User Environment Tracking and Problem Detection with XALT

Kapil Agrawal, Gregory Peterson (Advisor), Reuben Budiardja
National Institute for Computational Sciences (NICS)
University of Tennessee, Knoxville
Knoxville, TN 37996
kagrawa1, gdp, reubendb@vols.utk.edu

Mark Fahey
Argonne National Laboratory
Argonne, IL 60439
mfahey@anl.gov

Robert McLay
Texas Advanced Computing Center (TACC)
University of Texas at Austin
Austin, TX 78712
mclay@tacc.utexas.edu

ABSTRACT
This work enhances our understanding of individual users' software needs, then leverages that understanding to help stakeholders conduct business in a more efficient, effective and systematic manner. XALT is designed to track linkage and execution information for applications that are compiled and executed on any Linux cluster, workstation, or high-end supercomputers. XALT allows administrators and other support staff to consider demand when prioritizing what to install, support and maintain. Datasets, dashboards, and historical reports generated by XALT and the systems with which it interoperates will preserve institutional knowledge and lessons learned so that users, developers, and support staff need not reinvent the wheel when issues arise that have already been encountered.

Keywords
XALT, user environment, library tracking.

1. INTRODUCTION
Developed through collaborative efforts of the National Institute for Computational Sciences (NICS) and the Texas Advanced Computing Center (TACC), XALT is a mechanism for following users' jobs and environments on a computer cluster. It provides a census of libraries and applications and automatically filters user issues, yielding exactly the type of job tracking information that most computing centers need or want.

Computing centers can help researchers in especially important ways. Service providers at both national and university level can provide automatic mechanism to collect information on the versions of software used by the researcher. XALT will flag jobs that require the attention of support staff, deliver alerts to users regarding the fundamental causes of problems preventing their jobs from running, and collect metrics that improve training, documentation, and outreach programs.

2. XALT OVERVIEW
XALT [1] is designed to track linkage and execution information for applications that are compiled and executed on any Linux cluster, workstation, or high-end supercomputers. Our approach is based on wrappers that intercept both the GNU linker (ld) and get linkage information and the code launcher (like mprun, aprun or ibrun) when any code is executed.

The main objectives of XALT were: 1) Must work seamlessly on any cluster, workstation or high-end supercomputers 2) Must support both dynamic and static libraries 3) Lightweight solution 4) Avoid impacting user experience. XALT has achieved its entire stated objective.

2.1 Wrappers
2.1.1 Linker
The linker (ld) wrapper intercepts the user link line. Since in many cases more libraries are included at link time than are actually used, we go through a multiple step process to get all desired linkage information:

1. Generate assembly code
2. Generate link text
3. Generate link data
4. Use arbitrary 'transmission methods' to store the information in XALT database.

2.1.2 Code Launcher
Launching a parallel job on compute nodes is often done via batch systems like PBS, Slurm, or LoadLeveler through a parallel job launcher such as apron, mpirun, mpiexec, or ibrun. These job launchers are intercepted to provide a measure of application usage and a secondary measure of "library usage" (dynamic libraries needed at run time). Similar to linker wrapper this involves multiple steps:

1. Find executable
2. Get actual launcher and command line option
3. Collect link time, job, and shared library information
4. Use arbitrary 'transmission methods' to store the information in XALT database

2.2 Transmission Methods
XALT supports three different types of transmission methods depending upon what some sites may desire based on security and/or performance. XALT is flexible in a way to store the gathered information to its database as per individual's site's requirements. Below are the transmission methods supported by XALT:

1. Files: This is the default method for XALT – all information is dumped into '.json' files (one for each compile time and run time), then a script parses these files and stores the data to XALT database.
2. SYSLOG: As stated earlier some sites may see storing these files in user's home directory as a security and/or performance issue. An alternative is to write to SYSLOG files and later use a script to upload the results to XALT database.
3. Direct to Database: All the linkage and execution information can be inserted directly into the XALT database in real time as and when a user tries to compile an executable or run a job.

The complete overview of XALT can be seen in Figure 1. Currently XALT supports three types of portability aspects to store information in its database.

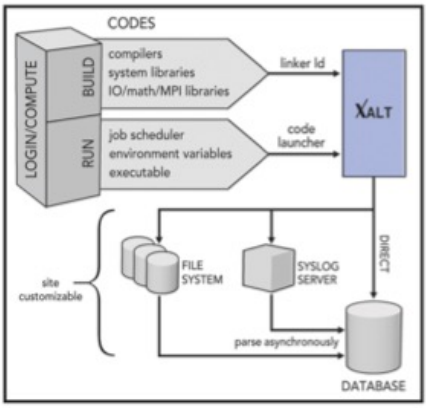

Figure 1. XALT Overview.

3. Preliminary Reports

In this section, we show a few examples of reports generated from the XALT database at NICS.

3.1 Most used compilers

Figure 2 shows the number of compilations performed by users using different compilers on Darter. The most used compilers are C++ compiler followed by GNU Fortran. Although these results are user dependent, they do provide a clear picture of trends on a given machine.

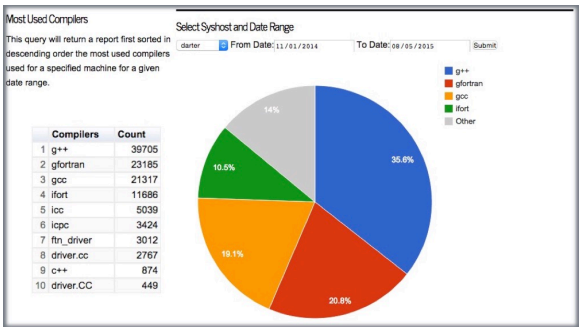

Figure 2. Most Used Compilers

3.2 Identify users who linked to a given library

This report shows a list of users and libraries (both static and dynamic) that used a given library. This result can come handy when the support staff wants to know who all were using that given library. There can be incidents when a user might be using a deprecated library, having this information first-hand could help resolving errors in a timely manner.

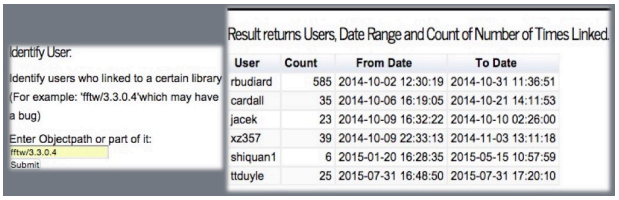

Figure 3. Identify Users

3.3 Get user info:

Sometimes for researchers/users using high-performance computers for their research, it is a painstaking exercise to find what went wrong with their executable. It might happen that a simple recompile somehow causes a job to fail. It would be a lot better if they had some information about how did they build/compile their code earlier. XALT comes handy to resolve these issue by giving a map of how a user built an executable some time ago. Figure 4 shows how a user in a given timeline built an executable.

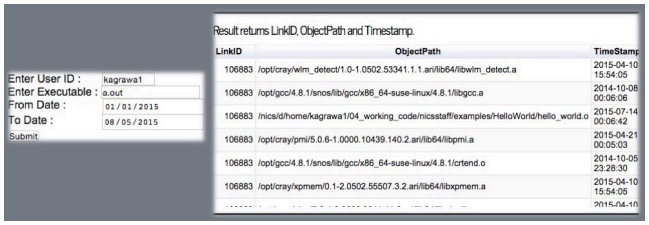

Figure 4. Get User Info

4. CURRENT CAPABILITIES

XALT current capabilities include a host of features like 1) Track if maintained library is used and how often 2) Identify users and code that used a buggy library 3) Support tracking of both static and dynamic libraries 4) Track if center provided packages are used more or less that user-installed packages 5) Track how many users and projects use a library or executable 6) Identify applications that are using deprecated libraries or just identify old binaries 7) Provide information on how an executable was built (provenance data).

5. FUTURE WORK

With the data collected by XALT, a detailed and accurate survey of the usage of software installed by vendors, staff, and user support quality can be improved. We strongly believe that every center should be doing this for a variety of reasons, including better user support, provenance data collection, and security related concerns.

Other new features such as function tracking at link time to track function call resolved by libraries external to user code, and checking runtime environment against compile-time environment are worked upon for future release.

6. ACKNOWLEDGMENTS

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

Resources supported by University of Tennessee's Joint Institute of Computational Sciences and Texas Advanced Computing Center (TACC) at University of Texas at Austin.

7. REFERENCES

[1] K. Agrawal, M. Fahey, R. McLay, D. James, User Environment Tracking and Problem Detection with XALT, SC14, http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=7081224