HTCondor workflows at Utility Supercomputing Scale: How?

Ian D. Alderman
Cycle Computing
Thundering Herd Problem
Thundering Herd Problem

- Classical OS problem: multiple processes are waiting for the same event, but only one can respond at a time.
- In the cloud, what happens to the (underlying) infrastructure when you start 10k servers is someone else's problem.
- What happens at the platform and application level is your problem.
- Experience is helpful.
Ramping up to 50,000 cores
I DON'T OFTEN RUN 50,000 CORES

BUT WHEN I DO I USE CONDOR
while true bottleneck.next()

- From Miron:
  - A bottleneck is a (system) property that once removed creates a new bottleneck.
- Related to theory of constraints from industrial engineering.
- Corollary: Every component in a distributed system can be a bottleneck.
Bottlenecks we have seen

- Scheduler. Forking, transferring data, etc.
- Shared filesystem (NFS).
- Web server/backend/provisioning system – client.
- Provisioning system - server (AWS). Need delta mechanism for ec2-describe-instances.
- Configuration management system. Designed to handle updates in large systems, not provision large systems all at once.
Message in a bottleneck?
Find the right problem: Aim high.

- Predict costs, runtime. Understand I/O and memory requirements. Users don't always know this.
- Zach says: **Understand your job**. Users don’t often have the tools to do this.
- We were surprised to find out that Flexera license server can handle this scale given enough file handles.
- The right bottleneck is CPU: that’s what we’re paying for.
Distributing jobs

- Distribute tasks among several schedds. (Manure spreaders)
- CycleServer manages tasks across several environments.
- Multi-region, heterogeneous clusters.
- Goals:
  - Keep queues filled (but not too full)
  - Keep queues balanced
  - Minimize complexity
  - Reduce server overhead costs
### Cluster Summary for All Clusters

**Show: CPU Usage by Host**

<table>
<thead>
<tr>
<th>Host</th>
<th>Cluster</th>
<th>Clock Speed</th>
<th>Memory</th>
<th>CPU Usage</th>
<th>Mem Usage</th>
<th>Net In</th>
<th>Net Out</th>
<th>Last Reported</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>internal</td>
<td>2.06 GHz</td>
<td>16.00 GB</td>
<td>100%</td>
<td>5%</td>
<td>38.53 B/s</td>
<td>763.60 B/s</td>
<td>7:11 PM</td>
</tr>
<tr>
<td>2</td>
<td>internal</td>
<td>2.06 GHz</td>
<td>16.00 GB</td>
<td>100%</td>
<td>5%</td>
<td>41.17 B/s</td>
<td>1.28 B/s</td>
<td>7:17 PM</td>
</tr>
<tr>
<td>3</td>
<td>2 internal</td>
<td>2.06 GHz</td>
<td>16.00 GB</td>
<td>100%</td>
<td>5%</td>
<td>33.79 B/s</td>
<td>1.39 B/s</td>
<td>7:19 PM</td>
</tr>
<tr>
<td>4</td>
<td>internal</td>
<td>2.27 GHz</td>
<td>8.50 GB</td>
<td>100%</td>
<td>6%</td>
<td>77.72 B/s</td>
<td>634.67 B/s</td>
<td>7:12 PM</td>
</tr>
</tbody>
</table>

### Cluster Performance Stats

**Time Frames:** Hour

- **CPU:**
  - Chart showing CPU usage over time.
- **Memory:**
  - Chart showing memory usage over time.
- **Network:**
  - Chart showing network activity over time.
CycleCloud: Auto-start and auto-stop at the cluster level

- Automation is the goal: nodes start when jobs are present, nodes stop when jobs aren't there (5 minutes before the billing hour mark).
- Select instance types to start in rank order to maximize price-performance.
- Use pre-set spot prices to minimize costs.
Zero-impact job wrapper

Goal: Don’t hit the file server, don’t have HTCondor transfer anything.

- No file transfer
- No input
- No results
- No output, error or log
- So how does the job do anything?
Use S3 instead of file server

- **B3**: bottomless bit bucket.
- Eventual consistency is well suited for the type of access patterns we use:
  - Read (big) shared data
  - Read job-specific data
  - Write job-specific results
  - Jobs can be made to except (hold) when inputs aren’t available (rare)
- Some systems do scale; this is one.
Don’t overwrite results
Actual check to see if results are there already
Exponential back-off for data transfer
Actual command line captures stdout and stderr
If command succeeds, save stdout and stderr
universe  = vanilla
Requirements = (Arch =?= “X86_64”) && (OpSys =?= “LINUX”)
executable = /ramdisk/glide_job_wrapper.rb
should_transfer_files = if_needed
when_to_transfer_output = on_exit
environment = "..."
leave_in_queue = false

arguments  = $(process)
queue 325937
DAGMan is your friend
Configuration management system

- OpsCode Chef.
- Chef-solo.
- Chef Server 11 from OpsCode.
- Deploy changes to wrapper scripts, HTCondor configuration, etc during a run.
- Run OOB task on all hosts (knife ssh). Very cool but realistically can be a bottleneck.
### Chef overview for chef-server-11.ecosys.com

#### Current host stats
- **# Chef Servers:** 1
- **# Hosts:** 1043
- **# Converged Hosts:** 10312
- **# Unconverged Hosts:** 31

#### Converge stats (last hour)
- **Total Converges:** 3544
- **Successful Converges:** 3532
- **Failed Converges:** 92

#### Recent converges

<table>
<thead>
<tr>
<th>Time</th>
<th>Host</th>
<th>Status</th>
<th>Start Time</th>
<th>End Time</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>6m 15m</td>
<td>esx-20-1.19</td>
<td>Success</td>
<td>6m 15m</td>
<td>6m 36m</td>
<td>11m 11m</td>
</tr>
<tr>
<td>6m 10m</td>
<td>esx-21-3.19</td>
<td>Success</td>
<td>6m 10m</td>
<td>6m 31m</td>
<td>21m 11m</td>
</tr>
<tr>
<td>6m 15m</td>
<td>esx-22-1.19</td>
<td>Success</td>
<td>6m 15m</td>
<td>6m 36m</td>
<td>21m 11m</td>
</tr>
<tr>
<td>6m 10m</td>
<td>esx-21-3.19</td>
<td>Success</td>
<td>6m 10m</td>
<td>6m 31m</td>
<td>21m 11m</td>
</tr>
</tbody>
</table>

### Alerts

### Converge status by host

[Image of converge status by host]
Design principle: Planning to handle failure is not planning to fail nor failing to plan

- Wrapper checks to see if its result is present and correct.
- There are a lot of moving parts. Different things break at different scales.
- Testing is essential but you’ll always find new issues when running at scale.
- Data is stale.
- Make sure you have enough file handles!
- HTCondor can be overwhelmed by too many short jobs.
- Spots fail: HTCondor is designed to handle this.
Additional advice

- Keep tight with your friends. (Keep your friends close and your enemies closer.)
- **DAGMan is your friend**
  - Even when there aren't dependencies between jobs
- **CycleServer is your friend**
  - What the heck is going on?
  - The race: Jason wins.
- **Additional advice: maintain flexibility, balance**
  - Keep it simple
  - Throw stuff out
    - Elegant job wrapper with cached data
  - Keep it fun
Thank you, Questions?

- Utility Supercomputing 50 to 50,000 cores
- Visualization, Reporting
- Data scheduling: internal ↔ cloud
- Workload portability