Tracking Library and Application Usage: Recent Enhancements to the Automatic Library Tracking Database infrastructure at CSCS

CUG, Napa Valley, 9 May 2013

Tim Robinson, CSCS
Application-level accounting

• The good old days: Cray XT3 under Catamount
  – Cray comprehensive system accounting recorded “yod line”

• Deployment of the Cray XT5 and Compute Linux Environment
  – Job launch commands recorded in system logs but difficult to associate with specific batch jobs and/or users

• We cannot easily answer questions like
  – How often has application x been launched in the last month?
  – Is anyone using a legacy version of application x?
  – Which compilers are/aren’t being used to build applications?
  – How many applications make use of library y?
  – Which users are using legacy/buggy versions of library y?
Let’s have a user survey!
But... what applications are users *really* running on our systems?
The software available on the Cray systems is provided by modules...
Tracking module loads?

- Can only track software that is actually provided through a module
- Loading a module doesn’t mean the software is actually being used
- Not loading a module doesn’t mean the software is not being used
The Automatic Library Tracking Database

• Written by Fahey, Jones, and Hadri (Cray User Group meeting in 2010)

• ALTD records information every time an application is linked and every time the resulting executable is launched on the compute nodes

• This is done by intercepting the GNU linker and the aprun job launcher
  – ALTD records the entire link line so it can be used to determine ancillary information about the compilation, such as which compiler suite was used to build the application

• Everything is transparent to the user

• Extremely lightweight – little or no overhead

• Only tracks libraries that are actually used in the application
The Automatic Library Tracking Database

• Data is stored in three tables in an SQL database

• altd_<machine>_link_tags
  – An entry for every execution of the linker

• altd_<machine>_linkline
  – An entry for every unique link line

• altd_<machine>_jobs
  – An entry for every job launched with aprun command

• The ld wrapper
  – Generates assembly code and links it into application (ALTD header)
  – Determines link line with tracemap option to real linker
  – Updates link_tags and linkline tables

• The aprun wrapper
  – Performs an objdump on executable to retrieve the ALTD header: the build machine and tag id are checked to trace the executable back to its entry in the link_tags table
  – Updates the jobs table
  – Calls the real aprun
How does it work?: link lines

robinson@rosa101:~> ftn dgemm.f -o dgemm
robinson@rosa101:~> cc hello.c -o hello_cug_2013

<table>
<thead>
<tr>
<th>tag_id</th>
<th>linkline_id</th>
<th>username</th>
<th>exit_code</th>
<th>link_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>438581</td>
<td>82474</td>
<td>robinson</td>
<td>0</td>
<td>2013-05-05</td>
</tr>
<tr>
<td>438582</td>
<td>82475</td>
<td>robinson</td>
<td>0</td>
<td>2013-05-05</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>linking_inc</th>
<th>linkline</th>
</tr>
</thead>
<tbody>
<tr>
<td>82474</td>
<td>dgemm</td>
</tr>
<tr>
<td>82475</td>
<td>hello_cug_2013</td>
</tr>
</tbody>
</table>
How does it work?: application launch

robinson@rosa101:~> aprun -n 1 ./dgemm
robinson@rosa101:~> aprun -n 1 ./hello_cug_2013

<table>
<thead>
<tr>
<th>run_inc</th>
<th>tag_id</th>
<th>executable</th>
<th>username</th>
<th>run_date</th>
<th>job_launch_id</th>
<th>build_machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>2410158</td>
<td>438581</td>
<td>/users/robinson/dgemm</td>
<td>robinson</td>
<td>2013-05-05</td>
<td>834805</td>
<td>rosa</td>
</tr>
<tr>
<td>2410189</td>
<td>438582</td>
<td>/users/robinson/hello_cug_2013</td>
<td>robinson</td>
<td>2013-05-05</td>
<td>834805</td>
<td>rosa</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tag_id</th>
<th>linkline_id</th>
<th>username</th>
<th>exit_code</th>
<th>link_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>438581</td>
<td>82474</td>
<td>robinson</td>
<td>0</td>
<td>2013-05-05</td>
</tr>
<tr>
<td>438582</td>
<td>82475</td>
<td>robinson</td>
<td>0</td>
<td>2013-05-05</td>
</tr>
</tbody>
</table>
How to mine data: a hypothetic situation

A performance bug has been identified in Cray’s LibSci version 12.0.00 (GNU) !!!
How to mine data: a hypothetic situation

• **How can we determine which users might be affected?**
  – Start by checking which users have linked this library into their codes

```sql
mysql> select distinct username from altd_rosa_link_tags,altd_rosa_linkline where altd_rosa_link_tags.linkline_id=altd_rosa_linkline.linking_inc and exit_code=0 and linkline like '%libsci/12.0.00/gnu%';
+----------+
| username  |
+----------+
| tkachenn  |
| boswald   |
| subedi    |
| scman     |
| . . .     |
| liang     |
| **robinson** |
| yuding    |
| kraused   |
| pkiryl    |
| zilia     |
+----------+
60 rows in set (4.33 sec)
```
How to mine data: a hypothetic situation

- We could also check if the user “robinson” is actually running these application(s)

```
mysql> select altd_rosa_jobs.* from altd_rosa_link_tags,altd_rosa_linkline,altd_rosa_jobs
where altd_rosa_jobs.tag_id=altd_rosa_link_tags.tag_id and altd_rosa_link_tags.linkline_id=altd_rosa_linkline.linking_inc and exit_code=0 and linkline like '%libsci/12.0.00/gnu%' and altd_rosa_jobs.username="robinson";

+---------+--------+--------------------------------------------------+----------+------------+---------------+---------------+
| run_inc | tag_id | executable                                       | username | run_date   | job_launch_id | build_machine |
|---------+--------+--------------------------------------------------+----------+------------+---------------+---------------+
| 2279195 | 371698 | /scratch/rosa/robinson/LAPACK_scalling/a.out     | robinson | 2013-03-04 | 728048        | rosa          |
| 2279198 | 371698 | /scratch/rosa/robinson/LAPACK_scalling/a.out     | robinson | 2013-03-04 | 728048        | rosa          |
| 2279199 | 371698 | /scratch/rosa/robinson/LAPACK_scalling/a.out     | robinson | 2013-03-04 | 728048        | rosa          |
| 2279200 | 371698 | /scratch/rosa/robinson/LAPACK_scalling/a.out     | robinson | 2013-03-04 | 728048        | rosa          |
| 2279202 | 371700 | /scratch/rosa/robinson/LAPACK_scalling/a.out     | robinson | 2013-03-04 | 728048        | rosa          |
| 2279203 | 371700 | /scratch/rosa/robinson/LAPACK_scalling/a.out     | robinson | 2013-03-04 | 728048        | rosa          |
| 2279214 | 371709 | /scratch/rosa/robinson/LAPACK_scalling/a.out.gnu| robinson | 2013-03-04 | 728071        | rosa          |
| 2279215 | 371709 | /scratch/rosa/robinson/LAPACK_scalling/a.out.gnu| robinson | 2013-03-04 | 728071        | rosa          |
| 2279222 | 371709 | /scratch/rosa/robinson/LAPACK_scalling/a.out.gnu| robinson | 2013-03-04 | 728071        | rosa          |
| 2279282 | 371709 | /scratch/rosa/robinson/LAPACK_scalling/a.out.gnu| robinson | 2013-03-04 | 728071        | rosa          |
| 2279283 | 371709 | /scratch/rosa/robinson/LAPACK_scalling/a.out.gnu| robinson | 2013-03-04 | 728071        | rosa          |
| 2279284 | 371709 | /scratch/rosa/robinson/LAPACK_scalling/a.out.gnu| robinson | 2013-03-04 | 728071        | rosa          |
| 2279286 | 371709 | /scratch/rosa/robinson/LAPACK_scalling/a.out.gnu| robinson | 2013-03-04 | 728071        | rosa          |
| 2279301 | 371709 | /scratch/rosa/robinson/LAPACK_scalling/a.out.gnu| robinson | 2013-03-04 | 728071        | rosa          |
| 2410158 | 438583 | /users/robinson/dgemm                            | robinson | 2013-05-05 | 834805        | rosa          |
+---------+--------+--------------------------------------------------+----------+------------+---------------+---------------+
24 rows in set (0.65 sec)
Extending the framework

• The jobs table holds only the following information
  – tag_id, executable, username, run_date, job_launch_id, build_machine

• We would like to know more about *how* applications are being run
  – Processes, threads, processes per node, and so on.
  – We call this **application-level accounting**
  – A new table in ALTD database: altd_<machine>_accounting

• Accounting table contains
  – account_or_group
  – begin_time and end_time
  – linking
  – aprun_line
  – num_pes
  – depth_per_pe
  – used_cores
  – claimed_cores
  – num_nodes
  – pes_per_cu
  – exit_code
  – some_env_vars
  – app_name
  – notes
Systems with ALTD deployed at CSCS

• **Cray XE6 Rosa**
  - 1496 compute nodes with AMD Interlagos CPUs (50K cores, 400 Tflops)
  - Main production system

• **Cray XK7 Tödi**
  - 272 nodes with AMD Interlagos CPU and Nvidia K20X GPU (4K cores and 272 GPUs)
  - Research and development, and production for GPU-enabled code

• **Cray XC30 Daint**
  - 2256 nodes with Intel SandyBridge CPUs (40K phys. cores, 750K Tflops)
Case studies
Predicting the impact of change

Results from ALTD could guide procurements or give a prediction of how disruptive a given change would be to the user community.
Compiler usage in 2013: % users

- XE6
- XK7
- XC30

PGI
Intel
Cray
GNU
Chapel uptake?

Brad Chamberlain et al. look away now.

Mysql> select * from altd_rosa_linkline where linkline like '%/opt/chapel/%'
Empty set (1.27 sec)

Mysql> select * from altd_todi_linkline where linkline like '%/opt/chapel/%'
Empty set (1.07 sec)

Mysql> select * from altd_daint_linkline where linkline like '%/opt/chapel/%'
Empty set (0.12 sec)
User characteristics: developer vs “black-box”

Results can be used to guide support:
Should effort be put into managing large application portfolios? Or, should more focus be placed on optimizing users’ applications?
Mode of linking: number of jobs run

Only 20% of applications launched on XK7 are linked dynamically?

Potential misuse of a system?

Shell script
Dynamically linked
Statically linked
CUDA and/or OpenCL usage on XK7

- Never built CUDA or OpenCL code
- Built CUDA and/or OpenCL code
The ALTD **accounting table** provides further information about the application, and the way it is being run: the code is CP2K (cp2k.popt) and was compiled by the user; it is being run in pure MPI mode (no OpenMP) using 16 processes per node and hyperthreading turned off.
Summary

• We have extended the Automatic Library Tracking Database to record additional information about applications launched on Cray systems
  – number of processing elements and threads used
  – mode of linking
  – site-definable metadata like mappings between executable names and applications or application domains

• We’ve shown example scenarios where ALTD could assist application support specialists by alerting them to unusual usage patterns or potential misuse of resources

• Moving forward, there is a strong need for further development of ALTD to provide fully automated data mining, reporting and alerting.
  – Ideally, the tool should alert application specialists to situations such as the use of legacy or buggy libraries, or potential wastage of available compute resources.
Acknowledgements

Depth per process: applications launched