HPC Workload Monitoring

Prakhar Gupta

ILLINOIS
NCSA | National Center for Supercomputing Applications

Objectives

The SEAS groups provides telemetry and support for various applications run on the Delta Cluster. While they have some degree of telemetry on resource utilization and scheduler activity, they would like more information about the jobs being run.

Overview of Current SW Stack

LMOD

- Modifies user environment variables like `$HOME`
- Uses the environment to change the software/libraries being used by the system
- Offers provisioned software such as anaconda, cuda, python, etc.

Slurm

- Job scheduler
- `sbatch`, `salloc`, `srun` to get access to compute resources
- OpenOnDemand wraps around slurm to simplify interactive jobs (by abstracting slurm and ssh-tunnels)

Existing telemetry gives us basic information about loaded modules, job-id, and resource utilization. However more telemetry, particularly on the libraries used would be beneficial

Why?

- **Telemetry for Python and OOD**
- More information for user support/troubleshooting and incident analysis
- More informed decisions for software provisioning
  - New modules, deprecating modules, etc
- Better informed scheduling policies
- Time Series analysis of library usage over time

- Track user executables and library usage on a cluster
- Track package usage for Python, Matlab, R
- Track activity inside a cluster
- Integrates with Slurm and LMOD

https://xalt.readthedocs.io/en/latest/#

https://github.com/xalt/xalt

How it Works

- `$LD_PRELOAD` is an environment variable that points to a `.so` file that is linked to any ELF at runtime
- Inject XALT into the `myinit` and `myfini` arrays
- Effectively hijacks ALL executables on a system

- `myinit`: does some filtering (hostname and path) and book-keeping (start record creation, disabling fork tracking, sets up signal handlers)
- `myfini`: saves the end-record, moves logs from temporary to final location. Called after `exit()` in user code

Section	Description
ELF Header	Metadata about the ELF file
Program Headers	Describes segments to be loaded into memory - Segment 1: Code (.text) - Segment 2: Data (.myinit_array, .myfini_array)
Section Headers	Metadata about sections in the ELF file (e.g., .text, .data, .bss, .init_array, ...)

myinit	Initialization functions executed before usercode
usercode	Main executable code
myfini	Finalization functions executed after usercode

How it Works

`$PYTHONPATH`: Environment Variable that supports loading “site-packages”

- Python script executed at interpreter startup (effectively running code before user code is run)
- Run code to get a list of modules
- Run through a package filter
- Execute an executable with arguments to save record to a temporary directory

Design Considerations

- Speed
  - Don’t want to slow-down user code with bloat
  - Majority of the system runs LUSTRE (parallel fs), which is not great with small files
  - Save logs in `/dev/shm` and move to a ‘global’ location at end of job
- Graceful Exit
  - Don’t want to break user code
  - Provisioned XALT as a sticky module

Design Considerations

- Robustness
  - Don’t want to miss logs or collect incomplete logs
  - File collection method
  - Integrated with Telegraf ingestion service already used for other telemetry. Offers circular buffer
- Memory Overhead
  - Don’t want to flood the filesystem with logs
  - Each job can produce anywhere from ~5kB to 400kB of logs
  - Lots of pre-ingestion filtering and sampling
  - Post-ingestion log-deletion

Challenges

Slurm:

- Most jobs time-out and are killed by `slurmd` as opposed to calling `exit()` normally. `myfini()` not always called

- Proof of Concept: Enabled Signal Handling on OOD slurm scripts to send `USR2` to shell launching processes.
  - Shell then sends `USR2` to the process 60s before end
  - XALT intercepts signal, preempts execution of `myfini(`
  - Also preempted by `INT`, `ABRT`, `TERM`, `HUP` ensuring logs always saved before `/dev/shm` is flushed

- `slurm.conf` might need to be modified by admins, `USR2` no longer available to users?

Apptainer:

- By default, only loads in `$HOME`, so XALT will not be available on the container
- Solution: Module to include an `$APPTAINER_BINDPATH` and `$APPTAINER_LD_PRELOAD` to include XALT into the container

Collected Telemetry

- Run logs
  - XALT metrics
  - Environment Variables
  - Command Line
  - Libraries in Use
  - Process Tree
  - Resources used
  - Slurm metadata
  - Module information
- Link logs
  - Linked libraries
  - Linker arguments
  - XALT watermark
- Pkg Logs
  - Package name
  - Package path
  - Package Version
  - Xalt Run Record UUID

Future Work

- Integrate with Ingestion Service
- Post-ingestion filtering
- Analytics Dashboard
- Finetune Sampling and Filtering
- Include package support for R, Matlab

References

- Agrawal, K., Fahey, M. R., McLay, R., & James, D. (2014). User Environment Tracking and Problem Detection with XALT. In Proceedings of the First International Workshop on HPC User Support Tools (pp. 32–40). Piscataway, NJ, USA: IEEE Press. http://doi.org/10.1109/HUST.2014.6
- Ved Arora, Nayeli Gurrola, Amiya Maji, and Guangzhen Jin. 2023. A Fast and Responsive Web-based Framework for Visualizing HPC Application Usage. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12--17, 2023, Denver, CO, USA.* ACM, New York, NY, USA 4 Pages. https://doi.org/10.1145/3624062.3624148