Understanding Aprun Use Patterns

Hwa-Chun Wendy Lin, *NERSC*

ABSTRACT: *On the Cray XT, aprun is the command to launch an application to a set of compute nodes reserved through the Application Level Placement Scheduler (ALPS). At the National Energy Research Scientific Computing Center (NERSC), interactive aprun is disabled. That is, invocations of aprun have to go through the batch system. Batch scripts can and often do contain several apruns which either use subsets of the reserved nodes in parallel, or use all reserved nodes in consecutive apruns. In order to better understand how NERSC users run on the XT, it is necessary to associate aprun information with jobs. It is surprisingly more challenging than it sounds. In this paper, we describe those challenges and how we solved them to produce daily per-job reports for completed apruns. We also describe additional uses of the data, e.g. adjusting charging policy accordingly or associating node failures with jobs/users, and plans for enhancements.*

KEYWORDS: XT4, CNL, ALPS, apsched, aprun, Torque, Moab

1. Introduction

NERSC is a Department of Energy (DOE) computing facility providing resources to researchers in a wide range of disciplines, including climate/weather studies, high energy physics, chemistry, materials sciences, Engineering, and computer sciences. Researchers have to have their proposals approved by DOE to receive allocations to compute at NERSC. How to make best use of limited funds is always a challenge.

The newest addition to the NERSC computing facilities is nicknamed Franklin, which went into production in October 2007. It then went through a series of upgrades and settled in as a 9,532 node quad-core Cray XT4 system. With so many nodes available and per DOE, NERSC has adopted a policy giving discounts to large jobs, to encourage users to scale up their programs. But then the question becomes whether users are gaming the system to take advantage of this policy. That is, are users asking for a large number of nodes at the job level, but using subsets of them in parallel at the application level?

The initial attempt to answer this question was just to find large jobs that launched applications in parallel, and generate a daily report. But the more we dug into it, the more we thought the information we were able to gather about applications could be useful in many other ways. As a result, we abandoned the original approach, and decided to split the tasks into two separate parts. The first script would collect all information for all invocations of applications and group it based on job ID and save it to a file. This is the data gathering part. A second script would post-process the resulting data file and generate a report. This is the data consumption part.

2. The Players

2.1 ALPS (Application Level Placement Scheduler)

According to the ALPS introduction man page, ALPS (Application Level Placement Scheduler) is the Cray supported mechanism for placing and launching applications on the Cray XT compute nodes. This support is limited to the CNL (Compute Node Linux) environment, where nodes run a stripped down Linux kernel and very few daemons.

There is not much documentation about ALPS other than man pages. Michael Karo of Cray gave presentations at the last three CUG conferences. His slides for the 2006 presentation provide a good source for understanding ALPS. ALPS is designed to work with multiple workload managers, i.e., batch systems. Among the core services listed, resource reservation management is the service of interest in managing batch jobs. The daemon that supports this functionality is *apsched*, which runs on Franklin's SDB (System Database) node to coordinate all reservation requests. The user command in ALPS that places and launches applications to the compute nodes is called *aprun*, which is functionally equivalent to the generic application launcher *mpiexec* or the AIX specific application launcher *poe*.

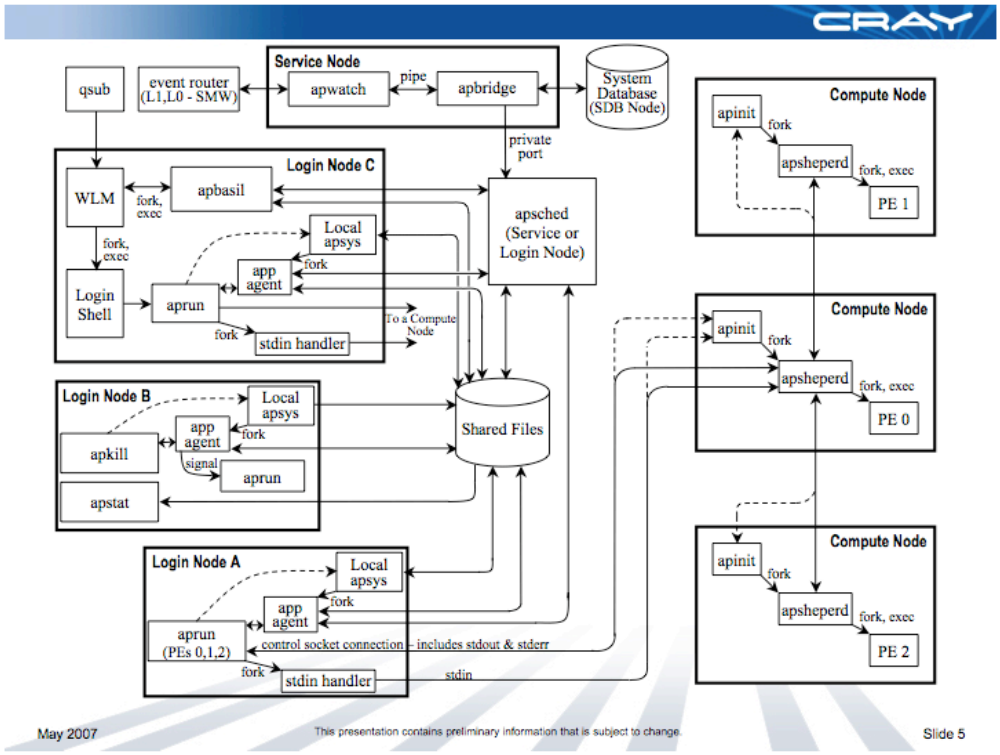
ALPS also supports interactive application executions. However the focus of this project is on applications running under a batch system, because NERSC policy disables interactive use of the compute nodes, for more effective resource control.

2.2 Torque/Moab

Cray officially supports three major batch systems: Altair's PBSPro, Cluster Resources Incorporated's Torque, and Platform Computing's LSF (Load Sharing Facility). The batch system choice at NERSC is Torque, with Moab as the scheduler. Torque is an OpenPBS derivative, with the typical server and execution hosts (aka MOMs, where user job scripts run). However, on the XT, compute nodes are execution hosts but they don't run the MOM daemon. Instead, there are service nodes set aside to be MOM nodes, on which applications are launched via the *aprun* command. This is different from the one-execution-host/one-MOM setup for systems running full-blown OS on compute nodes. The Moab job scheduler on the XT manages the scheduling policy enforcement as it normally does, but it doesn't manage the compute nodes. The responsibility of allocating and de-allocating the computing resource falls on ALPS.

2.3 Job Life Cycle

Karo, in his CUG 2006 and 2007 presentations, provides a comprehensive picture to show the life cycle of a job.



The WLM (Work Load Manager) in the diagram refers to all three pieces in a batch system: PBS server, MOM, and the scheduler. When a job is submitted to the batch system, it stays in an input queue until the WLM receives a go-ahead from ALPS, indicating a reservation consisting of nodes suitable to run the job is available to the job. Only then the MOM launches the desired shell and starts the job script. When a script runs, commands other than the ones preceded with the *aprun* keyword get executed on the MOM node. An application is launched to all or a subset of the reserved nodes via the *aprun* command

A reservation can be in "FILED", "CONFIRMED", "CLAIMED" state, or some combination of the three. According to Karo in his CUG 2006 presentation, FILED means a reservation request is filed (registered with ALPS), CONFIRMED means resources are locked, and CLAIMED means resources are in use. The FILED state did not appear to be relevant to this project, and was not studied. This was verified with Karo. Apparently, the design of ALPS was based on a two-phase commit. The job scheduler, e.g. Moab on Franklin, gets a reservation for a job from ALPS, which puts the reservation in the FILED state. Moab then instructs the PBS server to start the job script on a MOM node. As part of starting up the script, MOM confirms the reservation, which puts the reservation in the CONFIRMED state. The current Torque/Moab implementation, though, does the filing and confirming back to back, and the FILED state can be ignored.

Once a reservation enters the CONFIRMED state, it can be used by an application (or applications for MPMD (Multiple Programs Multiple Data) jobs. When applications are launched via *aprun*, the reservation enters the CLAIMED state. At the conclusion of an *aprun* run, the reservation goes back to CONFIRMED until it's cancelled by WLM. We use the timestamps for confirming and cancelling a reservation as markers for the job starts/ends, and those of claiming and releasing the reservation as the application starts/ends. Multiple apruns can share a reservation simultaneously or consecutively. We will illustrate difference uses of *aprun* in the Examples section.

3. Data Gathering

3.1. Reservation information sources

The primary source for reservation information is the *apsched* log. On Franklin, the log lives in *sdb:/var/log/alps* and is switched each day around midnight. The records we are interested in are: "Confirmed", "Bound", "Placed", "Released", and "Canceled". The Confirmed, Bound, and Canceled records are job specific. The information from the Bound record is merged with the Confirmed record to mark the starting of a reservation/job script. The Confirmed and Canceled, which marks the end of a job script are linked by an ALPS's "apid". The Placed and Released records are application specific, they are also linked by an apid, and they mark the start and end of an application. All five records have the same "resId" (Reservation ID) for the same job, while the last four also have the same "pagg" (session ID). There are as many pairs of Placed and Released records as there are apruns. Below is an example of these records.

```
20:26:11: Confirmed apid 437380 resId 1749 \
    pagg 0 nids: 6454-6513
20:26:11: Bound Batch System ID 5828057 \
    pagg 61513 to resId 1749
20:26:40: Placed apid 437384 resId 1749 \
    pagg 61513 uid 32407 MPMD cmd cpl \
    nids: 6454-6513
01:28:30: Released apid 437384 resId 1749 \
    pagg 61513 claim
01:30:04: Canceled apid 437380 resId 1749 \
    pagg 61513
```

How to associate applications with a job then? In XT 2.1, the Bound record has the job ID, as shown above, thus the *apsched* log has all information needed to re-construct the application information for jobs. But in XT 2.0, job ID is not stored in the *apsched* log. It has to come from the set_job record in the *syslog* in *sdb:/syslog/var/log/messages*. Here is a set_job record:

```
Apr 11 20:26:11 nid00576 pbs_mom: set_job,\
    /opt/moab/default/tools/partition.create.xt4.pl \
    –confirm -p 1749 -j 5828057.nid00003 -a 61513
```

where the "-j" specifies the job ID while the "-p" the reservation ID and "-a" the session ID. The reservation ID and session ID are the key to linking job pieces, regardless of where the job ID comes from.

Notice this very same set_job record is also present in the *syslog* in XT 2.1. It's true that it's easier to process just one file, the *apsched* log. In fact, after the XT 2.1 upgrade, we removed *syslog* scanning from the picture. However we plan to revert back to using both files. We will discuss why in the Future Enhancements section.

Scenarios for different *aprun* use patterns are provided in the Examples section, which should help better understand how the source records are tied together.

3.2 The aprundat script

The data gathering program is called *aprundat*, and is a Perl script. It runs every day early in the morning to post-process the previous day's log files. As mentioned previously, the apsched log is switched daily; however, the *syslog* is not. The *syslog* is switched manually every so often when the system boots. A complete set of records that allow us to group apruns by job consist of five records from the *apsched* log and the set_job record from the *syslog*, as described earlier. We collectively call them information source records, or just source records.

The source records might be, and often are, spread out across days. Relevant information about specific apruns is accumulated as the daily processing progresses. At the end of a run, *aprun* records for completed jobs, one per *aprun*, are written to a data file, which is used by the report generating facilities (to be discussed in the Data Consumption section). For jobs that are still in progress, information from various source records that have been processed are written to an overflow file. This overflow file is read in the following day to re-build the past events.

Due to system crashes and occasional ALPS mess-ups, some source records will never be able to bind together to describe a job. We discard old source records after two days (the walltime limit for Franklin jobs is 36 hours). The discarded records are saved to a file with _expired as the suffix. On occasions, we get complete *aprun* information records, but can't locate job IDs. We save these to a file with _incomplete as the suffix. The primary reason for having _incomplete records stems from matching difficulty with two logs—information goes out of sync more easily when it's available in two different places. After we stopped using the *syslog* for job ID, we see fewer _incomplete records. The _expired and _incomplete files are usually results of system troubles. System administrators might be able to make use of these files.

In the directory to store *aprundat* run outputs, four new files come to existence each day. The names of the files for the same day all start with the same date stamp, followed by a file type identifier. For example, for May 9, 2008, the file names are 20080509_aprundat, 20080509_overflow, 20080509_expired, and 20080509_incomplete.

Here are a few _aprundat records that are used in the Data Consumption section to demonstrate report generation. The fields are ";" separated, and give, in the order they appear, job ID, nodes assigned, start time, end time, user login, command name, and nodes used.

```
504757;3445-3572;\
    1210346785;1210375032;\
    userzzz;lesmpi.a;3445-3572
504758;12677-12693,12698-12735,12800-12829;\
    1210339493;1210339500;\
    useryyy;ddt-debugger;\
    12677-12693,12698-12735,12800-12829
504759;12704-12735,12800-12831;\
    1210339704;1210339708;\
    userxxx;RadHyd3D;\
    12704-12735,12800-12831
504759;12704-12735,12800-12831;\
    1210339892;1210339936;\
    userxxx;RadHyd3D;\
    12704-12735,12800-12831
504759;12704-12735,12800-12831;\
    1210339963;1210339988;\
    userxxx;RadHyd3D;\
    12704-12735,12800-12831
504759;12704-12735,12800-12831;\
    1210341258;1210341393;\
    userxxx;RadHyd3D_check;\
    12704-12735,12800-12831
```

4. Data Consumption

4.1. The aprunrpt script

The *aprunrpt* script can be run anytime against a <date>_aprundat file to generate a report. With the 20080509_aprundat as input, this report generator, by default, produces output as shown below. With the "-m" option, it will only report jobs with multiple apruns. That means, with the example, the entries for jobs 504757 or 504758 will be suppressed. The script doesn't report nodes list because there is no good way to display it cleanly. However the script is written in Perl, it's easy to add more options to get, for example, job/user specific entries, or all jobs completed during a specific time period.

```
504757   128   128 \
```

```
    08/05/09 08:26:25 08/05/09 16:17:12 \
    userzzz  lesmpi.a
504758   85    85 \
    08/05/09 06:24:53 08/05/09 06:25:00 \
    useryyy ddt-debugger
504759   64    64 \
    08/05/09 06:28:24 08/05/09 06:28:28 \
    userxxx RadHyd3D
                64 \
    08/05/09 06:31:32 08/05/09 06:32:16 \
    userxxx RadHyd3D
                64 \
    08/05/09 06:32:43 08/05/09 06:33:08 \
    userxxx RadHyd3D
                64 \
    08/05/09 06:54:18 08/05/09 06:56:33 \
    userxxx RadHyd3D_check
```

4.2 The Franklin completed job status page

On its website, NERSC provides a completed job status page, one for each system. The data generated by the *aprundat* script are used to provide nodes list and to populate the aprun section on the Franklin page. Below is a web display copy for job 504759.

Job details

Step ID	504759.nid00003	**Job Name**	STDIN		
Owner	userxxx	**Account**		**Status**	265
Execution queue	interactive	**Submit class**	interactive	**Job type**	
Nodes	64	**Wall secs**	1,928	**Wall hrs**	0.54
Available cores per node	2	**Requested secs**	1,800	**Requested hrs**	0.50
MPP secs	1,604,096	**MPP hrs**	445.58	**Raw Secs**	246,784
Submit	May-09-08 06:26:41	**Start**	May-09-08 06:26:54	**Wait**	00:00:13
Completion	May-09-08 06:59:02	**systime**	0	**usrtime**	0
Nodelist	12704-12735, 12800-12831				

*Indicates dispatch time

List of aprun commands executed in this job

Number of *aprun* commands: 4

Command	Nodes Used	Run Time (secs)	Start	Complete	Nodelist
RadHyd3D	64	4	May-09-08 06:28:24	May-09-08 06:28:28	12704-12735, 12800-12831
RadHyd3D	64	44	May-09-08 06:31:32	May-09-08 06:32:16	12704-12735, 12800-12831
RadHyd3D	64	25	May-09-08 06:32:43	May-09-08 06:33:08	12704-12735, 12800-12831
RadHyd3D_check	64	135	May-09-08 06:54:18	May-09-08 06:56:33	12704-12735, 12800-12831

Examples

We ran the ping_pong program under batch with four slightly different *aprun* requests to show how reservations were made and claimed for each scenario. For each example, the first block is the batch job script; the second, source records from the *apsched* log; the third, the set_job record from the *syslog*; the last, the resulting aprun information entry in the <date>_aprundat file.

5.1 Job with a single application instance

```
#PBS -q debug
#PBS -l mppwidth=64
cd $PBS_O_WORKDIR
```

```
aprun -n 64 ./ping_pong


17:37:35: Confirmed apid 411088 resId 349 \
    pagg 0 nids: 12622-12627,12632-12641
17:37:36: Bound Batch System ID 5820466 \
    pagg 73126 to resId 349
17:37:37: Placed apid 411089 resId 349 \
    pagg 73126 uid 40877 cmd ping_pong \
    nids: 12622-12627,12632-12641
17:37:57: Released apid 411089 resId 349 \
    pagg 73126 claim
17:38:15: Canceled apid 411088 resId 349 \
    pagg 73126


Apr  7 17:37:36 nid00576 pbs_mom: set_job, \
    /opt/moab/default/tools/partition.create.xt4.pl \
    --confirm -p 349 -j 5820466.nid00003 -a 73126


5820466;12622-12627,12632-12641; \
    1239151057;1239151077; \
    hclin;ping_pong; \
    12622-12627,12632-12641
```

5.2 Job with multiple application instances launched in succession

```
#PBS -q debug
#PBS -l mppwidth=64
cd $PBS_O_WORKDIR
aprun -n 64 ./ping_pong
aprun -n 32 ./ping_pong
aprun -n 48 ./ping_pong


17:42:12: Confirmed apid 411111 resId 356 \
    pagg 0 nids: 12800-12815
17:42:13: Bound Batch System ID 5820474 \
    pagg 852 to resId 356
17:42:13: Placed apid 411112 resId 356 \
    pagg 852 uid 40877 cmd ping_pong \
    nids: 12800-12815
17:42:34: Released apid 411112 resId 356 \
    pagg 852 claim
17:42:34: Placed apid 411113 resId 356 \
    pagg 852 uid 40877 cmd ping_pong \
    nids: 12800-12807
17:42:45: Released apid 411113 resId 356 \
    pagg 852 claim
17:42:45: Placed apid 411115 resId 356 \
    pagg 852 uid 40877 cmd ping_pong \
    nids: 12800-12811
17:43:00: Released apid 411115 resId 356 \
    pagg 852 claim
17:43:11: Canceled apid 411111 resId 356 \
    pagg 852


Apr 7 17:42:13 nid04096 pbs_mom: set_job, \
    /opt/moab/default/tools/partition.create.xt4.pl \
    --confirm -p 356 -j 5820474.nid00003 -a 852


5820474;12800-12815; \
    1239151333;1239151354; \
    hclin;ping_pong; \
    12800-12815
5820474;12800-12815; \
    1239151354;1239151365; \
    hclin;ping_pong; \
    12800-12807
5820474;12800-12815; \
    1239151365;1239151380; \
    hclin;ping_pong; \
    12800-12811
```

5.3Job with multiple application instances launched in parallel

```
#PBS -q debug
#PBS -l mppwidth=64
cd $PBS_O_WORKDIR
aprun -n 8 ./ping_pong &
aprun -n 32 ./ping_pong &
aprun -n 16 ./ping_pong
wait


17:43:14: Confirmed apid 411117 resId 357 \
    pagg 0 nids: 12800-12815
17:43:14: Bound Batch System ID 5820475 \
    pagg 1162 to resId 357
17:43:15: Placed apid 411119 resId 357 \
    pagg 1162 uid 40877 cmd ping_pong \
    nids: 12800-12803
17:43:15: Placed apid 411120 resId 357 \
    pagg 1162 uid 40877 cmd ping_pong \
    nids: 12804-12805
17:43:15: Placed apid 411121 resId 357 \
    pagg 1162 uid 40877 cmd ping_pong \
    nids: 12806-12813
17:43:18: Released apid 411120 resId 357 \
    pagg 1162 claim
```

```
17:43:20: Released apid 411119 resId 357 \
    pagg 1162 claim
17:43:25: Released apid 411121 resId 357 \
    pagg 1162 claim
17:44:14: Canceled apid 411117 resId 357 \
    pagg 1162


Apr 7 17:43:14 nid04096 pbs_mom: set_job, \
    /opt/moab/default/tools/partition.create.xt4.pl \
    --confirm -p 357 -j 5820475.nid00003 -a 1162

5820475;12800-12815; \
    1239151395;1239151398; \
    hclin;ping_pong; \
    12804-12805
820475;12800-12815; \
    1239151395;1239151400; \
    hclin;ping_pong; \
    12800-12803
5820475;12800-12815; \
    1239151395;1239151405; \
    hclin;ping_pong; \
    12806-12813
```

5.4 MPMD job

```
#PBS -q debug
#PBS -l mppwidth=64
cd $PBS_O_WORKDIR
aprun -n 8 ./ping_pong : -n 32 ./ping_pong : \
    -n 16 ./ping_pong


17:54:29: Confirmed apid 411173 resId 370 \
    pagg 0 nids: 5787-5789,6586-6598
17:54:30: Bound Batch System ID 5820529 \
    pagg 4171 to resId 370
17:54:31: Placed apid 411174 resId 370 \
    pagg 4171 uid 40877 MPMD cmd ping_pong \
    nids: 5787-5789,6586-6596
17:54:51: Released apid 411174 resId 370 \
    pagg 4171 claim
17:55:10: Canceled apid 411173 resId 370 \
    pagg 4171


Apr  7 17:54:30 nid04096 pbs_mom: set_job, \
    /opt/moab/default/tools/partition.create.xt4.pl
    --confirm -p 370 -j 5820529.nid00003 -a 4171


5820529;5787-5789,6586-6598; \
    1239152071;1239152091; \
    hclin;ping_pong; \
    5787-5789,6586-6596
```

5.5 The aprunrpt display

At NERSC, the batch queue structure includes a routing queue called *regular*, that dispatches jobs to either the *reg_small* or *reg_big* execution queue, depending on the job size, i.e., the *mppwidth* specification. The *reg_big* jobs, those asking for 1024 or more nodes, get discounts. Users can't submit jobs directly to *reg_big*, the only way to select the queue destination is through the resource request.

Below is the *aprunrpt* display for the four example jobs. Notice that job 5820475 is a job that ran *aprun* in parallel, but it's not gaming the system because of the job size—it did not push the size to go from small to large to get a discount

Job ID	Reserved	Used	Start	End	User	Command
5820466	16	16	09/04/07 17:37:37	09/04/07 17:37:57	hclin	ping_pong
5820474	16	16	09/04/07 17:42:13	09/04/07 17:42:34	hclin	ping_pong
		8	09/04/07 17:42:34	09/04/07 17:42:45	hclin	ping_pong
		12	09/04/07 17:42:45	09/04/07 17:43:00	hclin	ping_pong
5820475	16	2	09/04/07 17:43:15	09/04/07 17:43:18	hclin	ping_pong
		4	09/04/07 17:43:15	09/04/07 17:43:20	hclin	ping_pong
		8	09/04/07 17:43:15	09/04/07 17:43:25	hclin	ping_pong
5820529	16	14	09/04/07 17:54:31	09/04/07 17:54:51	hclin	ping_pong

6. Challenges

6.1 Constructing timestamps

The *apsched* log is switched daily, therefore the developer probably felt it's not necessary to record the date in the log. Fortunately the month and day can be derived from the file name, which is something like apached0407. Unfortunately, the year is not available anywhere, thus the current year is assumed. This is usually okay except when processing the December 31 log on January 1 the following year, or processing the previous year's *apsched* log in general. A "-y" command line option is provided to the *aprundat* program to allow the manual specification of a year to work around the issue.

6.2 Finding job ID for apruns in syslog

When the *aprundat* script was being developed, the Bound record in the *apsched* log did not have the corresponding job ID. We had to get it from the set_job record in the *syslog*. With the script being Perl, it's instinctive to use a hash to store the reservation ID and job ID pairs. As mentioned previously, the *syslog* is switched at boot time whenever needed, which means each *syslog* contains multiple days of data, and as a result, the daily run of the *aprundat* script builds a hash for reservations occurred over multiple days. We quickly found then, there were only a limited number of reservation IDs available, when a reservation ID ws recycled during the lifetime of a *syslog*, we wiped out information for jobs using the same reservation ID previously.

Looking closely at the source records, we discovered another common piece among the *apsched* records and the *syslog* set_job record. We name this piece session ID, which is unique during the life time of a job. On the *apsched* records, it's the number after the "pagg" keyword, on the set_job record, it's the specification to the "-a" parameter. Using the combination of reservation ID and session ID as the key for the job ID hash appeared to have solved the issue of keeping all job IDs until one day, when we had to process a *syslog* that spanned over a longer than normal period of time. In order to handle this atypical but not impossible situation, we added a third dimension: time. The timestamp of a set_job record is not used as part of the key, but is kept along with the job ID in a chain identified by the key. The time information is only fetched to break a tie.

7. Future Enhancements in Data Gathering

As mentioned in the Data Gathering section, in XT 2.1, job ID is available in the *apsched* log, there is no need to process the *syslog* just to get job ID. But there is some other information about applications that we'd like to collect. The eventual goal is to provide a one-stop shop for everything we ever want to know about NERSC user applications. One thing requested by NERSC consultants is complete *aprun* command line options specified for each *aprun*. This turns out to be trivial, we just need to go back to still processing two files. In the *syslog*, there is one entry for each *aprun* where the command line is displayed as specified. Below is such an entry for an MPMD program.

```
Apr 11 20:26:40 nid00576 aprun[63195]:\
    apid=437384, Starting, user=32407,\
    cmd_line="aprun -n 32 -d 1 cpl : \
                    -n 32 –d 1 csim : \
                    -n 16 -d 1 clm : \
                    -n 96 -d 1 pop : \
                    -n 64 -d 1 cam",\
    num_nodes=60, node_list=6454-6513
```

Another piece of information we'd like to have for applications is their exit status. From examining the console log kept on the SMW (System Management Workstation), we identify at least two types of difficulties applications ran into: OOM (out of memory) and segmentation fault. Here are examples of such entries:

```
[2009-04-14 13:22:15][c5-4c0s2n0] \
    Out of memory: Killed process 30142 (jfdtd3d). \
    apid: 453270
[2009-04-14 13:16:42][c10-3c0s2n3] \
    nwchem[30104]: \
    segfault at 00000003204b1dd0 \
    rip 0000000000ff5e35 rsp 00007fffffffb930 \
    error 4
```

The challenge in including information supplied in the console log is not having to parse yet another date format, or converting the node specification from a physical node location to nid number, it is how to get the information out. The SMW is behind a firewall on a private network. Currently we extract OOM entries from the console log daily and e-mail them to interested parties. We are looking for a way to automatically save the e-mail

body to a file to be processed along with the *apsched* log and *syslog*. We'll look into using *procmail* for this.

8. Conclusion

Understanding *aprun* use patterns of NERSC researchers was the motivation behind this project—we were to answer the question whether users ran multiple applications in parallel just to take advantage of the NERSC policy in favoring large jobs. We screened the daily reports from the *aprunrpt* script carefully, and concluded that, no, users did not game the system. We see apruns running in parallel, but users did not appear to do that just so their jobs would run in the large queue. Thus we did not do further systematic analysis, and also did not change our charging policy.

The daily reports from the *aprunrpt* script provide insight into how individual users/groups do their research. The very first report showed a 15-node, 24-hour job that launched *aprun* 41,007 times, with one node for each aprun most of the time; but once in a while, the *aprun* would use all 15 nodes. We got really concerned and checked with NERSC consultants to confirm that they knew about this research and why its jobs ran the way they ran. But then if the MOM node crashed due to high volume of apruns, we would know whom to blame! The reports also provide an easy way to quantify the use of software packages. NERSC pays for commercial software packages, such as DDT, Q-Chem, and Molpro, it needs to justify that the money is well spent. Even for non-commercial products, it takes manpower to support them, we need to justify the effort as well.

In addition to the intended use of the *aprun* data files, we've found that the data files compiled daily can be useful other ways. For example, we are currently trying to see whether there is correlation between UMEs (uncorrectable memory errors) and user applications, i.e., are some applications inclined to trigger memory chip errors? Because the *aprundat* data files have the start/end time and nodes list for each application, and there is a time and node associated with each UME incident, it should be trivial to match them up.

The decision to adopt a two-step approach is proven a wise one. We'll continue looking for ways to collect more information about applications and to build even more resourceful data files. We expect to find more uses for the data.

Acknowledgments

This work was supported by the Director, Office of Science, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC02-05CH11231.

This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy.

The author would like to thank Michael Karo of Cray for giving permission to use his slide and providing additional information.

References

Man pages: intro_alps(1), apsched(1), aprun (1)

Michael Karo: ALPS Application Level Placement Scheduler, CUG 2006 (slides)

Michael Karo: ALPS User Tutorial (Base Camp), CUG 2007 (slides)

Michael Karo: ALPS Tutorial " Ascent", CUG 2008 (slides)

About the Author

Wendy Lin is a Systems Engineer at NERSC and the Backup Analyst of Franklin. E-mail: hclin@lbl.gov.