Improving User Accounting and Isolation with Linux Kernel Features

Brian Bockelman

Condor Week 2011

Case Study: MPD

- The MPICH2 library is a common implementation of the MPI interface, a popular parallel programming paradigm.
- One issue in MPI is spawning processes: some entity external to the library needs to spawn one process per core in the job.
- Process spawning is typically handled by the batch system.

Case Study: MPD

- If MPICH2 isn't integrated with the batch system, it has a small python program, MPD, that does process spawning for you.

- *Common occurrence*: The tenured faculty thinks they are smarter than those lazy sysadmins and decides to compile their private copy of MPI with dangerous compiler settings. Their private copy of MPI is not integrated with the batch system, and MPD is used.

MPD

- MPD is designed for single-user clusters without batch systems:
  - Automatically double-forks and backgrounds itself.
  - User job communicates with MPD via a world-writable socket in /tmp; uses this to launch actual running processes.

MPD Issues:

- World writable socket: Any user who can write to /tmp can launch processes as your username.
- Double-forking: batch systems (even Condor!) lose track of the MPI processes.
  - Job can't be cleanly killed anymore.
  - Accounting is incorrect.

MPD

- So, the unsuspecting user has made the pool hard to manage and insecure!
- Today, we'll talk about ~~new~~ almost ready Condor capabilities on Linux that could have helped in this situation.
  - Improved process **accounting** with cgroups (requires RHEL6 or kernel 2.6.24).
  - Improved **isolation** (requires RHEL6 or kernel 2.6.24).

Accounting

- Accounting, *in this context*, is tracking the system resources used by the user in a job slot.

- What do we keep track of in Condor?
  - Processes associated with the job slot.
  - CPU time used by the job processes.
  - Memory usage of the job processes.

    Wishlist: Block I/O, network I/O

State of the Art

- The condor_procd is responsible for process accounting. Its job is hard because:
  - It is poll-based (out of lack of better alternatives); it has poor visibility into short-lived processes.
  - Linux ~~has~~ had no ability to track arbitrary sets of processes *that can't be "escaped"*.
  - Memory accounting for a set of processes is a mess.

Introducing cgroups

- Cgroups are control structures for aggregating/partitioning processes in a system
- Different cgroups subsystems may act on these structures to control scheduler policy, allocate/limit resources, account for usage.

http://en.wikipedia.org/wiki/Cgroups

http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt

Cgroups

- The cgroups structures - presented as a filesystem like /proc - are a kernel-level mechanism for doing the process tracking the procd was doing from user-space.
- There exists cgroup subsystems for CPU accounting, memory accounting, block I/O, and killing processes.
  - That's everything the procd does!

Condor and Cgroups

- Condor ProcD ~~has~~ will have the ability to create a cgroup per job using the memory, block, and CPU accounting subsystems.
    - The “freezer” subsystem then provides the ability to deliver signals atomically.
- If cgroups fail, the procd can fall back to the traditional methods.

Future Work?

- Well, it does need to be committed first...
- Network I/O needs to be accounted.
- We haven't touched on resource limiting:
  - CPU affinity/scheduler (allows us to monitor amount of swapping done per batch slot).
  - Hard limits on memory used per batch slot enforced by the OS.
  - Block I/O can be bandwidth-limited or fair-shared between batch slots.

Isolation

- Two running jobs are isolated if the throughput and outcome of each job is independent of the other.
- Typically, two running multiple jobs on the same node are not isolated.
  - Likely never will be 100% isolated until we start running on more expensive hardware...
- Same is true for shared networks, shared storage or submit hosts.

We can't provide complete isolation, but we can often do a better job!

Isolation

- Condor provides some level of isolation already:
  - CPU affinity.
  - Preemption based upon memory use.
  - Preemption based upon disk use.

Isolation

- Cgroups likely will improve this:
  - More accurate memory accounting.
  - We could easily monitor the per-batch-slot swap rates.
  - Has ability to do per-job scheduling (network, block, memory, CPU).

Other Isolation

- Besides isolating system resources, we can also remove the job's ability to interact with each other:
  - *PID namespaces*: The job can only see (and signal) processes in the same namespace.
  - *FS namespace*: Each job gets its own filesystem mounts.

PID Namespaces

- Each process within the namespace gets a PID local to the namespace.
- I.e., the top-level process has PID 1; outside the namespace, it has a normal-looking PID.
- POSIX calls (getpid, kill) will work within the namespace using the local PIDs.
  - "ps aux" will only show local processes.

Example Sessions

```
[bbockelm@rcf-bockelman condor]$ condor_run ps faux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
bbockelm     1  0.0  0.0 106200  1032 ?        SN   12:51   0:00 /bin/bash /home/bbockelm/
projects/condor/.condor_run.29620
bbockelm     2  0.0  0.0 108052   964 ?        RN   12:51   0:00 ps faux
```

```
[bbockelm@rcf-bockelman condor]$ ls /tmp/
condorLocks                                hsperfdata_condor  hsperfdata_root  keyring-amZ5iJ
gnome-system-monitor.bbockelm.1758109379  hsperfdata_hsqldb  keyring-Agrhyl   keyring-imIakq
[bbockelm@rcf-bockelman condor]$ condor_run ls /tmp
[bbockelm@rcf-bockelman condor]$
```

PID Namespaces

- Pitfalls: Small change in semantics!
  - A job doesn't know its PID outside the namespace. Hence, the job's logs will record PIDs that are not meaningful to the sysadmin.
  - PID uniqueness is no longer guaranteed: You can't create a unique filename using the PID (note: don't do this anyway; it's insecure).

FS Namespaces

- FS namespaces (available since RHEL4) allow jobs to have a unique mount table.
  - Per-job mount of storage: don't have to drain the node to make mount changes.
  - Unique /tmp and /var/tmp: don't have to worry about jobs leaving world-writable sockets or temporary files outlasting the job.

Condor vs. System

- It's possible to run the condor_master itself in a separate PID namespace.
  - Prevents batch system users from seeing what else is running on the system.
  - Narrows an attack vector: it's harder to attack specific services if you don't know what's running. Reduces the ability for a partially compromised Condor to be used to attack the host system.

Future Work

- Both PID and FS namespaces aren't yet committed.
- Both need integration with PrivSep.
- Allow generic per-job mounts.
- Hide the working directories of other running jobs.
  - At this point, two jobs running under the same Unix username shouldn't be able to interfere with each other.

Returning to MPD

- The world-writable socket no longer accessible with FS namespaces.
- Cgroups can track, account for, and kill the MPD daemon.
- Issues like MPD are difficult to avoid: they're created by eager and well-meaning users.
  - Better isolation is not a "security thing" - it helps users from inadvertently colliding.
  - Cgroups and namespaces are a welcome addition to our toolkit for managing resources.