XALT

Tutorial: Tracking users codes and environments on a cluster with XALT

Mark R. Fahey, LCF Division, ANL
Robert McLay, TACC, U. Texas Austin
Reuben D. Budiardja, NICS, U. Tennessee Knoxville

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.”

- This material is based upon work performed using computational resources provided by the University of Tennessee’s Joint Institute for Computational Sciences and the Texas Advanced Computing Center (TACC) at the University of Texas at Austin.

- Argonne National Laboratory's work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under contract DE-AC02- 06CH11357.

Outline

- Introduction and Motivation, 10 min
- Prepare to install, 20 min
  - Things to consider
- Installation and online demo, 50 min
  - Configure, make, install
  - Post install changes
- Break, 30 min
- Testing, 30 min
  - Debugging
- Production, 15 min
  - Wrappers
  - Data transmission
- Data mining, 30 min

Introduction and Motivation

XALT Overview

- Goal is a census of libraries and applications
- Collecting job-level and link-time level data and subsequent analytics

Overview (2)

- NSF funded project
- Trying to balance the need for portability with support for site-specific capabilities
- Have a fairly straightforward process system administrators use to install, configure, and manage
- Attempting to build a community around analytics of software usage
  - this would be potentially one of many tools
- Making it available to the community
  - Sourceforge and github

Why

Most every computing center needs or wants

- How many users and projects use a particular library or executable
- If a library they maintain is used (and how often)
- If center provided packages are used more or less than user-installed packages
- After the fact identify users and code that used a buggy library
- Provide information on how an executable was built (provenance data) - how did I build my code 6 months ago
- Catch compile time/run time differences
- Identify applications that are using deprecated libraries or just identify old binaries

Goals

- Avoid impacting the user experience
- Must work seamlessly on any cluster, workstation or high-end computer
- Supports both static and dynamic libraries
- Lightweight solution

Tutorial Format

- **Will intersperse slide lecture with online demonstration**
- **You are highly encourage to follow along doing the install on your own Cray (or cluster)**
  - **Everything we do today can be done in user space**

Prepare to install

Installation is mostly automated with some manual steps

Obtaining

- Released tar file from sourceforge
  - http://sourceforge.net/projects/xalt/
- Github repo
  - https://github.com/Fahey-McLay/xalt.git

Prerequisites

- Clients:
  - Python 2.6 or later
  - Python MySQLDb module
- Servers
  - MySQL with proper IP ranges opened for client machines

Unpacking

- Choose an installation directory
  - Likely to be new versions as we provide patches and add functionality
  - Might want to provide for multiple versions
- Directory will be referred to as XALT_DIR

Decisions: Database(s)

- If you are running XALT on multiple machines, must decide if you want to have one database for all or multiple databases (e.g. one per machine)
  - Either mode is easily support
  - Developers use one database
- Regardless, will need file/syslog to database scripts for each machine

Decisions: Lmod/ReverseMap

- If your machines have different software installations (module lists) *AND* you want the **ReverseMap** support in XALT, then you will need an Lmod ReverseMap per machine
  - If machines are different architectures, then need a build of Lmod per machine
  - Lmod provides the spider utility which creates the map
  - TCL module system does not need to be replaced
  - Lmod can be added later, a script is available to update table entries with ReverseMap after the fact

Decisions: Intercepting

- If you have multiple code launchers or linkers, then you have to pick a method to intercept them
- This also depends on how you “install” XALT: with a module or as a replacement
- If they are already wrapped/intercepted, then add XALT as a plugin

Decisions: Intercepting (cont)

- With a single linker/launcher, one can replace them – rename the originals and put XALT wrappers in place
  - Not recommended as it requires modifications to our wrappers
  - If in /usr/bin, then the modulefile for each MPI must set some environment variable that the ld and mpirun wrappers recognize
  - If, however, the XALT wrappers are placed in each each MPI installation directory (not recommended), then the wrappers just call "./ld.x" and "./mpirun.x"
    - Within the ld and mpirun wrappers, you still need to set the XALT_DIR properly to where-ever it was unpacked and configured

Decisions: Intercepting (cont)

- If you have multiple launchers, then there must be a way to swap them (assuming modules)
  - Create a modulefile for XALT (for each version)
  - An example modulefile is provided with the xalt.tar file. This can be used to help make XALT part of the default environment. The modulefile modifies the default user PATH and puts the ld and mpirun wrappers first in the PATH.

- This requires addressing an issue: any change to the MPI library by a module swap will not keep the XALT wrapper first in the path
  - **Have each MPI modulefile reload the XALT modulefile (or inline the XALT modulefile contents) OR**
  - **Use Lmod as the module system which allows one to specify priorities on a PATH setting. Lmod has the ability to prioritize modules to keep them first in the path**
    - Set the XALT DIR priority to ensure that the MPI modulefile swaps will not be a problem.

Suggestion: Intercepting Linker on Cray

- Cray provides (undocumented ?) environmental variable `ALT_LINKER` to use (called by cc, CC, ftn wrappers)
- Set `ALT_LINKER` to `$XALT_DIR/bin/ld` in modulefile

Decisions: etc dir

- Pick a location for a few files
  - xalt_db.conf
  - reverseMapD directory
- Suggest for simplicity that these go in XALT_DIR/etc, but this is up to you
  - It may be that the site wants the xalt_db.conf file somewhere more hidden/secure
  - This can be chosen with the configuration option --with-etcDir=ans or overridden at runtime with XALT_ETC_DIR
  - Note that you will need to have a XALT_ETC_DIR directory for each machine

Installation

Installation Prep

- For this tutorial, we will install everything to $XALT_DIR
- Choose a directory to be used as $XALT_DIR (can be in your Home dir), then

```
% export XALT_DIR=[directory]
% export PATH=$XALT_DIR/bin:$PATH
```

or

```
% setenv XALT_DIR [directory]
% setenv PATH $PATH\:$XALT_DIR/bin
```

Lmod Installation (optional)

- Optional installation of Lmod
  - Needed for reverse map capability (optional, can be done after the fact)
  - Can be used to replace TCL module system (optional)
- Lmod requires Lua with *posix* and *lfs* module
  - Get Lua from Lmod's site (already include needed Lua modules): http://sourceforge.net/projects/lmod/files/lua-5.1.4.8.tar.gz/download

```
./configure --prefix=$XALT_DIR; make; make install
```

- Get Lmod-5.8rc2 or greater from…
  - git clone https://github.com/TACC/Lmod.git or download Lmod-<ver>.tar.gz from github.

```
./configure --prefix=$XALT_DIR; make install;
```

- Documentation for Lmod can be found at: https://www.tacc.utexas.edu/tacc-projects/lmod

XALT Installation: Automated part

- Again, mostly automated, but a few things need to be done by hand
- configure
  - Will determine if a newer psmisc installation is needed
  - Might want to specify
    - --prefix=[$XALT_DIR]
    - --with-etcDir=[directory]
    - --with-transmission=[file,syslog,directdb]
- make
- make install
  - Installs mpirun, aprun, srun, ibrun
    - might want to delete inappropriate launchers

Installation: Manual part (1)

- Modify the site/xalt_syshost_default.py file
  - This file must return the name or names of the hosts you want stored in the database
  - We suggest one name for an entire machine, but you could go with a name for each compute node or partition
  - We provide a stub xalt_syshost_default.py file
    - Examples work on Stampede (TACC) and Darter (NICS) to provide one name for each node on the machines
    - Up to installer to modify it to return what the site wants

Installation: manual part (2)

- Might need to develop your own code launcher if not part of XALT
  - We provide mpirun, aprun, srun, and ibrun launchers
  - Might also require creating site/xalt_find_exec_**xxxx**.py
- We hope sites will contribute new launchers or fixes to these wrappers if issues are detected

This is already provided for Cray (XE, XC)

Installation: manual part (3)

- Might require edits to site/xalt_site_pkg.py
  - provides LSF, SGE, SLURM and PBS hooks
  - should review for your site
  - will add as we get exposure to other batch schedulers and hope to see sites contribute
  - more data for the “run” table
    - e.g. number of cores requested

This is already provided for Cray (XE, XC)

Installation: manual part (4)

- Create modulefile for XALT
  - unless you want to do something else
  - Example modulefile is provided
  - A “priority” setting is included if you are using Lmod
    - Again this can keep XALT first in the path
  - Set ALT_LINKER for Cray (XC, XE) wrapper
- Choose database transmission method
  - If you didn’t choose at configure time or want to change your choice, set XALT_TRANSMISSION_STYLE in the modulefile
  - File, syslog or directdb
    - File is best for initial testing/debugging

Modulefile

- Create modulefile to put XALT's aprun in front of PATH

```
prepend-path PATH          /sw/xc30/xalt/0.5.3-1/sles11.2/bin
prepend-path PATH          /sw/xc30/xalt/0.5.3-1/sles11.2/libexec
prepend-path PYTHONPATH    /sw/xc30/xalt/0.5.3-1/sles11.2/libexec
prepend-path PYTHONPATH    /sw/xc30/xalt/0.5.3-1/sles11.2/site

# ALT_LINKER makes this work with the Cray compiler
setenv  ALT_LINKER     /sw/xc30/xalt/0.5.3-1/sles11.2/bin/ld


# options can be (file,directdb,syslog)
setenv  XALT_TRANSMISSION_STYLE  file

setenv  XALT_ETC_DIR  /sw/xc30/xalt/0.5.3-1/sles11.2/etc
```

Installation: manual part (5)

Set up database

- run sbin/conf_create.py to create your own xalt_db.conf file to point to your MySQL server
  - This file needs to be put in the XALT_ETC_DIR location that was either specified at configuration(--with-etcDir=ans) or by the environment variable XALT_ETC_DIR

- Use sbin/createDB.py to create the initial db schema
  - may need to be run from where xalt_db.conf reside
- For this tutorial, open MySQL server is provided

```
host=104.236.169.194
user=cugxalt
passwd=chicago
db=xalt
```

Installation

- At this point, the XALT code base is installed and will work. But there is more you likely want to do. We will call the rest post-installation.

Break

Testing

Testing: setup

- Highly suggest setting transmission style to “file” for initial testing (default on install)
  - File is best for initial testing/debugging
  - Can set it on command line or in modulefile via XALT_TRANSMISSION_STYLE environment variable
- Load your new XALT modulefile
  - Unload old XALT or ALTD modulefile (first)
  - Remember to do this inside batch jobs too!

Testing: setup

- Create reverse map with lmod
  - *optional* but nice feature
  - Not necessary now, but since we are testing a good time to do it
- Run the spider tool that comes with lmod
  - Will map all lib and exe directories to modulefiles
  - On Cray's a bit more complicated but still works
    - An example script is provided in the contrib directory
  - This resulting map (text file) must go in the XALT_ETC_DIR directory set at configure or set by the env var in the modulefile

Testing: cray_build_rmapT.sh

- To create the reverse map on a Cray XC, XE series, run the contributed script cray_build_rmapT.sh
  - This uses the spider utility from Lmod
    - need to set LMOD_DIR=[lmod's prefix during install]
  - You might need to edit the PrgEnvA setting in the script
  - Typically, place the reverse map file in $XALT_ETC_DIR

```
% ./cray_build_rmapT.sh $XALT_ETC_DIR
```

- On a cluster, more simple

```
% spider -o jsonReverseMapT $LMOD_DEFAULT_MODULEPATH > rmapD/jsonReverseMapT.json
```

Testing: setup

- If you plan to test XALT with multiple MPIs, then you will need to **keep XALT first in the path**
- **Options**
  - Edit MPI modulefiles
  - Use Lmod as TCL module system replacement
  - Move mpirun wrappers and put ours in place
  - Same for ld
- For now, you can reload the XALT modulefile after each MPI swap

Testing

- Testing is now simple and straightforward
- Compile a code and run it in a job
  - Remember to have XALT loaded in the job
- There should be two files in ~/.xalt.d/
  - link.machine.XXXX
  - run.machine.XXXX
- Lets look at both of them now

Link.machine.*


```
{
  "build_epoch": 1428693356.71,
  "build_syshost": "darter",
  "build_user": "faheymr",
  "exec_path":
"/nics/d/home/faheymr/examples/HelloWorld/h
ello_world",
  "exit_code": 0,
  "hash_id":
"6fac4296d9be1407315abc33660bda6daa824b01",
```

Link.machine.* (2)

```
"linkA": [
    [
"/nics/d/home/faheymr/examples/HelloWorld/h
ello_world.o",

"42aa18fc85e8f05fe89b6f501c87ad3e26843130"
    ],
    [ "/opt/cray/mpt/7.0.3/gni/mpich2-
cray/83/lib/libmpich_cray.a",

"fc7ff1c7c6ff9c326e261037a28cef7ce2fc1d8e"
    ],
```

Link.machine.* (3)

```
    [
      "/usr/lib64/libpthread.a",
"51883ac8b5e65f19135203e0ddd80e3e302a71f1"
    ]
  ],
  "link_program": "driver.cc",
  "uuid": "995980ad-7daa-4841-be80-
21b6e05fe375",
  "wd":
"/nics/d/home/faheymr/examples/HelloWorld"
}
```

Run.machine.*

```
{
  "envT": {
    "ALT_LINKER": "/sw/xc30/xalt/0.5.3-1/sles11.2/bin/ld",
    "ASSEMBLER_X86_64": "/opt/cray/cce/8.3.3/cray-binutils/x86_64-
unknown-linux-gnu/bin/as",
    "ATP_HOME": "/opt/cray/atp/1.7.5",

...
```

Run.machine.* (2)


```
},
"hash_id": "7b88e3f497365d6bc2f86f634a2a26a8d886056e",
"libA": [],
"userT": {
  "currentEpoch": 1428766964.8458829,
  "cwd": "/lustre/medusa/faheymr",
  "end_time": 1428766964.73,
  "execModify": "Sat Apr 11 11:42:43 2015",
  "exec_epoch": 1428766963.0,
  "exec_path": "/lustre/medusa/faheymr/hello_world",
  "exec_type": "binary",
  "exit_status": 0,
  "num_tasks": 1,
  "num_threads": 0,
  "run_time": 1.3559999465942383,
  "run_uuid": "e3adf193-5f09-46c6-8c0d-18ff0d555dde",
  "start_date": "Sat Apr 11 11:42:43 2015",
  "start_time": 1428766963.3740001,
  "syshost": "darter",
  "user": "faheymr"
},
```

Run.machine.* (3)

```
  "xaltLinkT": {
    "Build.Epoch": "1428693356.71",
    "Build.OS": "Linux 3.0.101-0.46-default",
    "Build.Syshost": "darter",
    "Build.UUID": "995980ad-7daa-4841-be80-21b6e05fe375",
    "Build.User": "faheymr",
    "Build.Year": "2015",
    "Build.compiler": "driver.cc",
    "Build.date": "Fri Apr 10 15:15:56 2015",
    "XALT_Version": "0.5.3"
  }
}
```

Things that could be wrong

- Missing files
- Syshost name
  - Need to go back and work on xalt_syshost_default.py
- Num_cores
  - Something likely wrong with site_pkg or launcher

Testing

- If all the output in the files looks right, then we should test the parsing of the data and transmission to the database
- If you have files that have bad data, then delete them – maybe delete them all and create some new ones when you think everything looks right
- Now we can use the sbin/xalt_file_to_db.py script

Xalt_file_to_db.py

- Example of how to use xalt_file_to_db.py

```
% export PATH=/sw/tools/lmod/builds/darter/lmod/lmod/lmod/libexec:${PATH}

% export XALT_USERS=faheymr:reubendb

% ln -s $XALT_DIR/etc/xalt_db.conf .

% ln -s $XALT_DIR/sbin/xalt_file_to_db.py .

% $XALT_DIR/sbin/xalt_file_to_db.py --reverseMapD
/sw/tools/xalt/builds/darter/etc/reverseMapD --delete --timer


Time:  00:00:00
total processed :  25 , num links:  10 , num runs:  15 , badCnt:  0
```

Testing: file to db issues

- You may have access issues like only certain accounts can insert or access is only granted to hosts
- There is a simple summary report generated
  - Total records parsed, links, runs, bad entries
  - Bad could be several things:
    - Couldn't decode file (or syslog)
      - For each bad entry there is a number printed that is the length of the number of characters – this number may be a clue to syslog limit issues
    - Couldn't insert data into db

Things that could be wrong

- Modulefile name
  - The reverse map is likely not right
  - Check the spider output
  - Make sure to compile a code that uses hdf5 or netcdf or some other package that should have a modulefile name

Testing: directdb

- Set XALT_TRANSMISSION_STYLE to directdb either on command line or in modulefile
- This method has the database updated immediately when ld or mpirun wrappers are used
  - This is how ALTD worked for years at many sites
- Even if you don't want to use it, good to test to make sure it works
  - No files generated, so no trace of what was done other than what is in the database

Testing

- Lets log into the mysql server and find our own entries

```
% mysql -h 104.236.169.194 -u cugxalt \
-pchicago xalt
```

Production

Production: reverseMap

- The reverse map has been mentioned several times before, but exactly how do you create it? As described above, Lmod can be used as a module replacement. But even if you don't want to replace TCL modules, you can use Lmod to create what we refer to as the reverse map. Basically, it maps libraries (with paths) back to modulefiles.

- If you have your modulefiles set up with a one-to-one to mapping of modules to package installations, then Lmod can probably create the reverse map without issue.

- But on some machines, a modulefile (with appropriate if-tests) can point to a variety of installations and set environment variables depending on the currently loaded compilers and MPI. In this scenario, the reverse map can be created, but it is a looser reverse map with many-to-one relationships (e.g. this is how Cray's provided software / modulefile works)

Production: reverseMap

- And further on some machines (like Crays), for spider to work, you have to run it multiple times (one for each Programming Environment) to get multiple reverse maps, which then have to be combined together for a master reverse map. Below you will see a sample script for how to do this.

- *The reverse map needs to be created/updated per machine every time a new modulefile or package is installed.* So either 1). it has to become part of the software installation process, or 2). run as a cron job every week for example, then use a (provided) script to update the module name entries in DB

- And if you have multiple machines and one XALT installation, then you will need to have a reversemap for each machine. That means as an example the etc directory will likely need to have subdirectories for each machine and a reverseMapD in each of those directories. And you will need to set XALT_ETC_DIR in the modulefile for each machine to point to the appropriate place.

Production: reverseMap

Examples

- **Simple command to create reverseMap**
- spider -o jsonReverseMapT $LMOD_MODULEPATH > rmapD/jsonReverseMapT.json
- **Cray script to create reverseMap**
- An example script, darter_build_rmapT.sh, for a Cray XC30 that uses Lmod (namely the spider utility) to create the reverseMap is provided in the contrib/build_reverseMapT_cray/ directory.

Production:

- **Keep XALT first in the path**
  - Edit MPI modulefiles
    - Unload and load XALT, or
    - Put XALT modulefile contents in MPI modulefile
  - Use Lmod at TCL module system replacement
  - Both of these in use at sites
  - If one launcher, then another option is to rename ld and mpirun to ld.x and mpirun.x and put our wrappers in their place

Production: wrappers

- Some tools have issues with wrappers
  - Totalview for instance
  - It is easy to unload xalt automatically when loading totalview if you use modules
  - Much harder to automatically load xalt when totalview is not being used
    - So some links and runs are missed – small percent

Production: syslog

- Arguably the best method for production
  - All records first go to syslog
  - Then the records must be parsed later (like file method) to go to database
- Set XALT_TRANSMISSION_STYLE to syslog
- Set syslog settings (actually rsyslog)
- To use syslog (and in this context we mean rsyslog since that is all we have tested)
  - set up a configuration file for syslog and place in /etc/rsyslog.d/ that we called xalt_syslog.conf

```
$MaxMessageSize 256k

if $programname contains 'XALT_LOGGING' then
/var/log/xalt.log

& ~
```

- This example shows that the log file is set up as /var/log/xalt.log. Note that /var is probably local for the node where the linker or job launcher is run.
  - This is fine, but you will have to run the syslog parser on each node for this setup.
  - Alternatively, it might be easier if you had say one node/server where all the log files could be put by syslog and then you would only have to run the parser on that node, but for each file.
  - For testing, you might want to also put the files in your home or scratch directory for some short period of time
- Also note that 256k is the maximum message size as given. We have hit a case where a link line was larger than 64k and this results in the XALT log message being incomplete and as a result the parser will have to skip those entries as incomplete

Production: syslog (2)

- modify /etc/rsyslog.conf to use this new configuration

```
# Include all config files in
/etc/rsyslog.d/

$IncludeConfig /etc/rsyslog.d/*.conf
```

- Restart rsyslog
- Can't read the entries directly as they are encoded
- Will have to parse the log file and see the results in the database (or modify the parser to print the results)

Testing: parse syslog file

- use xalt_syslog_to_db.py to collect data from syslog

```
python xalt_syslog_to_db.py –
reverseMapD=$BASE/etc/reverseMapD
/var/log/xalt.log.1
```

- Can be rerun if desired – will not make duplicate entries

Production: syslog rotation

- set up rotation on the /var/log/xalt.log log file with a logrotate configuration file like

```
/etc/logrotate.d>  cat xalt
/var/log/xalt.log{
  copytruncate
  rotate 4
  daily
  create 0644 root root
  missingok
}
```

- The above sets a 4 day rotation on the files. We suggest nothing less than 2. The above is also setting the log file to be readable by all. This is a site dependent setting - in this case, a non-root account can be used to parse the data and put it in the database.

Production: syslog notes

- *A site might want to "send" their syslog records to a separate server and then parse the records there*
  - *Make sure to use the appropriate reverseMap*
- *We assume the installers know that all of these steps for syslog will have to be done on each of the nodes where the linker and the job launcher (mpirun, aprun, etc) will be run.*

Data Mining

Usage

- There are many analyses that can be done with XALT data
  - Most/least and trends reports for
    - Libraries
    - Modulefiles
    - Applications
    - Based on user or project or time used
  - Last time a library was used
  - Provenance/reproducibility reports
  - Providing usage statistics to developers and vendors
  - Restoring the program environment where user applications were built

Program usage

```
mysql> select link_program, build_syshost, count(*) from
xalt_link group by link_program, build_syshost;
+--------------+---------------+----------+
| link_program | build_syshost | count(*) |
+--------------+---------------+----------+
| configure    | darter        |        7 |
| driver.CC    | darter        |        8 |
| driver.cc    | darter        |      180 |
| ftn_driver   | darter        |      173 |
| g++          | darter        |     2396 |
| gcc          | darter        |     6190 |
| gfortran     | darter        |      959 |
| icc          | darter        |     1890 |
| icpc         | darter        |      562 |
| ifort        | darter        |      915 |
| make         | darter        |      123 |
+--------------+---------------+----------+
11 rows in set (0.02 sec)
```

Identifying users or codes or libraries

A critical bug was identified in FFTW version 3.3.0.2, affecting code correctness

Find which users have linked this library

```
mysql> select distinct build_user from xalt_link,xalt_object
where xalt_object.object_path like'%fftw/3.3.0.2/%' ;

+----------+
| username |
+----------+
| user1    |
| user2    |
| user3    |
| user4    |
| user5    |
+----------+
5 rows in set (1.33 sec)
```

- Querying the database reveals that several users have applications linked to the buggy library

Was the buggy library used?

```
mysql> select distinct xalt_run.run_id, xalt_run.job_id, xalt_run.date, xalt_run.syshost,
xalt_run.user, xalt_run.exec_path from xalt_run, xalt_object, join_run_object where
xalt_object.object_path like '%fftw/3.3.0.2/%' AND xalt_object.obj_id=join_run_object.obj_id
AND join_run_object.run_id=xalt_run.run_id;

+--------+-------------+---------------------+---------+--------+------------------+
| run_id | job_id      | date                | syshost | user   | exec_path        |
+--------+-------------+---------------------+---------+--------+------------------+
|   7273 | 350840.ocoe | 2014-10-23 13:22:29 | darter  | user4  | ~/cp2k/cp2k.psmp |
+--------+-------------+---------------------+---------+--------+------------------+
1 rows in set (0.08 sec)
```

And it's confirmed that user "user4" has run the application linked to the buggy library

It's now up to the user services group to contact the user and recommend relinking their applications against the newer version of FFTW, which has fixed the bug

How did I build my program 2 months ago?

```
mysql> select xalt_link.* from xalt_link where build_user like '%faheymr%' AND
exec_path like '%hyperslab%';

+---------+---------------------+-------------+-----------+--------------
+---------------+-----------+-------------------------------------------+

| link_id | date                | link_program | build_user | build_syshost |
build_epoch   | exit_code | exec_path                                 |

+---------+---------------------+-------------+-----------+--------------
+---------------+-----------+-------------------------------------------+

|       4 | 2014-09-23 14:17:29 | ftn_driver   | faheymr    | darter        |
1411496249.58 |         0 | /nics/d/home/faheymr/examples/hdf5/hyperslab |

+---------+---------------------+-------------+-----------+--------------
+---------------+-----------+-------------------------------------------+

3 rows in set (0.01 sec)
```

How did I build my program 2 months ago? (cont)

```
mysql> select object_path, timestamp from xalt_object, join_link_object where join_link_object.
link_id="4" AND join_link_object.obj_id=xalt_object.obj_id;

+-----------------------------------------------------------------------+---------------------+
| object_path                                                           | timestamp           |
+-----------------------------------------------------------------------+---------------------+
| //usr/lib64/libc.a                                                    | 2014-09-26 15:49:53 |
| //usr/lib64/libdl.a                                                   | 2014-09-26 15:49:53 |
| //usr/lib64/libhugetlbfs.a                                            | 2014-09-26 15:49:53 |
| //usr/lib64/libm.a                                                    | 2014-09-26 15:49:53 |
| //usr/lib64/libpthread.a                                              | 2014-09-26 15:49:53 |
| //usr/lib64/librt.a                                                   | 2014-09-26 15:49:53 |
| //usr/lib64/libz.a                                                    | 2014-09-26 15:49:53 |
| /opt/cray/atp/1.7.2/lib/libAtpSigHCommData.a                          | 2014-09-26 15:49:53 |
| /opt/cray/atp/1.7.2/lib/libAtpSigHandler.a                            | 2014-09-26 15:49:53 |
| /opt/cray/cce/8.2.5/craylibs/x86-64/libcsup.a                         | 2014-09-26 15:49:53 |
| /opt/cray/cce/8.2.5/craylibs/x86-64/libf.a                            | 2014-09-26 15:49:53 |
| /opt/cray/cce/8.2.5/craylibs/x86-64/libfi.a                           | 2014-09-26 15:49:53 |
| /opt/cray/cce/8.2.5/craylibs/x86-64/libtcmalloc_minimal.a             | 2014-09-26 15:49:53 |
| /opt/cray/cce/8.2.5/craylibs/x86-64/libu.a                            | 2014-09-26 15:49:53 |
| /opt/cray/cce/8.2.5/craylibs/x86-64/no_mmap.o                         | 2014-09-26 15:49:53 |
| /opt/cray/hdf5/1.8.12/CRAY/81/lib/libhdf5_cray.a                      | 2014-09-26 15:49:53 |
| /opt/cray/hdf5/1.8.12/CRAY/81/lib/libhdf5_fortran_cray.a              | 2014-09-26 15:49:53 |
| /opt/gcc/4.4.4/snos/lib/gcc/x86_64-suse-linux/4.4.4/crtbeginT.o       | 2014-09-26 15:49:53 |
| /opt/gcc/4.4.4/snos/lib/gcc/x86_64-suse-linux/4.4.4/crtend.o          | 2014-09-26 15:49:53 |
| /opt/gcc/4.4.4/snos/lib/gcc/x86_64-suse-linux/4.4.4/crtfastmath.o     | 2014-09-26 15:49:53 |
| /opt/gcc/4.4.4/snos/lib/gcc/x86_64-suse-linux/4.4.4/libgcc.a          | 2014-09-26 15:49:53 |
| /opt/gcc/4.4.4/snos/lib/gcc/x86_64-suse-linux/4.4.4/libgcc_eh.a       | 2014-09-26 15:49:53 |
| /opt/gcc/4.4.4/snos/lib64/libstdc++.a                                 | 2014-09-26 15:49:53 |
| /tmp/pe_14932/hyperslab_1.o                                           | 2014-09-26 15:49:53 |
| /usr/lib64/crt1.o                                                     | 2014-09-26 15:49:53 |
| /usr/lib64/crti.o                                                     | 2014-09-26 15:49:53 |
| /usr/lib64/crtn.o                                                     | 2014-09-26 15:49:53 |
+-----------------------------------------------------------------------+---------------------+
27 rows in set (0.00 sec)
```

Cray cce/8.2.5
Hdf5/1.8.12

How did I build my program 2 months ago? (cont)

```
...
| /opt/cray/cce/8.2.5/craylibs/x86-64/libcsup.a                | 2014-09-26 15:49:53 |
| /opt/cray/cce/8.2.5/craylibs/x86-64/libf.a                   | 2014-09-26 15:49:53 |
| /opt/cray/cce/8.2.5/craylibs/x86-64/libfi.a                  | 2014-09-26 15:49:53 |
| /opt/cray/cce/8.2.5/craylibs/x86-64/libtcmalloc_minimal.a    | 2014-09-26 15:49:53 |
| /opt/cray/cce/8.2.5/craylibs/x86-64/libu.a                   | 2014-09-26 15:49:53 |
| /opt/cray/cce/8.2.5/craylibs/x86-64/no_mmap.o                | 2014-09-26 15:49:53 |
| /opt/cray/hdf5/1.8.12/CRAY/81/lib/libhdf5_cray.a             | 2014-09-26 15:49:53 |
| /opt/cray/hdf5/1.8.12/CRAY/81/lib/libhdf5_fortran_cray.a     | 2014-09-26 15:49:53 |
...
```

Modulefile usage

```
mysql> SELECT xalt_object.module_name, count(xalt_run.
date) AS Jobs, ROUND(SUM(run_time*num_cores)/3600) as
TotalSUs FROM xalt_run, xalt_link, join_link_object,
xalt_object WHERE xalt_run.syshost='darter' AND
xalt_object.module_name is NOT NULL AND xalt_run.uuid =
xalt_link.uuid AND xalt_link.link_id = join_link_object.
link_id AND join_link_object.obj_id = xalt_object.obj_id
AND xalt_run.date >= '2014-11-01' AND xalt_run.date <=
'2014-11-09' GROUP BY xalt_object.module_name ORDER BY
Jobs DESC;
```

Modulefile usage (2)

module_name	Jobs	TotalSUs
alps/5.2.1-2.0502.8712.10.32.ari	26458	258684
cray-mpich/7.0.3	26456	259040
wlm_detect/1.0-1.0502.51217.1.1.ari	13229	129342
udreg/2.3.2-1.0502.8763.1.11.ari	13229	129342
xpmem/0.1-2.0502.51169.1.11.ari	13229	129342
ugni/5.0-1.0502.9037.7.26.ari	13229	129342
pmi/5.0.5-1.0000.10300.134.8.ari	13227	129341
gcc/4.8.1	10868	59680
rca/1.0.0-2.0502.51491.3.92.ari	10852	59675
dmapp/7.0.1-1.0502.9080.9.32.ari	10852	59675
fftw/3.3.4.0	**3123**	**1482**
cray-libsci/13.0.1	**2357**	**69848**
craype-intel-knc	**1758**	**522**
hdf4/4.2.9	**1180**	**667**
cray-hdf5/1.8.12	**586**	**174**
cray-netcdf/4.3.1	**586**	**174**
szip/2.1	**295**	**167**
fftw/3.3.0.4	**274**	**78293**

Other

Automating the process of alerting?

- Ideally, user support specialists would be alerted automatically to “situations of interest”
  - Users running applications linked to legacy, less-performant, or buggy libraries
  - Users running legacy versions of applications
  - Users building code with legacy compilers
  - Users making use of their own libs or apps, when more optimized versions are available centrally

TACC_Stats (and SUPReMM)

- Job-level transparent performance monitoring from HPC compute nodes
  - CPU performance counters
  - IB statistics
  - Lustre statistics
  - Scheduler job statistics
  - Host data
  - OS statistics
- Analyses integrate available Lariat data (and will be XALT in the near future)

Future

- Will add a feature to track function calls
  - Only those function calls resolved by external libraries
  - Will not track
    - User defined functions
    - Auxiliary functions in a library
- Ability to compare run time environment against compile time environment
  - Provide warning messages to users
  - May help users self-diagnose run time problems
- Much, much more data analysis

Acknowledgments - Thanks

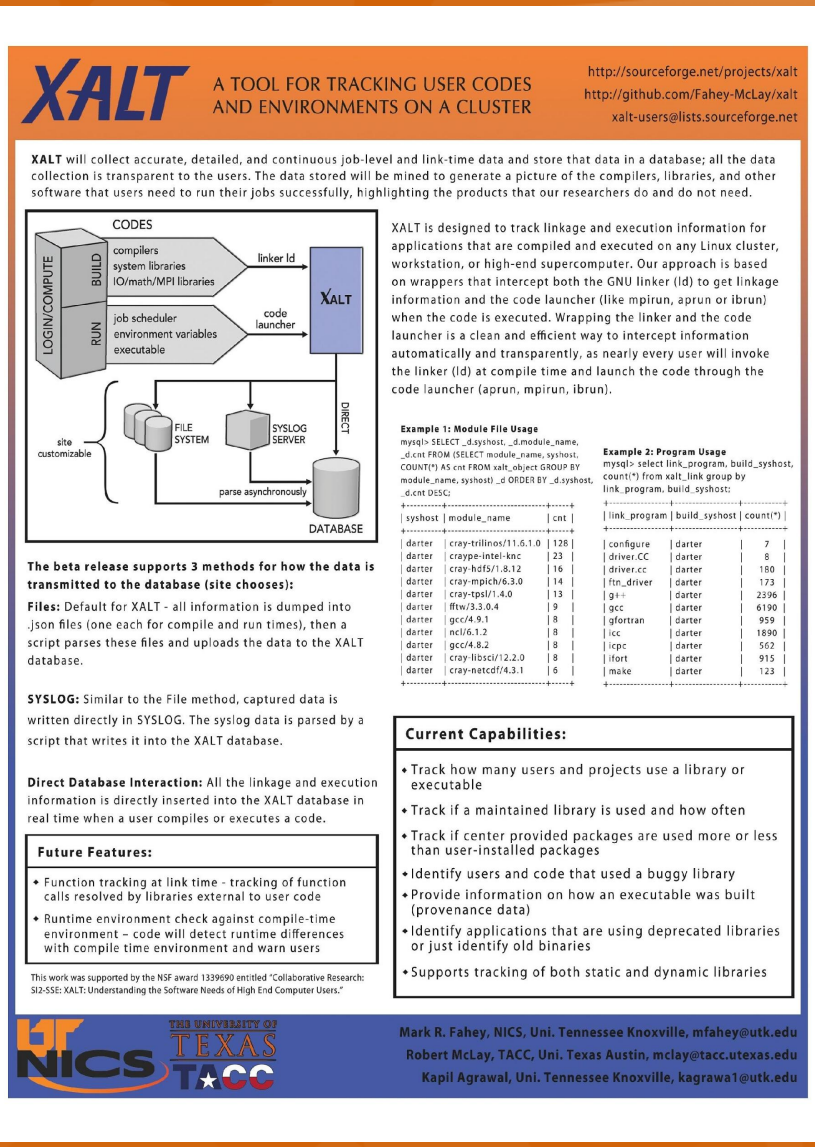
- provided valuable feedback or [alpha] tested code
  - Tim Robinson, CSCS
  - Bilel Hadri, KAUST
  - Zhengji Zhao, NERSC
  - Julius Westerman, LANL
  - Davide Del Vento, NCAR
  - Andrew Elwell, iVEC

Current issues

- International character support
  - In progress to support UTF8 characters, not fully working yet

Thank You

- Contacts
  - mfahey@anl.gov
  - mclay@tacc.utexas.edu
  - reubendb@utk.edu



XALT

A TOOL FOR TRACKING USER CODES AND ENVIRONMENTS ON A CLUSTER

http://sourceforge.net/projects/xalt
http://github.com/Fahey-McLay/xalt
xalt-users@lists.sourceforge.net

XALT will collect accurate, detailed, and continuous job-level and link-time data and store that data in a database; all the data collection is transparent to the users. The data stored will be mined to generate a picture of the compilers, libraries, and other software that users need to run their jobs successfully, highlighting the products that our researchers do and do not need.

XALT is designed to track linkage and execution information for applications that are compiled and executed on any Linux cluster, workstation, or high-end supercomputer. Our approach is based on wrappers that intercept both the GNU linker (ld) to get linkage information and the code launcher (like mpirun, aprun or ibrun) when the code is executed. Wrapping the linker and the code launcher is a clean and efficient way to intercept information automatically and transparently, as nearly every user will invoke the linker (ld) at compile time and launch the code through the code launcher (aprun, mpirun, ibrun).

Example 1: Module File Usage

```
mysql> SELECT _d.syshost, _d.module_name, _d.cnt FROM (SELECT module_name, syshost, COUNT(*) AS cnt FROM xalt_object GROUP BY module_name, syshost) _d ORDER BY _d.syshost, _d.cnt DESC;
+---------+-----------------------------+-----+
| syshost | module_name                 | cnt |
+---------+-----------------------------+-----+
| darter  | cray-trilinos/11.6.1.0      | 128 |
| darter  | craype-intel-knc            |  23 |
| darter  | cray-hdf5/1.8.12            |  16 |
| darter  | cray-mpich/6.3.0            |  14 |
| darter  | cray-tpsl/1.4.0             |  13 |
| darter  | fftw/3.3.0.4                |   9 |
| darter  | gcc/4.9.1                   |   8 |
| darter  | ncl/6.1.2                   |   8 |
| darter  | gcc/4.8.2                   |   8 |
| darter  | cray-libsci/12.2.0          |   8 |
| darter  | cray-netcdf/4.3.1           |   6 |
+---------+-----------------------------+-----+
```

Example 2: Program Usage

```
mysql> select link_program, build_syshost, count(*) from xalt_link group by link_program, build_syshost;
+--------------+----------------+----------+
| link_program | build_syshost  | count(*) |
+--------------+----------------+----------+
| configure    | darter         |        7 |
| driver.CC    | darter         |        8 |
| driver.cc    | darter         |      180 |
| ftn_driver   | darter         |      173 |
| g++          | darter         |     2396 |
| gcc          | darter         |     6190 |
| gfortran     | darter         |      959 |
| icc          | darter         |     1890 |
| icpc         | darter         |      562 |
| ifort        | darter         |      915 |
| make         | darter         |      123 |
+--------------+----------------+----------+
```

The beta release supports 3 methods for how the data is transmitted to the database (site chooses):

Files: Default for XALT - all information is dumped into .json files (one each for compile and run times), then a script parses these files and uploads the data to the XALT database.

SYSLOG: Similar to the File method, captured data is written directly in SYSLOG. The syslog data is parsed by a script that writes it into the XALT database.

Direct Database Interaction: All the linkage and execution information is directly inserted into the XALT database in real time when a user compiles or executes a code.

Current Capabilities:

- Track how many users and projects use a library or executable
- Track if a maintained library is used and how often
- Track if center provided packages are used more or less than user-installed packages
- Identify users and code that used a buggy library
- Provide information on how an executable was built (provenance data)
- Identify applications that are using deprecated libraries or just identify old binaries
- Supports tracking of both static and dynamic libraries

Future Features:

- Function tracking at link time - tracking of function calls resolved by libraries external to user code
- Runtime environment check against compile-time environment – code will detect runtime differences with compile time environment and warn users

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

Mark R. Fahey, NICS, Uni. Tennessee Knoxville, mfahey@utk.edu
Robert McLay, TACC, Uni. Texas Austin, mclay@tacc.utexas.edu
Kapil Agrawal, Uni. Tennessee Knoxville, kagrawa1@utk.edu