



TEXAS ADVANCED COMPUTING CENTER

WWW.TACC.UTEXAS.EDU



TEXAS

The University of Texas at Austin

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

Robert McLay

January 27, 2021

XALT: Outline



XALT

- ▶ What is XALT and what it is not?
- ▶ How it works: Three Parts
- ▶ Memory allocation issues
- ▶ Container issues
- ▶ Conclusions

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
- ▶ 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.
- ▶ Having the LD Wrapper is helpful but not required.

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)

hello.c:

```
#include <stdio.h>
int main() { printf("Hello World!\n"): return 0; }
```

xalt.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)

```
% ./hello
```

```
Hello World!
```

```
% gcc -c -fPIC xalt.c;
```

```
% gcc -o libxalt.so -fPIC -shared xalt.o
```

```
% 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)`
- ▶ Tmod Sites can still use Lmod to build the reverse map.

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 \Rightarrow 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:
 - ▶ $0 \text{ mins} < 30 \text{ mins} \Rightarrow 0.01\% \text{ recorded}$
 - ▶ $30 \text{ mins} < 120 \text{ mins} \Rightarrow 1\% \text{ recorded}$
 - ▶ $120 \text{ mins} < \infty \Rightarrow 100\% \text{ recorded}$
- ▶ 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.

XALT is now linking with everything!

- ▶ I feel like a developer on every team.
- ▶ XALT shares name space -> obfuscation
- ▶ XALT shares memory allocation
- ▶ Containers: XALT cannot depend on all system libs.

Memory Collusion w/ user code (I)

- ▶ User serial code creates a link list
- ▶ Their program failed to NULL end of list.
- ▶ All memory is zeroed before program start.
- ▶ User works w/o XALT! Fails with!
- ▶ Result 1: Modified user code
- ▶ Result 2: Any memory freed by XALT is zeroed.

Memory Collusion w/ user code (II)

- ▶ Not all user programs allocate and free memory correctly (Surprise!)
- ▶ XALT would sometimes fail when freeing memory after `main()`
- ▶ Result: XALT allocates but doesn't free memory after `main()`

Containers

- ▶ XALT requires libuuid.so on host
- ▶ Not on Containers
- ▶ XALT has to use dlopen()/dlsym() to use libuuid.so

Containers (II)

- ▶ XALT copies system libuuid.so to XALT install dir.
- ▶ XALT does:
`dlopen("XALT_INSTALL_DIR/lib64/libuuid.so")`
- ▶ XALT then uses `dlsym()` to connect to libuuid routines

Using XALT Data

- ▶ Targeted Outreach: Who will be affected
- ▶ Largemem Queue Overuse
- ▶ XALT and TACC-Stats
- ▶ Who is running NWChem or ...?

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 were used.
- ▶ Where was it built.

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

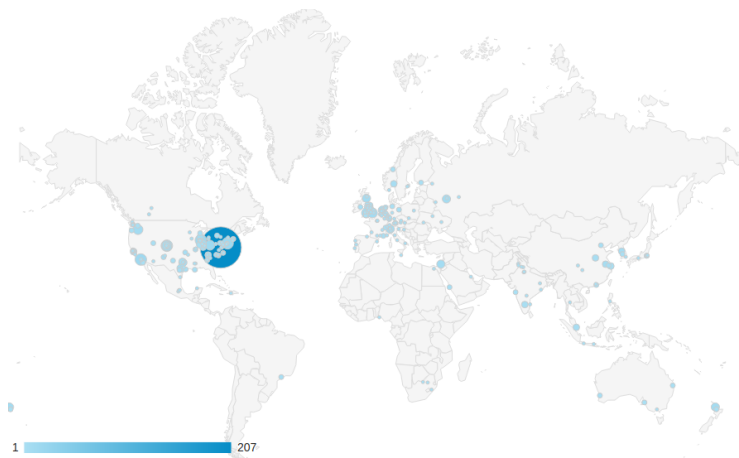
```
$ XALT_TRACING=yes XALT_SAMPLING=no ./hello
```

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

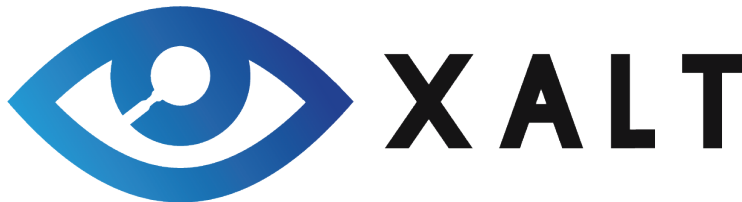
Hello World!

```
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 ...  
  
    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  
    }  
    -> leaving myfini  
}
```

XALT Doc usage by City



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>