Automatic Library Tracking Database at NERSC

Zhengji Zhao
National Energy Research Scientific Computing Center,
Lawrence Berkeley National Laboratory
Berkeley, USA
ZZhao@lbl.gov

Abstract—**Automatic Library Tracking Database infrastructure is a lightweight tool that can automatically track the libraries that are linked into the applications. It can also track the applications that are launched on the Cray systems (on other systems as well). It was previously developed by Fahey, *et al.* [1], and was further developed at NERSC to be used at computing centers like NERSC, which has more than 6,000 active users. In this paper, we will present the ALTD tool and its enhancement at NERSC. We will also present how it is used at NERSC for various purposes.**

Keywords—ALTD; link line; library usage; ld; linker; aprun; job launcher; wrapper

I. Introduction

Understanding software usage is an important task for HPC centers in order to provide a better software service to users and to plan for future software need. While there are ways to track application software from the various logs available on the HPC systems, there were not good ways to track the libraries used in the applications, especially when the applications are linked statically, until Fahey, *et al.,* developed the Automatic Library Tracking Database infrastructure (ALTD) [1]. ALTD can track the libraries used in both statically and dynamically linked applications, and also for the libraries provided by vendors, support staff and users. ALTD intercepts the link line of the GNU linker (ld), which is invoked by all compilers at the backend, and stores the link line along with other user information into a database for later query and analysis. ALTD also intercepts the job launchers, aprun or mpirun (for simplicity omit mpirun for the job launcher hereafter), to track the application binaries launched on the systems and stores the application as well as the job information into a database. By searching the characteristic patterns in the link lines stored in the database, one can obtain the library usage statistics. Since ALTD stores the whole link line which contains the paths to the library files, not only the library names but also the exact versions of the libraries can be tracked as well. ALTD also adds a unique tag to the application binaries at linking time, therefore it is possible to track the libraries used by an application actually run on the system as well. ALTD was implemented with two main scripts that wrap the ld and the aprun commands. Where the aprun wrapper enables arbitrary prologue and epilogue scripts for the job, which can be used for other workload monitoring purpose in addition to tracking library usage. ALTD is a lightweight tool. It has a negligible overhead to the compile time, and has zero overhead to the application run time. Therefore, ALTD does not change user experience on the systems, which is essential for ALTD to be used as a system-wide tracking tool on a HPC system.

ALTD has been used in several HPC centers [2-3], including the National Institute for Computational Sciences (NICS), Oak Ridge National Laboratory, the Swiss National Supercomputing Centre (CSCS), and NERSC. NERSC imported ALTD on its Cray XE6 in 2012. Compared to NICS where the tool was first developed, NERSC has a much larger number of users and applications. It has more than 6,000 active users, and has more than 500 applications running on each machine. When ALTD was first imported at NERSC, we ran into a few issues. First, ALTD accesses an external database at the place where the link line was intercepted, so each invocation of the ld command accesses an external database server. While this works fine for a center with a smaller user base, it did not work well at NERSC. Since many users were simultaneously pounding the database server, which serves other databases as well, it frequently put a high load on the database server, and made it fail to function properly. In addition, to keep only unique link lines in the database ALTD checks if a link line already exists in the database or not for each link line intercepted by the ld command. We noticed this query takes longer and longer time when database table sizes grow. Moreover, the system admins and the server team staff at NERSC had a strong concern about the fact that a fundamental command like ld relies on the status of an external server. Therefore NERSC implemented an ALTD to address these issues and concerns.

II. ALTD enhancement at NERSC

Before introducing the NERSC enhancement to ALTD, we would like to provide a relevant overview of the ALTD implementation at NICS. More details about ALTD implementation can be found in [1]. ALTD consists of the ld wrapper (a bash and a python scripts) and the aprun wrapper (a perl and a python scripts). The ld wrapper intercepts the link line, and stores the linkage information into two tables, altd_machine_link_tags and altd_machine_linkline (denote as link_tags and linkline tables hereafter), in the external database (MySQL) server. At the same time an ELF section header is added in the binary, which includes a unique id (the tag_id field in the link_tags table) that connects the binary with its link line in the database. The ld wrapper assigns this unique id for each successful linking and the resulting binary by incrementing an integer primary key in the link_tags table. The ld wrapper accesses the external database server each time the ld command is invoked. The ALTD aprun wrapper reads the ELF section header of a binary and stores its unique id (tag_id) and other job information into a database table, altd_machine_jobs, and then executes the binary. Therefore the aprun wrapper also accesses the database server for each aprun invocation.

To avoid the live access of the ld command to an external database server, we generated the primary key that is unique to each application binary without querying the link_tags table as implemented in the original ALTD. We used the Linux command, uuidgen, to generate unique primary key outside of the link_tags table, and saved the intercepted linkage data to the disk (See Fig. 1 for the tables used by ALTD at NERSC). Then a cron job runs every 5 minutes to push the collected data into the database. Fig. 2 is the flowchart of the ALTD ld wrapper. We also separated the data intercepting and database updating parts for the job launcher wrapper as well. Fig. 3 shows the flowchart of the ALTD aprun wrapper. This removed the "external" component in the ld and aprun commands, also prevented many users from pounding the database server simultaneously. To speed up the link line searching for the unique entries in the database, we added a field in the linkline table with the hash values of the link line, and used the hash value instead of the whole link line when searching for the unique link line entries.

III. Applications of ALTD

The ALTD tables contain information about by whom and when what application was compiled, and what libraries (and the exact versions) were used, which compiler (and version) was used, and how the application was linked (dynamically or statically), on which system the application was built, by whom and when the application was run, etc. Various library and application usage statistics can be generated by querying the ALTD databases.

ALTD has been used for multiple purposes since it was imported to NERSC. It can provide the library usage statistics which is important for the center to understand the current software need and to provide more focused software support for users. This information was also important for the system procurement. Fig. 4 shows the library usage on Hopper, a Cray XE6 system, which was used to prioritize the library need in the NERSC-8 procurement (which resulted in the next-generation system, Cori, a Cray system based on the Intel MIC architecture).

ALTD provides the usage statistics for the library and application developers and vendors, which help them to focus development effort on the more commonly used software packages.

ALTD helps restoring the program environment where user applications were built, which was helpful for users who have problems to use the upgraded software. See Fig. 5. This information was also helpful for debugging system issues that were potentially caused by the applications using certain libraries.

The ALTD aprun wrapper allows any scripts to run before and/or after the job execution, which can be used to collect many other information about the applications launched on the system. For example, by adding a simple prologue script in the aprun wrapper, ALTD was able to capture user input files for an application, VASP, a materials science code that consumes the most computing cycles at NERSC. This helps to identify the most commonly used computation types, so to focus code optimization effort on the most commonly used routines in the application to prepare for the next generation system.

Acknowledgment

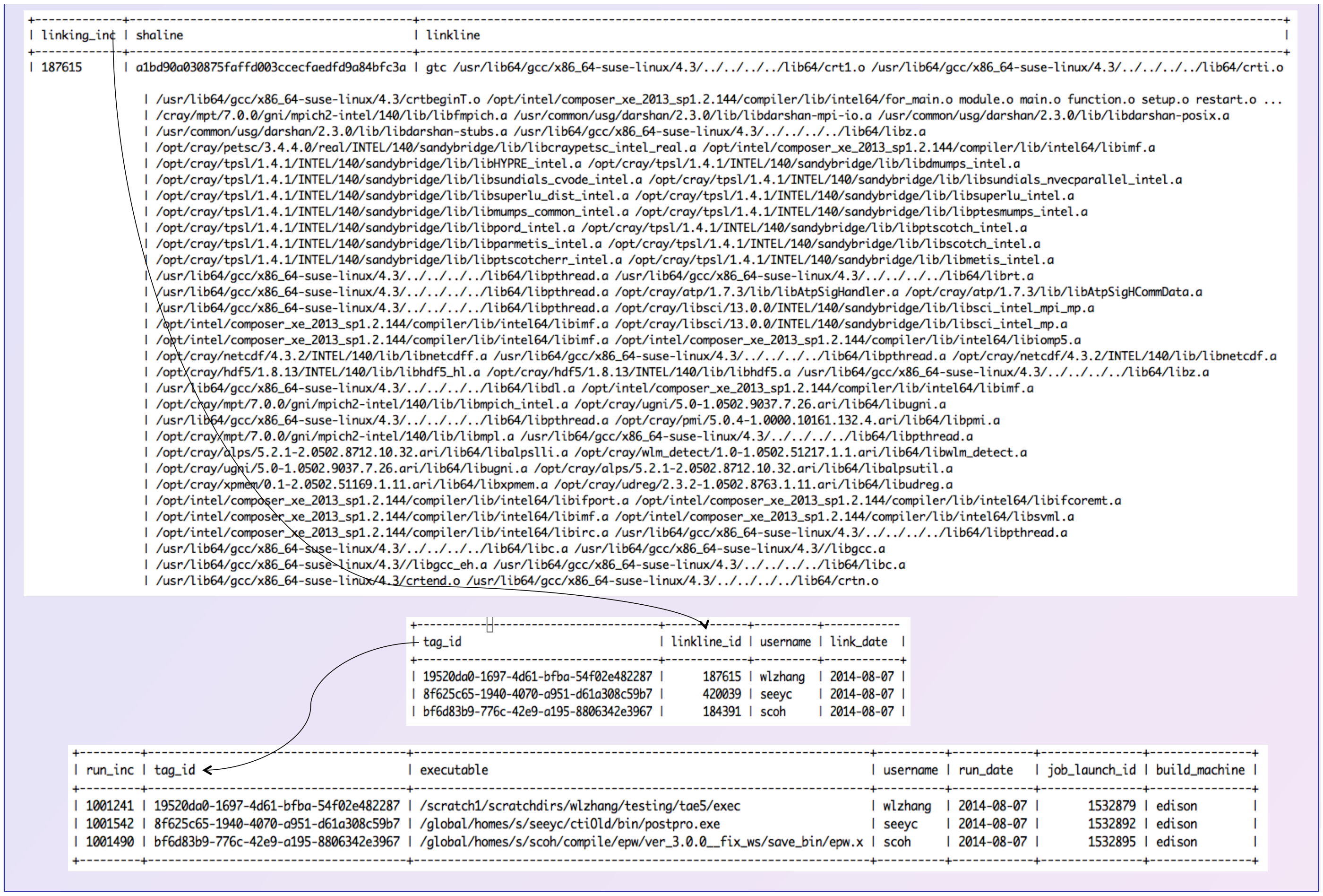
This work was supported by the ASCR Office in the DOE, Office of Science, under contract number DE-AC02-05CH11231. It used the resources of the National Energy Research Scientific Computing Center (NERSC).

References

[1] M. Fahey, N Jones, and B. Hadri, *The Automatic Library Tracking Database*, Proceedings of the Cray User Group 2010, Edinburgh, United Kingdom.

[2] B. Hadri, M. Fahey, T. Robinson, and W. Renaud, *Software Us- age on Cray Systems across Three Centers (NICS, ORNL and CSCS)*, Proceedings of the Cray User Group 2012, Stuttgart, Germany.

[3] *Recent enhancements to the Automatic Library Tracking Database infrastructure at the Swiss National Supercomputing Centre, proceedings of the Cray User Group 2013, Napa Valley, CA.*



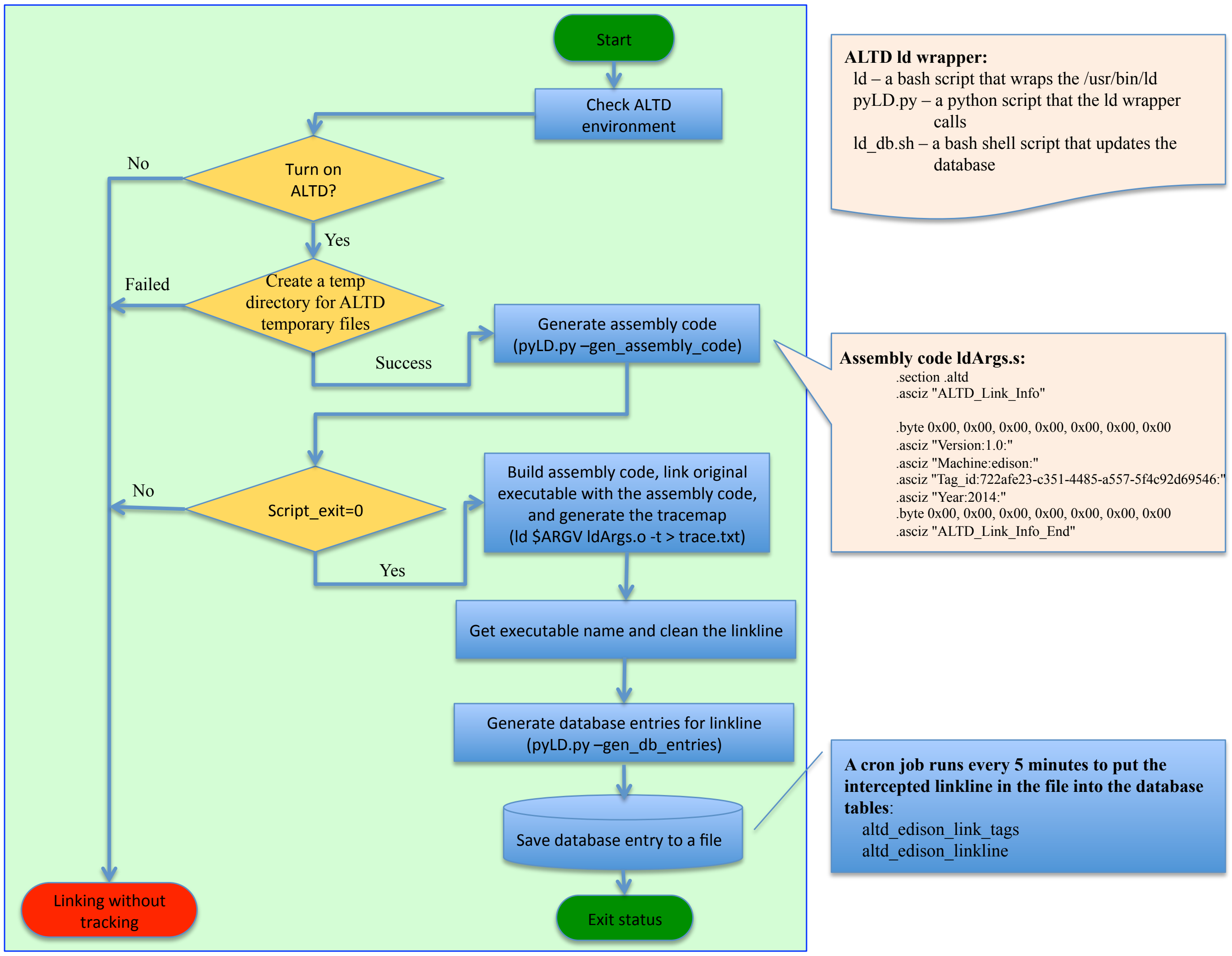
```
+-------------+-----------------------------------------+---------------------------------------------------------------------------------------------------------------------+
| linking_inc | shaline                                 | linkline                                                                                                            |
+-------------+-----------------------------------------+---------------------------------------------------------------------------------------------------------------------+
| 187615      | a1bd90a030875faffd003ccecfaedfd9a84bfc3a | gtc /usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../lib64/crt1.o /usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../lib64/crti.o

  | /usr/lib64/gcc/x86_64-suse-linux/4.3/crtbeginT.o /opt/intel/composer_xe_2013_sp1.2.144/compiler/lib/intel64/for_main.o module.o main.o function.o setup.o restart.o ...
  | /cray/mpt/7.0.0/gni/mpich2-intel/140/lib/libfmpich.a /usr/common/usg/darshan/2.3.0/lib/libdarshan-mpi-io.a /usr/common/usg/darshan/2.3.0/lib/libdarshan-posix.a
  | /usr/common/usg/darshan/2.3.0/lib/libdarshan-stubs.a /usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../lib64/libz.a
  | /opt/cray/petsc/3.4.4.0/real/INTEL/140/sandybridge/lib/libcraypetsc_intel_real.a /opt/intel/composer_xe_2013_sp1.2.144/compiler/lib/intel64/libimf.a
  | /opt/cray/tpsl/1.4.1/INTEL/140/sandybridge/lib/libHYPRE_intel.a /opt/cray/tpsl/1.4.1/INTEL/140/sandybridge/lib/libdmumps_intel.a
  | /opt/cray/tpsl/1.4.1/INTEL/140/sandybridge/lib/libsundials_cvode_intel.a /opt/cray/tpsl/1.4.1/INTEL/140/sandybridge/lib/libsundials_nvecparallel_intel.a
  | /opt/cray/tpsl/1.4.1/INTEL/140/sandybridge/lib/libsuperlu_dist_intel.a /opt/cray/tpsl/1.4.1/INTEL/140/sandybridge/lib/libsuperlu_intel.a
  | /opt/cray/tpsl/1.4.1/INTEL/140/sandybridge/lib/libmumps_common_intel.a /opt/cray/tpsl/1.4.1/INTEL/140/sandybridge/lib/libptesmumps_intel.a
  | /opt/cray/tpsl/1.4.1/INTEL/140/sandybridge/lib/libpord_intel.a /opt/cray/tpsl/1.4.1/INTEL/140/sandybridge/lib/libptscotch_intel.a
  | /opt/cray/tpsl/1.4.1/INTEL/140/sandybridge/lib/libparmetis_intel.a /opt/cray/tpsl/1.4.1/INTEL/140/sandybridge/lib/libscotch_intel.a
  | /opt/cray/tpsl/1.4.1/INTEL/140/sandybridge/lib/libptscotcherr_intel.a /opt/cray/tpsl/1.4.1/INTEL/140/sandybridge/lib/libmetis_intel.a
  | /usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../lib64/libpthread.a /usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../lib64/librt.a
  | /usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../lib64/libpthread.a /opt/cray/atp/1.7.3/lib/libAtpSigHandler.a /opt/cray/atp/1.7.3/lib/libAtpSigHCommData.a
  | /usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../lib64/libpthread.a /opt/cray/libsci/13.0.0/INTEL/140/sandybridge/lib/libsci_intel_mpi_mp.a
  | /opt/intel/composer_xe_2013_sp1.2.144/compiler/lib/intel64/libimf.a /opt/cray/libsci/13.0.0/INTEL/140/sandybridge/lib/libsci_intel_mp.a
  | /opt/intel/composer_xe_2013_sp1.2.144/compiler/lib/intel64/libimf.a /opt/intel/composer_xe_2013_sp1.2.144/compiler/lib/intel64/libiomp5.a
  | /opt/cray/netcdf/4.3.2/INTEL/140/lib/libnetcdff.a /usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../lib64/libpthread.a /opt/cray/netcdf/4.3.2/INTEL/140/lib/libnetcdf.a
  | /opt/cray/hdf5/1.8.13/INTEL/140/lib/libhdf5_hl.a /opt/cray/hdf5/1.8.13/INTEL/140/lib/libhdf5.a /usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../lib64/libz.a
  | /usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../lib64/libdl.a /opt/intel/composer_xe_2013_sp1.2.144/compiler/lib/intel64/libimf.a
  | /opt/cray/mpt/7.0.0/gni/mpich2-intel/140/lib/libmpich_intel.a /opt/cray/ugni/5.0-1.0502.9037.7.26.ari/lib64/libugni.a
  | /usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../lib64/libpthread.a /opt/cray/pmi/5.0.4-1.0000.10161.132.4.ari/lib64/libpmi.a
  | /opt/cray/mpt/7.0.0/gni/mpich2-intel/140/lib/libmpl.a /usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../lib64/libpthread.a
  | /opt/cray/alps/5.2.1-2.0502.8712.10.32.ari/lib64/libalpslli.a /opt/cray/wlm_detect/1.0-1.0502.51217.1.1.ari/lib64/libwlm_detect.a
  | /opt/cray/ugni/5.0-1.0502.9037.7.26.ari/lib64/libugni.a /opt/cray/alps/5.2.1-2.0502.8712.10.32.ari/lib64/libalpsutil.a
  | /opt/cray/xpmem/0.1-2.0502.51169.1.11.ari/lib64/libxpmem.a /opt/cray/udreg/2.3.2-1.0502.8763.1.11.ari/lib64/libudreg.a
  | /opt/intel/composer_xe_2013_sp1.2.144/compiler/lib/intel64/libifport.a /opt/intel/composer_xe_2013_sp1.2.144/compiler/lib/intel64/libifcoremt.a
  | /opt/intel/composer_xe_2013_sp1.2.144/compiler/lib/intel64/libimf.a /opt/intel/composer_xe_2013_sp1.2.144/compiler/lib/intel64/libsvml.a
  | /opt/intel/composer_xe_2013_sp1.2.144/compiler/lib/intel64/libirc.a /usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../lib64/libpthread.a
  | /usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../lib64/libc.a /usr/lib64/gcc/x86_64-suse-linux/4.3/libgcc.a
  | /usr/lib64/gcc/x86_64-suse-linux/4.3//libgcc_eh.a /usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../lib64/libc.a
  | /usr/lib64/gcc/x86_64-suse-linux/4.3/crtend.o /usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../lib64/crtn.o
```

```
+--------------------------------------+-------------+----------+------------
| tag_id                               | linkline_id | username | link_date  |
+--------------------------------------+-------------+----------+------------+
| 19520da0-1697-4d61-bfba-54f02e482287 |      187615 | wlzhang  | 2014-08-07 |
| 8f625c65-1940-4070-a951-d61a308c59b7 |      420039 | seeyc    | 2014-08-07 |
| bf6d83b9-776c-42e9-a195-8806342e3967 |      184391 | scoh     | 2014-08-07 |
```

```
+---------+--------------------------------------+------------------------------------------------------------------+----------+------------+---------------+---------------+
| run_inc | tag_id                               | executable                                                       | username | run_date   | job_launch_id | build_machine |
+---------+--------------------------------------+------------------------------------------------------------------+----------+------------+---------------+---------------+
| 1001241 | 19520da0-1697-4d61-bfba-54f02e482287 | /scratch1/scratchdirs/wlzhang/testing/tae5/exec                  | wlzhang  | 2014-08-07 |       1532879 | edison        |
| 1001542 | 8f625c65-1940-4070-a951-d61a308c59b7 | /global/homes/s/seeyc/cti0ld/bin/postpro.exe                     | seeyc    | 2014-08-07 |       1532892 | edison        |
| 1001490 | bf6d83b9-776c-42e9-a195-8806342e3967 | /global/homes/s/scoh/compile/epw/ver_3.0.0__fix_ws/save_bin/epw.x | scoh     | 2014-08-07 |       1532895 | edison        |
+---------+--------------------------------------+------------------------------------------------------------------+----------+------------+---------------+---------------+
```

Fig. 1. Tables used in the ALTD implementation at NERSC. From the top, they are the altd_edison_linkline, altd_edison_link_tags and altd_edison_jobs tables, respectively, for Edison, a Cray XC30 system at NERSC. The first two tables are used by the ALTD ld wrapper. Compared to the tables used in the NICS implementation, the unique primary key, tag_id, in the altd_edison_link_tags table is a 36 character string instead of an integer, and is generated by the Linux command, uuidgen, outside of the table. This is to avoid the ld command from accessing the external database server. The shaline field in the altd_edison_linkline table is the hash function of the linkline, which is used to check if a linkline already exists in the linkline table or not. This is a new addition in the NERSC ALTD linkline table.


Start
Check ALTD environment
Turn on ALTD?
No
Yes
Create a temp directory for ALTD temporary files
Failed
Success
Generate assembly code (pyLD.py –gen_assembly_code)
Script_exit=0
No
Yes
Build assembly code, link original executable with the assembly code, and generate the tracemap (ld $ARGV ldArgs.o -t > trace.txt)
Get executable name and clean the linkline
Generate database entries for linkline (pyLD.py –gen_db_entries)
Save database entry to a file
Exit status
Linking without tracking

ALTD ld wrapper:
ld – a bash script that wraps the /usr/bin/ld
pyLD.py – a python script that the ld wrapper calls
ld_db.sh – a bash shell script that updates the database

Assembly code ldArgs.s:
```
        .section .altd
        .asciz "ALTD_Link_Info"

        .byte 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
        .asciz "Version:1.0:"
        .asciz "Machine:edison:"
        .asciz "Tag_id:722afe23-c351-4485-a557-5f4c92d69546:"
        .asciz "Year:2014:"
        .byte 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
        .asciz "ALTD_Link_Info_End"
```

A cron job runs every 5 minutes to put the intercepted linkline in the file into the database tables:
altd_edison_link_tags
altd_edison_linkline

Fig. 2. The flowchart of the ALTD ld wrapper at NERSC. Now the ld wrapper does not access the external database server at invocation of the ld command. Instead, it saves the database entry for the linkage information to a file, and then a cron job, which runs every 5 minutes, pushes the linkage information from the file to the database.

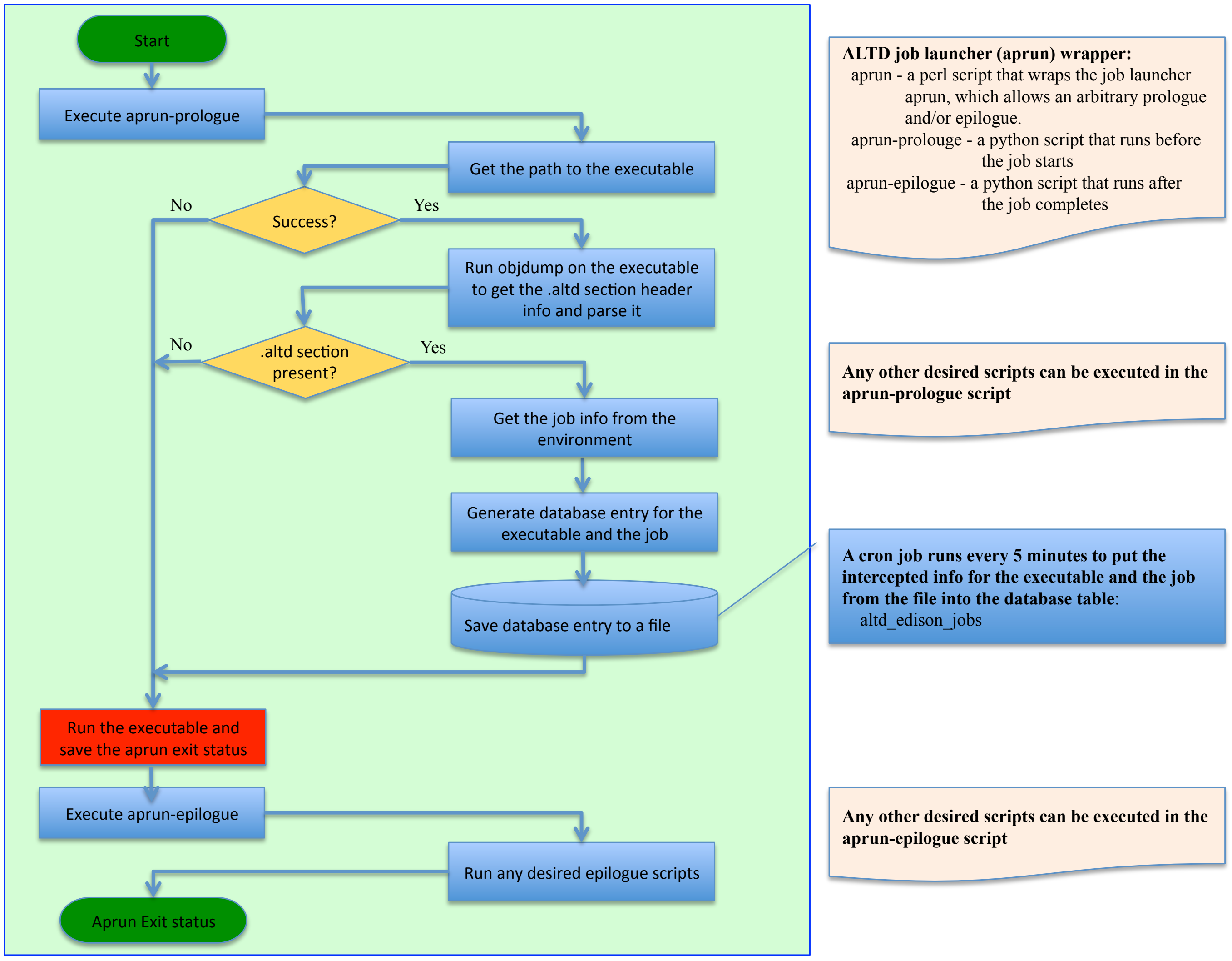

Start
Execute aprun-prologue
Get the path to the executable
Success?
No
Yes
Run objdump on the executable to get the .altd section header info and parse it
.altd section present?
No
Yes
Get the job info from the environment
Generate database entry for the executable and the job
Save database entry to a file
Run the executable and save the aprun exit status
Execute aprun-epilogue
Run any desired epilogue scripts
Aprun Exit status
ALTD job launcher (aprun) wrapper:
aprun - a perl script that wraps the job launcher aprun, which allows an arbitrary prologue and/or epilogue.
aprun-prolouge - a python script that runs before the job starts
aprun-epilogue - a python script that runs after the job completes
Any other desired scripts can be executed in the aprun-prologue script
A cron job runs every 5 minutes to put the intercepted info for the executable and the job from the file into the database table: altd_edison_jobs
Any other desired scripts can be executed in the aprun-epilogue script

Fig. 3. The flowchart of the ALTD aprun wrapper at NERSC. Now the aprun wrapper does not access the external database server at invocation of the aprun command. Instead, it saves the database entry about the application and the job to a file, and then a cron job, which runs every 5 minutes, pushes the entry from the file to the database.

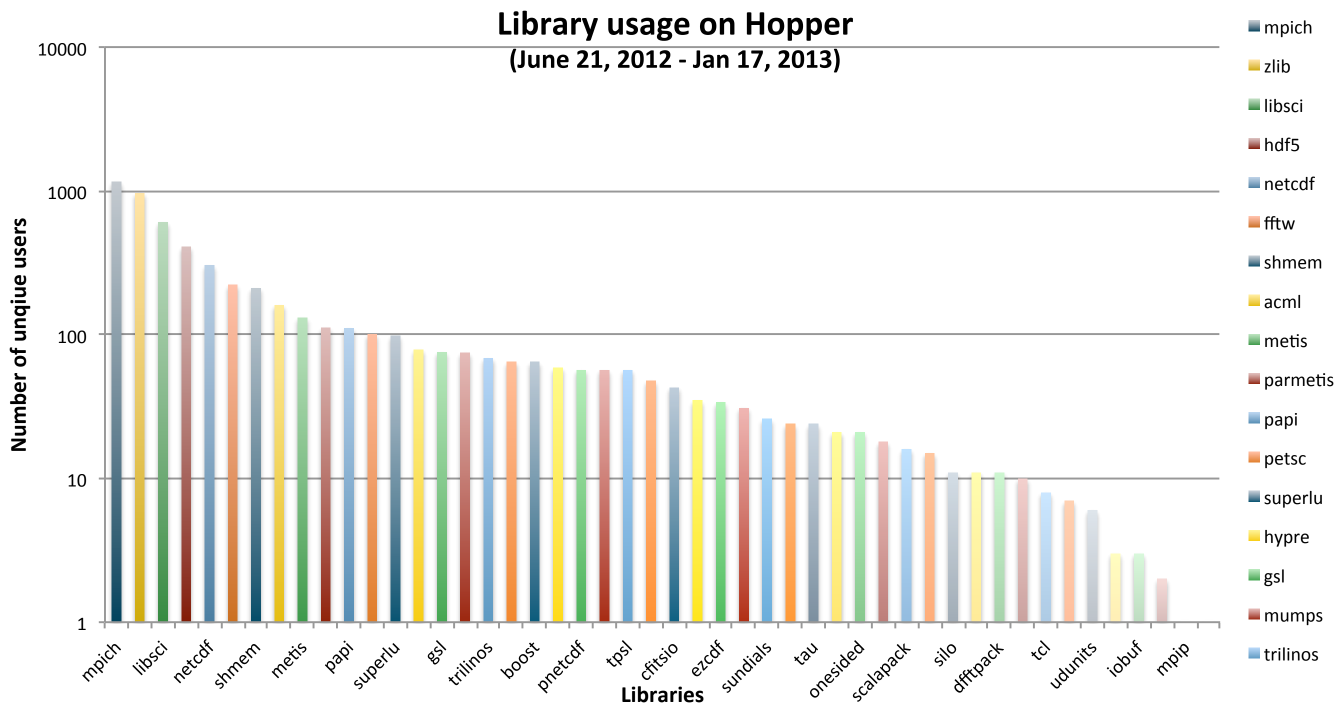


Figure 4. This figure shows the number of unique users for the libraries supported by Cray and NERSC staff on Hopper during Jun 21, 2012 and Jan 17, 2013 time period, in the order of from the most to the least used libraries.

```
aryal@edison12:~> linkinfo.sh /global/homes/a/aryal/bin/gvasp5.3.2
User          : zz217
Linked on     : 2013-01-03
Executable Name: vasp
Libraries Used :
//usr/lib64/libhugetlbfs.a
../vasp.5.lib/libdmy.a
/opt/cray/atp/1.6.0/lib//libAtpSigHCommData.a
/opt/cray/atp/1.6.0/lib//libAtpSigHandler.a
/opt/cray/libsci/12.0.00/cray/81/sandybridge/lib/libsci_cray_mp.a
/opt/fftw/3.3.0.1/x86_64/lib/libfftw3.a
/opt/cray/mpt/5.6.0/gni/mpich2-cray/74/lib/libmpich_cray.a
/opt/cray/mpt/5.6.0/gni/mpich2-cray/74/liblibmpl.a
/opt/cray/xpmem/0.1-2.0500.36799.3.6.ari/lib64/libxpmem.a
/opt/cray/pmi/4.0.0-1.0000.9282.69.4.ari/lib64/libpmi.a
/opt/cray/ugni/4.0-1.0500.5836.7.58.ari/lib64/libugni.a
/opt/cray/udreg/2.3.2-1.0500.5931.3.1.ari/lib64/libudreg.a
/opt/cray/alps/5.0.1-2.0500.7663.1.1.ari/lib64/libalpslli.a
/opt/cray/alps/5.0.1-2.0500.7663.1.1.ari/lib64/libalpsutil.a
/opt/cray/cce/8.1.2/craylibs/x86-64/libpgas-dmapp.a
/opt/cray/cce/8.1.2/craylibs/x86-64/libu.a
/opt/cray/dmapp/4.0.1-1.0500.5932.6.5.ari/lib64/libdmapp.a
/opt/cray/pmi/4.0.0-1.0000.9282.69.4.ari/lib64/libpmi.a
/opt/cray/cce/8.1.2/craylibs/x86-64/libfi.a
/opt/gcc/4.4.4/snos/lib64/libstdc++.a
/opt/gcc/4.4.4/snos/lib/gcc/x86_64-suse-linux/4.4.4/libgcc_eh.a
/opt/cray/cce/8.1.2/craylibs/x86-64/libf.a
/opt/cray/cce/8.1.2/craylibs/x86-64/libcraymath.a
/opt/cray/cce/8.1.2/craylibs/x86-64/libcraymp.a
/opt/cray/cce/8.1.2/craylibs/x86-64/libu.a
/opt/cray/cce/8.1.2/craylibs/x86-64/libcsup.a
//usr/lib64/librt.a
/opt/cray/cce/8.1.2/craylibs/x86-64/libtcmalloc_minimal.a
//usr/lib64/libpthread.a
//usr/lib64/libc.a
/opt/gcc/4.4.4/snos/lib/gcc/x86_64-suse-linux/4.4.4/libgcc_eh.a
//usr/lib64/libm.a
/opt/gcc/4.4.4/snos/lib/gcc/x86_64-suse-linux/4.4.4/libgcc.a
```

Fig. 5. An ALTD tool to display the build environment for an application binary.