8-2016

Extending Capability and Implementing a Web Interface for the XALT Software Monitoring Tool

Kapil Agrawal
University of Tennessee, Knoxville, kagrawa1@vols.utk.edu

To the Graduate Council:

I am submitting herewith a thesis written by Kapil Agrawal entitled "Extending Capability and Implementing a Web Interface for the XALT Software Monitoring Tool." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Engineering.

Gregory Peterson, Major Professor

We have read this thesis and recommend its acceptance:

Audris Mockus, Michael Jantz

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Extending Capability and Implementing a Web Interface for the XALT Software Monitoring Tool

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Kapil Agrawal

August 2016

dedicated to my parents Shri. Gopal Das Agrawal and Smt. Geeta Agrawal

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Prof. Gregory Peterson for providing the great opportunity to work at the Joint Institute for Computational Sciences, and for continuous support for my Master study and related research work.

I appreciate Dr. Mark Fahey and Dr. Reuben Budiardja for their patience, motivation, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis.

I would like to thank Prof. Audris Mockus and Asst. Prof. Michael Jantz for being on my graduate committee.

Last but not the least, I would like to thank my friends Mohit, Eduardo, and my family for their support.

Abstract

As high performance computing centers evolve in terms of hardware, software, and user-base, the act of monitoring and managing such systems requires specialized tools. The tool discussed in this thesis is XALT, which is a collaborative effort between the National Institute for Computational Sciences and Texas Advanced Computing Center. XALT is designed to track link-time and job level information for applications that are compiled and executed on any Linux cluster, workstation, or high-end supercomputer. The key objectives of this work are to extend the existing functionality of XALT and implement a real-time web portal to easily visualize the tracked data. A prototype is developed to track function calls resolved by external libraries which helps software management. The web portal generates reports and metrics which would improve efficiency and effectiveness for an extensive community of stakeholders including users, support organizations, and development teams. In addition, we discuss use cases of interest to center support staff and researchers on identifying users based on given counters and generating provenance reports. This work details the opportunity and challenges to further push XALT towards becoming a complete package.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

High-performance computing (HPC) is the use of supercomputers and parallel processing techniques for solving complex computational problems. HPC technology focuses on developing parallel processing algorithms and systems by incorporating both administration and parallel computational techniques [1]. The most common users of HPC systems are scientific researchers, engineers, and academic institutions. Some government agencies, particularly the military, also rely on HPC for complex applications.

High-performance computing systems often use custom-made components in addition to commodity components. On these systems a collection of program libraries and software packages are maintained to support HPC users with their research activities. Over time, these software packages, each with multiple versions, with each version potentially built with different compilers, grows and it becomes a mountainous task for the support staff to maintain/support these packages.

Building, maintaining, and supporting a large software stack on these HPC systems is a demanding task. Historically, many separate and diverse monitoring tools have been used within supercomputing centers to address the diverse needs of end-users, system administrator, and center directors, a few of them are discussed in section 2.2.

Figure 1.1: The Oak Ridge Leadership Computing Facility (OLCF) is home to Titan, the world's most powerful supercomputer for open science with a theoretical peak performance exceeding 20 petaflops. Image courtesy Oak Ridge National Laboratory.

The tool here in discussion is XALT [2] which is designed to track linkage and execution information for applications that are compiled and executed on any Linux cluster, workstation, or high-end supercomputer. XALT monitors and tracks individual code executions, which would in turn help get the attention of support staff and/or deliver alerts to users regarding the basic causes of problems preventing their jobs from running, and collect metrics that improve training, documentation, and outreach programs.

The work presented in this thesis focuses on leveraging our understanding of an individual user's software needs and help different stakeholders conduct business in a more efficient, effective, and systematic way. We aim at challenges and opportunities of developing a prototype to track function call resolved by external libraries along with implementing the XALT web portal.

1.1 Motivation

HPC systems are rapidly evolving and are complex in nature. The cost associated with operating and maintaining these systems is high, and there is an urgent need to utilize these systems as effectively as possible. It is essential to measure and monitor the software usage and forecast needs (if possible) of the packages and libraries used on HPC systems. The motivation of this works comes from the idea of implementing a web-interface to easily mine data collected by XALT to get a better understanding of various metrics, reports, and use cases. Furthermore, these reports and metrics can be used to highlight the products researchers need and do not need, and alert users and support staff to the root causes of software configuration issues as soon as the problem occurs. It is also beneficial for centers to have access to a web portal, which could give direct access to real-time data. This in turn enables not only support staff but users (often without a compute science background) to look into their work history along with a host of detailed information without going through the pain of writing scripts manually.

1.2 Thesis Outline

Chapter 2 presents background research in why we selected XALT, and other related work done in this area, including XALT's predecessors and various other tools. Chapter 3 presents the design, methodology, and implementation of the function tracking functionality and web portal. Next, we present various reports and metrics generated using XALT data, and discuss use cases which were part of this study in Chapter 4. Finally, Chapter 5 discusses conclusions and future work.

Chapter 2

Background and Related Work

2.1 Background

XALT is designed to track linkage and execution information. It collects accurate and continuous data on every job and store that in a database; all the data collection is transparent to the users. XALT not only tracks static, shared, and dynamic libraries but also detects function calls that need to be resolved by external libraries. In this section we briefly discuss XALT predecessors ALTD [3] and Lariat [4] and other open source and commercial resource usage monitoring tools.There are a few approaches that are related, however they were designed for other purposes and as such are not good solutions as discussed in 2.2.

2.1.1 XALT Overview

XALT is a lightweight solution with essentially no overhead at compilation time and runtime [5]. It was designed to work seamlessly on any cluster, workstation, or high-end computer. XALT intercepts both the GNU linker (ld) at link time and the code launcher (e.g. aprun, mpirun, srun, or ibrun) at runtime as shown in Figure 2.1. XALT not only tracks static, shared, and dynamically linked libraries but also function calls that need to be resolved by external libraries. The wrapper for the linker

(ld) intercepts the user link line and parses the command line, storing the results in the JSON [6] file. At the same time an ELF section header (Figure 2.2) is inserted into the user's code. The job specific environment variables from the batch system and dynamic libraries loaded during the runtime are detected by the job launcher wrapper and stored in another JSON file.

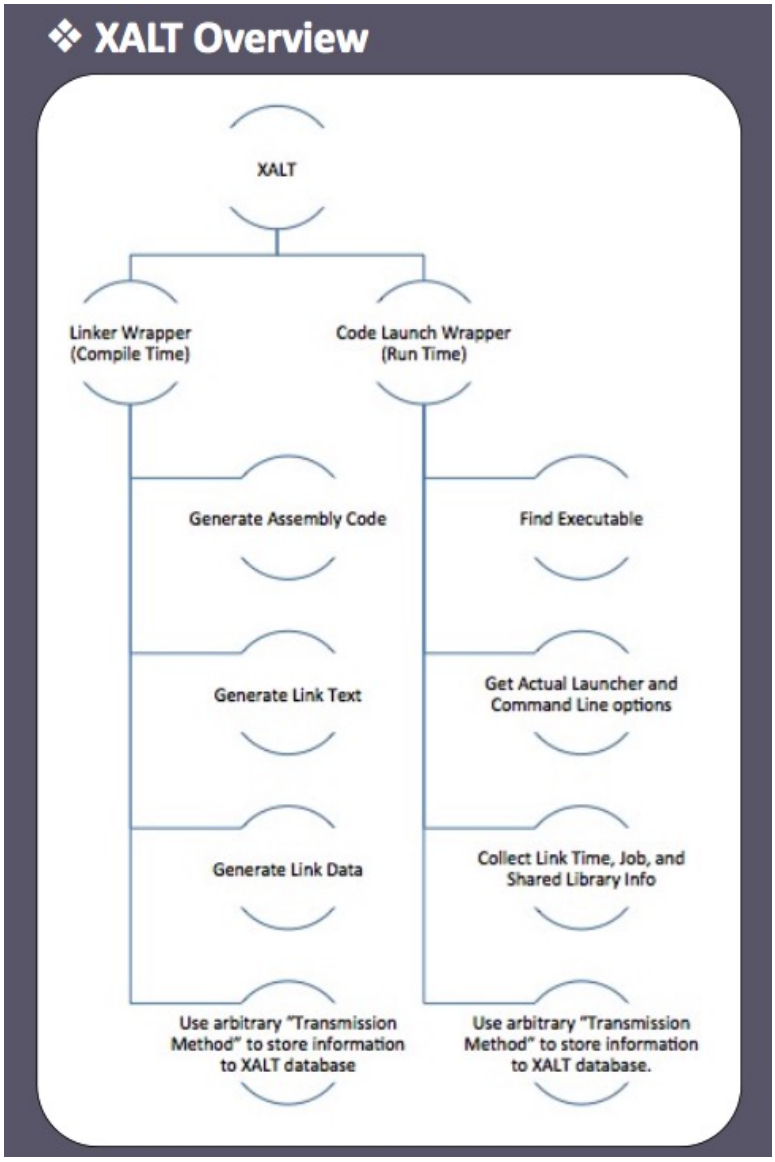


Figure 2.1: XALT linker and code launcher wrappers that intercept tracking information at compile time and run time, respectively.

```
.section .xalt
.asciz "XALT_Link_Info"

.byte 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.asciz "<XALT_Version>%%%%"
.asciz "<Build.Syshost>%%darter%%"
.asciz "<Build.compiler>%%driver.cc%%"
.asciz"<Build.OS>%%Linux_%_%_3.0.101-0.29-default%%"
.asciz "<Build.User>%%kagrawal%%"
.asciz "<Build.UUID>%%bd97b98b-2169-416e-85c1-762be8846dd2%%"
.asciz "<Build.Year>%%2014%%"
.asciz "<Build.date>%%Fri_%_%_Jul_%_%__%_%_4_%_%_13:37:01_%_%_2014%%"
.asciz "<Build.Epoch>%%1406914621.1%%"
.byte 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
.asciz "XALT_Link_Info_End"
```

Figure 2.2: ELF section header with fields necessary to accurately track executable in the jobs table back to the correct machine, the user who built it, and the machine on which it was built.

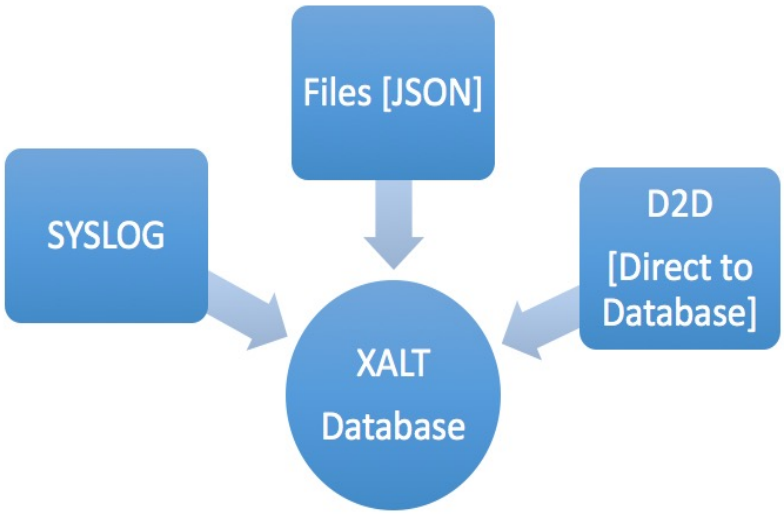


Figure 2.3: XALT transmission methods.

These JSON files are then stored in a database by running a script or by one of the built-in transmission methods deployed by the given site. XALT can transmit data in one of the three ways to a database (Figure 2.3): (1) write to an intermediate JSON file and parse later, (2) store directly in the database, or (3) write to *SYSLOG* and parse later. The idea behind having three different transmission methods is to let each site choose for themselves what works best for their environment based on their preferred file system, and database accesses patterns, and security policies.

2.2 Related Work

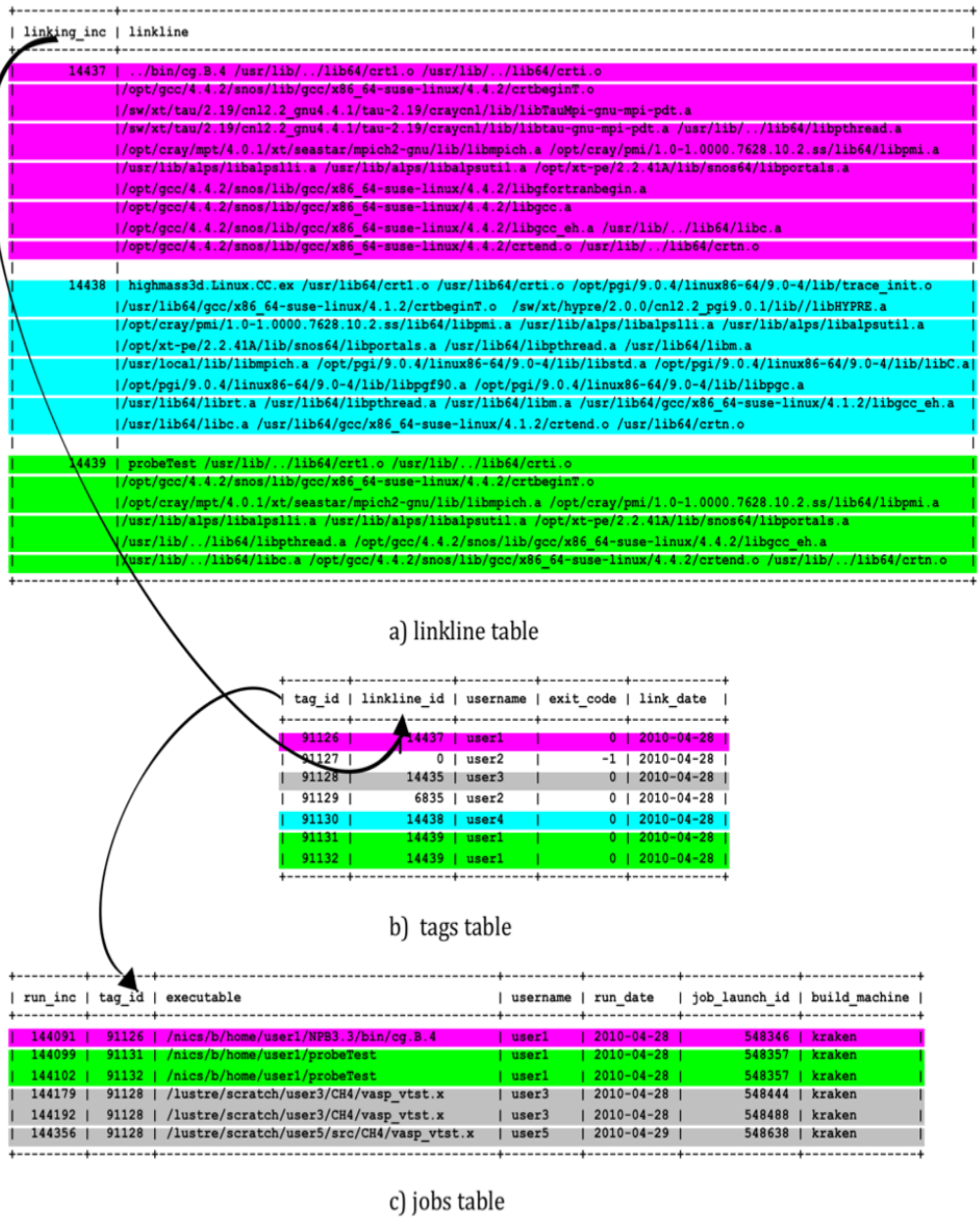
2.2.1 XALT Predecessors (ALTD and Lariat)

Automatic Library Tracking Database

ALTD is able to track both static and shared libraries; however, it does not track function calls which the latest version of XALT does [5]. ALTD intercepts the job launcher as a secondary measure to track usage by counting how many times an executable is run and then how many times the libraries (at linktime) are used [3]. However, libraries that are loaded and unloaded at runtime, such as dynamically linked libraries, are not tracked [7]. In addition, only a limited set of counters are captured, as show in Figure 2.4.

Lariat

Lariat [4] collects a set of details about each job that includes executable names for parallel jobs, their working directories, size, creation date, and SHA1 hash [8], and environment modules that they may employ [9]. However, Lariat is incapable of tracking static and shared libraries. XALT is a single infrastructure that provides the best of both successful predecessors. XALT incorporates all of their capabilities and adds more.



linking_inc	linkline
14437	../bin/cg.B.4 /usr/lib/../lib64/crt1.o /usr/lib/../lib64/crti.o /opt/gcc/4.4.2/snos/lib/gcc/x86_64-suse-linux/4.4.2/crtbeginT.o /sw/xt/tau/2.19/cnl2.2_gnu4.4.1/tau-2.19/craycnl/lib/libTauMpi-gnu-mpi-pdt.a /sw/xt/tau/2.19/cnl2.2_gnu4.4.1/tau-2.19/craycnl/lib/libtau-gnu-mpi-pdt.a /usr/lib/../lib64/libpthread.a /opt/cray/mpt/4.0.1/xt/seastar/mpich2-gnu/lib/libmpich.a /opt/cray/pmi/1.0-1.0000.7628.10.2.ss/lib64/libpmi.a /usr/lib/alps/libalpslli.a /usr/lib/alps/libalpsutil.a /opt/xt-pe/2.2.41A/lib/snos64/libportals.a /opt/gcc/4.4.2/snos/lib/gcc/x86_64-suse-linux/4.4.2/libgfortranbegin.a /opt/gcc/4.4.2/snos/lib/gcc/x86_64-suse-linux/4.4.2/libgcc.a /opt/gcc/4.4.2/snos/lib/gcc/x86_64-suse-linux/4.4.2/libgcc_eh.a /usr/lib/../lib64/libc.a /opt/gcc/4.4.2/snos/lib/gcc/x86_64-suse-linux/4.4.2/crtend.o /usr/lib/../lib64/crtn.o
14438	highmass3d.Linux.CC.ex /usr/lib64/crt1.o /usr/lib64/crti.o /opt/pgi/9.0.4/linux86-64/9.0-4/lib/trace_init.o /usr/lib64/gcc/x86_64-suse-linux/4.1.2/crtbeginT.o /sw/xt/hypre/2.0.0/cnl2.2_pgi9.0.1/lib//libHYPRE.a /opt/cray/pmi/1.0-1.0000.7628.10.2.ss/lib64/libpmi.a /usr/lib/alps/libalpslli.a /usr/lib/alps/libalpsutil.a /opt/xt-pe/2.2.41A/lib/snos64/libportals.a /usr/lib64/libpthread.a /usr/lib64/libm.a /usr/local/lib/libmpich.a /opt/pgi/9.0.4/linux86-64/9.0-4/lib/libstd.a /opt/pgi/9.0.4/linux86-64/9.0-4/lib/libC.a /opt/pgi/9.0.4/linux86-64/9.0-4/lib/libpgf90.a /opt/pgi/9.0.4/linux86-64/9.0-4/lib/libpgc.a /usr/lib64/librt.a /usr/lib64/libpthread.a /usr/lib64/libm.a /usr/lib64/gcc/x86_64-suse-linux/4.1.2/libgcc_eh.a /usr/lib64/libc.a /usr/lib64/gcc/x86_64-suse-linux/4.1.2/crtend.o /usr/lib64/crtn.o
14439	probeTest /usr/lib/../lib64/crt1.o /usr/lib/../lib64/crti.o /opt/gcc/4.4.2/snos/lib/gcc/x86_64-suse-linux/4.4.2/crtbeginT.o /opt/cray/mpt/4.0.1/xt/seastar/mpich2-gnu/lib/libmpich.a /opt/cray/pmi/1.0-1.0000.7628.10.2.ss/lib64/libpmi.a /usr/lib/alps/libalpslli.a /usr/lib/alps/libalpsutil.a /opt/xt-pe/2.2.41A/lib/snos64/libportals.a /usr/lib/../lib64/libpthread.a /opt/gcc/4.4.2/snos/lib/gcc/x86_64-suse-linux/4.4.2/libgcc_eh.a /usr/lib/../lib64/libc.a /opt/gcc/4.4.2/snos/lib/gcc/x86_64-suse-linux/4.4.2/crtend.o /usr/lib/../lib64/crtn.o

a) linkline table

tag_id	linkline_id	username	exit_code	link_date
91126	14437	user1	0	2010-04-28
91127	0	user2	-1	2010-04-28
91128	14435	user3	0	2010-04-28
91129	6835	user2	0	2010-04-28
91130	14438	user4	0	2010-04-28
91131	14439	user1	0	2010-04-28
91132	14439	user1	0	2010-04-28

b) tags table

run_inc	tag_id	executable	username	run_date	job_launch_id	build_machine
144091	91126	/nics/b/home/user1/NPB3.3/bin/cg.B.4	user1	2010-04-28	548346	kraken
144099	91131	/nics/b/home/user1/probeTest	user1	2010-04-28	548357	kraken
144102	91132	/nics/b/home/user1/probeTest	user1	2010-04-28	548357	kraken
144179	91128	/lustre/scratch/user3/CH4/vasp_vtst.x	user3	2010-04-28	548444	kraken
144192	91128	/lustre/scratch/user3/CH4/vasp_vtst.x	user3	2010-04-28	548488	kraken
144356	91128	/lustre/scratch/user5/src/CH4/vasp_vtst.x	user5	2010-04-29	548638	kraken

c) jobs table

Figure 2.4: ALTD database tables.

Reports generated by ALTD

XALT supports all existing reports generated by its predecessors ALTD and Lariat. Table 2.1 shows the number of compilations performed with the various compilers available on the different systems, and Table 2.2 shows library usage on Kraken. The

data corresponds to a one-year period (Jan-Dec' 2011) and was part of data collection performed at NICS in 2012 [10]. The reports were generated from the scripts written manually which again requires a great deal of understanding of how the database is structured and how to run these scripts. In addition, the scripts and the knowledge of extracting these reports may be lost if no care has been taken in assuring a centralized repository. This might add up to an overhead to the centers that are using these tools.

Table 2.1: Usage of Compilers (Number of Instances)

Compiler	Kraken	Jaguar	Rosa
GNU	26689	70854	9407
PGI	51154	132345	6116
Intel	6321	55182	1729
CCE	69	343	1415
Pathscale	14	1486	389

Table 2.2: Library Usage Ranked by Number of Instances and Number of Users

Library	Instances	Users	Library-version	Instances	Users
Libsci	42271	291	libsci/10.5.02	29787	220
atlas	35954	8	fftw/3.2.2.1	15987	128
fftw	24494	235	xt-libsci/10.4.5	12167	169
acml	3537	59	fftw/2.1.5	3710	64
petsc	2460	20	acml/4.4.0/	3088	39
sprng	1745	13	sprng/2.0b/	1739	12
arpack	1721	11	petsc/3.1.05	1571	13
tspl	1517	14	arpack/2008	1543	1
gsl	1451	48	tpsl/1.0.0/	1517	1
fftpack	1317	35	gsl/1.14	1063	39

2.2.2 Profiling and tracing tools

Profiling and tracing tools such as IPM (Integrated Performance Monitoring) [11] provides a performance profile on batch jobs while maintaining low overhead by using

a unique hashing approach that allows a fixed memory footprint and minimal CPU usage. XALT's aim is to track all libraries used at the link time and at execution time rather than track all the code executed. Likewise, tools such as TAU [12], Vampir [13],CrayPat [14] perform analysis for only one user and provide all the function calls in the application. These tools can be used to provide system-level information as a byproduct, nevertheless they are heavy-weight and introduce compile time and runtime overheads, which is highly undesirable [2].

2.2.3 Signature matching tools

Efforts [15] had been made to extract information (such as compilers and libraries used) using the open-source anti-virus package ClamAV [16]. These tools use a signature matching approach to automatically uncover the program build information. The implementation comprises of two tools: a signature generator and a signature scanner. The signature generator takes ELF files and automatically outputs ClamAV-formatted signature files. The signature scanner takes as input the signature files and the executable binaries and outputs all possible matches. However, this approach has a few drawbacks: (1) this approach takes a noticeable amount of time to extract signatures (e.g., 28 seconds to scan through a library of 210 MB size [15]); and (2) one has to be vigilant to make sure that all the signatures are up-to-date [2].

2.2.4 Performance management tools

Performance management tools such as TOPAS [17] (T3E Observative Performance Analysis System) monitor usage and performance of every parallel job executed on a CRAY T3E. The measurement consists of executing special code immediately before and after the execution of the actual program. The technique TOPAS employs is by modifying the UNICOS/mk compiler wrapper scripts to automatically link the TOPAS measurement module to every user application whenever it is recompiled. However, the way TOPAS calculates MFlop rates (Million Floating point operations

per second) are based on total wall clock execution time of the applications, i.e., it covers also input, output, initialization, wrap-up, and checkpointing phases of the program and not just inner loops or kernels. It is basically not an ideal way to calculate MFlop rates [17]. Tools such as RUR (Resource Utilization Reporting) developed by Cray collects statistics on how system are used by collecting data about the usage of a particular resource [18]. Cray provides plugins to support several sets of collected data. These plugins support multiple methods to output collected data. However, these tools are specific to Cray systems.

Other projects such as PAPI (Performance Application Programming Interface) provides a portable interface to the hardware performance counters available on all modern CPUs and other components like GPUs, network, and I/O systems. PAPI [19] produces performance data that originates from within the processing cores. It also pays more attention to the power consumption of the system. However, these counters are specific to the underlying architecture and require greater attention on how to use/build newly added components to existing frameworks to explore its usefulness.

2.2.5 Process accounting and resource managing tools

Resource managing tools such as TORQUE (Terascale Open-source Resource and Queue manager) [20] provides control over batch jobs and distributed computing resources. It is based on the original open source PBS (Portable Batch System) project which helps manages jobs that users submit to various queues on a computer system. This method of gathering information is known as *process accounting*. The information gathered usually is how many times the software has been used or CPU hours [2].

On the other hand, one can parse the output generated by environment commands supported by Linux such as *lastcomm* and retrieve summaries on software usage [21]. Tools developed for accurately identifying *masqueraders* such as NVision-PA [22]

produces text and graphic statistical summaries describing input *process accounting* logs.

Other open-source system such as SUPReMM which stands for Integrated HPC **S**ystems **U**sage and **P**erformance of **Re**sources **M**onitoring and **M**odeling [23] provides resource management capabilities to users and managers of HPC systems. SUPReMM was created by integrating data collected by the TACC Stats [9] with XDMoD [24] (**X**SE**D**E Metrics on Demand). Unlike conventional profilers, TACC Stats continuously collects and analyzes resources usage data for every job run on a system transparent to the user. This data can be used to automatically generate analyses and reports such as average cycles per instruction (CPI), average and peak memory usage, average and peak memory bandwidth usage. XDMoD [25] is an open source project which utilizes information obtained from the XSEDE central database, and further support the analyses and reporting of HPC job data from multiple sources, such as resource manager log files, and campus Lightweight Directory Access Protocol (LDAP) services. However, library usage, functions called, or applications called inside a program or script can not be traced by these tools. XALT provides in-depth details overcoming all these limitations.

Chapter 3

Methodology and Implementation

In this chapter we describe the approach we took to develop a prototype for tracking function calls. We then discuss requirements, design, and implementation of the XALT web portal.

3.1 Function Tracking

Some additional functionality that was proposed as part of the earlier releases of XALT was the need to track function calls resolved by the external libraries for the executable build. In this section we discuss components of the existing XALT [v0.5.4] package and what we changed to track function calls.

3.1.1 Database Changes

Two new tables are introduced to the existing set of tables in XALT's database named `join_link_function` and `xalt_function` see Figure 3.1. To avoid data redundancy, we made use of the `join_link_function` table, as the same function can be linked to different executables. The `xalt_function` is our core table that stores function names along with corresponding `func_id` which can again be traced back to the corresponding executable using `link_id` from the `join_link_function` table.

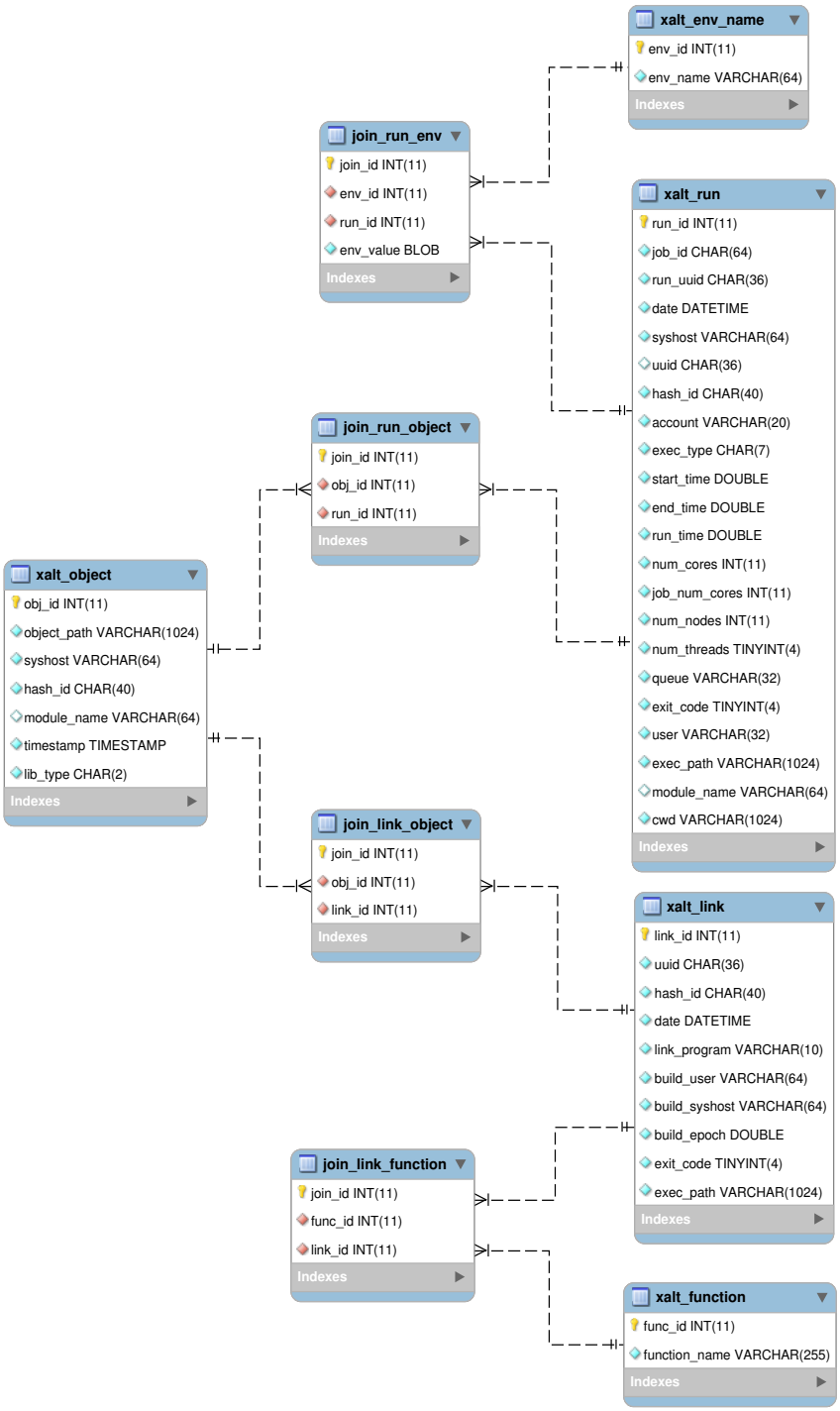


Figure 3.1: XALT Database Layout: Tables `join_link_function` and `xalt_function` are new tables added to the existing database to enable function tracking.

Changes have been made to the python script `createDB.py` that creates existing tables for XALT to support the new tables.

3.1.2 Create LibMap

As stated earlier we are interested in function calls resolved by the external libraries for the executable build; user-defined functions and auxiliary functions in the external libraries have no appeal to this work.

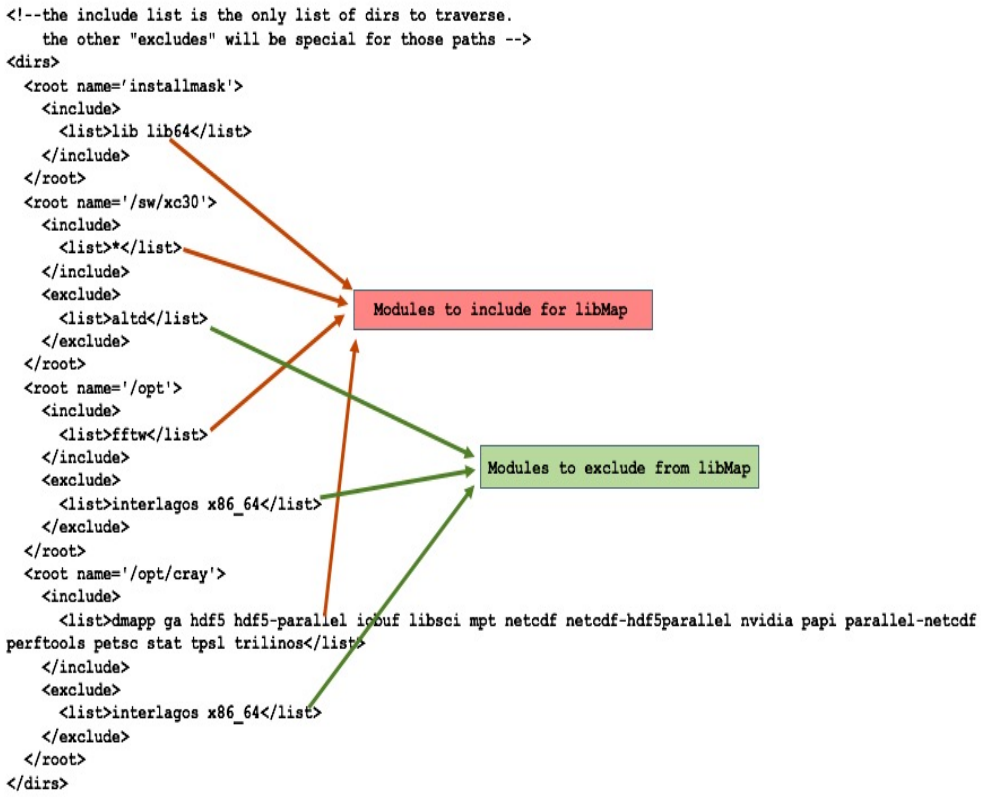


Figure 3.2: Directories: List of modules to be included/excluded to create LibMap defined in dirs.xml

To ensure that we track function calls of the libraries that are of interest we create a *libMap* of all the libraries listed. To achieve this we make use of the python script `xalt_create_libmap.py`. This script iterates through the file called `dirs.xml` (see Figure 3.2), which gives list of directories to traverse while creating the *libMap* JSON

file (see Figure 3.3). The `dirs.xml` files and *libMap* need to be created/updated per machine every time a new module or package is installed. This can be done as a part of software installation process, or run as a cron job every week. As part of improving the code base and removing dependencies we can integrate *libMap* with existing *reverseMap* [26] functionality of XALT.

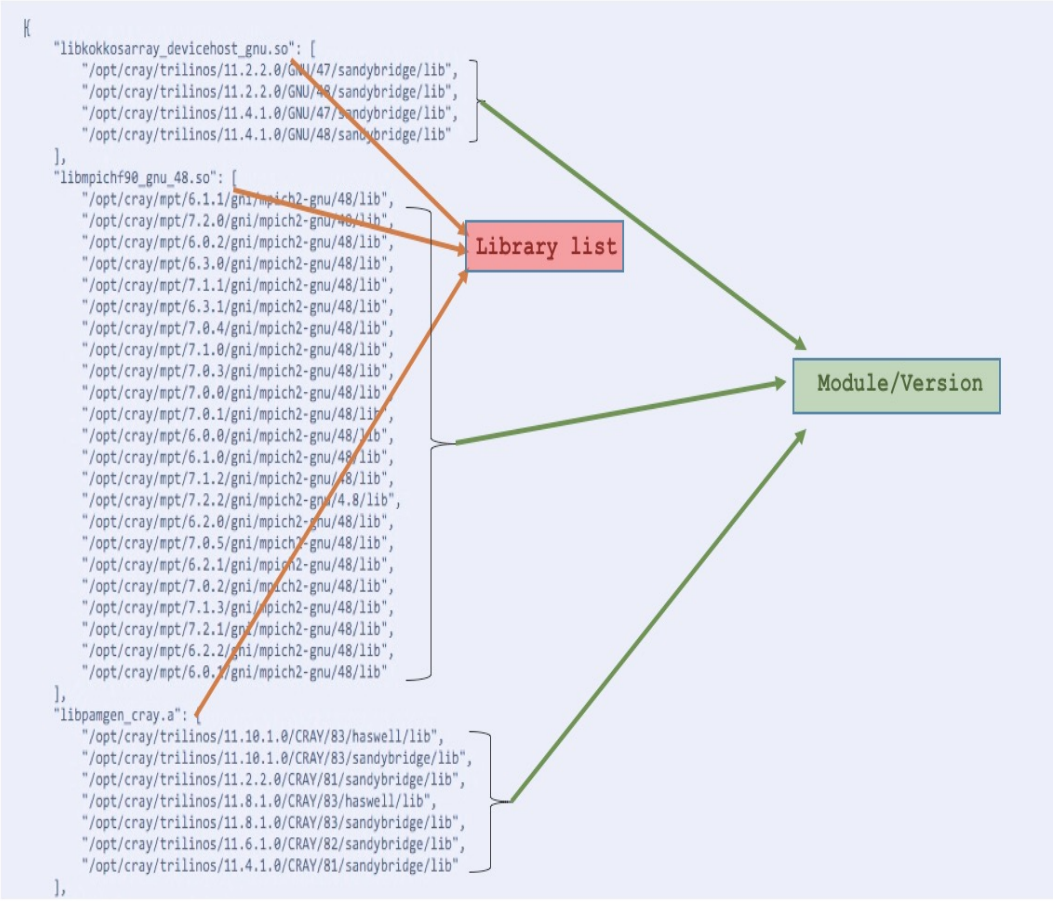


Figure 3.3: LibMap: A map of libraries used as a cross-reference for function tracking.

3.1.3 Modify Linker (ld) Wrapper

The linker (ld) wrapper which intercepts the user link line, has four main functions [2] as shown in Figure 2.1.

1. *Generate assembly code*

2. *Generate link text*

3. **Generate function list *(new)***

4. *Generate link data*

5. *Upload JSON file to XALT database*

We modify the existing process flow to get all function calls by adding a new step *Generate function list*. We strip the link line from all the libraries by calling the python script `xalt_strip_libs.py` which strips out libraries from a link line given a reference list of libraries in the *libmap* file produced in 3.1.2.

Once we have the list of functions we then pass this list along with the link data generated in step 4 to one of the transmission methods (see Figure 2.3). Depending upon the transmission method selected by the given center, the data generated would either be uploaded to JSON file, *SYSLOG*, or directly to the database.

3.2 XALT Web Portal

The principle requirement of web portal is to deliver library usage and job tracking data in a real-time fashion that eliminates the painstaking manual reporting employed by the centers. We do this using the XALT tool. The XALT web portal has been developed as a customizable and extensible tool to support the analysis and reporting of the data collected by XALT at a given high-performance computing center.

As shown in Figure 3.4, the web portal architecture is made up of two main components: the data is looked up from XALT database using a traditional MySQL, PHP software stack, and the user-facing portion of the web portal is developed using the free HTML5 Gentella template [27], including Google Charts [28] and Bootstrap 3 [29]. Bootstrap 3 is one of the most popular HTML, CSS, and JS frameworks for developing responsive mobile first projects on the web, whereas Google charts is a

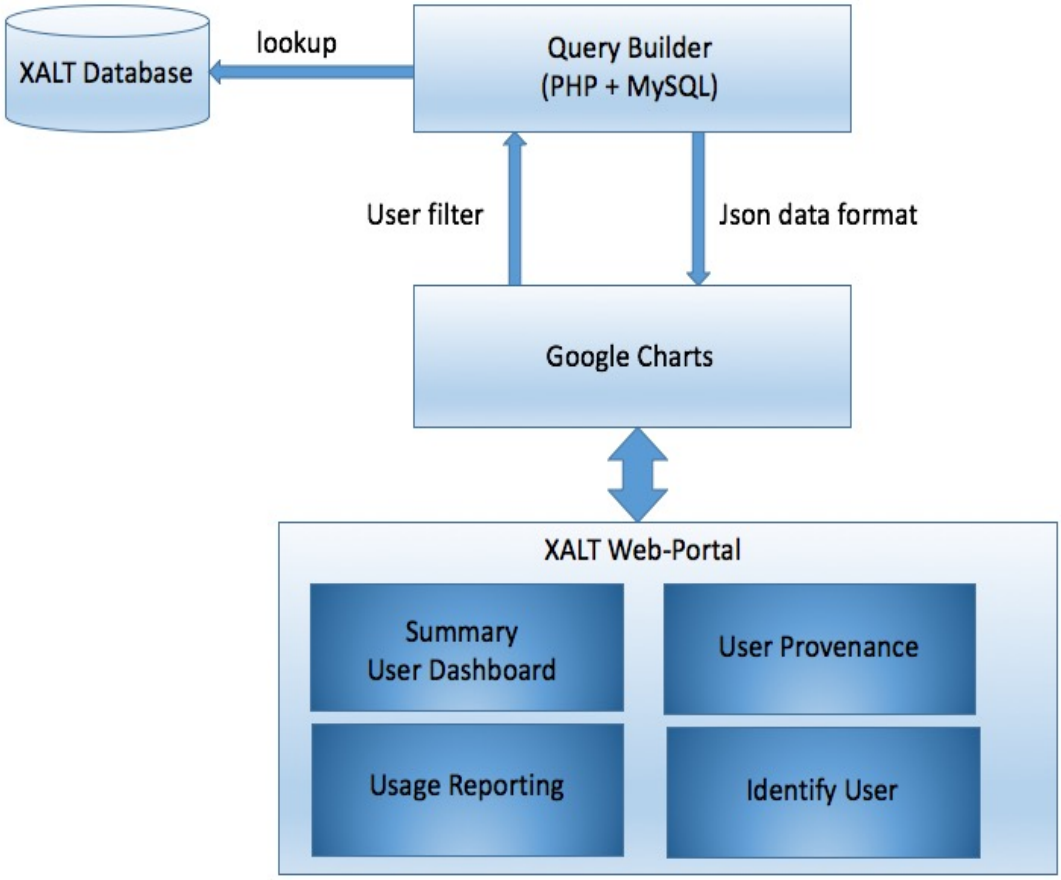


Figure 3.4: XALT web portal: It has two main compnent: XALT database lookup, and an interactive web-based charts/tables for viewing and analyzing data.

powerful, simple to use, and interactive client-side charting tool that allows users to view and interact with multiple data series directly from the browser.

Figure 3.5 shows a screen capture of a typical display page of the XALT web portal at NICS. The user interface for the web portal features a two panel navigation format. The left-panel provides users the ability to traverse through various tabs on the portal, and the right-panel shows various charts and reports customized to user needs.

When the web portal is launched, the user initially sees the Dashboard page, which contains a collection of charts and data presenting an overview of HPC center

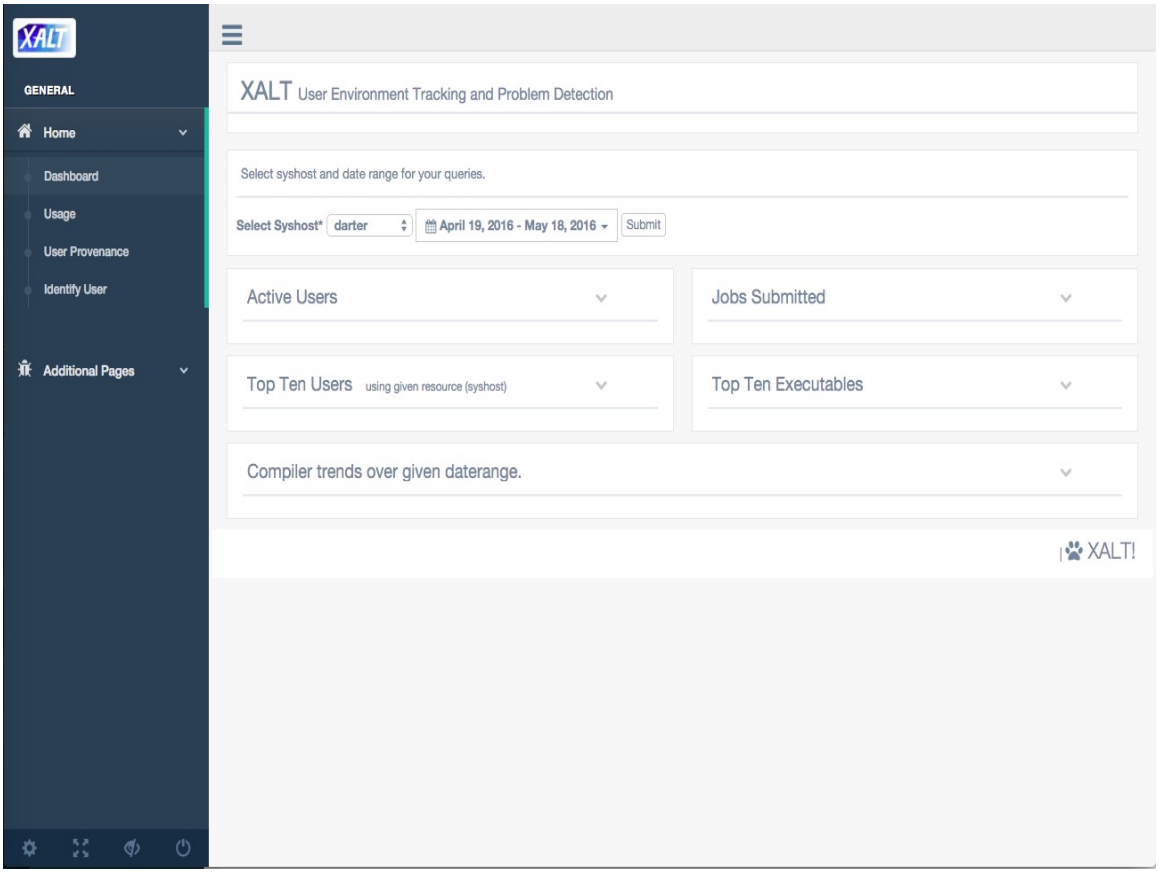


Figure 3.5: XALT Dashboard: Shows overview of the Center by showing different charts that can be generated for a given system and daterange.

operations. Users can select system host (syshost) and give a specific date range. Once users submits the request, five different charts are presented to the user; (1) Active Users; (2) Jobs Submitted; (3) Top Ten Users; (4) Top Ten Executables; and (5) Compiler trends over time.

The Usage tab lets users produce interactive table-charts. By selecting syshost and a date range users gets the top echelon table on Module usage and Link Program Usage. One can traverse through a list of table charts which in turns provides in-depth information such as; which users have linked to what all module versions?; how many times did user compiled the same executable?; whether there was a job run using a particular executable?; what libraries are linked to the given executable?; or

how was the run environment set?. More detailed analysis of this information and what use does it bring to the stakeholders is presented in section 4.

The User Provenance tab lets support staff/users get run and compile time details for a given user. Researchers can use the information mined by XALT's data to dive into one's work history. One starts with a list of executables corresponding to the job submission by the given user for the given syshost over a period of time. One can start navigating through the given list just by clicking on the executable. The list expands giving further details such as run date, link program, build user, build date, list of libraries linked to a unique executable, list of all job runs for a given executable, list of run environment variables with corresponding values, and finally list of libraries linked at the run time (if any). The later part of the details pertaining to run time information can be useful in tracking down undesirable behavior that may occur where the user comes across a scenario when run time loaded libraries are different from compile time. This may be a possibility when there is a change in run time environment variables which again can be traced and matched to compiles for specific days.

The Identify User tab gives access to support staff/users to identify users based on a given object path (whole or part of it) or by a given executable name. This information can be useful when the support staff wants to get a list of users using a deprecated library and/or wants to track how an executable was built.

Chapter 4

Results and Use Cases

The plots generated from the web portal demonstrates the extensible capability of the framework which can used to generate more reports as and when required with minimal effort. The most recent version of XALT web portal is currently available on Github at [30]. Here, one can find generic source bundle suitable for all operating systems. Extensive instructions detailing prerequisites and code maintenance is well documented using Doxygen [31] which will be helpful for future work as and when needed.

4.1 Results and discussions

4.1.1 XALT Dashboard tab

The XALT web portal provides a rich set of features accessible through an intuitive graphical interface. To optimize the UI responsiveness, the dashboard automatically presents data granularity to the user based on the time period being analyzed, with data aggregated by day, week, and month. All the results discussed below are part of the data collected by XALT at NICS. User can select any date range and any syshost for the given center for generating reports and metrics, however for our discussion we have selected the data for the year 2015 on Darter [32].

The dashboard tab presents a collection of charts and data presenting an overview of HPC center operations. Once the user submits the request, five different charts are presented to the user.

Active Users

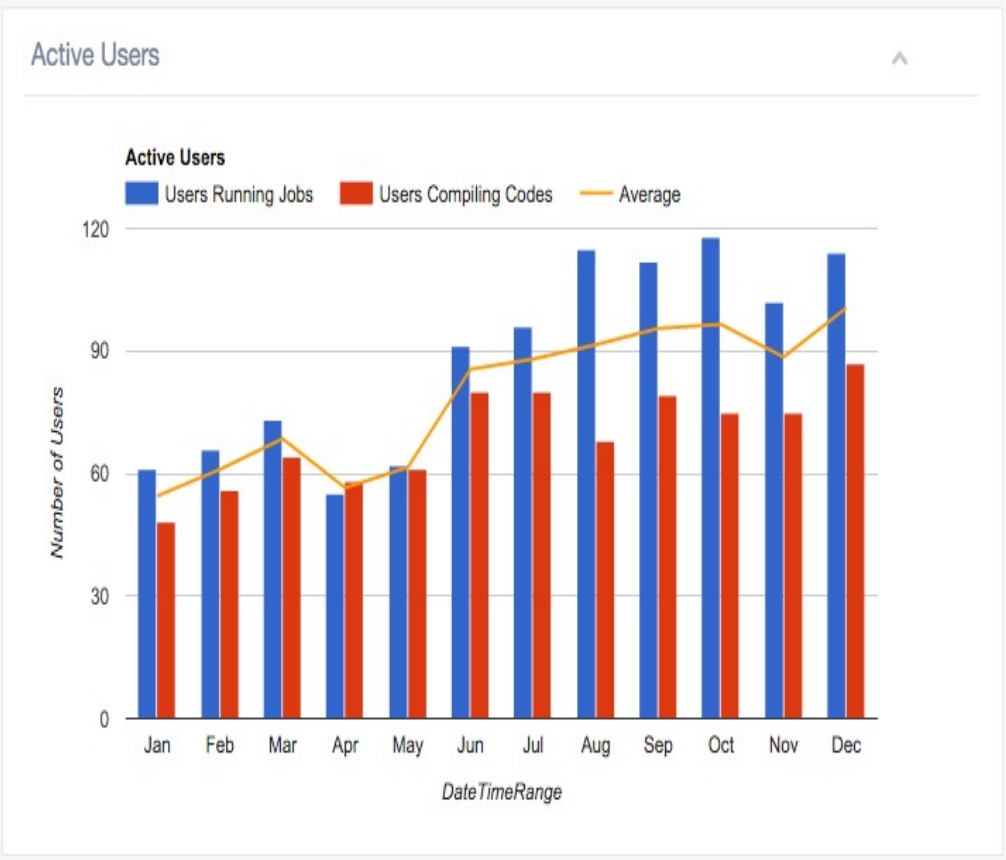


Figure 4.1: Active Users: The plot shows all distinct users over the year 2015, with blue bars indicating the number of distinct users running jobs and red bars indicating number of users who have built their code on Darter. The yellow line indicates an average of the two sets to understand the trends.

Figure 4.1 shows distinct active users based on users running jobs and users compiling their code on Darter for the year 2015. The data is aggregated by the date range. In this case the portal automatically aggregates data on a monthly basis. The plot provides an easier way to see a trend of increasing distinct user base both in

terms of users compiling their code and the ones who are running jobs on Darter. We can see an upward increase of user base from 54.5 in January' 15 to almost double 100.5 by the end of the year 2015.

Jobs Submitted

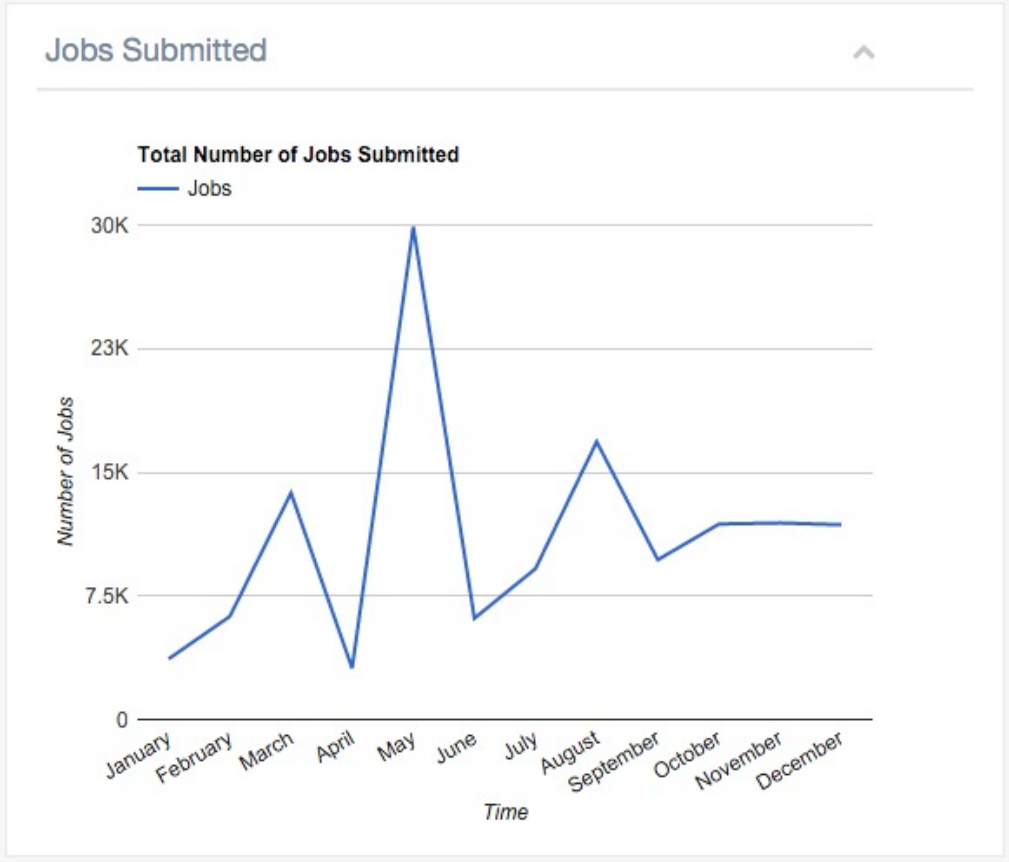


Figure 4.2: Jobs Submitted: The plot shows total number of jobs submitted during the year 2015, with a sharp increase in the numbers during May-2015 on Darter.

Figure 4.2 shows number of jobs submitted aggregated on monthly basis, May-2015 with 22.3% stands out to be the month with highest number of jobs submitted. However, there is no correlation between active users and number of jobs submitted as there is hardly any significant change in the number of active users using NICS resources during the same time period, i.e., from an average of 56.5 in April-2015 to 61.5 in May-2015.

It is difficult to draw any conclusion from the aggregated data as one tends to lose information on a granular level. A more holistic approach is needed to view the trend in its totality. However, one can always look up detailed information provided on other tabs of the web portal. In addition, one can find similar plots which are generated by an open source project called XDMoD [33].

Top Ten Executables and Top Ten Users

Figure 4.3: Top ten executables: The horizontal axis specify total number of jobs submitted (in log scale) for that executable, while the vertical axis indicates the core-hours. The ratio of these two represents average core-hours per submission. The size of the bubble represents the number of unique users that run that particular executable.

In Figure 4.3 we present data for executables run on Darter for year 2015. This kind of information helps a given center to report resource utilization by showing

what and how executables occupy the system resources. The plot shows the number of jobs, total core-hours in log scale, and number of unique users along with average core-hours (ratio of core-hours and number of jobs) for that executable. XALT records each build as an unique record. For generating below plot we have combined known executables residing in different file system paths into a common name.

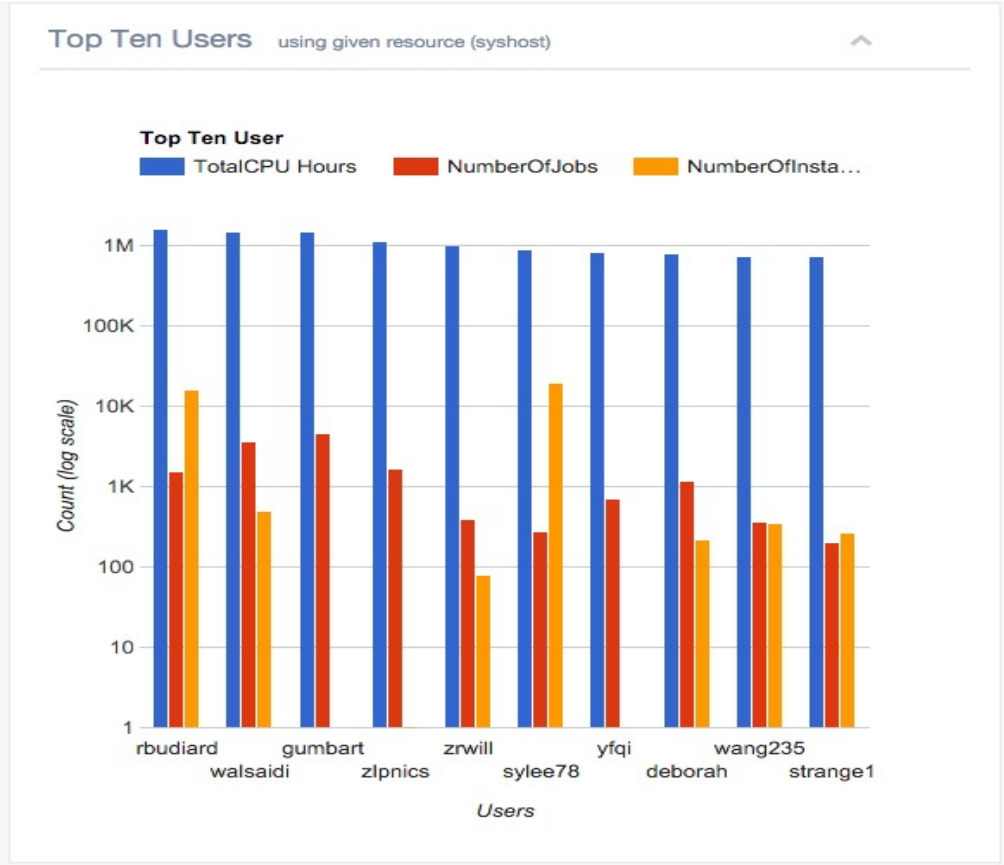


Figure 4.4: Top Ten Users: The plot shows top ten users using NICS resources (Darter for year 2015). Blue bars indicate total CPU hours, red bars indicate number of jobs users has submitted during the given time frame and orange bars indicate number of instance user has comipled the code on the given resource.

For example, the executable NWChem, software for electronic structure calculations and molecular dynamics simulations, can be built by different users, therefore having different directories. For this particular plot we have combined all NWChem executables into NWCHEM*. Four major applications (NAMD [34], VASP [35], LAMMPS [36], NWCHEM [37]) utilized a major chunk of resources on Darter. These

application fall under Material Science and Chemistry, given center can take proper measures on scheduling maintenance activity which can hamper normal functioning of the researchers using these applications.

Figure 4.4 shows the top ten users utilizing resources at NICS for year 2015. An interesting observation can be made by looking at the given plot is that: It is not necessary for a user to build a code in order to run it. Users *'gumbart', 'yfqi'* had never compiled a code on Darter, however they are the ones who are using resources extensively, showing up on the top ten list. A center can identify researchers and task them to set up a community support group for other researchers who are new to HPC environment and are working on similar applications. In addition, this community support group can come up with seminar series to increase HPC awareness among respective stakeholders. This alone can take some load off from a center's support staff.

Compiler Trends Over Time

Figure 4.5 shows a trend of all the 'link programs' on Darter for year 2015. The horizontal axis specify time range and vertical axis indicates number of times each link program calls the linker. The plot generated is an interactive plot, one can select/de-select a link program by clicking on legend shown on the right side of the plot, and generate the trend of the link program which interests him. This feature can be useful for reporting different compiler collections like GNU (g++, gcc, gfortran), compilers by Intel (ifort, icc, icpc) and the Cray family of compilers (ftn_driver, driver.cc, driver.CC).

As the plot suggests most of the link programs are mainly compilers which have followed a usual trend throughout the year. However, we observe an unusual trend for GNU compilers 'gfortran' for the month of July-2015 with #instances 19,823 and 'g++' for the month of Feb-2015 with #instance 11,642. As keeping these two compilers in the plot overshadows to what is happening with others we would for time being remove these two from our trend analysis.

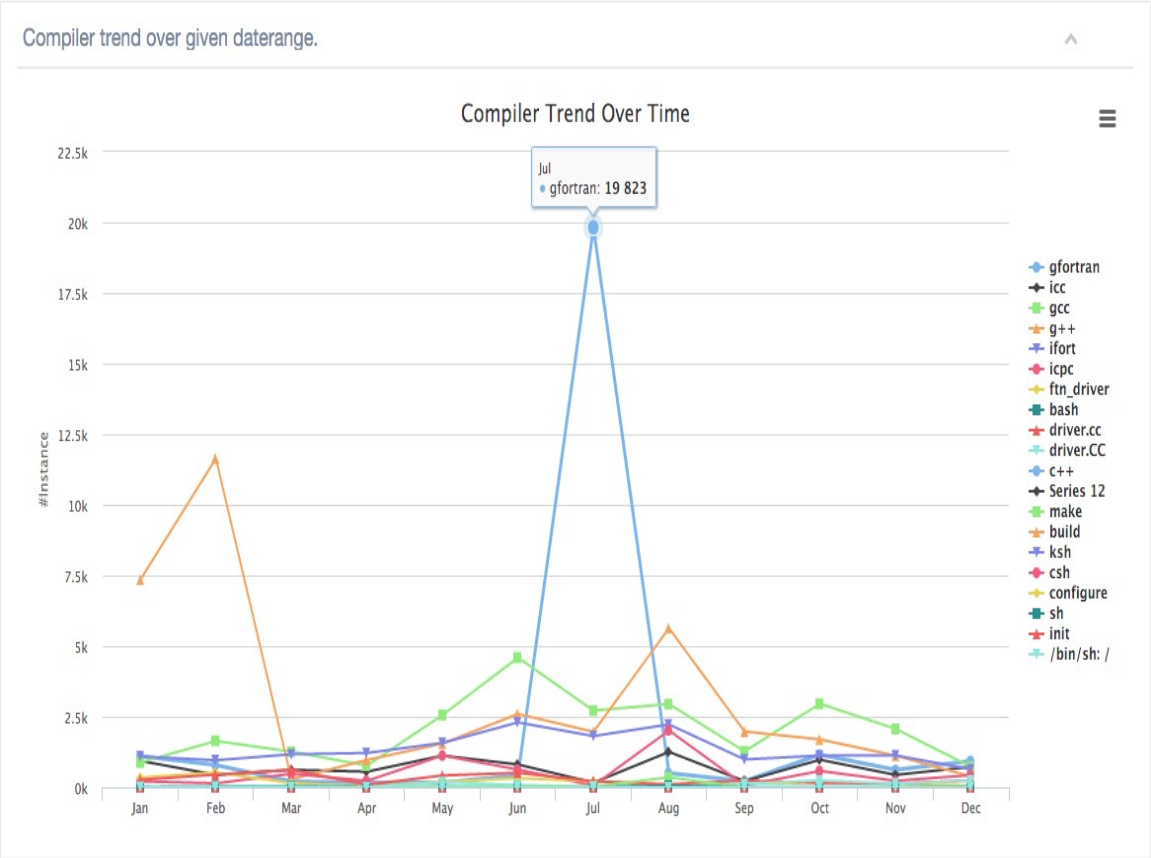


Figure 4.5: Compiler Trends Over Time: The plot shows a time trend of all the compilers on Darter for year 2015 based on number of times each link program calls the linker.

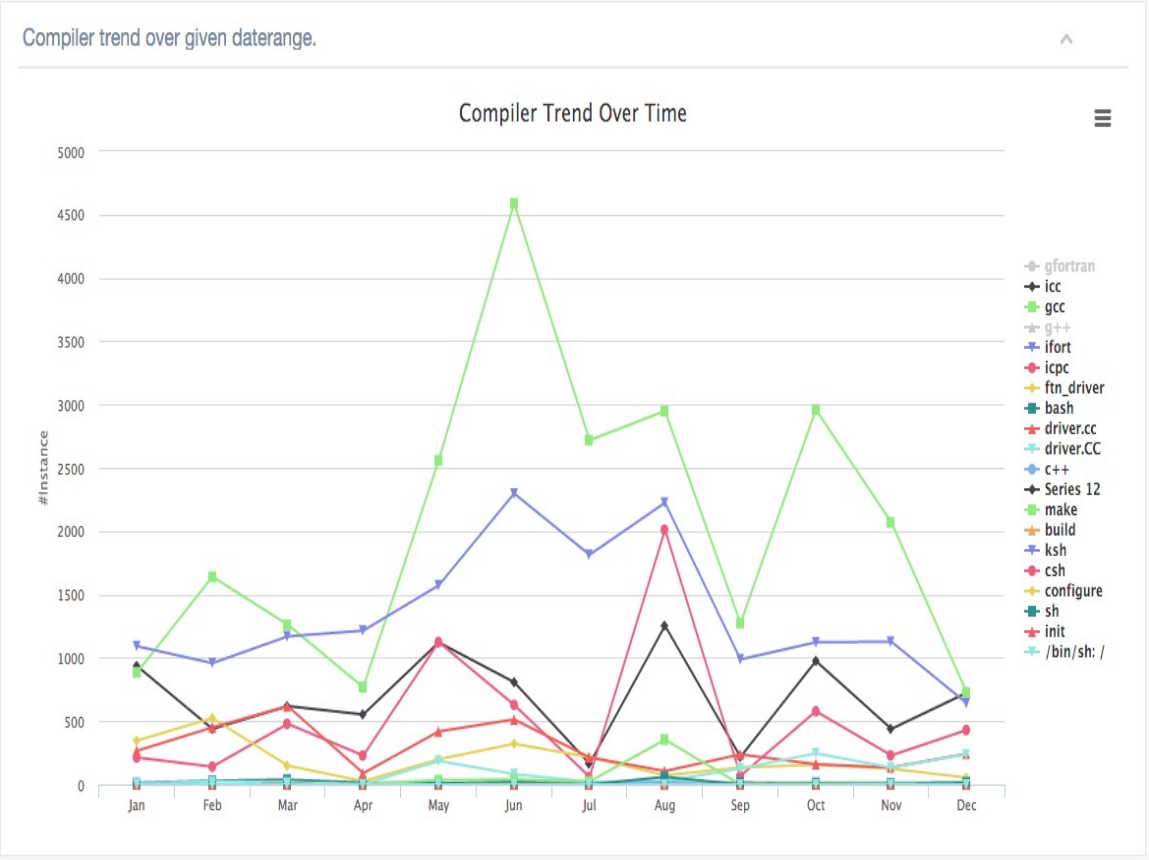
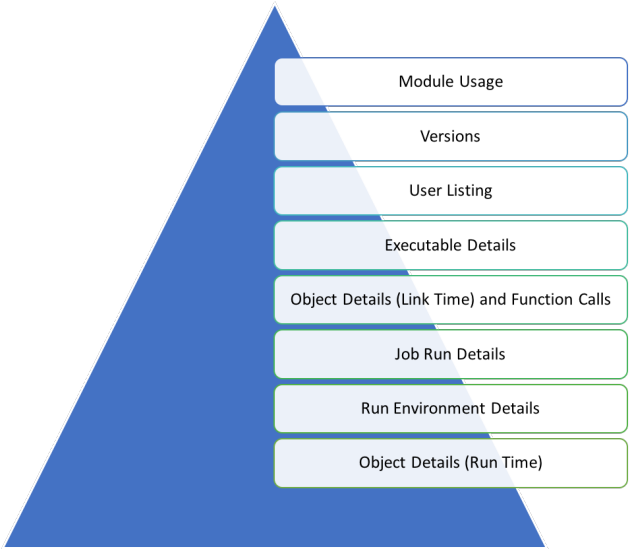


Figure 4.6: Compiler Trends Over Time: The two outliers (gfortran and g++) compiler are removed by de-selecting respective legend shown on the right side of the plot.

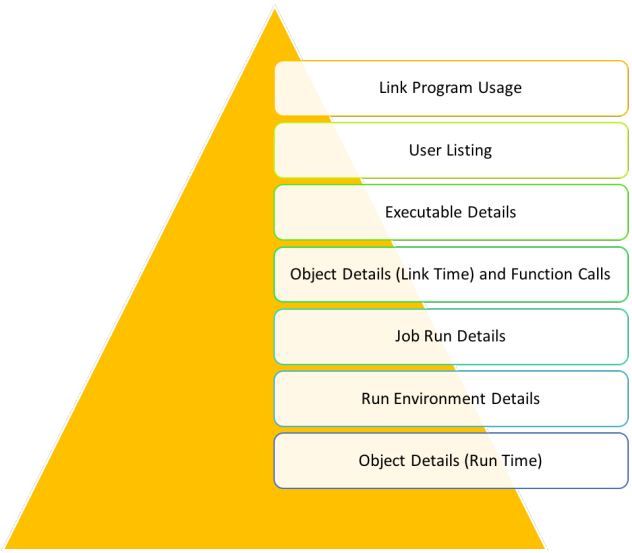
The support staff can gauge what is the cause of an unexpected hike in the number of instances for these two compilers by using the web portal's usage tab. One can dive deep into the wealth of information pulled from the XALT database. This discussion is continued in section 4.1.2 to show how support staff can benefit from readily available information.

4.1.2 XALT Usage Tab

The usage tab lets users produce plots showing modules usage and link program usage on the given syshost, date time range and number of records user wants to retrieve. This tab lets user explore stacks of information that is tracked by XALT, one can



(a) Module Usage.



(b) Link Program Usage

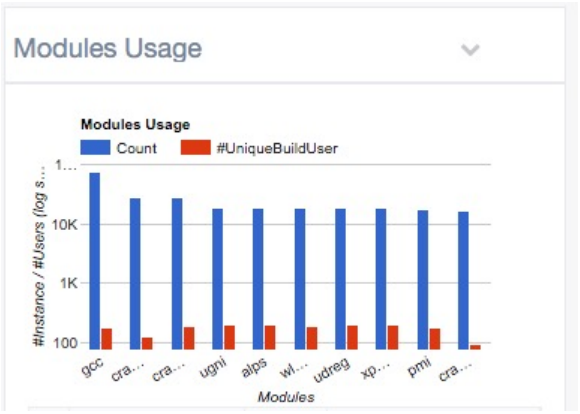
Figure 4.7: Various Listing on Usage Tab.

simply click on these interactive table chart for a given record to go to next detail listing.

Figure 4.7a and Figure 4.7b shows a pictorial representation of information that XALT tracks by traversing top down in the pyramid structure.

Module Usage

A module is defined as a group of most popular libraries which are provided by a center and/or vendor having an associated *modulefile* which is categorized using XALT's *ReverseMap* [5] functionality. In above plots, we have combined multiple versions of the same library to a single entry. The plot 4.8 focuses on, the number of time object was linked, and the number of unique users associated with those linkings. Figure 4.8a and 4.8b shows top ten modules usage on Darter for year 2015.



(a) Top Ten Module Usage: Blue bar shows number of time object was linked and red bar indicates number of unique users both in log scale.

	Modules	Count	#UniqueBuildUser
1	gcc	75190	179
2	craype-intel-knc	27955	125
3	cray-mpich	27346	185
4	ugni	19085	199
5	alps	19073	199
6	wlm_detect	18959	195
7	udreg	18561	198
8	xpmem	18242	197
9	pmi	17085	182
10	cray-libsci	16962	97

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

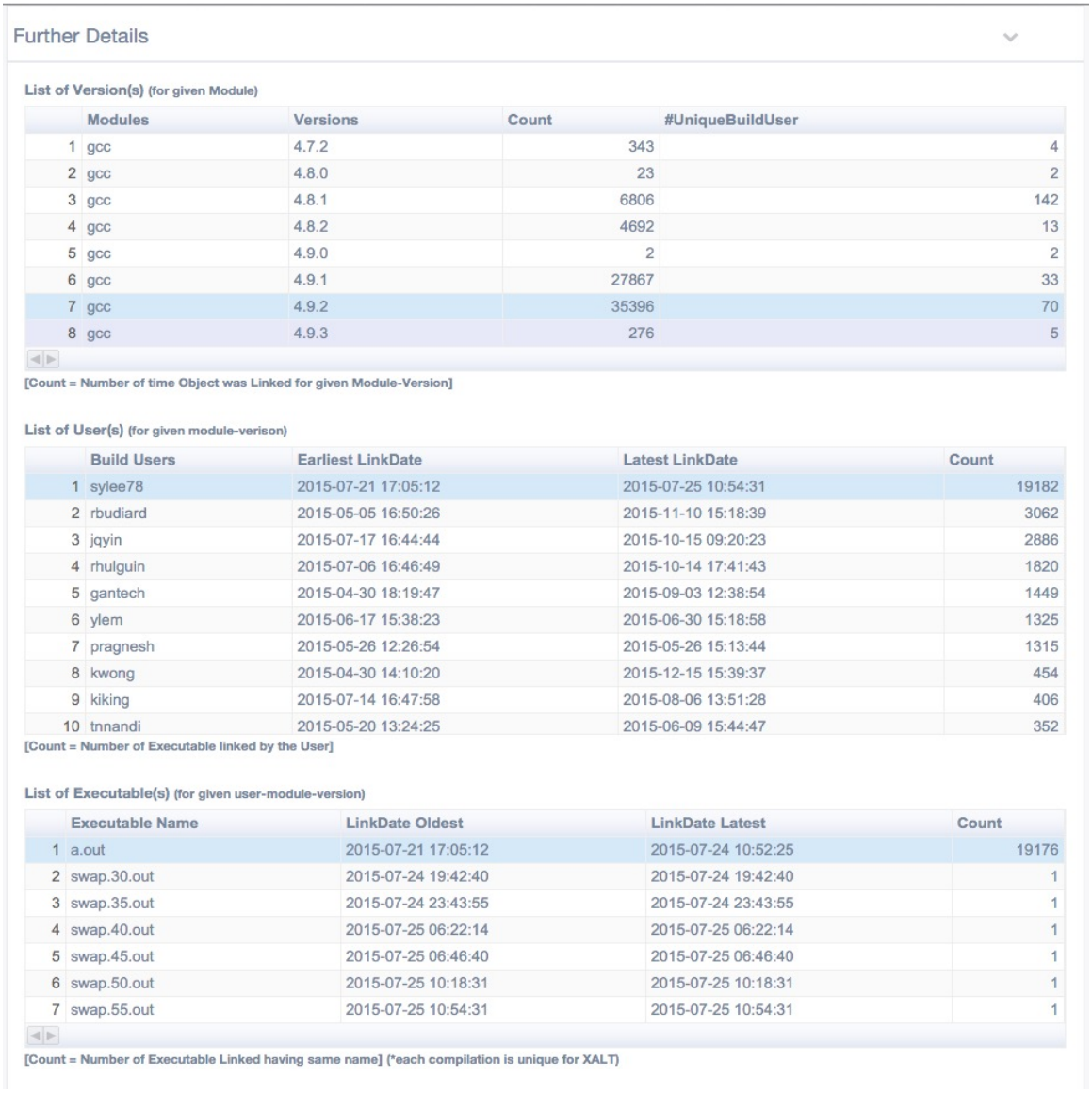
(b) Top Ten Module Table Chart: Sorted in descending order of #instances linked.

Figure 4.8: XALT Module Usage.

Moreover, Figure 4.8b shows same information in tabular format which is more functional as user can sort any given column just by clicking on the column header. By default, the table chart is sorted in descending order of number of times objects was linked. Observe that the gcc module is used thrice as much as the second most used module (cray-intel-knc). We can narrow down our lookup to address what might be the cause of this irregularity.

Figure 4.9 shows further details pertaining to the gcc module. Version 4.9.2 stands out to be the one contributing to the sudden hike, where in user 'sylee78' had linked an executable 'a.out' 19,182 times within the span of three days i.e., 21-July-2015 (Oldest link date) to 25-July-2015 (Latest link date). A center's support staff would

have this information handy and can make informed decisions when they decide to update/deprecate the given module (per se). They can get in touch with the users asking for a feedback on what sort of support is further needed to make their work easy.



Further Details

List of Version(s) (for given Module)

	Modules	Versions	Count	#UniqueBuildUser
1	gcc	4.7.2	343	4
2	gcc	4.8.0	23	2
3	gcc	4.8.1	6806	142
4	gcc	4.8.2	4692	13
5	gcc	4.9.0	2	2
6	gcc	4.9.1	27867	33
7	gcc	4.9.2	35396	70
8	gcc	4.9.3	276	5

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

List of User(s) (for given module-verison)

	Build Users	Earliest LinkDate	Latest LinkDate	Count
1	sylee78	2015-07-21 17:05:12	2015-07-25 10:54:31	19182
2	rbudiard	2015-05-05 16:50:26	2015-11-10 15:18:39	3062
3	jqyin	2015-07-17 16:44:44	2015-10-15 09:20:23	2886
4	rhulguin	2015-07-06 16:46:49	2015-10-14 17:41:43	1820
5	gantech	2015-04-30 18:19:47	2015-09-03 12:38:54	1449
6	ylem	2015-06-17 15:38:23	2015-06-30 15:18:58	1325
7	pragnesh	2015-05-26 12:26:54	2015-05-26 15:13:44	1315
8	kwong	2015-04-30 14:10:20	2015-12-15 15:39:37	454
9	kiking	2015-07-14 16:47:58	2015-08-06 13:51:28	406
10	tnnandi	2015-05-20 13:24:25	2015-06-09 15:44:47	352

[Count = Number of Executable linked by the User]

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

	Executable Name	LinkDate Oldest	LinkDate Latest	Count
1	a.out	2015-07-21 17:05:12	2015-07-24 10:52:25	19176
2	swap.30.out	2015-07-24 19:42:40	2015-07-24 19:42:40	1
3	swap.35.out	2015-07-24 23:43:55	2015-07-24 23:43:55	1
4	swap.40.out	2015-07-25 06:22:14	2015-07-25 06:22:14	1
5	swap.45.out	2015-07-25 06:46:40	2015-07-25 06:46:40	1
6	swap.50.out	2015-07-25 10:18:31	2015-07-25 10:18:31	1
7	swap.55.out	2015-07-25 10:54:31	2015-07-25 10:54:31	1

[Count = Number of Executable Linked having same name] (*each compilation is unique for XALT)

Figure 4.9: Further Details: List of versions for the given module, list of users using the particular module-version, list of executables build by the given users using given module-versions.

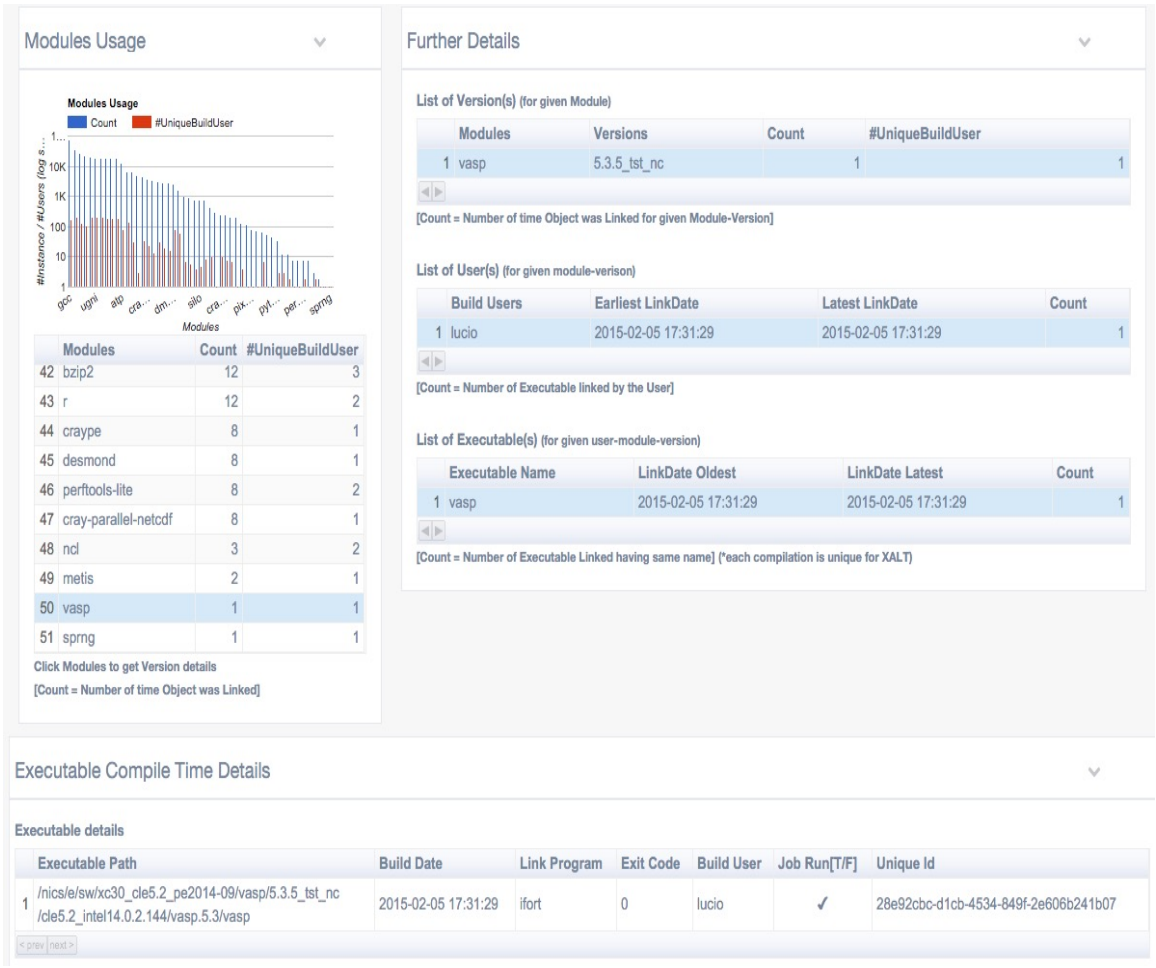
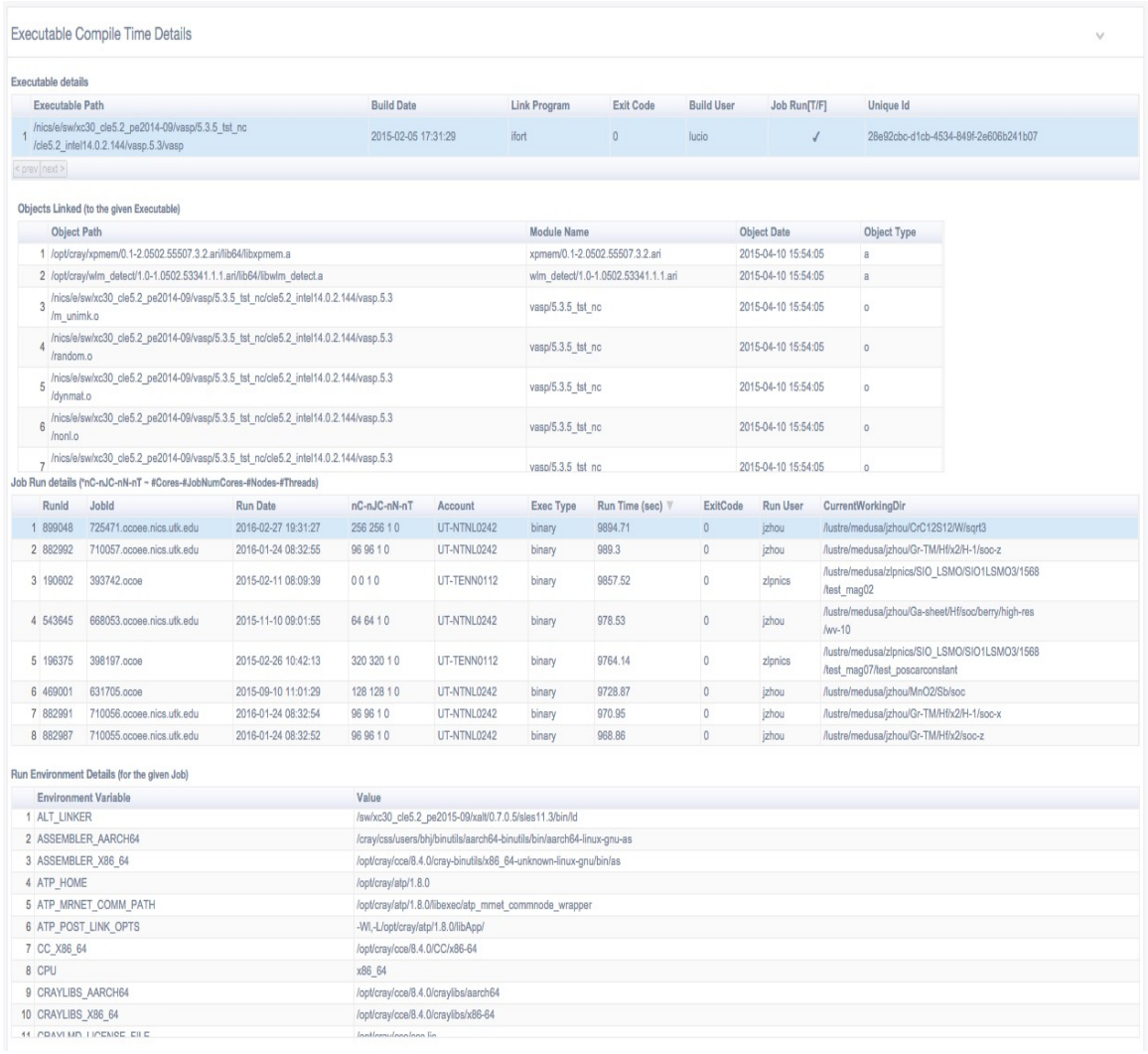


Figure 4.10: VASP Module Usage: As shown above only one executable is been built in year 2015.

Another interesting observation about module usage that came into light is: It is not always recommended to deprecate the modules which are least used i.e., there is hardly any build using VASP modules. For example, as shown in section 4.1.1 Figure 4.3 VASP [35] (a molecular dynamics package) is listed as one of the top ten executable on Darter for year 2015 utilizing around 4.5M total CPU hours with 30,305 number of jobs submitted, it is however one of the least (to be precise second last in the list of 51 modules) used module package on Darter for the year 2015 as shown in Figure 4.10.



Executable Compile Time Details

Executable details

	Executable Path	Build Date	Link Program	Exit Code	Build User	Job Run[T/F]	Unique Id
1	/nics/e/sw/xc30_cle5.2_pe2014-09/vasp/5.3.5_tst_nc /cle5.2_intel14.0.2.144/vasp.5.3/vasp	2015-02-05 17:31:29	ifort	0	lucio	✓	28e92cbc-d1cb-4534-849f-2e606b241b07

< prev next >

Objects Linked (to the given Executable)

	Object Path	Module Name	Object Date	Object Type
1	/opt/cray/xpmem/0.1-2.0502.55507.3.2.ari/lib64/libxpmem.a	xpmem/0.1-2.0502.55507.3.2.ari	2015-04-10 15:54:05	a
2	/opt/cray/wlm_detect/1.0-1.0502.53341.1.1.ari/lib64/libwlm_detect.a	wlm_detect/1.0-1.0502.53341.1.1.ari	2015-04-10 15:54:05	a
3	/nics/e/sw/xc30_cle5.2_pe2014-09/vasp/5.3.5_tst_nc/cle5.2_intel14.0.2.144/vasp.5.3 /m_unimk.o	vasp/5.3.5_tst_nc	2015-04-10 15:54:05	o
4	/nics/e/sw/xc30_cle5.2_pe2014-09/vasp/5.3.5_tst_nc/cle5.2_intel14.0.2.144/vasp.5.3 /random.o	vasp/5.3.5_tst_nc	2015-04-10 15:54:05	o
5	/nics/e/sw/xc30_cle5.2_pe2014-09/vasp/5.3.5_tst_nc/cle5.2_intel14.0.2.144/vasp.5.3 /dynmat.o	vasp/5.3.5_tst_nc	2015-04-10 15:54:05	o
6	/nics/e/sw/xc30_cle5.2_pe2014-09/vasp/5.3.5_tst_nc/cle5.2_intel14.0.2.144/vasp.5.3 /nonl.o	vasp/5.3.5_tst_nc	2015-04-10 15:54:05	o
7	/nics/e/sw/xc30_cle5.2_pe2014-09/vasp/5.3.5_tst_nc/cle5.2_intel14.0.2.144/vasp.5.3	vasp/5.3.5_tst_nc	2015-04-10 15:54:05	o

Job Run details (*nC-nJC-nN-nT ~ #Cores-#JobNumCores-#Nodes-#Threads)

	RunId	JobId	Run Date	nC-nJC-nN-nT	Account	Exec Type	Run Time (sec)	ExitCode	Run User	CurrentWorkingDir
1	899048	725471.ocoee.nics.utk.edu	2016-02-27 19:31:27	256 256 1 0	UT-NTNL0242	binary	9894.71	0	jzhou	/lustre/medusa/jzhou/CrC12S12/W/sqrt3
2	882992	710057.ocoee.nics.utk.edu	2016-01-24 08:32:55	96 96 1 0	UT-NTNL0242	binary	989.3	0	jzhou	/lustre/medusa/jzhou/Gr-TM/Hf/x2/H-1/soc-z
3	190602	393742.ocoe	2015-02-11 08:09:39	0 0 1 0	UT-TENN0112	binary	9857.52	0	zlpnics	/lustre/medusa/zlpnics/SIO_LSMO/SIO1LSMO3/1568 /test_mag02
4	543645	668053.ocoee.nics.utk.edu	2015-11-10 09:01:55	64 64 1 0	UT-NTNL0242	binary	978.53	0	jzhou	/lustre/medusa/jzhou/Ga-sheet/Hf/soc/berry/high-res /wv-10
5	196375	398197.ocoe	2015-02-26 10:42:13	320 320 1 0	UT-TENN0112	binary	9764.14	0	zlpnics	/lustre/medusa/zlpnics/SIO_LSMO/SIO1LSMO3/1568 /test_mag07/test_poscarconstant
6	469001	631705.ocoe	2015-09-10 11:01:29	128 128 1 0	UT-NTNL0242	binary	9728.87	0	jzhou	/lustre/medusa/jzhou/MnO2/Sb/soc
7	882991	710056.ocoee.nics.utk.edu	2016-01-24 08:32:54	96 96 1 0	UT-NTNL0242	binary	970.95	0	jzhou	/lustre/medusa/jzhou/Gr-TM/Hf/x2/H-1/soc-x
8	882987	710055.ocoee.nics.utk.edu	2016-01-24 08:32:52	96 96 1 0	UT-NTNL0242	binary	968.86	0	jzhou	/lustre/medusa/jzhou/Gr-TM/Hf/x2/soc-z

Run Environment Details (for the given Job)

	Environment Variable	Value
1	ALT_LINKER	/sw/xc30_cle5.2_pe2015-09/xalt/0.7.0.5/sles11.3/bin/ld
2	ASSEMBLER_AARCH64	/cray/css/users/bhj/binutils/aarch64-binutils/bin/aarch64-linux-gnu-as
3	ASSEMBLER_X86_64	/opt/cray/cce/8.4.0/cray-binutils/x86_64-unknown-linux-gnu/bin/as
4	ATP_HOME	/opt/cray/atp/1.8.0
5	ATP_MRNET_COMM_PATH	/opt/cray/atp/1.8.0/libexec/atp_mrnet_commnode_wrapper
6	ATP_POST_LINK_OPTS	-Wl,-L/opt/cray/atp/1.8.0/libApp/
7	CC_X86_64	/opt/cray/cce/8.4.0/CC/x86-64
8	CPU	x86_64
9	CRAYLIBS_AARCH64	/opt/cray/cce/8.4.0/craylibs/aarch64
10	CRAYLIBS_X86_64	/opt/cray/cce/8.4.0/craylibs/x86-64
11	CRAYLMD_LICENSE_FILE	/opt/cray/cce/ce.lic

Figure 4.11: Job Run Details for UUID '28e92cbc-d1cb-4534-849f-2e606b241b07'.

XALT tracks link time information by providing each build a unique universal ID (UUID), it seems only user 'lucio' had build vasp executable once on darter on 05-Feb-2015, also 'Job Run' = *True* indicates that jobs were submitted using this executable (having UUID = '28e92cbc-d1cb-4534-849f-2e606b241b07'). The portal allows users to click and select any record shown in that table and obtain additional information such as Objects Linked (at compile time and run time), Job Run Details, Function Calls, and Run Environment Details as shown in Figure 4.11.

One can sort the order of the displayed results just by clicking on the column header. For example: Job Run Details table is sorted in descending order of Run Time (sec). As one can see executable that user 'lucio' built with UUID '28e92cbc-d1cb-4534-849f-2e606b241b07' is been used (790 jobs submitted) by number of different users (almost 7 unique users) to conduct their research. This again would help support staff and other users to get in touch with the build user in case the executable fails for some unknown reason. Apart, from this users running the jobs using this executable can find other users and discuss ways to perform experiments in more informed manner which can help developing a support community/group for their research work.

Link Program Usage

Link Program usage follows the same hierarchy of displaying information in table structure format as that of module usage as shown in Figure 4.7b

Talking about irregularity in section 4.1.1 as shown in Figure 4.5. The compiler trend for 'g++' has seen an unusual hike for the first quarter of 2015 with 11,642 #instances and for 'gfortran' in the third quarter of 2015 with exceptionally high 19,823 #instances.

Figure 4.12 shows link program usage on Darter for a specific date time range, i.e., June-2015 to August-2015. As shown in the figure it turns out that user 'sylee78' built 'a.out' 19,175 times within the span of 3 days i.e., 21-July-2015 (Oldest Linkdate) and 25-July-2015 (Latest LinkDate), which seems to be the cause of the hike.

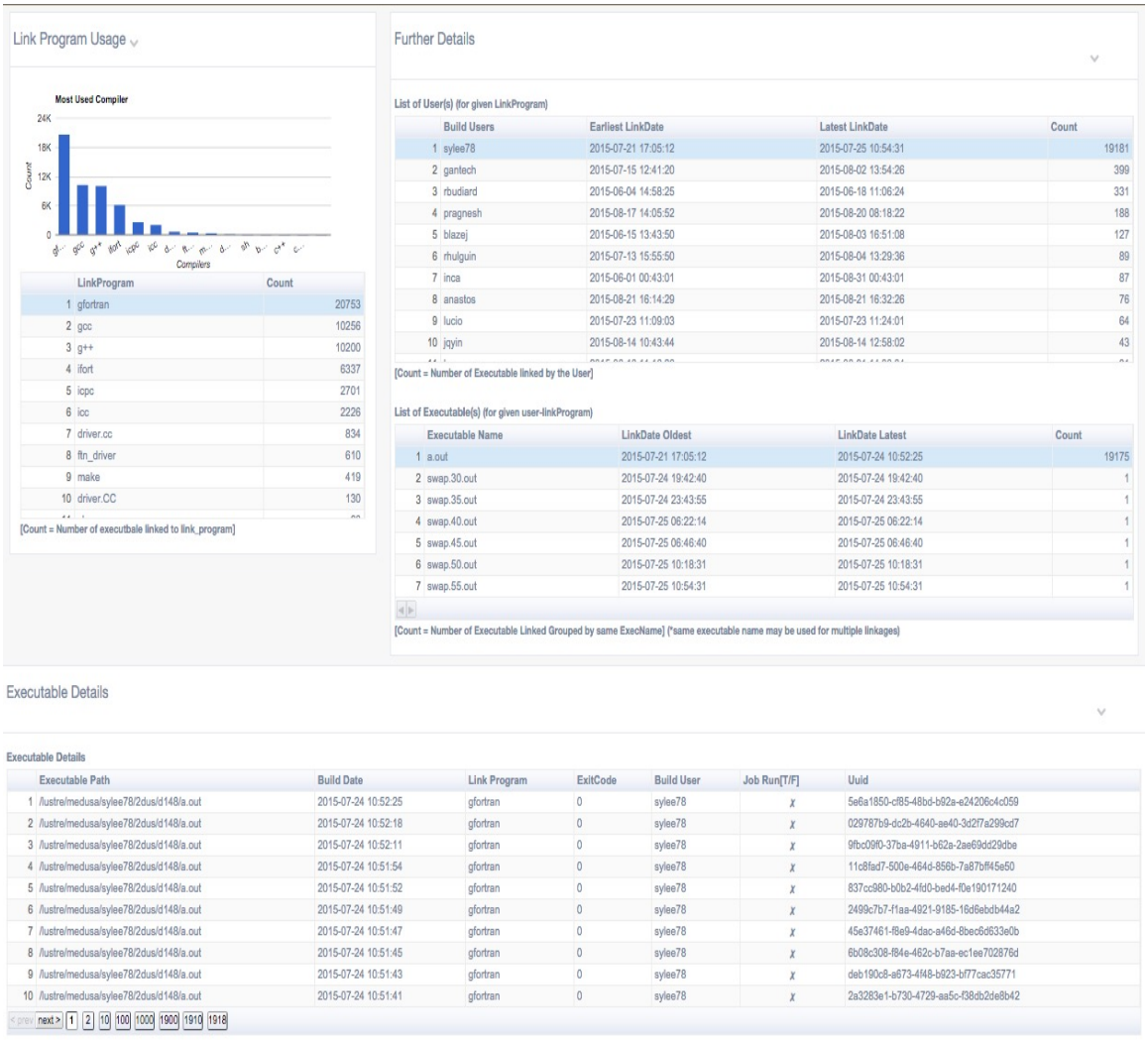


Figure 4.12: Link Program Usage on Darter for period Jun-Aug' 2015

We got a chance to get back to the user asking what was he trying to achieve as 'a.out' does not give any specific pointers. This user responded that he was trying to analyze the trajectories in the simulations for the replica-exchange umbrella sampling (REUS [38]). Similarly for compiler 'g++' Figure 4.13 the usage reports shows that 'g++' (GNU C++ compiler) was used almost five times more than the second most used compiler 'gcc' (GNU C compiler) for the same duration i.e., Jan-2015 to March-2015. A similar case can be seen here with user 'hirsh' running a simulation.

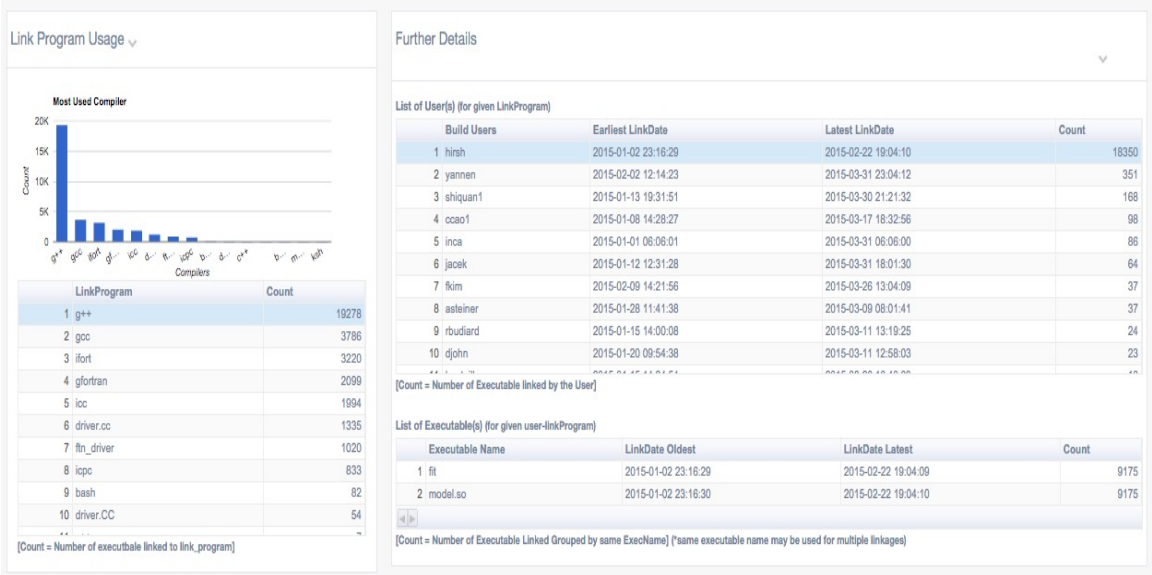


Figure 4.13: Link Program Usage on Darter for period Jan-Mar' 2015

As the time line suggests, the work was done almost a year back and it is difficult for users to recall what they were trying to achieve. While analyzing the compiler trends support staff can dive into the readily available information by accessing the XALT web portal and further decide about the outliers in more confident manner whether to ignore the trend or take necessary action on it.

However, monitoring usage can be done monthly, quarterly, semi-annually, or yearly. Doing so would restrict the probability of users not having enough information about their work. Also, support staff can communicate with the users asking for feedback to better understand user's needs which further leads to better training, effective documentation, and more desirable outreach programs.

4.2 Use Cases

This section presents few cases that can faced by support staff/researchers working on supercomputers.

User Software Provenance

Taking care of ones computational research work history without any personal efforts is the most desirable byproduct which interests scientific researchers. The volume of data XALT tracks for users using HPC machines is one of the most remarkable characteristics and building principle of the tool. Computers now a days process a huge amount of data, and some of the most intense and productive tasks involve simulations. Simulations considers multiple variables and use artificial intelligence to analyze them and examine outcomes, eventually becoming an integral part of scientific research.

By user software provenance or knowing the earliest known history of their work. We can provide researchers a window to look into their own work promptly. One starts by giving a 'userid', syshost and the date time range to portal's user provenance tab. For our discussion we would give user as 'rbudiard', syshost as 'Darter' and date time range for year 2015.

As shown in Figure 4.14 user would be able to see information that has been tracked automatically by XALT. Overall 7 different tables will be presented to the user which are:

1. *List of Executable(s)*: Lists all executables that user had run on the given system within given date range. It also shows #count i.e., number of times executable was run. Here executable having same name are grouped together. For example executable 'sssimp' has been run twice on Darter. On selecting the given executable, a new table showing details of each unique executable would be presented.

Figure 4.14: User Software Provenance: Complete work history of user 'rbuidiard' for year 2015 on Darter.

2. *Executable Details*: Multiple job runs using same executable can be traced back and shown in this table. User can look up for individual job run details in addition to link time details like who is the build user, when was it build and what compiler was used along with universal unique ID (UUID). Once the user determines which job run they want to inspect, they can just click on the given record (say record with Job Id = 666809) as shown in Figure 4.14 and three more tables will be presented to the user.

3. *Object Linked (Compile Time)*: This table displays all the objects what were linked to the executable at the link time. This information is static, and would hardly change over time unless user recompiles their code.

4. *Function Called*: This table displays all the functions calls resolved by external libraries for executable build. However, as this is a new functionality which was introduced in later version of XALT [v0.7.1]. Therefore, for earlier builds we have no record of function calls, which means the table would not be shown if in case the build is old.

5. *Job Run Details*: Lists additional information of the job run for given user-uuid combination.

6. *Run Environment Details*: Here we have all the counters which represents the state of the run environment at the time when the executable was run. How this information can help user is discussed further below.

7. *Objects Linked (Run Time)*: XALT tracks objects which are dynamically linked at the run time. This table displays all the objects that are linked dynamically at run time *(if in case any)*.

Typically on a given high performance computing machine, there are different nodes to which a user has access, e.g., compute nodes, login nodes, or service nodes. The way these nodes are set up is entirely at the discretion of the center. One has

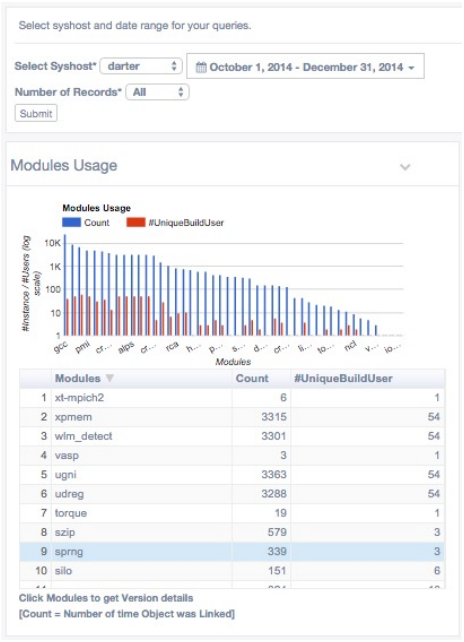
to be careful when using shared object libraries, as there is a good chance that login nodes do not run the same operating system as the compute nodes, and thus many shared libraries which are available on login nodes are not available on the compute nodes. This means that the executable may appear to compile correctly on a login node, but will fail to start on a compute node because it is unable to locate the shared libraries [39].

Sometimes users might even need to copy all necessary libraries from their project scratch area and then update the `LD_LIBRARY_PATH` environment variable to include this directory. XALT tracks all important counters for each individual job run and build. Having access to the complete picture on how the executable was built and how the run environment was setup could help users and support staff to detect errors. Using the web portal's software provenance tab, users can have most accurate and detailed information available in real time.

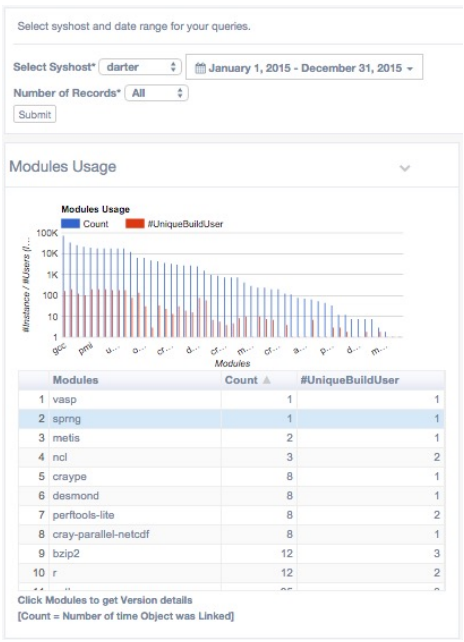
Identify User

The identify user tab on web portal gives access to support staff and users to identify users based on a given object path (whole or part of it) or by a given executable name. This tab can be used for different purpose like auditing packages/licenses, identify users using deprecated libraries, optimizing HPC system operations by asking users to recompile their executable with the updated version of software package, find who built the executable by performing a lookup with executable name.

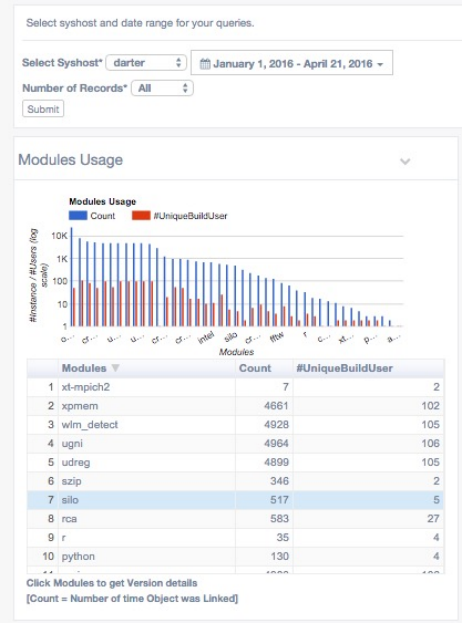
Lets consider a hypothesis, a center decides to remove the support for a module 'sprng' [40] (A scalable parallel random number generator library) based on the usage over the past three years. This time frame is selected as XALT went live in Sep-2014 at NICS, in future centers would be able to excavate more information depending on what year XALT was installed. Figure 4.15 shows the module usage for last quarter of year 2014 4.15a, for complete year 2015 4.15b, and first quarter of year 2016 4.15c. As shown in the figure module 'sprng' is hardly used over the period of year 2014-2015 and absolutely no use in first quarter of year 2016. However, as discussed in section



(a) Module Usage: For last quarter of year 2014.



(b) Module Usage: For year 2015.



(c) Module Usage: For first quarter of year 2016.

Figure 4.15: Module usage for 'sprng' module for 2014, 2015, and 2016

4.1.2, it may not be the best recommended practice to migrate a software to either a newer version or to different, compatible software (deprecating the existing software) based only on software usage. A center's support staff can make informed decisions by identifying target users who have used that particular module (in this case 'sprng') rather than contacting all users.

Centers can even optimize HPC system operations by asking users to recompile their existing codes with an upgraded version of a software package by identifying users still using old versions on this tab. This information is usually stated in 'whats new' section of the software update. On the identify user tab one can get all the information as provided in other tabs of the portal.

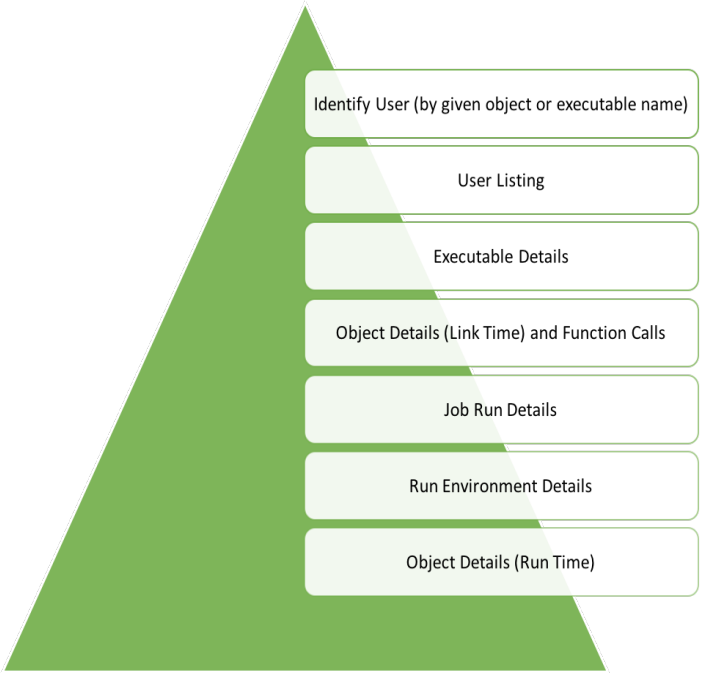


Figure 4.16: Identify User: Based on given object path, or executable name.

A pictorial representation is shown in Figure 4.16 of all sort of listing one can access on XALT's identify user tab.

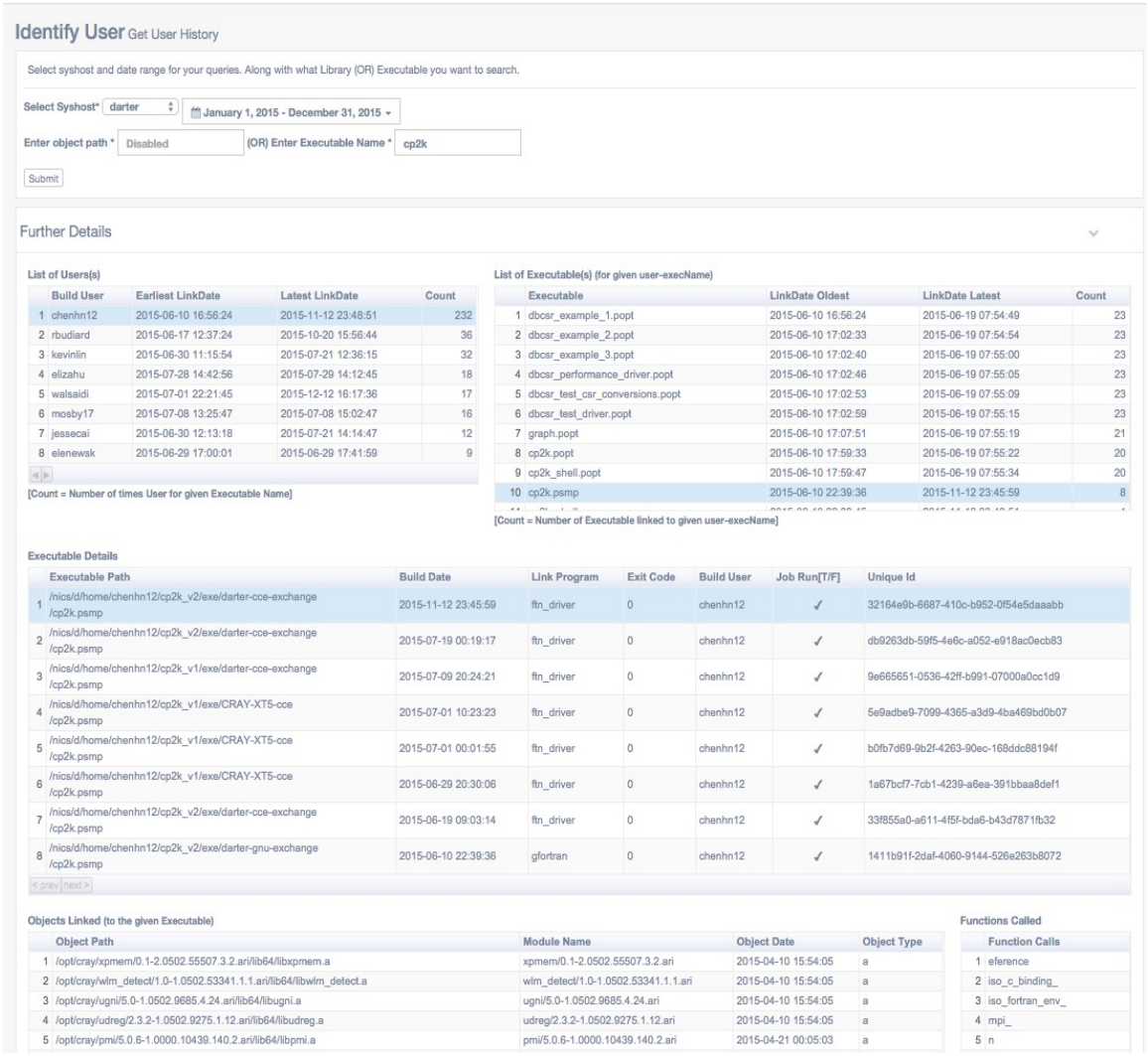


Figure 4.17: Identify users who have built executable 'cp2k' for year 2015 on Darter.

Another hypothesis revolves around the fact that not all researchers who work on high performance computing machines comes from a computer science background. This makes things more difficult, as a researcher might be using an executable built by some other user to carry out their research work. With time researchers may equip themselves with knowledge of running the executable and performing small changes to the existing code; however, there are times when a researcher wishes they could trace the actual build user and discuss why an executable has failed and/or some new changes they might want to incorporate into the existing build.

The identify tab on the portal provides this functionality based on the given executable name. It provides a list of all users for a given syshost and date time range. Researchers/users can make use of the interactive charts and trace the actual build user. This can help resolve unforeseen errors with guided supervision. For example, as shown in Figure 4.17, we wanted to track all users who have build an executable named 'cp2k' for the year 2015 on Darter. It seems there are 8 distinct users who have build an executable 'cp2k' with associative earliest link date and latest link date. Apart from researchers getting in touch with build user whose executable they are using, they can get in touch with different users working on the same type of application and further have better understanding of their work.

Chapter 5

Conclusions and Future Work

We have demonstrated, through several reports and case studies, the utility of web portal for providing metrics regarding the resource utilization from the data collected by XALT. This portal can be used by centers to readily audit the given system weekly, monthly, quarterly, or annually. We anticipate that the full value of web portal will be realized when it is fully operational and used by center support staff and end-users. Furthermore, center support staff will be able to use the portal to conduct analyses to set and adjust policies that maximize system usability. In addition, users can proactively troubleshoot their HPC environment to optimize their work. We have tried to remove dependencies to write manual scripts to mine XALT's data and hence, provide a complete package which is easy to operate, is interactive, and has an extensible framework. Importantly, the instantaneous access to metrics will aid resource providers in making infrastructure decisions which translates into lowering associated costs.

There are a few things which we would like to carry forward as part of our future work for this web portal. We would like the portal to support user roles so that the HPC center director can determine the appropriate data access level for end users, support personnel, and administrators. We can also extend the capability of this framework by making it portable to users, that is, to allow users to download the

reports and metrics generated by the portal. This would give a degree of freedom to use the data for further analyses or present it in a custom manner.

Although functionality continues to be added to the web portal, the features summarized in this work can prove useful and provide a better foundation for future development. Ultimately we would like to make this portal a standard part of the infrastructure tool.

Bibliography

[1] High-Performance Computing (HPC). [Online]. https://www.techopedia.com/definition/4595/high-performance-computing-hpc/. 1

[2] K. Agrawal, M. Fahey, R. McLay, and D. James. User environment tracking and problem detection with XALT. In *Proceedings of 1st Workshop on HPC Tools for User Support (HUST14) Workshop held in Conjunction with the International Conference for High Performance Computing, Networking, Storage and Analysis (SC14)*, HUST14, New Orleans, LA, November 2014. 2, 10, 11, 16

[3] M. Fahey, N. Jones, and B. Hadri. The Automatic Library Tracking Database. In *Proceedings of the 2010 Cray User Group*, CUG10, Edinburgh, May 2010. 4, 7

[4] Lariat. [Online]. https://github.com/TACC/Lariat/. 4, 7

[5] Reuben Budiardja, Mark Fahey, Robert McLay, Prasad Maddumage Don, Bilel Hadri, and Doug James. Community use of xalt in its first year in production. In *Proceedings of the Second International Workshop on HPC User Support Tools*, page 4. ACM, 2015. 4, 7, 30

[6] JSON. http://www.json.org//. 5

[7] Mark Fahey, Nick Jones, Bilel Hadri, and Blake Hitchcock. The automatic library tracking database. In *Conference: Cray User Group*, 2010. 7

[8] http://linux.about.com/library/cmd/blcmdl1_sha1sum.htm/. 7

[9] Todd Evans, William L Barth, James C Browne, Robert L DeLeon, Thomas R Furlani, Steven M Gallo, Matthew D Jones, and Abani K Patra. Comprehensive resource use monitoring for hpc systems with tacc stats. In *Proceedings of the First International Workshop on HPC User Support Tools*, pages 13–21. IEEE Press, 2014. 7, 12

[10] Bilel Hadri, M Fahey, Tim Robinson, and William Renaud. Software usage on cray systems across three centers (nics, ornl and cscs). In *Proceedings of the Cray User Group Conference (CUG 2012)*, 2012. 9

[11] Julian Borrill, Jonathan Carter, Leonid Oliker, David Skinner, and Rupak Biswas. Integrated performance monitoring of a cosmology application on leading hec platforms. In *Parallel Processing, 2005. ICPP 2005. International Conference on*, pages 119–128. IEEE, 2005. 9

[12] Sameer S Shende and Allen D Malony. The tau parallel performance system. *International Journal of High Performance Computing Applications*, 20(2):287–311, 2006. 10

[13] Wolfgang Nagel. Vampir - Performance Optimization. https://www.vampir.eu/. 10

[14] Cray. Using Cray Performance Analysis Tools. http://docs.cray.com/books/S-2376-41/S-2376-41.pdf. 10

[15] Charng-Da Lu. Automatically mining program build information via signature matching. In *Proceedings of the 11th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, pages 25–32. ACM, 2013. 10

[16] T. Kojm. http://www.clamav.net/. 10

[17] Bernd Mohr. Automatic performance statistics collection on the cray t3e. 1999. 10, 11

[18] Andrew Barry. Resource utilization reporting. In *Proc. Cray Users Group Technical Conference (CUG)*, 2013. 11

[19] Heike McCraw, Joseph Ralph, Anthony Danalis, and Jack Dongarra. Power monitoring with papi for extreme scale architectures and dataflow-based

programming models. In *Cluster Computing (CLUSTER), 2014 IEEE International Conference on*, pages 385–391. IEEE, 2014. 11

[20] Garrick Staples. Torque resource manager. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 8. ACM, 2006. 11

[21] A Tam. Enabling process accounting on linux howto, version 1.1, 2001. 11

[22] Charis Ermopoulos and William Yurcik. Nvision-pa: A tool for visual analysis of command behavior based on process accounting logs (with a case study in hpc cluster security). *arXiv preprint cs/0606089*, 2006. 11

[23] James C Browne, Robert L DeLeon, Charng-Da Lu, Matthew D Jones, Steven M Gallo, Amin Ghadersohi, Abani K Patra, William L Barth, John Hammond, Thomas R Furlani, et al. Enabling comprehensive data-driven system management for large computational facilities. In *High Performance Computing, Networking, Storage and Analysis (SC), 2013 International Conference for*, pages 1–11. IEEE, 2013. 12

[24] James C Browne, Robert L DeLeon, Abani K Patra, William L Barth, John Hammond, Matthew D Jones, Thomas R Furlani, Barry I Schneider, Steven M Gallo, Amin Ghadersohi, et al. Comprehensive, open-source resource usage measurement and analysis for hpc systems. *Concurrency and Computation: Practice and Experience*, 26(13):2191–2209, 2014. 12

[25] Jeffrey T Palmer, Steven M Gallo, Thomas R Furlani, Matthew D Jones, Robert L DeLeon, Joseph P White, Nikolay Simakov, Abani K Patra, Jeanette Sperhac, Thomas Yearke, et al. Open xdmod: A tool for the comprehensive management of high-performance computing resources. *Computing in Science & Engineering*, 17(4):52–62, 2015. 12

[26] XALTUsersManual-0.5. https://github.com/Fahey-McLay/xalt/blob/master/doc/XALTUsersManual-0.5.pdf/, . 16

[27] Gentella. https://themewagon.com/themes/free-bootstrap-3-admin-dashboard-template/. 17

[28] Google Charts. https://developers.google.com/chart/. 17

[29] Bootstrap. http://bootstrapdocs.com/v3.0.3/docs/getting-started/. 17

[30] XALT Web-portal, Web interface to generate reports from the data collected by XALT [A software monitoring tool]. https://github.com/kpl-grwl/xalt-portal/, . 21

[31] Doxygen, Generate documentation from source code. http://www.stack.nl/~dimitri/doxygen/. 21

[32] Computing Resources at NICS. https://www.nics.tennessee.edu/computing-resources/darter/. 21

[33] Thomas R Furlani, Matthew D Jones, Steven M Gallo, Andrew E Bruno, Charng-Da Lu, Amin Ghadersohi, Ryan J Gentner, Abani Patra, Robert L DeLeon, Gregor Laszewski, et al. Performance metrics and auditing framework using application kernels for high-performance computer systems. *Concurrency and Computation: Practice and Experience*, 25(7):918–931, 2013. 24

[34] NAMD Molecular Dynamics Simulator. http://www.ks.uiuc.edu/Research/namd/. 25

[35] Vienna Ab initio Simulation Package. http://www.vasp.at/. 25, 32

[36] LAMMPS Molecular Dynamics Simulator. http://lammps.sandia.gov/. 25

[37] NWCHEM High-Performance Computational Chemistry Software. http://www.nwchem-sw.org/index.php/Main_Page/. 25

[38] REUS, One-dimensional replica-exchange umbrella sampling. http://www.ks.uiuc.edu/Training/Tutorials/science/umbrella/REUS-1D.pdf/. 36

[39] Compiling and Node Types on Titan. https://www.olcf.ornl.gov/kb_articles/compiling-and-node-types/?print=true/. 40

[40] The Scalable Parallel Random Number Generators Library. http://www.sprng.org/. 40

Appendix

Function Tracking Changes

Function tracking changes can be found at https://github.com/kpl-grwl/ftrack2015

XALT web-portal

Web portal source can be found at https://github.com/kpl-grwl/xalt-portal

Vita

Kapil Agrawal spent his childhood in Indore, a city located in the central part of India. He graduated from St. Paul Higher Secondary School at Indore, in 2003 and thereafter, enrolled in the Electronics and Communication Engineering undergraduate program at I.I.S.T. Indore. He received his undergraduate degree in 2007. He then moved to Mumbai, and worked for five years as a software developer. In Aug 2013, he started his Masters degree at the University of Tennessee, Knoxville. During his Masters he worked as an Graduate Assistant in the Joint Institute for Computational Sciences at UTK. In his spare time, he enjoys playing COD, cricket, and reading all things related to fighter jets and Indian Military.