Condor in Dynamic Environments

Ian Alderman
ialderman@cyclecomputing.com
A Little History...
Condor – A Hunter of Idle Workstations

- 1988 paper introducing Condor.
- Described system using cycle-scavenging: batch jobs run on desktop machines when their daytime users are idle.
- Before: 23 VAXstation II workstations utilized at ~30%.
- After: 200 CPU-days used over 5 months by 1000 jobs.
- Key features:
  - Transparent to interactive users, batch users.
  - Fair sharing among batch users.
  - Checkpointing and migration for batch jobs.
A busy week...

Show: Historical grid usage in Wisconsin pool

Legend:
- Unclaimed
- Capacity
- sihw
- unagpai
- caddiuwa
- davertt
- dai
- cweles
- gus
- mchavez
- dpike
- pnitra
- rathijit
- mpedraza
- ilanc
- kcathcar
- pbui
- ilvy

Version 3.2.13 licensed to Demo License (Internal Use Only)  Terms & Conditions  © 2009 Cycle Computing, LLC. All Rights Reserved

Leaders in Condor Grid and Cloud Solutions
http://www.cyclecomputing.com
Fast forward to today

- Condor widely used, very successful.
  - Look around you!

- Key reason: assumption of **dynamic environment**:
  - Design goals from 1988 still valid today.
    “Workstations are autonomous computing resources and should be managed by their own users.”

- High Throughput via automated policy management.

- There are differences between today’s dynamic environments and idle workstations.
Today’s Dynamic Environments?

- Virtualized Environments!
- Slots/machines come and go on demand.
- Essentially the same problem as autonomous user-managed workstations.
- Similar but different policy considerations…
  - e.g. Workstations: Round robin to distribute load.
    VMs: Fill entire host to minimize instances.
- Thesis: Even with these differences, the same features make Condor work well in both use cases…
Today: Notes from the field

- My work on CycleCloud – Managed Condor pools on Amazon EC2.
  - Instance Types/Profiles
  - Auto-start/stop
  - Spot instances
- VMWare Environments – tiered use of available resources.
  - Still have peak vs. median usage problem that leads to waste.
Wasted during Spring Break...
EC2 Background:
Amazon Elastic Compute Cloud

- Pay for use utility computing.
- BYOVM: you have root, they have kernel.
- Request instance; charged for each hour or portion.
- Range of machine configurations.
- Appearance of unlimited capacity.
- Proprietary highly-available storage backend: S3.
Condor pools on demand

- CycleCloud Service: at-cost or supported pools.
- Relies on dynamic features of Condor: instances stop and start as needed.

**BigTestPool**
- Size: Large Amazon Pool (<1000 cores)
- State: stopped
- Auto-Startup: on
- Auto-Shutdown: on

  - Enable Auto-stop: ✔
  - Enable Auto-start: ✔
  - Max Auto-Scale Nodes: 100

**Default Auto-start Machine:**
- High-CPU XLarge 8x Core (20 Compute Unit) & 7 GB RAM - 64bit
- Small 1x Core (1 Compute Unit) & 1.75 GB RAM - 32bit
- Large 2x Core (4 Compute Units) & 7.5 GB RAM - 64bit
- XLarge 4x Core (8 Compute Units) & 15 GB RAM - 64bit
- High-CPU Medium 2x Core (5 Compute Unit) & 1.7 GB RAM - 32bit
- High-CPU XLarge 8x Core (20 Compute Unit) & 7.5 GB RAM - 64bit

**MedHighMemTest**
- Size: Med. High-Memory SLES 8
- State: stopped
- Auto-Startup: on
- Auto-Shutdown: on

**Default Auto-start Machine:**
- High-Mem XXLarge 4x Core (13 Compute Unit) & 34 GB RAM - 64bit
- High-Mem Quad XLarge 8x Core (26 Compute Unit) & 68 GB RAM - 64bit
- High-Mem XLarge 2x Core (6.5 Compute Unit) & 17 GB RAM - 64bit


---

Leaders in Condor Grid and Cloud Solutions
http://www.cyclecomputing.com
## EC2 Instances Menu

<table>
<thead>
<tr>
<th>AWS Name</th>
<th>Arch</th>
<th>Cores</th>
<th>Memory (GB)</th>
<th>Price (VA AZ)</th>
<th>Price/Core</th>
<th>Memory (GB)/Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.small</td>
<td>32</td>
<td>1</td>
<td>1.7</td>
<td>$0.085</td>
<td>$0.085</td>
<td>1.7</td>
</tr>
<tr>
<td>m1.large</td>
<td>64</td>
<td>2</td>
<td>7.5</td>
<td>$0.340</td>
<td>$0.170</td>
<td>3.75</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>64</td>
<td>4</td>
<td>15</td>
<td>$0.680</td>
<td>$0.170</td>
<td>3.75</td>
</tr>
<tr>
<td>c1.medium</td>
<td>32</td>
<td>2</td>
<td>1.7</td>
<td>$0.170</td>
<td>$0.085</td>
<td>0.85</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>64</td>
<td>8</td>
<td>7</td>
<td>$0.680</td>
<td>$0.085</td>
<td>0.875</td>
</tr>
<tr>
<td>m2.xlarge</td>
<td>64</td>
<td>2</td>
<td>17.1</td>
<td>$0.500</td>
<td>$0.250</td>
<td>8.55</td>
</tr>
<tr>
<td>m2.2xlarge</td>
<td>64</td>
<td>4</td>
<td>34.2</td>
<td>$1.200</td>
<td>$0.300</td>
<td>8.55</td>
</tr>
<tr>
<td>m2.4xlarge</td>
<td>64</td>
<td>8</td>
<td>68.4</td>
<td>$2.400</td>
<td>$0.300</td>
<td>8.55</td>
</tr>
</tbody>
</table>
EC2 Instances Compared - Area ~ Price
High-CPU Amazon EC2 nodes have best price/performance
Making Condor work in EC2

- Naïve: Boot instances in EC2, add to pool.
- Better: Create a pool in EC2.
- No Condor configuration changes required, but…
  - Disable preemption.
  - Configure authentication, integrity, encryption.
  - Optimize for security, performance, scalability.
- Additional configuration:
  - Patches, information flow, installed applications, shared files, encrypt data at rest and in transit, etc…
Lots of intermediate data!
Why High Throughput leads to Efficient Computing
Auto-start, auto-stop mechanisms

**Auto-start:** Jobs present in queue $\Rightarrow$ machines start.
- Jobs run!
- User sets type of machine, limit # of instances to start.

**Auto-stop:** Before beginning of next hour:
- Check to see if jobs still running – if not, shut down.
- Users manually start machines that will auto-stop.
- Mechanisms for auto-starting different machine types based on user requirements.
- Users can supply *hints* about job run time for auto-start.
Spot Instances

- Same set of instance options.
- Lower cost, weaker service characteristics:
  - Could go away at any time.
  - If it goes away, you don’t pay.
- Bid for maximum you’re willing to pay, get machines at that price if available (i.e. going rate is $\leq$).
- If going rate goes above your maximum, your instances are terminated.
Spot Instance Policy Brainstorm

- Jobs expected to run for more than one hour need dedicated resources to avoid waste (getting billed but not finishing)
  - Job:
    - REQUIREMENTS = isSpotInstance == FALSE

  - Machine:
    - START = Target.EstimatedRuntime == UNDEFINED || Target.EstimatedRuntime >= 3600
    - isOwner = False

- Jobs run on the cheapest nodes possible
- Jobs prefer to run on machines up for lower fractions of an hour (to allow auto-stop to work)
  - RANK = 100 * SlotHourCost + FractionOfHourUP
Some folks don’t want EC2

• What about internal VM environments?
  • Most places have them, and they’re over provisioned

• VMWare environments:
  • Help with server consolidation (Run 40 VMs on beefy servers rather than 40 servers).
  • Still have peak vs. median usage problem.
  • For example, 500 Core VMWare environment running SAP that is 25-40% utilized on average, but still needs all that hardware for peak.
VMWare tiered applications

• Thankfully VMWare has tiers:
  • Production (PRD) usurps User Acceptance Testing (UAT) environment, which usurps Dev(DEV) environment.

• Perfect for harvesting (just like Spot Instances).

• Create a GRID tier that PRD/UAT/DEV usurp for resources and have the cores join locally.

• Add VMs to pool when there are jobs in queue, and remove GRID when PROD/UAT need them back.
VMWare Policy Use Cases

- Same high level policies work:
  - Jobs prefer dedicated nodes.
- Still cycle harvesting, just from a multi-core VM environment rather than just workstations.
- Just like auto-start, we’ll add VM requests to VMWare when there are jobs, remove them when they’re idle.
- Goal: Turn 40% Utilization to 90-95% utilization using technical computing workloads.
A lot has changed...

<table>
<thead>
<tr>
<th></th>
<th>1988</th>
<th>2010</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The original 200 CPU days</td>
<td>~11 CPU minutes</td>
</tr>
<tr>
<td></td>
<td>~3 CPU years</td>
<td>~1 core hour</td>
</tr>
</tbody>
</table>

Thanks to Moore’s Law
But some things have stayed the same:

<table>
<thead>
<tr>
<th></th>
<th>1988</th>
<th>2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quote</td>
<td>“We need more compute power”</td>
<td>“We need more compute power”</td>
</tr>
<tr>
<td>Notes</td>
<td>Dynamic Environments are plentiful</td>
<td>Dynamic Environments are plentiful</td>
</tr>
</tbody>
</table>