a specific LDAP group.

6. EXAMPLE WORKLOAD ANALYSES

The following workload analyses was done using existing functionality in `pbsacct`.

6.1 NICS Kraken

The Kraken system at NICS was a 100-cabinet Cray XT5 system with 9,408 dual-Opteron compute nodes (12 cores per node) and 96 service nodes. It was operated by NICS in production from February 4, 2009 to April 30, 2014. Allocations on the system were primarily made through the NSF's Teragrid and XSEDE projects' allocation processes.

Kraken's resource management and scheduling software consisted of TORQUE, Moab, and Cray's Application Level Placement Scheduler (ALPS) software, with Moab communicating with both TORQUE and ALPS through Moab's native resource manager interface. In TORQUE, the default queue was a routing queue called `batch` that routed jobs based on core count to a set of execution queues called `small`, `medium`, `large`, `capability`, and `dedicated`. For a time, there was a `longsmall` queue for small, long running jobs, but that was eventually disabled. There was also an `hpss` queue for scheduled data transfers to the NICS HPSS enclave. Limits for these queues can be found in Table 1.

Between February 4, 2009 and April 30, 2014, the Kraken XT5 system ran 4.14 million jobs consuming over 4.08 billion core-hours on behalf of 2,657 users in 1,119 project allocations, resulting in an average utilization of 85.6% (without compensating for downtime). Of these, 3.84 million of the jobs, 3.85 billion of the core hours, 2,252 of the users, and 793 of the project allocations came through the Teragrid and XSEDE allocation processes, with the rest coming from director-discretionary allocations. Breakdowns on a per-queue basis can be found in Table 2.

Queue	Min Core Count	Max Core Count	Max Wallclock Time
small	0	512	24:00:00
longsmall	0	256	60:00:00
medium	513	8,192	24:00:00
large	8,193	49,536	24:00:00
capability	49,537	98,352	48:00:00
dedicated	98,353	112,896	48:00:00
hpss	0	0	24:00:00

Table 1: Kraken Queue Limits

Queue	Jobs	Core Hours	Users	Proj
small	3,576,368	768,687,441	2,602	1,090
longsmall	3,570	2,782,681	169	122
medium	488,006	2,003,837,680	1,447	718
large	27,908	983,795,230	521	301
capability	2,807	306,724,698	117	73
dedicated	338	11,765,421	17	7
hpss	36,462	53,285	184	123
TOTAL	4,136,759	4,077,647,799	2,657	1,119

Table 2: Kraken Workload Breakdown By Queue

Jobs submitted by projects with negative balances were tagged in the submit filter with a negative balance quality of service (QOS) marker called `negbal` that decreased their priority to the point where they would run only when nothing else could. Jobs from allocations with negative balances made up 7.1% of the workload in terms of job count and 9.3% of the workload in terms of core hours consumed. The negative balance QOS had a sigificant impact on jobs' turnaround time; whereas jobs from projects with positive balances waited an average of 5.76 hours to run, jobs from projects with negative allocations waited an average of 17.7 hours. While other QOS markers were used on the system, they were used so infrequently that their effects were statistically insignificant, as seen in Table 3.

Almost 400 different applications were identified in Kraken's workload, ranging from those run hundreds of thousands of times by hundreds of users to those run a handful of times by one or two users. The top ten applications on Kraken in terms of job count and core hours consumed are shown in Tables 4 and 5, respectively.

QOS	Jobs	Core Hours	Mean Queue Time
Default	3,841,103	3,691,660,723	05:45:21
negbal	293,095	379,947,239	17:44:43
All others	2,561	6,039,837	N/A

Table 3: Kraken Workload Breakdown By QOS

App	Jobs	Core Hours	Users	Proj
arps	639,698	71,385,589	53	11
enkf	524,417	27,639,349	27	6
namd	347,535	421,255,609	358	164
calc1	197,542	3,925,921	9	6
wrf	152,544	7,133,848	74	38
vasp	148,188	94,872,455	147	85
lammps	137,048	94,398,554	187	127
myq	130,782	1,086,752	4	4
gromacs	115,589	89,794,782	159	105
amber	103,710	110,938,365	208	120

Table 4: Kraken Top 10 Applications By Job Count

App	Jobs	Core Hours	Users	Proj
namd	347,535	421,255,609	358	164
chroma	38,872	178,790,933	17	10
res	58,630	161,570,056	268	190
milc	22,079	146,442,361	37	21
gadget	6,572	131,818,157	29	21
cam	66,267	124,427,700	88	68
enzo	15,077	112,704,917	54	37
amber	103,710	110,938,365	208	120
vasp	148,686	94,872,455	147	85
lammps	137,048	94,398,554	187	127

Table 5: Kraken Top 10 Applications By Core Hours

6.2 OSC Oakley

The Oakley cluster at OSC is a HP cluster based on Intel Xeon processors. The system has a total of 693 compute nodes; of those, 692 compute nodes have 12 cores each, and 64 of those have dual NVIDIA M2070 GPUs. Additionally, 1 node has 32 cores and 1 TB of memory. The system has been in production since March 19, 2012. Allocations for the system are made to Ohio's academic user community and industrial users.

The resource management and scheduling for Oakley are provided by TORQUE and Moab respectively. A default queue named `batch` routes jobs to a set of execution queues named `serial`, `parallel`, `longserial`, `longparallel`, `dedicated`, and `hugemem`. Limits for the different queues can be found in Table 6.

Queue	Min Core Count	Max Core Count	Max Wallclock Time
serial	1	12	168:00:00
parallel	13	2,040	96:00:00
longserial	1	12	336:00:00
longparallel	13	2,040	250:00:00
dedicated	2,041	8,336	48:00:00
hugemem	32	32	48:00:00

Table 6: Oakley Queue Limits

Queue	Jobs	Core Hours	Users	Proj
serial	1,799,890	32,938,880	1,088	387
parallel	324,848	77,614,464	595	256
longserial	36	58,456	5	5
longparallel	158	1,574,567	5	3
hugemem	299	54,466	28	23
TOTAL	2,125,231	112,240,833	1,147	403

Table 7: Oakley Workload Breakdown By Queue

Queue	Jobs	Mean Queue Time	Mean Job Wall Time
serial	1,799,890	2:29:18	2:57:05
parallel	324,848	3:47:41	3:02:06
longserial	36	18:53:48	135:18:56
longparallel	158	6:09:23	6:04:38
hugemem	299	2:30:52	5:41:44

Table 8: Oakley Queue Wait Time

Between March 19, 2012 and March 14, 2014, the Oakley system ran 2.1 million jobs consuming over 112 million core-hours on behalf of 1,147 users in 403 projects, resulting in an average utilization of 77.6%. Breakdowns of utilization on a per-queue basis can be found in Table 7.

In contrast to Kraken, the QOS for projects with negative allocations on Oakley are applied through Moab. While OSC has a similar `negbal` QOS, jobs that have this QOS are not identified by `pbsacct` because the QOS does not appear in the PBS accounting logs. Average queue wait times, and job run time for the different Oakley queues can be found in Table 8. The average queue wait time for the `longserial` queue is high due to the relatively small number of jobs in this class, and the very long queue time of a small number of these jobs.

The top ten applications by core hours consumed on the Oakley cluster are shown in Table 9.

App	Jobs	Core Hours	Users	Projects
vasp	109,302	16,777,905	45	13
wrf	58,606	8,327,638	14	9
qchem	347,729	8,008,378	59	13
OpenFOAM	9,838	5,654,152	48	26
gaussian	163,173	4,782,065	110	51
gromacs	147,754	3,954,489	57	18
lammps	19,990	3,782,372	26	17
matlab	98,785	3,722,044	89	63
amber	264,191	2,902,836	33	20
namd	2,357	1,495,582	9	9

Table 9: Oakley Top 10 Applications By Core Hours

7. CONCLUSIONS

`pbsacct` is a feature rich and easily extensible system for workload analysis on PBS-based HPC systems. It scales to millions of job records across tens of systems with relatively modest resource requirements, with system sizes ranging from small clusters to petaflop scale systems. It is currently a bit labor intensive to deploy; however, most of that labor is in implementing job script capture functionality, which is also where much of its power lies.

8. FUTURE WORK

While `pbsacct` is more or less functionally complete, there are a number of areas where it can be improved. The most significant area for improvement from a deployment perspective is in packaging. Currently, deploying an instance of `pbsacct` or adding a new system to an existing instance requires a fair amount of hand configuration, particularly with regard to script capture. However, efforts to simplify this by distributing the software in native packaging formats such as RPMs have recently begun. Any person who is interested in helping with these efforts should contact the authors.

Another area where `pbsacct` could be improved is in its database back end. MySQL has a number of limitations that are not present in other open source databases such as PostgreSQL. For instance, MySQL does not support a statistical median operation, which would be useful for the sorts of distributions typically present in batch accounting data. Conversely, some of the queries used by `pbsacct` make use of MySQL idioms and functions not supported by other open source databases. The developers have discussed porting `pbsacct` to PostgreSQL or Oracle, but the level of effort required is daunting. Again, potential collaborators in these efforts are encouraged to contact the authors.

When the `pbsacct` MySQL DB contains large numbers of jobs, full text searches of job scripts can take considerable time. Furthermore, the flexibility of search patterns are limited. It would be productive to investigate alternative schemes for indexing the contents of the `script` field to allow for higher performance text searches of job scripts, searches for ad-hoc patterns, and the use of regular expressions in searches. The Apache Lucene project's Solr search server [6] or similar tools bear investigation. However, external indexes would add complexity to queries of the data, need to be frequently updated, and relies on an a set of services not typically available at HPC centers.

Finally, the ability to interface `pbsacct` to non-PBS systems such as Grid Engine or SLURM has come up from time to time. As with porting to another database engine, the level of effort required for this is daunting, but interested parties should contact the authors.

9. ACKNOWLEDGMENTS

This material is based in part upon work supported by the National Science Foundation under Grant numbers 0711134, 0933959, 1041709, and 1041710 and the University of Tennessee through the use of the Kraken computing resource at the National Institute for Computational Sciences (`http://www.nics.tennessee.edu/`). Additionally, this material was also supported by resources at the Ohio Supercomputer Center (`http://www.osc.edu/`). Any opinions, find-

ings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, the University of Tennessee, or the Ohio Supercomputer Center.

The authors would like to acknowledge the assistance of NICS and OSC staff in the preparation of this document, in particular Vince Betro, Stephen McNally, Rick Mohr, and Benny Sparks from NICS.

10. REFERENCES

[1] PBS Professional: Job scheduling and commercial-grade HPC workload management. http://www.pbsworks.com/Product.aspx?id=1.

[2] PHP classes – OpenOffice spreadsheet generation: Generate of OpenOffice spreadsheet documents. http://www.phpclasses.org/package/2858-PHP-Generate-of-OpenOffice-spreadsheet-documents.html.

[3] TORQUE resource manager. http://www.adaptivecomputing.com/products/open-source/torque/.

[4] Draft standard for information technology – portable operating system interface (POSIX®) draft technical standard: Base specifications, issue 7. 2008.

[5] PEAR – database abstraction layer, 2011. http://pear.php.net/package/DB.

[6] Apache Lucene. 2012. http://lucene.apache.org/.

[7] PEAR – package for generating Excel spreadsheets, 2012. http://pear.php.net/package/Spreadsheet_Excel_Writer.

[8] jQuery, 2014. http://jquery.com/.

[9] Ramon Bastiaans. Job Monarch. https://oss.trac.surfsara.nl/jobmonarch.

[10] Robert L Henderson. Job scheduling under the Portable Batch System. In *Job scheduling strategies for parallel processing*, pages 279–294. Springer, 1995.

[11] Scott Jackson. The Gold accounting and allocation manager, 2004. http://www.emsl.pnl.gov/docs/mscf/gold.

[12] OpenPBS Team. A batching queuing system. http://www.openpbs.org/.