XALT Users Manual

version 0.5

Mark Fahey

Robert McLay

Kapil Agrawal

Using XALT

This section describes how XALT is used in practice.

We assume XALT is installed (following Design and Installation Manual guidelines.) We will also assume that a modulefile for XALT exists and that the site is using the modulefile to get the linker and job launcher wrappers into the users path. (There are other ways to accomplish this discussed in the Design and Installation Manual.)

Decisions made previously

During the installation process, various choices had to be made. Here we quickly review the options.

1. If a site has multiple machines, they have to decide if they want one database (one set of common tables) to hold all the information for all machines or if they want to have multiple database (typically one per machine).

    a. **One database**: If you go with one database, then you *may* need a script per machine [samples provided in the distribution] to get the data into the database (depending on the method you choose),

    b. **Multiple databases**: In this scenario, a site just has to set up multiple databases (i.e., one server with multiple database like xalt-machine1 and xalt-machine2) You will need a script per machine to get the data into the database unless you are doing "direct to database." Note that if you are willing to edit the code directly, you could go with one database and multiple sets of tables per machine - not supported.

    We'll assume one database for the remainder, but it is an easy extension for multiple database.

    Regardless of the decision above, you *will* need the following if you have mulitple machines:

    a. if your machines have different software installations (module lists) *AND* want the ReverseMap support, then you will need a [Lmod] ReverseMap per machine which means having a build of Lmod per machine, and
    b. psmisc-22.21 or greater, this is discussed below.

2. The next choice was deciding where a few files are to be located; these files being the database xalt_db.conf file (mysql database access information) and the reverseMapD

directory and reverseMap file(s). We suggest for simplicity that that go in XALT_DIR/etc, but this is up to the site to pick the location. It may be that the site wants the xalt_db.conf file somewhere more hidden/secure. This can be chosen with the configuration option --with-etcDir=ans or overridden at runtime with XALT_ETC_DIR. We'll refer to this as XALT_ETC_DIR for the remainder.

3. Third, the site had decide on whether they want to support the reverseMap functionality. The reverseMap is our code word for mapping paths to libraries or executables back to the appropriate modulefile (if it exists). This ability requires the creation of the module reverseMap, which is a result of running the "spider" utility from the Lmod [9] module system. One does not have to replace the TCL module system with Lmod to get this functionality, it just needs to be installed.

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

   Furthermore, if a site has multiple code launchers or linkers, then the site has to decide how they want to intercept them. There are a variety of ways this can be accomplished, discussed below in the section on "*Intercepting linkers and job launchers.*" One possible way to do this is with Lmod in conjunction with XALT, but requires replacing TCL module system with Lmod.

   We'll assume that Lmod has been installed if desired.

Data transmission

A very important note is that the transmission method can be set at configuration time (defaults to file), but also can be set at any time later with an environment variable. So a site can first set it up to use json files, but then can switch to syslog or direct to database at any time. This is done with the XALT_TRANSMISSION_STYLE environment variable with options of: file, syslog, directdb. Note that if there is a typo on the transmission style setting, it reverts to the file method.

File method

By default, XALT uses json files written to the user's home directory in the ~/.xalt.d/ directory.

- The script sbin/xalt_file_to_db.py reads the json files in ~/.xalt.d and loads them into the database. It will delete files with --delete option.

    - Where ever you run this command, it needs to find the xalt_db.conf file.
    - XALT_USERS: colon separated list of users to find the json file. This can be used to to target test users instead of all users. If not set, then all users (found from getent command) directories will be searched and processed. In most circumstances, this command will need to be run as root.
- This isn't called automatically anywhere in XALT as we expect this to be called in a cron job by a privileged user to put data into the database.
    - The wrappers could be modified to call this function, but then the "direct to database" option would be better.

The file transmission method has a big concern from the developers - what happens in the case of someone running thousands upon thousands of small jobs concurrently on a large machine? This could generate thousands and thousands of json files in the home directory of that user -- will there be enough space? enough inodes? too much NFS traffic in general? If this is not a concern at your site, then the file method should be sufficient

syslog method

The syslog method for transmission is very similar to the file method. First the data goes to syslog, and then asynchronously (via a cron job) the data must be collected and put into the database.

We expect this method to be the best for production use. To use syslog (and in this context we mean rsyslog since that is all we have tested), you will need to

- set up a configuration file for syslog and place in /etc/rsyslog.d/ that we called xalt_syslog.conf

  ```
  $MaxMessageSize 256k
  if $programname contains 'XALT_LOGGING' then /var/log/xalt.log
  & ~
  ```

  This example shows that the log file is set up as /var/log/xalt.log. Note that /var is probably local for the node where the linker or job launcher is run. This is fine, but you will have to run the syslog parser on each node for this setup. Alternatively, it might be easier if you had say one node/server where all the log files could be put by syslog and then you would only have to run the parser on that node, but for each file.

  Also note that 64k is the maximum message size as given. We have already hit a case where a link line was larger than this and this results in the XALT log message being incomplete and as a result the parser will have to skip those entries as incomplete. We previously used 64k because all link lines (until this case) were much smaller and all our run examples were larger but less than 256k.
- modify /etc/rsyslog.conf to use this new configuration

  ```
  # Include all config files in /etc/rsyslog.d/
  $IncludeConfig /etc/rsyslog.d/*.conf
  ```
- restart rsyslog
- set up rotation on the /var/log/xalt.log log file with a logrotate configuration file like

  ```
  /etc/logrotate.d>  cat xalt
  ```

```
/var/log/xalt.log{
  copytruncate
  rotate 4
  daily
  create 0644 root root
  missingok
}
```

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

- use xalt_syslog_to_db.py to collect data from syslog

```
python xalt_syslog_to_db.py /var/log/xalt.log.1
```

We assume the installers know that all of these steps will have to be done on each of the nodes where the linker and the job launcher (mpirun, aprun, etc) will be run.

direct to database method

The direct to database method is probably the simplest. It basically inserts the data into the database as it is being collected. This is how the ALTD infrastructure worked for years.

This method has the security concerns of "users modify the database directly". Yes they do, but only through the wrappers and most if not all users have no idea that is happening. To ameliorate this concern, it is important to use an "insert-only" account for these transactions. There are also concerns about many database transactions going on all the time. Anecdotally, this has never been reported as a problem on several large HPC installations with thousands of users, but it certainly could be depending on how the database is set up.

To use this method, just set XALT_TRANSMISSION_STYLE to directdb. Really nothing else to do.

Mixing methods

It is possible to mix two methods, though clunky at best. First, the default method is set at configuration time (and defaults to file.) You can then set XALT_TRANSMISSION_STYLE in the modulefile or where ever appropriate to override configuration setting. Finally, you could also hardcode the environment variable in the linker or job launcher script. For example, you could set the env var to syslog in the modulefile, but set the env var to directdb in the job launcher script with the result that linker information goes to syslog and job launch information goes directly to the database.

Env Var BlackList

There is an environment variable blacklist, list of variables discarded, in the xalt_run_submission.py code. You can add or remove from this list, but note that as long as

this is part of the xalt_run_submission.py code, code updates will overwrite any changes a site does.

Creating the Reverse Map

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

If you have your modulefiles set up with a one-to-one to mapping of modules to package installations, then Lmod can probably create the reverse map without issue. But on some machines, a module points (with appropriate if tests) can point to a variety of installations and set environment variables depending on the currently loaded compilers and MPI. In this scenario, the reverse map can be created, but it is a looser reverse map with many-to-one relationships.

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

The reverse map needs to be created/updated per machine every time a new modulefile or package is installed. So it either has to become part of the software installation process, or run as a cron job every week for example.

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

Examples

Simple command to create reverseMap

spider -o jsonReverseMapT $LMOD_MODULEPATH > rmapD/jsonReverseMapT.json

Cray script to create reverseMap

Below is an example script, darter_build_rmapT.sh, for a Cray XC30 that uses Lmod (namely the spider utility) to create the reverseMap. This is provided in the contrib/build_reverseMapT_cray/ directory.

```
#!/bin/bash

################################################################################
# To get this to work please do the following:
#   a) Modify the Site Specific Settings to match your site
#   b) Make sure that this script and the python script
#      "merge_json_files.py" are in the same directory.
#   c) Make sure the module command is defined by using $BASH_ENV
#      or define the module command here.
#   d) Make sure that LMOD_DIR is defined as well
#         (it is defined by $BASH_ENV).
#
################################################################################
#  Site Specific Setting
################################################################################



BASE_MODULE_PATH=/opt/modulefiles:/opt/cray/ari/modulefiles:/opt/cray/craype/default/modulefiles:/sw/local/modulefiles:/sw/xc30/modulefiles:/sw/xc30/modulefiles:/sw/local/modulefiles:/opt/cray/craype/default/modulefiles:/opt/cray/modulefiles:/opt/modulefiles:/cm/local/modulefiles:/cm/shared/modulefiles

  ADMIN_DIR=$HOME/XALT/use_xalt
  RmapDir=$ADMIN_DIR/reverseMapD

  PrgEnvA=("PrgEnv-cray"  "PrgEnv-gnu" "PrgEnv-intel")

  moduleA=( "PrgEnv-cray/5.2.25" "PrgEnv-gnu/5.2.25" "PrgEnv-intel/5.2.25")

  ################################################################################
  # must define the module command and $LMOD_DIR:
  ################################################################################

  if [ -f "$BASH_ENV" ]; then
    source $BASH_ENV
  fi

################################################################################
#  End Site Specific Setting
################################################################################
```

```
if [ ! -d $RmapDir ]; then
  mkdir -p $RmapDir
fi

SCRIPT_DIR=$(cd $(dirname $(readlink -f "$0")) && pwd)
PATH=$SCRIPT_DIR:$LMOD_DIR:$PATH

cd $RmapDir

module unload "${PrgEnvA[@]}" 2> /dev/null
prev=""
for m in "${moduleA[@]}"; do
    sn=$(dirname $m)
    v=${m##*/}

    module unload $prev  2> /dev/null
    module load $m     2> /dev/null
    prev=$m

    echo -n '-'
    spider --preload -o jsonReverseMapT $BASE_MODULE_PATH >
rmapT_${sn}_${v}.JSON
    echo -n '*'

done

echo "*"

OLD=$RmapDir/jsonReverseMapT.old.json
NEW=$RmapDir/jsonReverseMapT.new.json
RESULT=$RmapDir/jsonReverseMapT.json

set -x
merge_json_files.py rmapT_*.JSON > $NEW
if [ "$?" = 0 ]; then
  chmod 644 $NEW
  if [ -f $RESULT ]; then
    cp -p $RESULT $OLD
  fi
  mv $NEW $RESULT
fi

rm rmapT_*.JSON
```

SCRIPTS to put syslog or json file data into DB

put examples here….

Database Queries

There are several tables in the XALT database and many ways to get information from them. In this section, we will provide many examples of queries that we think will be useful for many sites.

Database tables

List of tables in the XALT database. We have two types: (1) tables that hold information about a link, or a library, or a run and (2) tables that provide a "many-to-many" relationship between the data tables.

Join tables	**Data tables**
join_run_env	xalt_env_name (env vars from a run)
join_run_object	xalt_run (job launch details, exe, num cores, timestamp)
join_link_object	xalt_link (link line details)
join_function_link	xalt_object (library objects found in links and runs)
join_function_object	xalt_function (functions that need to be resolved by ext libs)

These are listed here to provide the system administrators a quick list. The following command will also provide the list of tables.

```
mysql> show tables;
```

If you want more information on the columns in any table, do the following:

```
mysql> show columns from <table>;
```

where <table> is one of the tables above.

Sample queries

For some of the reports below, you might need to change the maximum limit on GROUP_CONCAT

```
mysql> SET SESSION group_concat_max_len = 1000000;
```

Many of these queries will not have a date range specified. But clearly that might be desired. In most of them, it is fairly easy to add a where clause like

```
where date >= '2014-10-01' and date <= '2014-10-31'
```

In many of the queries that follow, you will see a core-hour calculation where most of the examples use

```
ROUND(SUM(run_time*num_cores)/3600)
```

as the method for computing it. This assumes that "num_cores" represents the total number of cores the code used on the platform. On some machines that will be true (Cray), while other machines (generic cluster) one will want to use

```
ROUND(SUM(run_time*num_cores*num_threads)/3600)
```

This is because on some machines, the total number of cores used by the job launcher can be determined by the options given to the job launcher (like aprun on a Cray). But on a generic cluster, the total number of cores used an executable will be determined by the -n option to mpirun times the number of threads in OMP_NUM_THREADS.

Examples

This query will return a report first sorted in descending order the most used libraries (objects) for a specified machine *based on number of times it appears in a link.*

```
mysql> SELECT object_path, module_name, count(date) AS cnt from
xalt_link, join_link_object, xalt_object where build_syshost='darter'
AND xalt_link.link_id = join_link_object.link_id AND
join_link_object.obj_id = xalt_object.obj_id GROUP BY object_path
ORDER BY cnt DESC;
```

The previous query can easily be changed to be grouped by occurrences of the modulefile name.

```
mysql> SELECT module_name, count(date) AS cnt from xalt_link,
join_link_object, xalt_object where build_syshost='darter' AND
xalt_link.link_id = join_link_object.link_id AND
join_link_object.obj_id = xalt_object.obj_id GROUP BY module_name
ORDER BY cnt DESC;
```

And above again without NULL module_name entries included:

```
mysql> SELECT module_name, count(date) AS cnt from xalt_link,
join_link_object, xalt_object where build_syshost='darter' AND
module_name is not NULL AND xalt_link.link_id =
```

```
join_link_object.link_id AND join_link_object.obj_id =
xalt_object.obj_id GROUP BY module_name ORDER BY cnt DESC;
```

The following query provides a report of modulefile usage based on shared library usage at run time ordered by number of occurrences used. The corehour calculation might need to include num_threads depending on what you are storing in num_cores and num_threads. If you mostly have static-ly built executables, see the next query after this one.

```
mysql> SELECT xalt_object.module_name, count(date) AS Jobs,
ROUND(SUM(run_time*num_cores)/3600) as TotalSUs from xalt_run,
join_run_object, xalt_object where xalt_run.syshost='mars' AND
xalt_object.module_name is NOT NULL AND xalt_run.run_id =
join_run_object.run_id AND join_run_object.obj_id =
xalt_object.obj_id AND date >= '2014-11-01' AND date <= '2014-11-09'
GROUP BY xalt_object.module_name ORDER BY Jobs DESC;
+----------------------+------+----------+
| module_name          | Jobs | TotalSUs |
+----------------------+------+----------+
| intel/2011_sp1.11.339 | 4147 |   191118 |
| openmpi/1.6.1-intel   | 1382 |    63706 |
| torque/4.2.6          | 1356 |    41263 |
| cuda/5.0              | 1259 |    29486 |
| openmpi/1.6.1-gnu     |   51 |       10 |
| intel/2011_sp1.8.273  |   24 |        5 |
| cuda/4.1              |    2 |       58 |
+----------------------+------+----------+
```

As opposed to the previous query, this one counts module files based on static library usage at runtime.

```
mysql> SELECT xalt_object.module_name, count(xalt_run.date) AS Jobs,
ROUND(SUM(run_time*num_cores)/3600) as TotalSUs from xalt_run,
xalt_link, join_link_object, xalt_object where
xalt_run.syshost='darter' AND xalt_object.module_name is NOT NULL AND
xalt_run.uuid = xalt_link.uuid AND xalt_link.link_id =
join_link_object.link_id AND join_link_object.obj_id =
xalt_object.obj_id AND xalt_run.date >= '2014-11-01' AND
xalt_run.date <= '2014-11-09' GROUP BY xalt_object.module_name ORDER
BY Jobs DESC;
+-----------------------------------+-------+----------+
| module_name                       | Jobs  | TotalSUs |
+-----------------------------------+-------+----------+
| alps/5.2.1-2.0502.8712.10.32.ari  | 26458 |   258684 |
| cray-mpich/7.0.3                  | 26456 |   259040 |
| ugni/5.0-1.0502.9037.7.26.ari     | 13229 |   129342 |
```

```
| wlm_detect/1.0-1.0502.51217.1.1.ari | 13229 |   129342 |
| udreg/2.3.2-1.0502.8763.1.11.ari    | 13229 |   129342 |
| xpmem/0.1-2.0502.51169.1.11.ari     | 13229 |   129342 |
| pmi/5.0.5-1.0000.10300.134.8.ari    | 13227 |   129341 |
| gcc/4.8.1                           | 10868 |    59680 |
| dmapp/7.0.1-1.0502.9080.9.32.ari    | 10852 |    59675 |
| rca/1.0.0-2.0502.51491.3.92.ari     | 10852 |    59675 |
| fftw/3.3.4.0                        |  3123 |     1482 |
| cray-libsci/13.0.1                  |  2357 |    69848 |
| craype-intel-knc                    |  1758 |      522 |
| hdf4/4.2.9                          |  1180 |      667 |
| cray-netcdf/4.3.1                   |   586 |      174 |
| cray-hdf5/1.8.12                    |   586 |      174 |
| szip/2.1                            |   295 |      167 |
| fftw/3.3.0.4                        |   274 |    78293 |
| gcc/4.8.2                           |   156 |    44739 |
| cray-mpich/6.3.0                    |    84 |    22373 |
| cray-libsci/12.2.0                  |    79 |    22370 |
| cp2k/2.5.1                          |    78 |    22369 |
| gcc/4.9.1                           |    12 |      155 |
| cray-hdf5/1.8.13                    |     5 |      236 |
| pmi/5.0.3-1.0000.9981.128.2.ari     |     2 |        1 |
+-------------------------------------+-------+----------+
25 rows in set (2.03 sec)
```

Quick look at what compilers are being used:

```
mysql> select link_program, count(*) from xalt_link group by
link_program;
```

Add in the build_syshost field to separate out the results by machine (if multiple machines.)

Simple join of xalt_run and xalt_object tables to produce a report of each job, the executable, and the libraries the code was run with

```
mysql> SELECT job_id, exec_path, GROUP_CONCAT(object_path) FROM
xalt_run, xalt_object WHERE user LIKE 'user' GROUP BY job_id;
```

Simple join of xalt_link and xalt_object tables to produce a report of each executable and the libraries the code was built with

```
mysql> SELECT exec_path, GROUP_CONCAT(object_path) FROM xalt_link,
xalt_object WHERE build_user LIKE 'user' GROUP BY exec_path;
```

By account (project), list executables [and syshost and dates run]:

```
mysql> select account, exec_path, syshost, group_concat(date),
count(exec_path) from xalt_run group by exec_path order by account;
```

Most used code by unique user (**not sure this is right yet)**:

```
mysql> SELECT U.account, U.exec_path, U.cnt FROM (SELECT account,
exec_path, user, COUNT(*) AS cnt FROM xalt_run GROUP BY account,
exec_path, user) AS U GROUP BY U.account, U.exec_path ORDER BY U.cnt;
```

Identify users who linked in a certain library (fftw/3.3.0.2 for example which had a bug)

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

And then see an executable ran that was linked with that library:

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

How did someone build their program some time ago (assume the user is **user1** and the code is **hyperslab**):

```
mysql> select xalt_link.* from xalt_link where build_user like
'%user1%' AND exec_path like '%hyperslab%';
```

Assuming found, this will produce one or more results with a "**link_id**". Use these link_id's to do a subsequent query which will produce a list of libraries/objects linked into the code for each link_id. Below we use a link_id of 4.

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

Multiple machines

When you have multiple machines, it is often the case that you will want your results sorted by machine or you have to run a query for each machine making sure syshost matches the machine you are interested in.

Below are a few examples when you want to do one query that separates the output by machine:

The following query will basically sort your codes into one of three categories: script, user built, or center (system) built. [User code is determined by the absence of a modulefile name associated with the executable. This could be done other ways as well - like searching for a certain path and if anything is in that path, then it is a system executable for instance.] It then counts up how many runs

and also the associated cputime (in hours). NOTE: you may need to include num_threads in the calculation for total CPU time (depends on your setup.)

```
mysql> SELECT syshost, exec_type, IF(module_name IS null, "user",
"system") AS codetype, ROUND(SUM(run_time*num_cores/3600)) AS cput,
count(exec_type) AS jobs FROM xalt_run WHERE date >= '2014-10-01'
GROUP BY syshost, exec_type, codetype ORDER BY syshost, cput DESC;
+--------------+-----------+----------+--------+------+
| syshost      | exec_type | codetype | cput   | jobs |
+--------------+-----------+----------+--------+------+
| darter       | binary    | user     | 155885 | 9130 |
| darter       | script    | user     |   9538 |   37 |
| darter       | binary    | system   |      0 |    3 |
| mars         | binary    | user     | 142897 | 2962 |
| mars         | binary    | system   |      2 |   16 |
+--------------+-----------+----------+--------+------+
5 rows in set, 1 warning (0.02 sec)
```

For the same report as above, but one machine only (say 'darter') and a percentage breakdown for cputime, we have

```
mysql> SELECT exec_type, IF(module_name IS null,"user","system") AS
codetype, LPAD(FORMAT(SUM(run_time*num_cores/3600),2),11,' ') AS
cput, LPAD(FORMAT(SUM(run_time*num_cores)/t.total*100,2),8,' ') AS
percentage, COUNT(1) AS jobs FROM xalt_run, (SELECT
SUM(run_time*num_cores) AS total FROM xalt_run WHERE date >=
'2014-10-01' AND xalt_run.syshost = 'darter') AS t WHERE date >=
'2014-10-01' AND syshost = 'darter' GROUP BY exec_type, codetype;
+-----------+----------+-------------+------------+------+
| exec_type | codetype | cputime     | percentage | jobs |
+-----------+----------+-------------+------------+------+
| binary    | system   |        0.27 |     0.00   |    3 |
| binary    | user     |  155,884.89 |    94.23   | 9130 |
| script    | user     |    9,537.76 |     5.77   |   37 |
+-----------+----------+---------0---+------------+------+
3 rows in set, 1 warning (0.03 sec)
```

You can see that WHERE clause has to be specified twice. This is because they are basically two separate queries and you have to specify the same WHERE clause to have the same set of data. If you only have one machine's data, then both syshost specifier statements can be removed. NOTE: you may need to include num_threads in the calculation for total CPU time (depends on your setup.) And if you have mulitple machines that calculate CPU time differently, then be careful doing a combined report like this.

The following shows a “link_program” usage report. Be careful with this, as we cannot distinguish when a code has multiple languages - we can only detect the compiler that invoked the linker.

```
mysql> SELECT link_program,  syshost,
ROUND(SUM(run_time*num_cores/3600)) FROM xalt_link, xalt_run WHERE
xalt_link.uuid = xalt_run.uuid GROUP BY link_program, syshost;
+--------------+---------------+-----------------------------------+
| link_program | syshost     | round(sum(run_time*num_cores/3600)) |
+--------------+---------------+-----------------------------------+
| driver.cc    | darter        |                                 0 |
| ftn_driver   | darter        |                             44231 |
| g++          | darter        |                                 1 |
| g++          | mars          |                              1026 |
| gcc          | darter        |                               585 |
| gfortran     | darter        |                                 3 |
| icc          | darter        |                                 1 |
| icc          | mars          |                                 0 |
| icpc         | darter        |                                 0 |
| icpc         | mars          |                                 1 |
| ifort        | darter        |                             38325 |
| ifort        | mars          |                             10531 |
| pgfortran    | mars          |                               596 |
+--------------+---------------+-----------------------------------+
15 rows in set (0.03 sec)
```

NOTE: you may need to include num_threads in the calculation for total CPU time (depends on your setup.)

The following shows how to get a list of the link_program and the associated executables and the amount of cpu hours they used, and if a modulefile corresponds to the executable (likely a center provided executable.) Note that the “substring_index” is used to strip off the path to the executable leaving just the executable name. This works by tie-ing the “uuid” during the link phase with the “uuid” grabbed from the xalt section header placed in the code retrieved by the code launcher. if the code was built with XALT loaded or if the code is run without use the code launcher, then there is no way to tie them together in the report.

```
mysql> SELECT link_program,
SUBSTRING_INDEX(xalt_link.exec_path,'/',-1) AS code,
ROUND(SUM(run_time*num_cores/3600)),  module_name FROM xalt_link,
xalt_run WHERE xalt_link.uuid = xalt_run.uuid GROUP BY link_program,
code;
+--------------+-----------------------------------------------------
-------------+--------------------------------------+-------------+
| link_program | code
| ROUND(sum(run_time*num_cores/3600)) | module_name |
```

```
+--------------+-------------------------------------------------------
-------------+-------------------------------------+-------------+
| ftn_driver   | StationaryAccretionShockAnalysis_Darter_Cray
|                                    1 | NULL        |
| ftn_driver   | StationaryAccretionShockCutout_Darter_Cray
|                                  239 | NULL        |
| ftn_driver   | ctqmc.e
|                                 3344 | NULL        |
| ftn_driver   | dca.e
|                                   77 | NULL        |
| ftn_driver   | df.e
|                                 2585 | NULL        |
| g++          | eoc
|                                    1 | NULL        |
| g++          | hoomd
|                                 1026 | NULL        |
| gcc          | driver
|                                  537 | NULL        |
| gcc          | orted
|                                   47 | NULL        |
```

NOTE: you may need to include num_threads in the calculation for total CPU time (depends on your setup.)