HTCondor: Virtualization (without Virtual Machines)

Brian Bockelman

HTCondor Week 2013

Dictionary Definition

- vir·tu·al·ize [vur-choo-uh-lahyz] verb
  - to create a virtual version of (a computer, operating system, data storage device, etc.), which is not itself an independent device but both works and appears to the user as a single, physical entity.
- HTCondor has been doing virtualization since about 1985 - and more recently it's been a hot topic in the Linux world.

Is this your worldview?

Batch Jobs

Virtual Machines

This is how we see things!

Batch Jobs

Virtual Machines

The Virtualization Continuum

- **Traditional Batch Process -** POSIX process tree. Same OS image, can easily interact with other jobs, little/no isolation between users. Resource management is minimal
- **Virtual Machine -** present a unique OS image per user. Hypervisor (such as Xen or Linux kernel) provides strong isolation and attempts to provide resource guarantees.

What's in between???

Does a process tree necessarily exclude strong isolation?

What's in between?

- I propose there's a long distance between the traditional batch process and virtual machines.
- Virtual machines carry a *lot* of baggage which makes them difficult to coexist:
  - Does a user *really* want to administer an OS? Who minds the security patches?
  - Site networking policies (do you let users open ports below 1024? Are you sure?).
  - Difficult to integrate safely with storage. Many storage types (NFSv3) assume root is privileged.
  - And don't forget the overhead of multiple kernels.
- It's *very* useful to give the site owner a rich interaction layer with the job; POSIX has treated us well for decades.
- It's a lot to pay for isolation, accounting, and resource management!

What makes VMs so Great?

- We really like the following pieces of a VM:
  - Precise resource management and isolation.
  - Separate OS image.
  - Appearance of privileged user (root).
  - Addressable by network.

What can be pitched?

- But truthfully, we rarely care about the following:
  - Running a separate kernel.
  - System daemons: Having another copy of portmap, auditd, udev, pdflush, sssd, mingetty, etc...
  - Another virtual memory hierarchy.
- There *are* cases where this is relevant - an Ubuntu-only application at a RHEL shop.
  - I'd posit that most batch clusters aren't one of them.

Containers

- Observation: For KVM-based virtual machines, the Linux kernel had to become a hypervisor
- The missing piece between a process and a VM is a container.
- A Linux container is a set of processes managed by the host kernel, isolated from the system, virtualized to make things appear as if they are running as a separate host.

Past Progress

- If you love HTCondor Week, you probably have heard this message before.
  - http://research.cs.wisc.edu/htcondor/CondorWeek2012/presentations/bockelman-condor-container.pdf
  - http://research.cs.wisc.edu/htcondor/CondorWeek2011/presentations/bockelman-user-isolation.pdf
  - http://www.biggrid.nl/fileadmin/documents/20120126_CondorContainers_by_Bockelman.pdf
  - http://research.cs.wisc.edu/htcondor/HTCondorWeek2013/presentations/ThainG_BoxingUsers.pdf
- Let's review where we've been!

Past Progress

- Items in a HTCondor release:
  - **Accounting**: CPU, memory usage, block IO.
  - **Isolation**: PID namespaces, chroots, per-job / tmp directories.
  - **Resource management**: CPU fairshare & affinity, memory limits, guaranteed process killing.
- These features are all borrowed from VMs and allow HTCondor to better virtual per-job.

Current Work

- We are currently working on a few items I'd like to highlight.
  - One I *won't* talk much about is virtualizing the batch job's network (covered this morning).
  - In the Lark project, we assign a network address per-job, allowing the network layer to reason about HTCondor.

OOM Killer

- The out-of-memory killer is the bane of cluster admins.
- Invoked when Linux believes it has run out of memory (*or when a cgroup limit is hit*), it is in charge of selecting a process to kill.
  - It assigns a score to each process - the highest score "wins" and is killed.
  - Due to its evaluation rules, it tends to prefer to kill HTCondor instead of the HTCondor job.
- Each process receives a multiplier to its score. As of HTCondor 7.9.6, we now assign high multipliers to the processes in a batch job. It's very likely the job itself will be killed.
- When using cgroups, the kernel will notify HTCondor of the out-of-memory situation instead of invoking the OOM. No action can happen in the cgroup until HTCondor resolves the situation (i.e., kills the job).
  - When this happens, the job is killed and put on hold in the schedd *with an appropriate message*.
  - No longer will an out-of-memory situation kill a node or result in a mysterious job death!

Per-job mounts

- Sysadmins can already set aside chroots which jobs can request to run within. They can also provide jobs with a separate /tmp.
- The natural "next step" is to allow jobs to request additional admin-whitelisted mounts. Examples:
  - Have /condor map to the schedd's filesystem via chirp/FUSE. Chroot into it?
  - Mount /mnt/hadoop for this job.
- Mounts will live for the duration of the job and be visible only to the job.
- **Driving idea**: Jobs should be able to specify their desired storage layout as easily as they describe their Unix environment variables.
- Patch is available; should land in 8.1.x.

Think of the Glideins!

- Resource management is great, but an important property is being *recursive*.
  - If I can hand you a resource, can you then sub-delegate and manage it?
  - This is the concept behind glideins/pilots.
- All of the virtualization features discussed *can* work recursively.
  - The real trick is to get permission to perform the relevant system operations. Many of the features discussed, when given to an arbitrary process, are easily turned into escalation vulnerabilities.

User Namespaces

- The *user namespace* feature gives a user process the impression it has full root-like capabilities.
  - But only for processes in the same namespace.
    - Any operation the process could not perform outside its namespace is still denied.
- Any user can create them, assign itself a UID within the namespace, and create other UID mappings.

User Namespaces

- For example, the unprivileged "condor" user could create a startd in a user namespace where it is mapped to "root".
  - The startd would have the power to do UID switching and spawn processes as other users.
    - To processes in the namespace, the kernel provides UID-based isolation and separation.
    - Outside the namespace, all processes have the same UID. Any condor-owned process could kill the jobs.

User Namespaces

- LWN tutorial for those interested: http://lwn.net/Articles/531114/#series_index
- I expect Fedora 19 will have all the kernel features necessary to use user namespaces.
- The Linux kernel has a sprawling ABI; any new feature typically has several unintended consequences when used with other parts of the system.
  - User namespaces are no exception - they've led to several straightforward kernel vulnerabilities.
- So, I expect there to be a few more years before this becomes "production". We should be ready the minute it hits the road.

Testing

- Right now, the HTCondor test coverage of cgroups / containers / namespaces is poor.
  - By which I mean, it is tested by developers before they submit the patch.
  - User namespaces give us a way to break forward. Once I get my hands on a F19 box, I hope to start writing non-root tests for namespaces.
  - Cgroups, since they are filesystem-based, should be able chown'd to non-root for testing.
    - We just haven't. :(

Future Feature Ideas

- We never did much with NFS mount monitoring.
- Enforcing size limits on data directories.
  - XFS has "project quotas".
  - More generally, would like to quantify the performance hit from having HTCondor build a sparse loopback device.
- Investigate the feasibility of running a job inside a VM image (without booting the VM).
- Setting per-job hostnames.
- Investigate the feasibility of Linux checkpoint-restore in userspace (http://criu.org). Requires Linux 3.7 or later; part of Fedora 19.
- As always, patches are accepted!

Food for thought

- If we're able to launch a job inside a raw disk image and a user namespace,
- can't that job be "init"?

One More Thing...

- Ever since the invention of multi-user systems, we've been "virtualizing" the use of CPU.
  - Each process gets a certain time-slice when they can use the CPU. We do accounting based on CPU time.
    - But how effectively are we using the CPU?
    - Are we getting value out of our CPU time?
- CPUs are complex beasts - time spent on the core does not mean much if we don't know how that time was used!

Which piece of code runs faster?

Version A:

```
#include <math.h>

int main(int argc, char *argv[])
{
  float sum=0;
  for (long long i=0; i<700000000; i++)
    sum+=sqrt(i);
  return 0;
}
```

Version B:

```
#include <math.h>

int main(int argc, char *argv[])
{
  float sum=0;
  for (long long i=0; i<700000000; i++)
    sum+=sqrt(i);
  return 0;
}
```

Answer: The one compiled with "-O3"

- With no optimization, the program takes 10.5 billion CPU instructions and achieves 0.29 instructions per CPU cycle.
- With optimization, the program takes 4.2 billion instructions and has 1.49 instructions per cycle.
  - About 12.8x faster!
- With the growing complexity of the hardware, knowing basic CPU-level statistics *is* an important part of our accounting!

Detailed CPU accounting

- Luckily, modern cores provide counters from the "Performance Monitoring Unit", or PMU, for hardware performance.
- Can account for cache usage, TLB usage, instructions, pipeline stalls, branch performance and more.
- Starting in the 2.6.32 kernel, these counters are available to userspace.

CPU Performance Tracking

- I've submitted a patch to the next HTCondor development series to report CPU performance metrics. I have the following ClassAd elements:
  - **RemoteInstructions** - number of CPU instructions performed by the job.
  - **RemoteIPC** - CPU instructions per cycle.
  - **RemoteCpuCacheHitRate** - hit rate for the last-level CPU cache.
  - **RemoteCpuMigrations** - number of times the process moved between CPUs.
  - **RemoteContextSwitches** - number of context switches the process performed.
  - **RemoteCpuBranchInstructionRate** - rate of instructions that involved a branch operation.
  - **RemoteCpuBranchPredictionMissRate** - rate of branches that were mispredicted by the CPU.
- Most accurate when measured via cgroups, but available when run as non-root HTCondor. Will work with privsep but not glexec.

Enjoy!

Appendix: Feature Matrix

Feature	OS Availability	Condor Availability	glideinWMS Availability
cgroup-based accounting	RHEL6	7.7.0	Possible with development
cgroup-based CPU fairshare	RHEL6	7.7.0	Possible with development
Reliable job killing	RHEL6	7.7.0	Available as glexec plugin
Per-job /tmp	RHEL5	7.7.5	Available as glexec plugin
chroots	RHEL5	7.7.5	Needs user namespaces
Per-job mounts	RHEL5	8.1 series?	Needs user namespaces
cgroup-based memory limits	RHEL6	7.9.2	Possible with development
user namespaces	Fedora 19	Needs investigation	When in Condor
checkpoint / restore	Fedora 19	Needs investigation	Unclear
PID namespaces	RHEL6	7.9.4	Needs user namespaces
Network namespaces	RHEL6	Separate branch; 8.1 series?	Unlikely
CPU performance tracking	RHEL6	8.1 series?	When in Condor
CPU affinity	RHEL5	7.9.3	Possible with development