Job-Based Accounting for UNICOS/mp

Jim Glidewell, Boeing Information Technology

ABSTRACT: *With the absence of CSA (Cray System Accounting) from UNICOS/mp, the ability of sites to provide job-based accounting is limited. We have developed tools for utilizing UNICOS process accounting to provide both end-of-job and end-of-day reporting based on jobid. We will provide a look at our strategies for doing so, and a detailed look at the contents of the UNICOS/mp process record and some suggestions for how sites might make use of this data.*

Introduction

Our organization provides HPC (High Performance Computing) services to a wide variety of customers within the Boeing Company. Using organizations are charged back for use of our services, which requires that we collect and report accounting data to the Finance organization for internal billing purposes. Over the years, we have used a variety of tools and techniques for gathering and reporting this data. This paper will detail how we created a local job-based accounting system for UNICOS/mp.

Why Accounting?

The most obvious reason for gathering accounting related information is for billing purposes, but even organizations that do not do charge-backs may find that taking advantage of the data provided by the system accounting processes is useful. Machine accounting data can also be used for trend analysis, both of overall demand and of detailed application usage. This can lead to improved load projections, and better hardware upgrade and acquisition planning.

Accounting data can be used in conjunction with other security tools and procedures, both to monitor for unusual events as they occur and to do detailed post-mortem analysis in the case of a suspected incident.

Using accounting data to supplement the standard system-wide performance tools can sometimes isolate troublesome application behavior, and help detect bottlenecks.

Users and developers can make use of end-of-job accounting reports to tune application parameters to maximize performance. At our site, we have ensured that job cost reflects the actual cost of delivering that particular service, so that our users can compare job costs on different HPC platforms and, by choosing the lowest cost option which meets their turnaround needs, they are in turn choosing the platform which offers the best price-performance for their particular application and datasets.

Copyright © 2006 The Boeing Company. All rights reserved.

Accounting Resources

There are a number of accounting resources available on UNICOS/mp. The primary resource is the set of process accounting files, which consist of a series of records, one for each completed process. The two most critical additions for our site, above the standard System V process accounting records are 1) the concept of a "job" or "session", with the job-id included in the process accounting record, and 2) support for multiple projects for a given user, and reporting of the project-id in the accounting record as well. These fields are included in the expanded standard process accounting record. In addition to the process records, the process accounting file also includes *application* records, which record additional information about resources and placement for applications only.

PBS Professional, which is the batch subsystem in use at this site, provides its own accounting log, which may be sufficient in itself for sites that choose to bill entirely based on batch usage. In our case, it provides supplemental information that we use in addition to process accounting data.

Finally, for the purposes of job accounting, it is essential to know when a job has actually completed. Previous Cray systems included a special "end-of-job" record in the process accounting stream to denote a job's termination. In the absence of such a record, we chose to examine the UNICOS/mp job table to determine whether a job still existed.

We based our job-based accounting system on the above three resources: the process accounting records, the PBS Pro accounting file, and job-status from the UNICOS job table.

Goals

Our goals in developing a job-based accounting methodology were fairly simple. We needed a job-based accounting system that was accurate and repeatable, that would provide our users the sort of visibility that they had had on previous HPC systems, and would provide the necessary records for the downstream billing processes. We wanted a system that was reliable, consumed minimal system resources, and would be easily recoverable in the event of system interruption or accounting process failure.

Initial Implementation

When the Cray X1 first arrived on our site, we had an immediate requirement to track usage for billing purposes. At that time, there was no concept of "job" or "session" in UNICOS/mp. It did, however, have process record support for project accounting. We created a C-based program, which reads the binary process accounting records, totals the resource counters into structures based on UID and ACID (the integer ID associated with a project), and generates a report to be used for downstream billing. We ran this program once a day, on the previous day's process accounting files. This resulted in a downstream accounting feed that contained one record for each active (user, project) pair, containing

Copyright © 2006 The Boeing Company. All rights reserved.

the total of all resources associated with processes that completed during that twenty-four hour period.

While this version met the minimal requirements for charge-back billing, it did not offer the user any visibility of the cost of an individual job, either in the job log or when viewing the accounting reports.

Job-based Accounting

With the addition of the job container structure in UNICOS, and the addition of a job/session ID in the process accounting records, we had the raw data needed to build a more robust and user-friendly accounting system. We used our existing accounting program as a starting point, and modified the sorting of processes to accumulate resources based on user, project, *and* job_id. This provided a basic job accounting report for downstream billing, but there were additional issues that needed to be addressed.

First, we only wanted to send accounting to downstream processes for completed jobs. To check the status of the job, we use the "ps –jleaf" command to report on all processes and their respective job_ids. From this output, we internally built a list of currently running jobs. Any job_id not in this list is assumed to be complete.

Active jobs need to have their data preserved for subsequent accounting runs, and this leads to one of the Achilles' heels of job accounting – the recycle file. In previous systems, every process record associated with an incomplete job was copied to the recycle file, which often grew without bounds as certain jobs (such as the console session, and various daemon-related jobs) never terminated. In our case, the recycle file is almost trivial in size, as it consists of a single record per (user, project, job_id) tuple.

Finally, there is job related data stored in the PBS accounting log. Fortunately, the job_id is recorded in this file, which allows easy correlation between the process-accounting-based records and the PBS accounting logs. For simplicity, we process the PBS accounting log with a Perl script that extracts the interesting data (including the PBS job ID, and the name assigned to the job by the user) and generates a tab-delimited file for easy processing by the main end-of-day program.

After all data from all sources is read in, the records marked as "job complete" are written to the downstream feed file, while the remaining records are written to the recycle file. PBS entries not associated with completed jobs are written out in the same tab-delimited format as when they were read in. To insure that jobs don't recycle indefinitely (which could occur if a job_id gets reused after a reboot), jobs are automatically treated as complete after a fixed period of ten days.

User Job Reporting

The other major function of job accounting is to provide a report at the end of the job, summarizing the resources used and the job cost. We named this utility "jobcost". Again,

Copyright © 2006 The Boeing Company. All rights reserved.

a large percentage of the code was reused, as we decided to use the same internal C structures for resource accumulation, therefore the code for reading the process records and accumulating process data to those records was essentially identical. At the start of the jobcost command, a query is made to the OS to determine the job_id. Next the recycle file is inspected to check whether any records matching that job_id are found – if so, they are imported. Next, all the process records from all the current pacct files are read and those that match the job_id are added to the in-memory structures. In the case of a batch job, PBS information is queried direct from pbs_server using the PBS Pro API, rather than extracting the data from the PBS accounting log. As before, when all data has been read, a pass over the in-memory structures is made, and output is generated. But in this case, the output is in a more human-friendly form.

Besides the basic command, the only options are those that allow the user to get a report on a subset of a job. By default, jobcost reports all usage since the beginning of the job. Adding the "-i" option makes the jobcost *incremental* – only processes since the last jobcost are reported upon. "jobcost –t" allows a user to set the start point for the next "jobcost –i" without generating any output or processing any process accounting data. Both these options rely on a hidden file created in the user's $TMPDIR directory which contains a timestamp of when jobcost was last executed.

One concern with this method of end-of-job reporting is that *every* user job that uses it reads up *all* of the process accounting data since the last end-of-day run. This means that a trivial "jobcost" near the end of the accounting day will read and examine all process records for all users for the entire day. CSA (Cray System Accounting) avoided this issue by creating a separate process accounting file containing only records associated with the job in $TMPDIR. We did not have that luxury. But as it turns out, the processing required is still small enough that its cost is a small fraction of one cent on our system. And for user benchmarking activities, they can use the "jobcost –t" option to essentially eliminate the overhead from their jobcost reports.

Experiences

Overall, users have been pleased with the return of the end-of-job cost reporting that they had come to know and expect. The benefits of having a more complete job history is only beginning to become clear to many of the data center folks, but already we have used our local archives of the downstream job accounting feed to resolve a few minor system anomalies and user errors. But job-based billing and reporting seems so natural that it is quickly becomes taken for granted.

The system has been reliable and robust. In the case of a "missed day", the calling scripts allow a single integer parameter of "days ago" that allows an admin to specify which day he wants to run. Running days out-of-order is designed to work without impact to total downstream charging, though it may result in splitting a single job or session into two or more parts. Maintaining the system requires very little system administrator intervention.

Copyright © 2006 The Boeing Company. All rights reserved.

Other Accounting Options

Writing C-based programs to read the process accounting files is not particularly difficult, but it is time consuming, and other sites might want to look at other techniques for gathering this data. The pacct (process accounting) files on the Cray X1 actually contain a mix of two different types of records: one type contains data associated with each process, while a second record type contains data pertaining to application use. Commands (non-applications) have only a process record, while applications have *both* a process record and an application record. One way to get easy access to the data in both of these record types is via the Cray-supplied "acctcom" command.

Process Accounting and "acctcom"

The acctcom command has effectively two major modes – one that reports on processes (the default), and a second mode, invoked by the "acctcom –A" parameter that reports solely on application records. The following table summarizes the fields available in the process records, the meaning of that field, and the acctcom parameter that is used to display that value on the acctcom output:

Accounting field	"acctcom" option	Definition
ah_flag	-f	Process flags
acc_uid	*	User
acc_status	-f	Exit status
acc_btime	*	Start time
acc_etime	*	Elapsed time
acc_comm	*	Command name
ac_gid		Group
ac_acid	-p	Account ID (project)
ac_tty	*	Terminal
ac_pid	-p	Process ID
ac_apid	-p	Application ID
ac_sid	-p	Session ID
ac_utime	-t	User CPU time
ac_stime	-t	System CPU time
ac_mem	-k	Memory integral
ac_himem	*	Memory hi-water
ac_io	-i	Characters transferred

Table 1. Process accounting fields and acctcom options

An asterisk denotes that the field is provided by the default acctcom output.

Copyright © 2006 The Boeing Company. All rights reserved.

As can be seen by the table above, every process accounting field except ac_gid (group) is available with the right set of option flags ("acctcom –fikpt", for example). In some cases, there is more than a single display option (such as average memory vs. memory integral), but the above table reflects the most direct relationship between process account file fields and acctcom output.

Application Accounting and "acctcom –A"

"acctcom –A" reports only on application records. The following table illustrates the relationship between application accounting record fields and "acctcom -A" output:

Accounting field	"acctcom –A" option	Definition
acc_uid	*	User
acc_status		Exit status
acc_btime	*	Start time
acc_etime	*	Elapsed time
acc_comm	*	Command name
acap_apid	*	Application ID
acap_sid		Session ID
acap_ltime	*	Launch time
acap_flags	*	Application flags
acap_width	*	Application width
acap_depth	*	Application depth
acap_ctime	*	Connect time
acap_acid		Account ID
acap_txt_pgsiz	-L	Text page size
acap_oth_pgsiz	-L	Non-text page size
acap_place_cnt	-L	Placement count

Table 2. Application record fields and acctcom options

In general, most fields are included by default. The "-L" option adds information about memory usage and placement count. No other options are allowed. For the application records, however, there are a few fields that are unavailable via acctcom: exit status, session (job) ID, and account ID. All of these fields are available in the corresponding process records, and the acctcom output for those records, but the lack of a direct output option using "acctcom –A" means that a site that wants to bill or track application usage cannot use the "acctcom -A" output alone for such processing if either job-based or project based reporting is desired.

Sites that need job or project based accounting, wish to use "acctcom" as their tool for data extraction, and require data from the "acctcom –A" output may wish to use the

Copyright © 2006 The Boeing Company. All rights reserved.

application ID (apid) which is common to both reports to tie the two report types together.

Conclusion

We have created a local job-based accounting system, which generates output which is useful to users, administrators, and downstream processes. Cray's inclusion of both a job and a project ID in each process record made this possible, and allowed us to tie together data from a variety of sources including process accounting records, the PBS accounting logs, and the UNICOS job table.

Building programs that process pact data is not difficult, and offers maximum completeness and flexibility in data collection. For sites that wish to avoid such coding, the acctcom command offers the vast majority of the process and application record data in an easy-to-process text form.

Job-based accounting offers a number of benefits, including relatively modest data size, natural fit for user perceptions, and far more options for workload analysis and forecasting. It has been well received by our users, and we expect to make increasing use of this data in the future.

About the Author

Jim Glidewell has been a member of Boeing's HPC group for over twenty years, working on a variety of systems from Cray, SGI, CDC, and others. He is currently serving as the CUG X1/E Systems SIG Chair. He can be reached at The Boeing Company, P.O. Box 3707 MC 7J-04, Seattle WA 98124-2207; E-mail: james.glidewell@boeing.com

Copyright © 2006 The Boeing Company. All rights reserved.