TEXAS ADVANCED COMPUTING CENTER

WWW.TACC.UTEXAS.EDU

TEXAS

The University of Texas at Austin

Extracting Useful Data from XALT DB

Robert McLay

Jan. 20, 2022

XALT: Outline

- XALT Parts: Generating and Storing
- Quick Discussion about what is stored
- Caveats about XALT Records
- xalt_usage_report.py
- Kinds of reports available
- What kinds of reports would sites like?
- Future Topics for XALT zoom mtg.

XALT Generating and Storing

- ALL programs on system have XALT via LD_PRELOAD
- Some prgms generate *.json records
- These records can be written to a MySQL DB (not required)
- Afterward *.json records are deleted.

Caveats about XALT records

- Core-Hours not Node-Hours
- XALT knows about mpi tasks and threads
- It doesn't know if a node is shared.
- User might run multiple single core prgms on a node
- User might run ingle core prgms on one or more nodes
- XALT runs inside each program.
- It is blind to what happens outside of user prgm.
- It would require a daemon on each node to know.

XALT time records won't match Accounting

- XALT will filter out or sample programs
- It won't catch them all.
- XALT can double count (rarely)
- If user forks off another prgm $\Rightarrow$ double counting
- If user runs > 1 prgm per core.
- XALT uses real time not cpu time.

Python prgm: sbin/xalt_usage_report.py

- An attempt to be system agnostic.
- It provides a name mapping file.
- It reports data that TACC has found useful:
  - Overall Job counts
  - Self Built vs. Not.
  - It reports Top Execs by Core-Hours, Number of Runs, Number of users for All, MPI Only, Scalar
  - Top Module usage
  - Compiler usage
  - Library Usage

xalt_usage_report.py (II)

```
$ python ./xalt_usage_report.py                \
        --confFn xalt_frontera_db.conf         \
        --syshost frontera                     \
        --start '2021-12-20'                   \
        --end   '2022-01-20'
```

- Would collect data for a month if today was '2022-01-20'
- See contrib/TACC/build_XALT_report.in report generator program on VM.

Overall Job counts

- Core hours for “system” prgms vs. user prgms.
- System prgms come from a module, User prgms don’t.
- We see about 5% for system prgms.
- Still have to teach how to build program.

Self-Built vs. Not.

- Beside system supplied program we have groups that share prgms
- Like to track that.
- Prgms built under XALT know build user.
- XALT knows the run user.
- We report 2 to 4 % non self-built prgm runs.

xalt_name_mapping.py

```
equiv_patternA = [
    [ r'^pmemd'                  , 'Amber*'       ],
    [ r'^sander'                 , 'Amber*'       ],
    [ r'^absorption.*real.*'     , 'BerkeleyGW*' ],
    [ r'^absorption.*cplx.*'     , 'BerkeleyGW*' ],
    [ r'^kernel.*real.*'         , 'BerkeleyGW*' ],
```

- Map prgms to projects
- Currently over 240 patterns
- Sites may need to modify to match their site
- Names get a when mapped
- Nothing stops a user naming Hello $\Rightarrow$ pmemd

Typical Reports: Top Execs Core-Hours

CoreHrs	# Runs	# Users	# Accts	Exec
289,924,113	710	13	7	CESM*
68,162,422	174,594	8	4	Chroma*
51,135,767	102,957	5	3	gene_fta
39,173,897	22,666	44	32	LAMMPS*
38,714,742	75,751	50	39	NAMD*
35,858,260	254,470	55	37	VASP*

Typical Reports: Top Execs Runs

CoreHrs	# Runs	# Users	# Accts	Exec
2,391,403	815,586	1	2	Rosetta*
2,402	583,091	136	97	mv
7,068	515,852	373	211	grep
11,942,647	499,463	391	218	Python*
24,176	414,788	1	2	MOPAC2016

Typical Reports: Top Execs Users

CoreHrs	# Runs	# Users	# Accts	Exec
11,942,647	499,463	391	218	Python*
7,068	515,852	373	211	grep
1,330	67,595	335	197	sed
12,779	121,983	322	182	gawk
2,402	583,091	136	97	mv
16,366	131,215	132	94	cp
137,580	21,731	103	62	perl

Typical Reports: Scalar Top Execs Core-Hours

CoreHrs	# Runs	# Users	# Accts	Exec
2,852,146	303,289	4	2	rockstr-glx
2,809,544	469,191	336	195	Python*
2,391,403	815,586	1	2	Rosetta*
908,214	186,031	23	11	R
747,719	22,940	1	2	squid

Typical Reports: Scalar Top Execs Users

CoreHrs	# Runs	# Users	# Accts	Exec
967	460,271	359	205	grep
2,809,544	469,191	336	195	Python*
962	41,880	314	187	sed
29	63,281	308	176	gawk
343	5,199	124	89	mv
137,145	19,162	95	59	perl
100	2,660	92	65	cp

Typical Reports: Top Modules Execs Core-Hours

CoreHrs	# Runs	# Users	# Accts	Modules
18,271,366	4,847	23	21	gromacs/2019.4
8,815,817	181	5	4	gromacs/2019.6
6,535,560	5,693	15	13	lammps/15Apr20
5,763,346	28,179	125	99	python3/3.7.0
5,077,825	1,082	17	11	namd/2.14
1,709,288	66,597	19	18	namd/2.13
1,303,299	1,054	15	16	python3/3.8.2
1,165,677	107	9	9	gromacs/2020.1

Typical Reports: Compiler usage by Count

Count	Link Program
4,017,109	x86_64-conda_co
242,232	gcc
204,782	icc
105,602	icpc
92,881	g++
85,090	ifort
40,970	c++
23,224	gfortran
9,250	mpiicc(icc)
7,377	mpiifort(ifort)
5,625	mpif90(ifort)
4,398	grim

- I have no idea what “grim” is.

Typical Reports: Compiler usage by Core-Hours

CoreHrs	# Users	# Accts	# Runs	Link Program
366,294,372	258	170	302,253	ifort
140,974,996	231	158	125,038	icpc
34,483,254	68	55	686,981	g++
32,660,529	149	98	138,838	icc
10,518,386	56	40	108,864	gcc
2,201,761	21	17	21,743	c++
269,002	31	24	19,562	gfortran
8,814	2	1	116	mpiifort(ifort)
1,684	3	3	1,809	mpif90(ifort)
865	3	3	17	mpiicpc(icpc)

- Recent change to capture mpi compiler(compiler)
- Obviously, most of the data is from before change.
- Also groups are still using fortran!!
- This is by core hours, maybe another measure would be better

Typical Reports: Core-Hours Libraries module families

CoreHrs	# Users	# Accts	# Runs	# Jobs	Library Module
1,183,467,488	1,064	651	3,544,533	281,720	gcc
1,140,838,707	984	624	1,062,676	235,540	impi
579,969,929	194	151	234,444	31,519	phdf5
441,465,631	550	372	1,269,790	121,421	intel
388,703,536	492	336	959,936	143,712	mkl
356,269,444	279	218	443,142	25,789	python3
322,972,860	37	30	6,380	6,175	parallel-netcdf
61,052,841	83	68	329,949	9,408	fftw3

- Have to know how to read this.
- Grouped by module families.
- This may depend on N/V naming scheme not C/N/V

Typical Reports: Core-Hours Libraries modules

1,183,467,488	1,064	651	3,544,533	281,720	gcc
1,140,838,707	984	624	1,062,676	235,540	impi
629,361,768	581	377	670,258	142,934	impi/19.0.9
579,969,929	194	151	234,444	31,519	phdf5
556,355,447	643	392	1,484,379	176,090	gcc/8.3.0
511,432,610	105	76	208,669	20,324	phdf5/1.10.4
475,379,381	253	132	298,122	66,894	impi/19.0.7
441,465,631	550	372	1,269,790	121,421	intel
395,220,235	459	310	1,012,327	103,776	intel/19.1.1
388,703,536	492	336	959,936	143,712	mkl
364,394,100	411	281	917,239	131,431	mkl/19.1.1

- Have to know how to read this.
- Grouped by module families.
- This may depend on N/V naming scheme not C/N/V
- Double counting or more is a problem.
- If you link with 3 phdf5 libraries $\Rightarrow$ 3x

Conclusions

- We can extract useful data from XALT
- It is not quick on a VM.
- Would love to have the DB on SSD
- We use data to know what codes add to next benchmark.

Future Topics?

- Package tracking
- Others?