XALT: Understanding HPC Usage via Job Level Collection

Robert McLay, Mark Fahey, Reuben Budiardja, Sandra Sweat

The Texas Advanced Computing Center, Argonne National Labs, NICS

Apr. 5, 2016
XALT: What runs on the system

- A U.S. NSF Funded project: PI: Mark Fahey and Robert McLay
- A Census of what programs and libraries are run
- Running at TACC, NICS, U. Florida, KAUST, ...
- Integrates with TACC-Stats.
Design Goals

• Be extremely light-weight
• Provide provenance data: How?
• How many use a library or application?
• Collect Data into a Database for analysis.
Design: Linker

- The linker (ld) wrapper intercepts the user link line.
  - A shell script wrapper, ld which uses python scripts
  - Generate assembly code: key-value pairs
  - Capture tracemap output from ld
  - Transmit collected data in *.json format
Design: Launcher

- Program Launcher: mpirun, aprun, ibrun ...
  - A shell script wrapper is called which uses python scripts
  - Find Executable by parsing command
  - Collect executable info, shared libraries, env.
  - Transmit collected data in *.json format
- The future is now. This is no longer necessary!
Design: Transmission to DB

- File: collect nightly
- Syslog: Use Syslog filtering
- Direct to DB.
Lmod to XALT connection

- Lmod spider walks entire module tree.
- Can build A Reverse Map from paths to modules.
- Can map program & libraries to modules.
- `/opt/apps/i15/mv2_2_1/phdf5/1.8.14/lib/libhdf5.so.9 ⇒ phdf5/1.8.14(intel/15.02:mvapich2/2.1)`
- Also helps with function tracking.
Database Changes (I)

- Tables sizes in XALT:

<table>
<thead>
<tr>
<th>Table</th>
<th>Size in MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>join_run_env</td>
<td>199603.00</td>
</tr>
<tr>
<td>join_run_object</td>
<td>9388.00</td>
</tr>
<tr>
<td>join_link_object</td>
<td>5013.00</td>
</tr>
<tr>
<td>xalt_run</td>
<td>4613.00</td>
</tr>
<tr>
<td>xalt_object</td>
<td>4175.00</td>
</tr>
<tr>
<td>xalt_link</td>
<td>814.00</td>
</tr>
</tbody>
</table>

- `join_run_env` has 2.1 billion rows
Database Changes (II)

- Environment variables are important.
- But mainly for reproducing results
- Not SQL tests (mostly)
Database Changes (III): New Design

- Store complete env $\Rightarrow$ compressed json blob
- Filter Env’s with Accept Test followed by Reject Test
- Instead of 250 vars per job $\Rightarrow$ 20 to 30.
- The Filter is site controllable!
Database Changes (IV): New Design

- The “join” tables are large
- Partition “join” tables by dates or index
- Precompute views nightly.
Protecting XALT (I): UTF8 Characters

- Linux supports UTF8 Characters in file names, env. vars.
- Python supports UTF8 if you know what you are doing.
- Switch XALT to use prepared statements
- Where query="INSERT INTO table VALUE(?,?)"
- This prevent SQL injection: “johnny drop tables;”
- Also supports UTF8 characters.
Protecting XALT (II): Python to C++

• Difficult to protect python from users in every case
• Solution: LD_LIBRARY_PATH="@ld_lib_path@"
  PATH=/usr/bin:/bin C++-exec ...
• Everything that depends on PATH must be hard coded
• basename ⇒ /bin/basename
• Unique install for each operating system.
• Programs move around: basename
Using XALT Data

- Targetted Outreach: Who will be affected
- Largemem Queue Overuse
- XALT and TACC-Stats
Publishing XALT Data

- Student Sandra Sweat
- Sanitized Data
- Community Codes Reported: Vasp*, WRF*, OpenFOAM*, users names: U012354, Charge Accounts: A12345
- Unique mapping, Added Field of Science
Tracking Non-mpi jobs (1)

- Originally we tracked only MPI Jobs
- By hijacking mpirun etc.
- Now we can use ELF binary format to track jobs
ELF Binary Format Trick

void myinit(int argc, char **argv)
{
    /* ... */
}
void myfini()
{
    /* ... */
}

__attribute__((section(".init_array")))
typeof(myinit) *__init = myinit;
__attribute__((section(".fini_array")))
typeof(myfini) *__fini = myfini;
Using the ELF Binary Format Trick

- This C code is compiled and linked in through the hijacked linker
- It can also be used with LD_PRELOAD
- We are using both...
Downsides

- Currently, we only track task 0 jobs.
- MPMD programs will only record the Task 0 job.
- We also lose the ability to capture return exit status.
Challenges (I)

- With both LD_PRELOAD and init.o linked in. $\Rightarrow$ double records
- Do not want to track mv, cp, etc
- Only want to track some executables on compute nodes
- Do not want to get overwhelmed by the data.
Why do both?

- We want both linking in and LD_PRELOAD, Why?
  - Data on programs built before XALT
  - Data on GUI debugger, ...
  - User sets LD_PRELOAD
Avoid Double counting

- `.init_array` and `.fini_array` work like an onion.
- `.init_array`: a Stack: LIFO
- `.fini_array`: a Queue: LILO
- Preload, Built-in, program, Built-in, Preload
- Use an env. var. to prevent double counting
Other Safety Features

- XALT Tracking only told to
- Compute node only
- Filter based on Path
- Protection against closing stderr before fini.
- Site configurable!
Path Filtering

- Accept test, following an Ignore Test,
- Two files containing regex patterns, converted to code.
- Accept List Tests: Track `/usr/bin/ddt`, `/bin/tar`
- Ignore List Tests: `/usr/bin`, `/bin`, `/sbin`, ...
XALT Demo

- Show modules hierarchy
- `ml -raw show xalt`
- Show debugging output
- `type -a ld,mpirun`
- Build programs
- Run tests
- Run utf8 program
- Show database results
Conclusion

• Lmod:
  – Source: github.com/TACC/lmod.git, lmod.sf.net
  – Documentation: lmod.readthedocs.org

• XALT:
  – Source: github.com/Fahey-McLay/xalt.git, xalt.sf.net
  – Documentation: doc/*.pdf, xalt.readthedocs.org