XALT: Job-Level Usage Data on Today's Supercomputers.

Robert McLay

June 19, 2020

XALT: Outline

- ► What is XALT and what it is not?
- ► Brief History
- ► How it works: Three Parts
- ► What can you do with it?
- ► How can I help you?

Understanding what your users are doing

- ▶ What programs, libraries are your users using?
- ▶ What imports from R, MATLAB, Python?
- ▶ What are the top programs by core-hours? by counts ? by users?
- ▶ System, User or Built by Other executables?
- ▶ Are Executables implemented in C/C++/Fortran?
- ▶ Track MPI task and/or Threading ($OMP_NUMTHREADS)
- ▶ Function Tracking
- ▶ Census Taker, Not a performance tool!

Brief History

- ▶ XALT was an U.S. NSF funded project (M. Fahey & R. McLay)
- ▶ Work continued at TACC: too useful
- ▶ Originally only tracked MPI execution.

Design Goals

- ▶ Be extremely light-weight
- ▶ How many use a library or application?
- ▶ What functions are users calling in system modules
- ▶ Collect Data into a Database for analysis.

How does XALT work?

- ► LD Wrapper
- ► ELF Trick to track execution
- ► Signal handling
- ► Generate Json records
- ► Transport to DB
- ► Analyze database.

Design: LD Wrapper

- ▶ XALT wraps the linker to enable tracking of exec's
- ▶ The linker (ld) wrapper intercepts the user link line.
- ▶ Generate assembly code: key-value pairs
- ▶ Capture tracemap output from ld
- ▶ Transmit collected data in *.json format
- ▶ Optionally add codes that executes before main() and after main() completes for static builds.

Signal handling

- XALT registers signal handlers before main()
- SIGSEGV, SIGFPE, SIGTERM, etc
- User code can override with their own.

Elf Trick (I)

- ► ELF is the binary format for Linux
- ► ELF has many hooks
- ► XALT uses two hooks to run before/after main()

ELF Trick (II)

```c
#include <stdio.h>
void myinit(int argc, char **argv)
{ fprintf(stderr, "This is run before main()\n"); }
void myfini()
{ fprintf(stderr,"This is run after main()\n");}
static __attribute__((section(".init_array")))
    typeof(myinit) *__init = myinit;
static __attribute__((section(".fini_array")))
    typeof(myfini) *__fini = myfini;
```

ELF Trick (III)

```
% LD_PRELOAD=./libxalt.so ./hello
This is run before main()
Hello World!
This is run after main()
```

Transmission to DB

- ▶ File: collect nightly/hourly/...
- ▶ Syslog: Use Syslog filtering (or ELK)
- ▶ Curl: send directly

Lmod to XALT connection (I)

- ▶ Optional support to connect paths to modules
- ▶ Lmod spider walks entire module tree.
- ▶ Can build a reverse map from paths to modules
- ▶ Can map program & libraries to modules.
- ▶ /opt/apps/i15/mv2_2_1/phdf5/1.8.14/lib/libhdf5.so.9 $\Rightarrow$ phdf5/1.8.14(intel/15.02:mvapich2/2.1)
- ▶ Also helps with function tracking.
- ▶ Tmod Sites can still use Lmod to build the reverse map.

Lmod to XALT connection (II)

- ▶ Need XALT's ld before compiler's ld
- ▶ User loads a new compiler module?
- ▶ Lmod support path priority:
- ▶ `prepend_path{"PATH", "/.../xalt/bin", priority = 100}`

Lmod path priority (I)

```
$ type -a ld
ld is /opt/apps/xalt/xalt/bin/ld
ld is /opt/apps/gcc/8.3.0/bin/ld
ld is /bin/ld
```

Lmod path priority (II)

```
$ module load gcc/9.1.0; type -a ld
ld is /opt/apps/xalt/xalt/bin/ld
ld is /opt/apps/gcc/9.1.0/bin/ld
ld is /bin/ld
```

XALT LD Wrapper Support w/o Lmod

- ▶ Move Compiler's ld to ld.x
- ▶ Small change in XALT's ld to find ld.x
- ▶ Must do this for every new compiler install.
- ▶ Or put xalt's bin path before compiler path in module
- ▶ Or treat every executable like `ls` or ABAQUS

Installing XALT

- ▶ Easy: `./configure ...; make install`
- ▶ Harder: Reverse Map from Lmod?
- ▶ Harder: Site config.py file
- ▶ Harder: Setup Transport Json records
- ▶ Harder: Setup VM to hold database
- ▶ Or: Setup your own way to handle the data

Site config.py (I)

- ▶ Each site must configure to match their setup
- ▶ Compute node names?
- ▶ What executables to track or ignore?
- ▶ What python packages to track or ignore?
- ▶ What sampling rules to use?

Site config.py (II)

- ▶ XALT use config.py to create *.h, *.lex *.py files during build.
- ▶ Provides xalt_configuration_report C++ program to know how configured.
- ▶ Config.py file only used when building XALT.
- ▶ Any changes to Config.py require a re-install of XALT.

Hostname, Path and Env Filtering

- ▶ Uses FLEX to compile in patterns
- ▶ Use regex expression to control what to keep and ignore.
- ▶ These Flex ⇒ C routine provide fast regex parsing.

TACC_config.py

```
hostname_patterns = [
  ['KEEP', '^c[0-9][0-9][0-9]-[0-9][0-9][0-9]\..*']
  ['KEEP', '^nid[0-9][0-9][0-9][0-9][0-9].*']
]
path_patterns = [
    ['PKGS',  r'.*\/python[0-9.]*'],
    ['PKGS',  r'.*\/R'],
    ['KEEP',  r'^/\usr\/bin\/ddt'],
    ['SKIP',  r'^\/bin\/.*'],
]
env_patterns = [
    [ 'SKIP', r'^MKLROOT=.*' ],
    [ 'SKIP', r'^MKL_DIR=.*' ],
    [ 'KEEP', r'^I_MPI_INFO_NUMA_NODE_NUM=.*'],
]
```

How sampling works

- ▶ Changed design to deal with the overload of XALT data
- ▶ Only generate records if plan to save.
- ▶ All Non-mpi executions only produce end records.
- ▶ Small MPI execution sample, Large MPI executions record.

Sampling Non-MPI programs

- ▶ XALT has sampling rules (site configurable!)
- ▶ TACC rules are:
- ▶ 0 mins $<$ 30 mins $\Rightarrow$ 0.01% recorded
- ▶ 30 mins $<$ 120 mins $\Rightarrow$ 1% recorded
- ▶ 120 mins $< \infty \Rightarrow$ 100% recorded
- ▶ Can now track/sample perl, awk, sed, gzip etc

Sampling MPI programs

- Some users are using many short MPI programs to train Deep Learning engine
- TACC rules are:
- Task counts < 128 tasks are sampled
  - 0 mins < 15 mins $\Rightarrow 0.01\%$ recorded
  - 15 mins < 30 mins $\Rightarrow 1\%$ recorded
  - 30 mins $< \infty \Rightarrow 100\%$ recorded
- Task counts $>= 128$ task are always recorded independent of runtime.
- Need to Capture long running MPI progs that never end.

Using XALT Data

► Targetted Outreach: Who will be affected
► Largemem Queue Overuse
► XALT and TACC-Stats
► Who is running NWChem or ...?
► Function Tracking: Who or What is using MPI-3?

Who is using MPI-3: MPI_I*

What codes link in MPI-3 routines. (Not necessarily Run)

Function Name	N Users	N Progs
MPI_Ibarrier	8	4
MPI_Ialltoall	24	4
MPI_Ineighbor_alltoall	4	3

What is new with XALT?

- Tracking R, Python, MATLAB
- Signal handler
- Optionally Track GPU Usage
- Track Singularity Container Usage
- Removed three system calls for improved speed

Tracking R packages

- ▶ XALT can now track R package usage
- ▶ James McComb & Michael Scott from IU developed the R part
- ▶ They do this by intercepting the "imports"

Tracking Python packages

- ► Help from Riccardo Murri
- ► sitecustomize.py
- ► It is run by any Python if found.
- ► All Pythons uses sys.meta_path to locate files to import
- ► Can register object to capture imports.
- ► Just add location to PYTHONPATH

Filtering python packages

```
{ 'k_s':'SKIP','kind':'path','patt': r"^[^/]" },
{ 'k_s':'SKIP','kind':'name','patt': r"^_"     },
{ 'k_s':'SKIP','kind':'name','patt': r".*\." },
{ 'k_s':'KEEP','kind':'path','patt': r".*/.local/" },
```

New program: xalt_extract_record

► This program reads the watermark.
► Find out who built this program on what machine
► Find out what modules where used.
► Where was it build.

Example of xalt_extract_record output

```
*****************************************
XALT Watermark: hello
*****************************************
Build_CWD            /home/user/t/hello
Build_Epoch          1510257139.4624
Build_LMFILES        /apps/mfiles/intel/17.0.4.lua:...
Build_LOADEDMODULES  intel/18.0.4:impi/18.0.3:TACC:..
Build_OS           Linux 3.10.0-514.26.2.el7.x86_64
Build_Syshost        stampede2
Build_UUID              586d5943-67eb-480b-a2fe-
35e87a1f22c7
Build_User           mclay
Build_compiler       icc
Build_date           Fri Jun 09 13:52:19 2019
Build_host         c455-011.stampede2.tacc.utexas.edu
XALT_Version         2.7
```

New Feature: Track GPU usage

- ▶ Optionally, XALT can know if a GPU was used.
- ▶ XALT will only know if one or more GPU's were accessed
- ▶ No performance data
- ▶ Thanks to Scott McMillan from NVIDIA for the contribution.

New Feature: Track Singularity Container Usage

- ▶ Sites can configure their Singularity script to include XALT
- ▶ It works well with syslog, file or curl transfer of data
- ▶ Thanks to Scott McMillan from NVIDIA for the contribution.

Debugging XALT

```
$ XALT_TRACING=yes XALT_SAMPLING=no ./hello

myinit(0/1,LD_PRELOAD,/path/to/hello){
    -> Setting up signals
    -> Leaving myinit
}

Hello World!

myfini(0/1,LD_PRELOAD,/path/to/hello){
    -> XALT_SAMPLING = "no" All programs tracked!
  Recording State at end of scalar user program:
    XALT_EXECUTABLE_TRACKING=no /path/to/xalt_run_submission ...
    -> leaving myfini

  xalt_run_submission(.zzz) {
    building UUID: ...
    Extracted recordT from executable
    Built userT, userDT, scheduler: SLURM
    Using XALT_TRANSMISSION_STYLE: file
    cmdlineA: ["./hello"]
    Built json string
  }
}
```
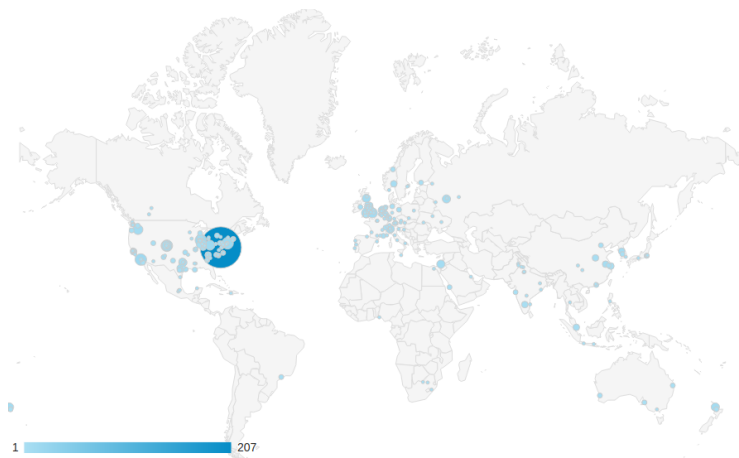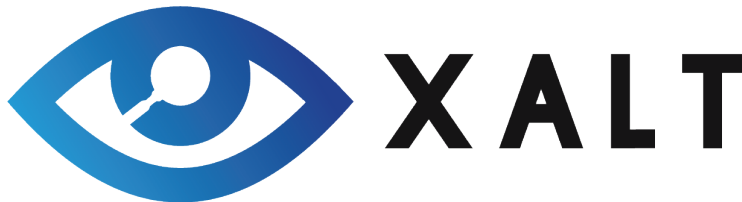
TACC

XALT Doc usage by Country

1 461
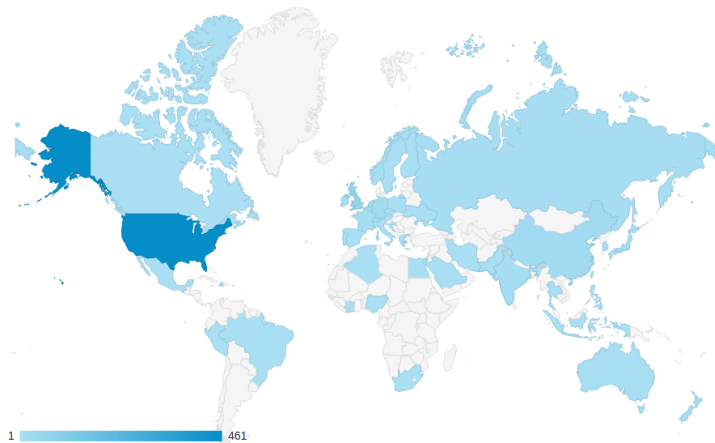
XALT Doc usage by City

1   207

Conclusion

- ▶ Lmod:
  - ▶ Source: github.com/TACC/lmod.git, lmod.sf.net
  - ▶ Documentation: lmod.readthedocs.org
- ▶ XALT:
  - ▶ Source: github.com/xalt/xalt.git, xalt.sf.net
  - ▶ Documentation: XALT 2 $\Rightarrow$ xalt.readthedocs.org
  - ▶ Join mailing list:
    https://sourceforge.net/projects/xalt/lists/xalt-users