Community Use of XALT in Its First Year in Production

HUST 2015
Austin, TX

Reuben D. Budiardja
National Institute for Computational Sciences
The University of Tennessee

with Mark Fahey (ANL), Robert McLay (TACC), Prasad Maddumage Don (FSU), Bilel Hadri (KAUST), Doug James (TACC)

https://github.com/Fahey-McLay/xalt
Talk Outline

• Introduction to XALT
  • Motivation
  • How It Works

Getting Data Out of XALT
  • Compilers, Libraries, Executables Usage Reports
  • Other Use Cases

• New Functionality
  • Function Tracking
  • GUI (Web)-Based Reports
    User Software Provenance
Introduction to XALT
Motivation

Most computing center needs to answer the questions:

• How many users and projects use a particular library or executable?

How many users use which compilers?

• Which center provided packages are used often? and which one are never used?

• Which users or applications still use old version of certain library, compiler, or executable?

Are there any widely used user-installed package that a center should provide instead?
XALT is a tool to collect accurate, detailed, and continuous job-level and link-time data, and store them in a database.
XALT is a tool to collect accurate, detailed, and continuous job-level and link-time data, and store them in a database.

**XALT collects information to answer questions on software usage**
Goals

• **Automatic, continuous** census of libraries and applications
• Collect job-level and link-time level data for subsequent analytics
  Must be transparent to user, avoid impacting the user experience
• Must work seamlessly on any system: workstation, cluster, high-end supercomputer
• Must be a lightweight solution
Approach: Link-time Level

Intercept linker at link-time:

- Wrap the (GNU) linker (ld) and parse the command line
  - Capture only the object files actually linked with the executable
- Stores the results using a chosen transmission style
- Insert an XALT’s ELF section header to the executable

SYSLOG

JSON files at ~/.xalt.d/

Direct DB

???

XALT Database

parser
Approach: Execution-time Level

Intercept job launcher to get execution environment:

• Wrap job-launcher (aprun, ibrun, mpirun, ... ) with a corresponding script

• Extract previously inserted XALT’s ELF header (if any)

• Extract environment variables
  • Job-specific environment (e.g. PBS_JOBID, etc)
  • Dynamics libraries loaded at runtime
  • Record job start and end time
Track shared libraries
Getting Data Out of XALT

Community Usage Reports
Compiler Usage

- XALT stores “link program”: the program that calls the linker
  - A proxy for the compiler \( \rightarrow \text{main()} \) compiler
  - Will miss mixed language compilation

- Can associate “compiler” with every linking event
SELECT link_program, count(*) as count
FROM xalt_link
WHERE build_syshost = [syshost]
GROUP BY link_program ORDER BY count desc
Compiler Usage Ratio per User

Is there a way to tell if someone used a compiler once (or a little), before giving up?
Compiler Usage: TACC, FSU, KAUST

**TACC**
- icc: 21.8%
- icpc: 13.6%
- gcc: 13.7%
- g++: 17.7%
- ifort: 4.5%
- c++: 30.7%
- cc: 11.1%
- gfortran: 16.8%
- clang++: 18%
- clang: 18%
- other: 45.8%

**FSU**
- icc: 21.8%
- icpc: 13.6%
- gcc: 13.7%
- g++: 17.7%
- ifort: 4.5%
- c++: 30.7%
- cc: 11.1%
- gfortran: 16.8%
- clang++: 18%
- clang: 18%
- other: 45.8%

**KAUST**
- g++: 45.8%
- gcc: 37.2%
- ifort: 6.5%
- ftn_driver: 11.9%
- icc: 11.1%
- icpc: 16.8%
- cc: 18%
- pgfortran: 18%
- pgcc: 18%
- other: 18%
Most Used Libraries

• What is “the most used”?  
  • By the number of linkings  
  • By the number of unique users  

• Use “module name” to identify library  
  • Multiple object files may be associated with a module  
  • Likely these libraries are provided via modulefile by vendor or center’s staff  
    Resistance to path changes as long as ReverseMap is maintained

• Script: contrib/library_usage.py
Most Used Libraries: Numerical

# Linkings scaled down by x100
Most Used Libraries: Prog. & I/O

# Linkings scaled down by x100
Top Executables

• Track only how much time spent by the parallel job
  • Not the entire job script
  • Can be correlated with other accounting to get the ratio of the parallel job over the entire job script
• Track the actual number of compute cores used in the parallel job
  • Done by parsing the argument given to parallel launcher
• Can show how the launched executable was built → provenance data
Top Executables

![Graph showing the number of jobs and core-hours for various executables. The graph includes labels for SGF, GENASIS*, NAMD*, VASP*, TTMMD*, NWChem*, KVC*, md.exe, pbhmd, trap.o, aims.150205.scal..., cm1.exe, LAMMPS*, even_a.out, gene_darter, exe, CP2K*, ctqmc.e, abinit, and CHIMERA*.]
Top Executables: KAUST
Software Pruning

• How or when to remove software (version) on the system?
  • Because newer versions are available
  • Because of lack of use
  • To free up disk space and/or support time

• XALT can provide data-driven decision
  Show when the last time each library was used (linked against), and by whom (user)
  • Allow for targeted notification to users (to upgrade version, migrate to different library, etc)
New Functionality
Function Tracking

- Recently added functionality (version >= 0.7.0)
- Only track functions (a.k.a. subroutines / symbol names) that are resolved by external libraries
  - Does not track user defined functions
  - Does not track auxiliary functions in libraries
- Currently does not track which library resolves the functions
  Although this can be done heuristically after the fact
Function Tracking (2)

• Collect the list of library / object files whose functions we are interested in tracking
  • Generated by traversing the directories of library files in modulefiles (typically used as argument to “-L” linker flag) already in ReverseMap file
### Example Query

**Most called functions**

```
SELECT trim(function_name), count(*)
FROM xalt_link xl,
     join_link_function lf,
     xalt_function xf
WHERE build_syshost = 'darter'
  AND xl.link_id = lf.link_id
  AND lf.func_id = xf.func_id
GROUP BY function_name
ORDER BY cnt DESC
LIMIT 100
```
### Example Query

- **BLAS’ mat-mul use**

```sql
SELECT distinct(SUBSTRING_INDEX(exec_path,'/','-1')) as exe,
    build_user
FROM xalt_link xl,
    join_link_function lf,
    xalt_function xf
WHERE build_syshost = 'darter'
    AND xl.link_id = lf.link_id
    AND lf.func_id = xf.func_id
    AND xf.function_name
    LIKE '%gemm%'
GROUP BY exe
```
XALT Portal

A web interface to more easily get XALT data:

• Used by center’s staff to easily get high level library, compiler, and executable usage

• From any of those “entry points”, can drill-down to users associated with library/compiler/executable, and their jobs and job environment

• Can search who uses a particular library or executable

  Allow targeted notification in case of buggy library, retired versions, etc
Usage

Results presented here are taken from XALT database at National Institute for Computational Sciences at Oak Ridge National Laboratory. XALT went live on November 2014 the data presented here is real-time.

Select syshost and date range for your queries.

Select Syshost: darter

**Select September 1, 2015 - October 31, 2015**

Submit

**Modules Usage**

![Bar chart showing modules usage]

**Further Details**

**List of Version(s) (for given Module)**

<table>
<thead>
<tr>
<th>Modules</th>
<th>Versions</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 alps</td>
<td></td>
<td>3777</td>
</tr>
<tr>
<td>2 boost</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>3 cce</td>
<td></td>
<td>922</td>
</tr>
<tr>
<td>4 cray-hdf5</td>
<td></td>
<td>308</td>
</tr>
<tr>
<td>5 cray-hdf5-parallel</td>
<td></td>
<td>470</td>
</tr>
<tr>
<td>6 cray-libsci</td>
<td></td>
<td>1435</td>
</tr>
<tr>
<td>7 cray-mpich</td>
<td></td>
<td>2790</td>
</tr>
</tbody>
</table>

Click Modules to get Version details [Count = Number of time Object was Linked]

**List of User(s) (for given module-version)**

<table>
<thead>
<tr>
<th>Users</th>
<th>Earliest_LinkDate</th>
<th>Latest_LinkDate</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2015-09-17 17:09:16</td>
<td>2015-10-14 17:41:43</td>
<td>262</td>
</tr>
<tr>
<td>3</td>
<td>2015-09-17 00:29:32</td>
<td>2015-10-28 11:57:39</td>
<td>158</td>
</tr>
<tr>
<td>7</td>
<td>2015-09-22 14:01:31</td>
<td>2015-10-08 18:45:03</td>
<td>94</td>
</tr>
</tbody>
</table>

[Count = Number of time Object was Linked for given Module-Version]

**List of Executable(s) (for given user-module-version)**

<table>
<thead>
<tr>
<th>Executable Name</th>
<th>LinkDate_Oldest</th>
<th>LinkDate_Latest</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 a.out</td>
<td>2015-09-18 14:40:49</td>
<td>2015-10-14 17:23:34</td>
<td>22</td>
</tr>
<tr>
<td>5 dist.x</td>
<td>2015-10-06 15:33:18</td>
<td>2015-10-07 13:23:56</td>
<td>4</td>
</tr>
</tbody>
</table>
XALT Portal for User Provenance

• “How did I build my exec x months ago?”
  “What was the default MPI / compiler / library at the time?”

• Allow user to know the history and origin, i.e. “provenance”, of the software they run

Different type of users:
  • Run their own executable
    Run executable provided by the Center
    Run executable built by another user
  • Helps with reproducibility of research conducted with such software
User Provenance

- Select an executable
- List of user’s executable
- List of jobs with executable
- List of object files / library linked to exec
- Environment variables for selected job
- Runtime loaded object files

Select a job

The UNIVERSITY of TENNESSEE
FORGING THE FUTURE OF COMPUTING

34
User Software Provenance  Get run/link details for given user

Select Syshost*  darter  October 20, 2015 - November 18, 2015  

Enter User ID * 

Submit

Further Details

List of Executable(s)

<table>
<thead>
<tr>
<th>Executable</th>
<th>No_Jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ChartAlterationInteriorProper_Form_Test_Darter_Cray</td>
<td>12</td>
</tr>
<tr>
<td>2 ChartAlteration_EP_Form_Test_Darter_GNU</td>
<td>12</td>
</tr>
<tr>
<td>3 ChartAlteration_EP_Form_Test_Darter_Cray</td>
<td>11</td>
</tr>
<tr>
<td>4 ChartAlteration_IP_Form_Test_Darter_Cray</td>
<td>9</td>
</tr>
<tr>
<td>5 Mesh_Form_Test_Darter_Cray</td>
<td>8</td>
</tr>
<tr>
<td>6 ChartAlterationInteriorProper_Form_Test_Darter_GNU</td>
<td>8</td>
</tr>
<tr>
<td>7 ChartStream_Form_Test_Darter_GNU</td>
<td>6</td>
</tr>
</tbody>
</table>

[Count = Number of times executable was run]
<table>
<thead>
<tr>
<th>Executable Path</th>
<th>Build Date</th>
<th>Link Program</th>
<th>Exit Code</th>
<th>Build User</th>
<th>Job Run[T/F]</th>
<th>Unique Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build Date: 2015-10-29 15:28:44, Build User: 0, Job Run[T/F]: True, Unique Id:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3ed67761-59f4-48df-8c7d-b08182999d3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build Date: 2015-10-29 15:23:02, Build User: 0, Job Run[T/F]: True, Unique Id:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2f568f4b-30-577-f1cd-9e63-4e880107be6b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build Date: 2015-10-29 15:20:19, Build User: 0, Job Run[T/F]: True, Unique Id:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c6817965-78f6-46b-be05-d010b96089f4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build Date: 2015-10-29 15:17:18, Build User: 0, Job Run[T/F]: True, Unique Id:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0c747f4a-6-ad2a-492e-8db-3c180099e4b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build Date: 2015-10-29 15:14:59, Build User: 0, Job Run[T/F]: True, Unique Id:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ca2efc1a-4450-4a99-aa7d-20e03019006</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build Date: 2015-10-29 15:01:57, Build User: 0, Job Run[T/F]: True, Unique Id:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5f23a9-e9-4dd-44b-9e17-cae9f6f930f</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Objects Linked (to the given Executable):

<table>
<thead>
<tr>
<th>Object Path</th>
<th>Module Name</th>
<th>Object Date</th>
<th>Object Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>/opt/cray/compv0.1.2.0502.55507.3.2.wav64/compn.a</td>
<td>xcompv0.1.2.0502.55507.3.2.wav</td>
<td>2015-04-19 15:54:05</td>
<td>a</td>
</tr>
<tr>
<td>/opt/cray/wkm_detect/1.0.1.0502.55541.1.1.wav/compn.detect.a</td>
<td>wkm_detect/1.0.1.0502.55541.1.1.wav</td>
<td>2015-04-19 15:54:05</td>
<td>a</td>
</tr>
<tr>
<td>/opt/cray/compv5.0.1.0502.6085.4.24.wav/compn.a</td>
<td>ucompv5.0.1.0502.6085.4.24.wav</td>
<td>2015-04-19 15:54:05</td>
<td>a</td>
</tr>
<tr>
<td>/opt/cray/ludgeg/2.3.2.1.0502.9275.1.12.wav/lbducg.a</td>
<td>ucompv2.3.2.1.0502.9275.1.12.wav</td>
<td>2015-04-19 15:54:05</td>
<td>a</td>
</tr>
<tr>
<td>/crave/crave/1.0.0.0-2.0502.53711.3.127.wav/compn.64/crave.a</td>
<td>ucompv1.0.0.0-2.0502.53711.3.127.wav</td>
<td>2015-04-19 15:54:05</td>
<td>a</td>
</tr>
<tr>
<td>/opt/cray/compv0.1.2.0502.55507.3.2.wav64/compn.a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RunId</td>
<td>JobId</td>
<td>Run Date</td>
<td>nCmTncM-nT</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>----------------</td>
<td>------------</td>
</tr>
<tr>
<td>535974</td>
<td>86096</td>
<td>2015-10-29 15:24:34</td>
<td>164.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Run Environment Details (for the given Job):**

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALT_LINKER</td>
<td>/sw/x30.cies_2.pe/2015/63/val/maste/sies11.3/binld</td>
</tr>
<tr>
<td>ASSEMBLER_x86_64</td>
<td>/opt/cray/lt/cce/8.3.9/cray-binutils/x86_64-unknown-linux-gnueabihf</td>
</tr>
<tr>
<td>ATP_HOME</td>
<td>/opt/cray/atp/1.8.9</td>
</tr>
<tr>
<td>ATP_MNPET_COMM_PATH</td>
<td>/opt/cray/atp/1.8.9/libexec/atp_mnpet_common deterrent</td>
</tr>
<tr>
<td>ATP_POST_LINK_OPTS</td>
<td>-Wl,-Lopt/cray/atp/1.8.9/libApp</td>
</tr>
<tr>
<td>CC_x86_64</td>
<td>/opt/cray/cce/8.3.9/cc/x86_64</td>
</tr>
<tr>
<td>CPU</td>
<td>x86_64</td>
</tr>
<tr>
<td>CRAYLIBS_X86_64</td>
<td>/opt/cray/cce/8.3.9/craylibs/x86_64</td>
</tr>
<tr>
<td>CRAYMD_LICENSE_FILE</td>
<td>/opt/cray/cce/lic</td>
</tr>
<tr>
<td>CRAYOS_VERSION</td>
<td>5.2.40</td>
</tr>
<tr>
<td>CRAYPE_DIR</td>
<td>/opt/cray/craype/2.2.1</td>
</tr>
<tr>
<td>CRAYXE_NETCDF_PROFILE</td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

- XALT has been in production for over a year
- XALT has been successfully deployed on multiple HPC centers to support their operations
- XALT helps stakeholders make data-driven decision on software support
- Further analysis on XALT data may yield more understanding of interesting users’ behavior
- Source: https://github.com/Fahey-McLay/xalt
Acknowledgment

• This work was supported by the NSF award 1339690 entitled “Collaborative Research: SI2-SSE: XALT: Understanding the Software Needs of High End Computer Users.”

• Thanks to the XALT community for feedback and bug reports