SAND2023-04881C

Process Tracking

Benjamin A. Allan

LDMSCON 2023 lightning talk

June 13, 2023

Boston, MA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Outline

- What? Tracking process creation and performance
- Why? Better: understanding, provisioning, reporting, design
- How? Light-weight, Linux kernel-based method
- Where? In testing on large Sandia production clusters
- Analyses? Preliminary example

Lightning topic: What other analyses should there be that are enabled by this data?

Tracking process creation and performance (what)

Collect time-stamped, unaliased data about process or thread start & end *events*

- Fast, highly configurable PID exclusion filtering is key for administrators
  - Many *uninteresting* PIDs exist, some sensitive PIDs or UIDs
  - *Exclude* matching combinations of program duration, location, or UID

Collect the desired per-process *metrics* from /proc/$pid/* *periodically*

- Highly configurable metric data selection.
- Metrics available in part:
  - Job identifiers
  - Process state, CPU times, I/O volume, memory usages, page faults, context switches
  - Name of blocking system call, UID, GID, oom score (files in use, argv, environment)

Understanding, provisioning, reporting, design (why)

What programs are our users actually running?

- Detect versions, configurations, associations with customers (WCIDs)

How should they be running the codes?

- Detect misconfiguration (allocated node under/over-usage)

What software (development) or allocation (management) changes are indicated?

- Detect wrong libraries in use; detect wrong cluster in use; detect misplaced programmatic loads.

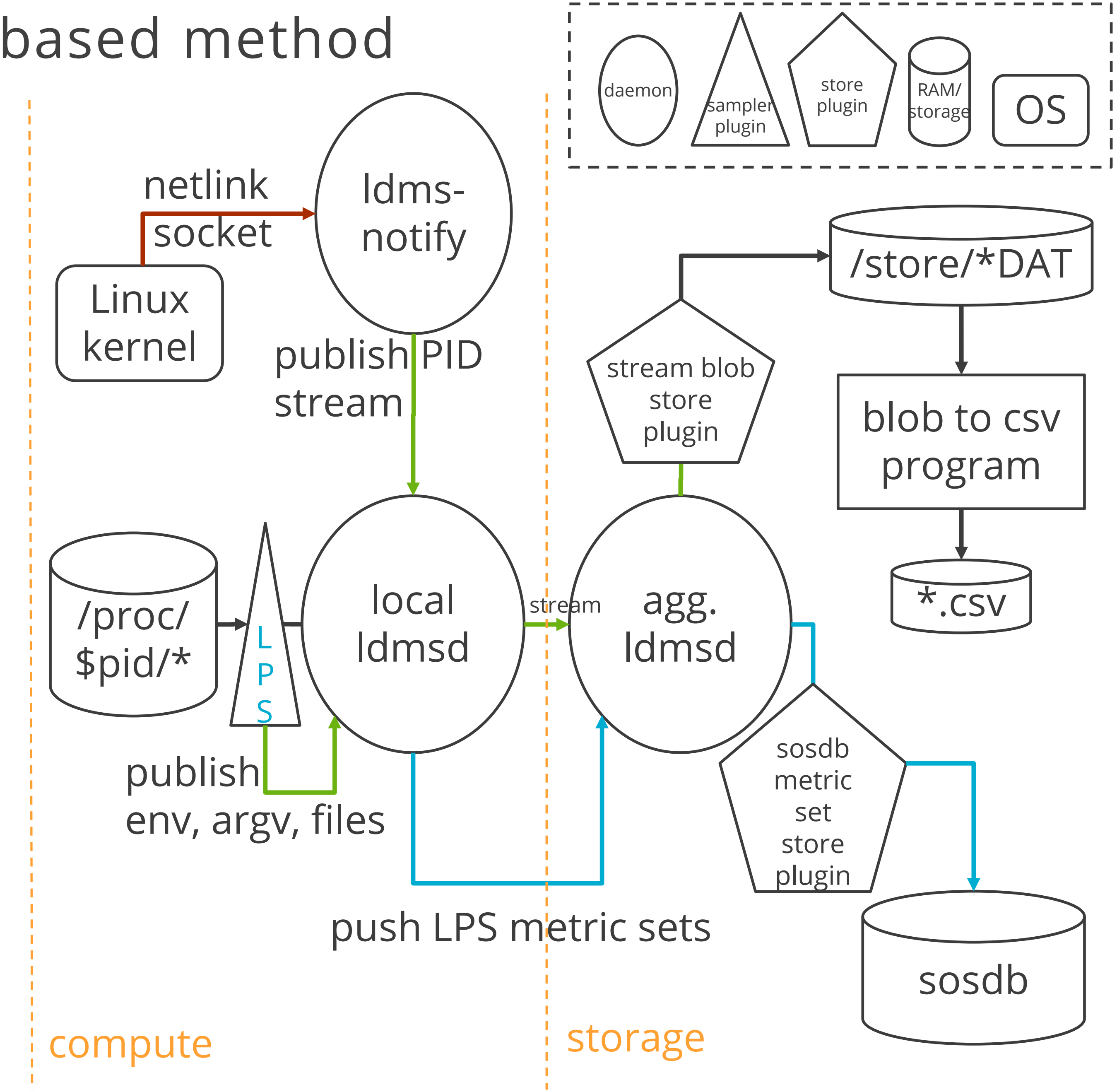
What are the performance characteristics of the total workload?

- Detect needs to shift the mix of memory, CPU, network, storage, GPU in the next HW (re)build.

Light-weight, Linux kernel-based method (how)

- ldms-notify (or slurmd) daemon publishes interesting PIDs via stream
  - **No application modifications**
  - Store & correlate start/end events
  - No periodic /proc search

- linux_proc_sampler (LPS) plug-in subscriber monitors only ‘interesting’ PIDS
  - Hardened & extended version of app_sampler (not fully compatible)
  - Argv, env, syscall, shared libraries, files used



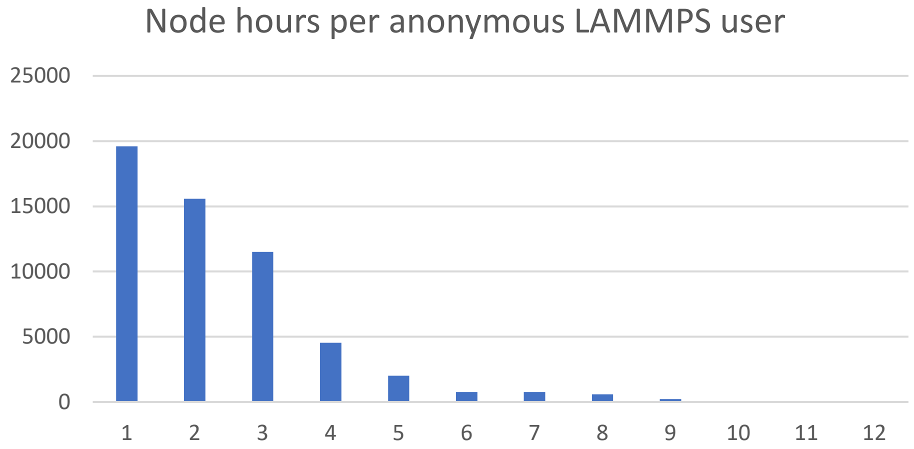
Testing on large production clusters (where)

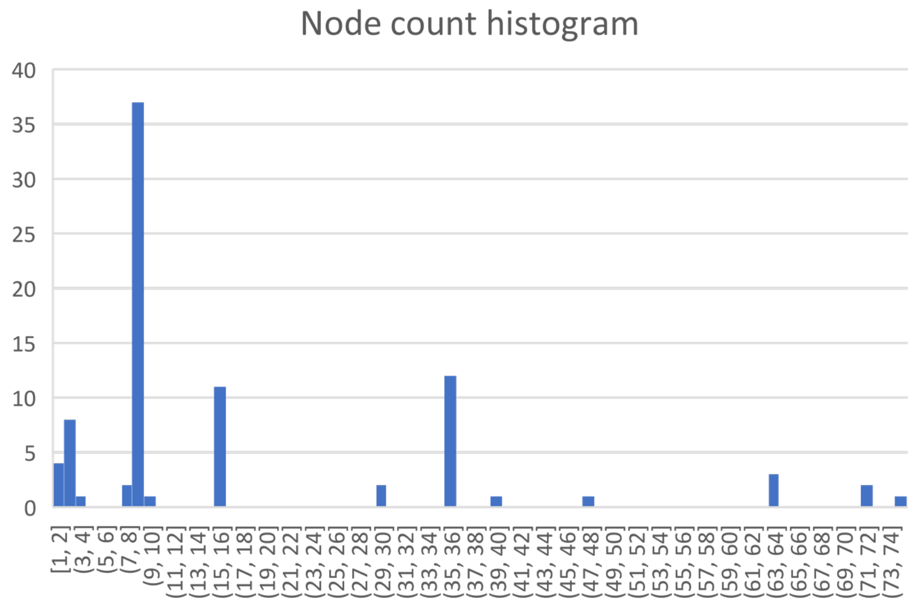
- Two 1500 node clusters configured to publish both slurm and ldms-notify PID events
  - both compute nodes and login nodes
- Filtering out:
  - security sensitive processes such as anti-virus
  - short duration processes with common paths (e.g /usr/bin)
  - processes with UID < 1000 [all the system daemons]
- Not yet running LPS: still qualifying the ldms-notify infrastructure (message rate)
- Observing nightly build jobs:
  - which compilers/pythons/file systems are in actual use
  - high rates of short and medium life programs
- Observing applications
  - LAMMPS demo on next slide

Preliminary results (analyses)

Data: two weeks of LAMMPS (molecular dynamics simulation tool) on one cluster

- 86 jobs identified by matching executable name regular expression
- 12 unique users
  - 1 *or more* unique binaries per user
- 55,000 total node-hours (9.2% of that cluster)
  - also broken down by user
  - also broken down by size





Conclusions/Discussion

Discussion

- Correlating per-process start/end (rather than job start/end) with node performance:

  What additional analyses should we be doing?

Conclusions from data:

- Collecting per-process information with LDMS is feasible with proper configuration
  - ***Cost***: 110 messages/sec (***39 kbyte/s***) averaged over a week (cluster with nightlies)
  - Login node users can be very naughty (*but you knew that*)
  - Some jobs are very complex (pre/run/post, nightly build, etc).
  - Mpiexec vs srun looks very different PID-wise and both are in use.
  - Slurm/spank identification of PIDs is incomplete compared to Linux kernel identification.
    - Simulation packages driven by python tend to see ***only*** the bin/python PID from slurm.

extras

linux task start message (ldms-notify published stream)

```
{
"msgno":29080, "schema":"linux_task_data", "event":"task_init_priv",
"timestamp":1683844505, "context":"*",
"data":{
        "ProducerName":“nid393", "start":"1683844505.706770",
        "start_tick":"183944289",
        "job_id":“12345", "serial":3083652, "os_pid":146732,
        "uid":95782, "gid":95782, "task_pid":146732, "task_global_id":-1,
        "is_thread":0,
        "exe":"/projects/a/bin/empire-pic.x“
}}
```

ldms-notify daemon

Manual page provided

Systemd wrapper provided

Derived from Canonical open-source forkstat utility

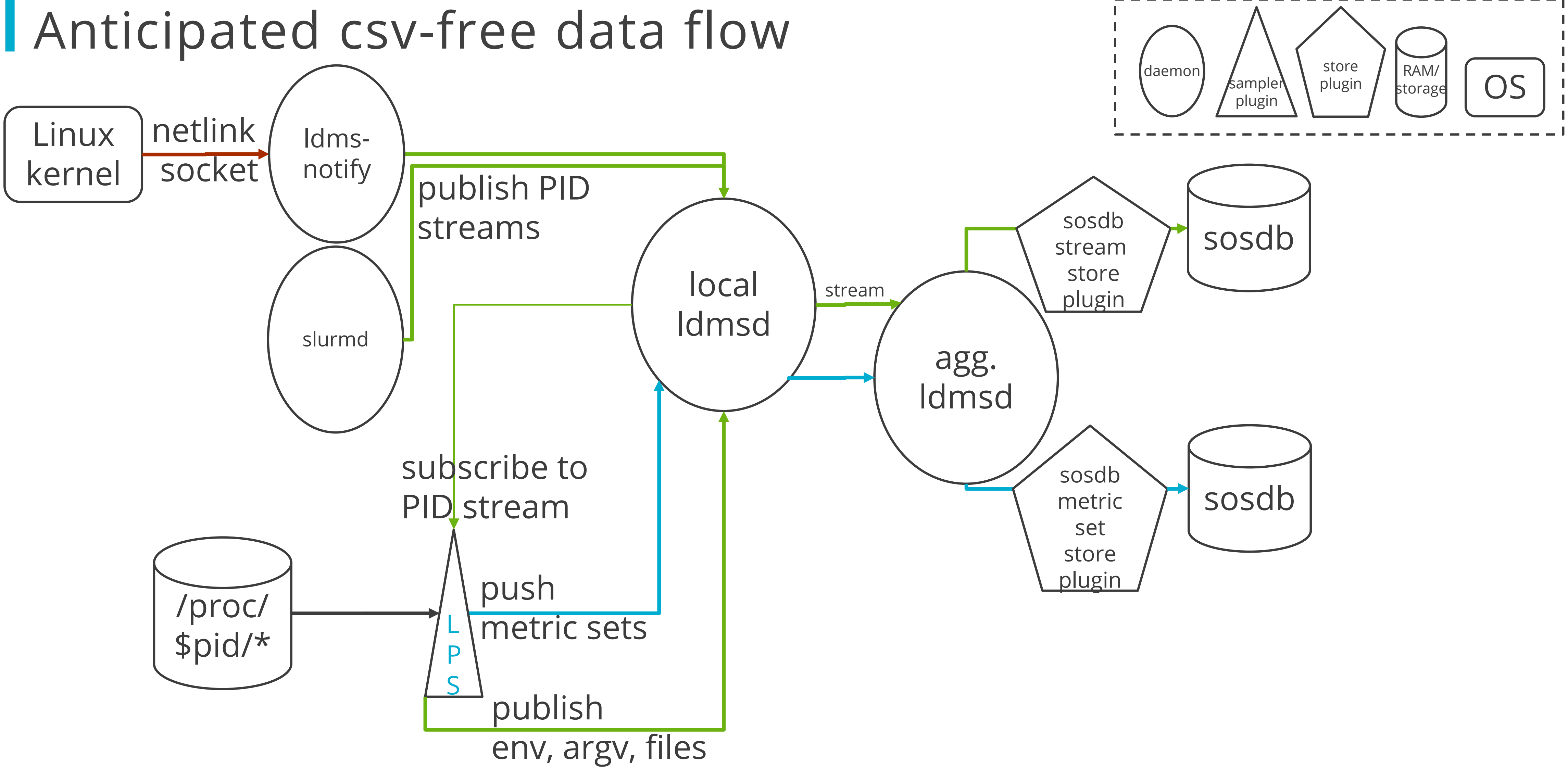
Start-up/restart of ldms-notify and ldmsd is designed to be fully asynchronous

- Stream connections are renegotiated as needed (self and ldmsd fault-tolerant)
- Small cache of ‘interesting’ PIDs for ldmsd restart handling is optional

Linux_proc_sampler new major features (since app_sampler)

- Manual page provided
- Can handle multiple PID sources and formats (currently ldms-notify, slurm)
  - Takes 'best-of-both' data when two sources present
- Captures enough data to uniquely identify a process across entire center for all time
  - Avoids comingling of data from distinct processes during analysis
- (option) Blocking system call *names* are captured
- (option) User/group *names* are captured
- (option) Publishes argv and environment as stream messages (pre-MPI_init)
  - Filtering the environment by regular expressions is allowed
- (option) Publishes approximate file open/close/delete events (/proc/$pid/fd/* scan)
  - Separately tunable scan interval
  - Filtering by path regular expressions is allowed

Anticipated csv-free data flow


daemon
sampler plugin
store plugin
RAM/ storage
OS
Linux kernel
netlink socket
ldms-notify
slurmd
publish PID streams
local ldmsd
stream
agg. ldmsd
sosdb stream store plugin
sosdb
sosdb metric set store plugin
sosdb
subscribe to PID stream
/proc/ $pid/*
LPS
push metric sets
publish env, argv, files