Dynamic Cloud-based clusters with HTCondor

John Hover

Grid Group, Brookhaven National Lab

HTCondor Week 2013

Madison, Wisconsin
Outline

Context & Goals

Dependencies/Limitations

System Components
- Workflow coordination: Job Factory
- Virtual Machines: Creation and Deployment
- HTCondor pool: Scaling for the WAN
- Cloud platform(s): EC2/OpenStack

Results

Glitches, Issues, and Problems

Next Steps
Context

ATLAS

- LHC Experiment at CERN
- In the US: BNL Tier 1, ~dozen Tier 2s, Tier 3s (local research groups)

Heavy users of OSG and HTCondor, but...

- No opportunistic use on OSG due to software constraints.
- Declining budget environment.
- Highly variable needs from Tier 3s/researchers
- Inefficient use of Tier 1 resources for simulation (low I/O)

Large-scale clusters have been done before. This work:

- Attempts to provide a complete workload solution. Wide-area pool + VM invocation.
- Allows repeatability via published docs/recipes.

How?

- $50K Amazon research grant. Thanks Michael!
Goal(s)
Run large Condor pool on multiple cloud platforms and/or providers.

- Spread across large area, possibly multi-continental.
- Include facility OpenStack instance(s).
- Utilize spot pricing on EC2.
- Allow the size of the distributed pool to be adjusted dynamically.
- Run typical ATLAS production (simulation) workloads on it.

Why?

- Free up high-performance Tier 1 resources for user analysis by moving low I/O work to EC2 and/or academic clouds.
- Pre-position ATLAS to utilize additional cloud-based resources that might become available.
Dependencies/Limitations

Inconsistent behavior, bugs, immature software:
- shutdown -h means destroy instance on EC2, but means shut off on OpenStack (leaving the instance to count against quota).
- EC2 offers public IPs, Openstack nodes behind NAT

VO infrastructures often not designed to be fully dynamic:
- E.g., ATLAS workload system assumes static sites.
- Data management assumes persistent endpoints
- Others? Any element that isn't made to be created, managed, and cleanly deleted programmatically.

EC2 imposes data export costs.
- Not appropriate for large-output work. (yet)

BNL imposes security and networking constraints.
Elastic Cluster Components

AutoPyFactory (APF): Coordinates submissions

- One APF queue observes a Panda queue, submits pilots to local Condor pool.
- Second APF queue
  - Observes a local Condor pool, when jobs are Idle, submits WN VMs to IaaS (up to a limit).
  - Notices spot terminations, submits additional VMs.

Worker Node VMs

- Condor starts join back to local Condor cluster.
- VMs are identical, don't need public IPs, and don't need to know about each other.

HTCondor pool

- Static Central Manager.
- Dynamic Execute hosts.
AutoPyFactory (APF)

APF (v2) is the ATLAS pilot factory utility

- Multi-threaded OO Python daemon. 1 thread per “APF queue”
- Uses HTCondor-G for grid submission
- Has done all ATLAS pilot submission in the US for 1 year+
- Migration to APFv2 nearly complete in Europe.
- Developed at BNL

Used by ATLAS, but modular and generic:

- Plug-in architecture for WMS, Batch, and scheduling functionality.
- No required ATLAS/Panda coupling or dependencies.
- Used within CloudScheduler at UVic.
Virtual Machines

Worker Node VM creation and deployment using Boxgrinder:

- http://boxgrinder.org/

Notable features:

- Modular appliance inheritance. The wn-atlas definition inherits from the wn-osg and wn-batch profile, which in turn inherit from base.
- “Baked in” HTCondor startd connects back to static Central Manager.
- BG uploads built images directly to Openstack (v3+), EC2, libvirt, or local directory via 'delivery plugins'.

Bad News! Boxgrinder being deprecated.

- Superceded by Aeolus/Oz/Imagefactory.
- Similar, but different model: XML, embedded resources, no inheritance.
- I prefer BG model, but don't have much choice.
WN Image Deployment

Build and upload VM:

```bash
svn co http://svn.usatlas.bnl.gov/svn/griddev/boxgrinder

<Add condor_password file to source tree.>

<Edit COLLECTOR_HOST to point to your collector>

boxgrinder-build -f boxgrinder/sl6-x86_64-wn-atlas.appl -p ec2 -d ami
boxgrinder-build -f boxgrinder/sl6-x86_64-wn-atlas.appl -p ec2 -d ami
   --delivery-config region:us-west-2,bucket:racf-cloud-2

#~.boxgrinder/config

plugins:

openstack:
   username: jhover
   password: XXXXXXXXXX
   tenant: bnlcloud
   host: cldext03.usatlas.bnl.gov
   port: 9292

s3:
   access_key: AKIAJRDFC4GBBZY72XHA
   secret_access_key: XXXXXXXXXXX
   bucket: racf-cloud-1
   account_number: 4159-7441-3739
   region: us-east-1
   snapshot: false
   overwrite: true
```
Condor Scaling: Naive attempt

Known requirements:
- Condor Connecton Broker (some startds are behind NAT)
- Password authentication (simplest secure setup on WAN)
- HTCondor 7.9.x

Naive Approach:
- Single Condor host (schedd, collector, etc.)
- Single process for each daemon

Result: Maxed out at ~3-4 thousand nodes.
- Collector load causing timeouts of schedd daemon.
  - WAN latencies + strong auth?
- Network connections exceeding open file limits/open ports
- Collector duty cycle regularly >= .99.
Collector & Schedd Tuning 1

OS-level Adjustments:

- Host-based firewall open on relevant ports
- Institutional firewall open on relevant ports
- Sufficient open files/ user process limits/ max connections:

```
#/etc/security/limits.conf
* - nofile 1000000
* - nproc unlimited
* - memlock unlimited
* - locks unlimited
* - core unlimited

#/etc/sysctl.conf
fs.file-max = 1000000
```
Collector & Schedd Tuning 2

Split (Collector, Negotiator, CCB) host from Schedd host
  - Protects schedd from Collector and network-related load

Multiple condor_collector processes
  - 20 collector processes reporting to single top-level collector.
  - (glideinWMS uses as many as ~200+.)
  - Execute hosts randomly choose one at boot-time.

Enable the shared port daemon everywhere possible
  - Reduces number of separate TCP connections between execute, schedd, and collector hosts.

Enable session auth.
  - Minimize security re-negotiation, which is very costly in high-latency scenarios.
# /etc/condor/config.d/50cloudcollector.config
# Available at http://svn.usatlas.bnl.gov/svn/griddev/boxgrinder/
# /condor/50cloudcollector.config

# Multiple collector processes
COLLECTOR_HOST=\$(CONDOR_HOST):29650
USE_SHARED_PORT = TRUE
COLLECTOR.USE_SHARED_PORT = FALSE
DAEMON_LIST = COLLECTOR, MASTER, NEGOTIATOR, SCHEDD, SHARED_PORT
COLLECTOR1 = \$(COLLECTOR)
COLLECTOR2 = \$(COLLECTOR)
COLLECTOR1_ARGS = -f -p 29660
COLLECTOR2_ARGS = -f -p 29661
DAEMON_LIST = \$(DAEMON_LIST) COLLECTOR1 COLLECTOR2

# Enable session auth.
SEC_ENABLE_MATCH_PASSWORD_AUTHENTICATION = True

# Don't preempt *ever*
PREEMPT = FALSE
KILL = FALSE
PREEMPTION_REQUIREMENTS = False
RANK = 0
NEGOTIATOR_CONSIDER_PREEMPTION = False
CLAIM_WORKLIFE = 3600
Execute Host Tweaks

Choose random Collector port to connect to.
  - Implemented via /etc/init.d/condorconfig script.

Similar tweaks to Central Manager:
  - Enable SHARED_PORT
  - Enable CCB
  - Enable session auth.

Collect and export cloud info from metadata server:
  - instance-id, public-hostname, public-ipv4

Allow the central management of startd state:
  - This will be used for retirement, shutdown.
  - E.g. via condor_off -peaceful
# /etc/condor/config.d/50cloudcondor.config
# Available at http://svn.usatlas.bnl.gov/svn/griddev/boxgrinder/

DAEMON_LIST = MASTER, STARTD, SHARED_PORT
CCB_ADDRESS = $(COLLECTOR_HOST)
UID_DOMAIN = localhost.localdomain

# Security
ALLOW_WRITE = condor_pool@*
SEC_DEFAULT_AUTHENTICATION = REQUIRED
SEC_DEFAULT_AUTHENTICATION_METHODS = PASSWORD, FS
SEC_PASSWORD_FILE = /etc/condor/password_file
SEC_DEFAULT_ENCRYPTION = REQUIRED
SEC_DEFAULT_INTEGRITY = REQUIRED
SEC_ENABLE_MATCH_PASSWORD_AUTHENTICATION = True

# Allow remote admin
ALLOW_WRITE = $(ALLOW_WRITE), submit-side@matchsession/*
ALLOW_ADMINISTRATOR = condor_pool@*/*

USER_JOB_WRAPPER = /usr/libexec/jobwrapper.sh
DEDICATED_EXECUTE_ACCOUNT_REGEXP = slot[1-8]+
EC2 and HTCondor

On-demand vs. Spot

- Spot: You declare *maximum* price. You pay current, variable spot price. When spot price exceeds your maximum, instance is terminated without warning.
- But partial hours are *not charged*.
- HTCondor handles spot by making one-time spot request, then cancelling it when fulfilled.

Problems:

- Memory provided in units of 1.7GB (less than ATLAS req).
- More (or less) memory than needed per “virtual core”
- On our private Openstack, we created a 1-core, 2GB RAM instance type.
EC2 Spot Considerations

Users utilizing spot pricing need to consider:

- Shorter jobs. Simplest approach. ATLAS originally worked to ensure jobs were at least a couple hours, to avoid pilot flow congestion. Now we have the opposite need.
- Checkpointing. Some work in Condor world providing the ability to checkpoint without linking to special libraries.
- Per-work-unit stageout (e.g. event server in HEP).

With sub 1-hour units of work, users could get significant free time!
## EC2 Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Memory</th>
<th>VCores</th>
<th>“CUs”</th>
<th>CU/Core</th>
<th>$Spot/hr Baseline</th>
<th>$On-Demand/hr</th>
<th>Slots?</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.small</td>
<td>1.7G</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>.007</td>
<td>.06</td>
<td>-</td>
</tr>
<tr>
<td>m1.medium</td>
<td>3.75G</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>.013</td>
<td>.12</td>
<td>1</td>
</tr>
<tr>
<td>m1.large</td>
<td>7.5G</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>.026</td>
<td>.24</td>
<td>3</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>15G</td>
<td>4</td>
<td>8</td>
<td>2</td>
<td>.052</td>
<td>.48</td>
<td>7</td>
</tr>
</tbody>
</table>

### Questions:

- We currently bid 3 * <baseline>. Is this optimal?
  - Seems to result in ~1/3 churn per day.
- Spot is ~1/10th the cost of on-demand. Nodes are ~1/2 as powerful as our dedicated hardware. **Based on estimates of Tier 1 costs, this is competitive.** But need exact numbers.
- Do 7 slots on m1.xlarge perform economically?
Results

HTCondor Scaling Results

- Smooth operation, even with bursts of new execute hosts.
- DaemonCoreDutyCycle ~.35. Lots more headroom.

Overall Project Results

- Ran ~5000 EC2 (1-slot) nodes for ~3 weeks.
- 3 EC2 zones (Virginia, California, Oregon)
- Added in ~250 Openstack slots to virtual pool as well.
- Spent approximately $13K. Only $750 was for data transfer.
- Poor EC2 efficiency poor due to long jobs. Otherwise reliable operation.
- Actual spot price paid very close to baseline, e.g. still less than $.01/hr for m1.small.
Results 2

“Replicate-able”

- Entire setup was duplicated on 2 other hosts in ~3 hours.
- Now running ~4000 slots on Google Compute Engine
- GCE does not permit image upload (yet), so execute adjustments added manually and snapshotted.

Public

- All recipes are in our Boxgrinder SVN repo. No “secret sauce”.
- APF is generic and modular enough to be used as a general-purpose conditional job factory.
Glitches, Issues, Problems

Don't want to preempt, ever!

- Rather tricky to express. Took several iterations.

Allowing admin access to pool from particular user account.

- Required somewhat hack-ish:

  #~/.bash_profile

  export _condor_SEC_PASSWORD_FILE=/var/home/apf/etc/password_file

Documentation

- In general, lack of high-level design documentation combined with detailed recipes (like this use case).
- Now being addressed as a result of this activity.
Glitches, Issues, Problems 2

Multi-collector setup is awkward

- Requires verbose, complex CM config
- Requires execute host coupling (i.e. port range to choose from).
- Team is considering simpler config, e.g.
  \[
  \text{NUM\_COLLECTORS}=20
  \]
  and auto-discovery of real collector port from CM by startd.

Shared port daemon should be the default?

- No harm. Great benefit.
Next Steps

Last piece of the Puzzle: Contraction

- Currently we ramp up and/or maintain (KeepNRunning) automatically with APF.
- APF needs to ramp down by retiring unneeded WNs.
  - Done by correlating condor_q and condor_status information, joining on instance ID.

Noticing terminations as they occur

- Will allow HTCondor to accurately track startd state. (Todd Miller)
- Currently testing Todd M's detection daemon on our WNs.

Drive ATLAS to create short-job workloads appropriate for Spot.

Run another large-scale test. Precisely test CU/slot efficiency.
Conclusions

HTCondor: Configurable, flexible, complete, scale-able.

- HTCondor is configurable enough to be scalable to very high levels, even over WAN.
- HTCondor is flexible enough to be programmatically integrated/controlled by an outside system (e.g. APF in this case).
- HTCondor is complete enough to serve as both local pool infrastructure and as cloud client framework (Condor-G).

Our work was done with a pilot system. But could work the same with real jobs submitted to local pool.
Acknowledgements

Jose Caballero: APF development

Xin Zhao: BNL Openstack deployment

Will Strecker-Kellogg: Local Condor pointers

Todd Miller, Todd Tennenbaum, Jaime Frey, Miron Livny: Condor scaling assistance

David Pellerin, Stephen Elliott, Thomson Nguy, Jaime Kinney, Dhanvi Kapila: Amazon EC2 Spot Team
Questions?
Extra Slides
Boxgrinder Base Appliance

name: sl5-x86_64-base
os:
  name: sl
  version: 5
hardware:
  partitions:
    "/":
      size: 5
packages:
  - bind-utils
  - curl
  - ntp
  - openssh-clients
  - openssh-server
  - subversion
  - telnet
  - vim-enhanced
  - wget
  - yum
repos:
  - name: "sl58-x86_64-os"
    baseurl: "http://host/path/repo"
files:
  "/root/.ssh":
    - "authorized_keys"
  "/etc":
    - "ntp/step-tickers"
    - "ssh/sshd_config"
post:
  base:
    - "chown -R root:root /root/.ssh"
    - "chmod -R go-rwx /root/.ssh"
    - "chmod +x /etc/rc.local"
    - " /sbin/chkconfig ssd on"
    - " /sbin/chkconfig ntpd on"
Boxgrinder Child Appliance

name: sl5-x86_64-batch

appliances:
  - sl5-x86_64-base

packages:
  - condor

repos:
  - name: "htcondor-stable"

files:
  "/etc":
    - "condor/config.d/50cloud_condor.config"
    - "condor/password_file"
    - "init.d/condorconfig"

post:
  base:
    - "/usr/sbin/useradd slot1"
    - "/sbin/chkconfig condor on"
    - "/sbin/chkconfig condorconfig on"
Boxgrinder Child Appliance 2

name: sl5-x86_64-wn-osg
summary: OSG worker node client.
appliances:
    - sl5-x86_64-base
packages:
    - osg-ca-certs
    - osg-wn-client
    - yum-priorities
repos:
    - name: "osg-release-x86_64"
    - name: "osg-epel-deps"
      baseurl: "http://dev.racf.bnl.gov/yum/grid/osg-epel-deps/rhel/5Client/x86_64"
files:
    "/etc":
        - "profile.d/osg.sh"
post:
    base:
        - "/sbin/chkconfig fetch-crl-boot on"
        - "/sbin/chkconfig fetch-crl-cron on"
Boxgrinder Appliance Inheritance

- **sl5-x86_64-base**
  - Adds static OS Yum repo.
  - Useful utilities: curl, ntp, vim, wget, subversion
  - SSH key/server setup.
  - Disable unneeded services.
  - Mounts cloud ephemeral disk

- **sl5-x86_64-batch**
  - Adds WISC Condor Yum repo
  - Adds Condor package
  - Condor configuration (security, slots)

- **sl5-x86_64-wn-osg**
  - Adds OSG Release and Epel deps
  - Yum repos.
  - Installs CA certs and OSG wn-client.
  - Configures OSG environment.
  - Enables CRL update.

- **sl5-x86_64-wn-osg-batch**
  - Sets up OSG wn slot directories.

- **sl5-x86_64-wn-atlas**
  - Adds CVMFS repository.
  - Installs all ATLAS OS dependencies (~45RPMs)
  - Configures CVMFS
#/etc/apf/queues.conf

[BNL_CLOUD]

wmsstatusplugin = Panda
wmsqueue = BNL_CLOUD
batchstatusplugin = Condor
batchsubmitplugin = CondorLocal
schedplugin = Activated

sched.activated.max_pilots_per_cycle = 80
sched.activated.max_pilots_pending = 100
batchsubmit_condorlocal.proxy = atlas-production
batchsubmit_condorlocal.executable = /usr/libexec/wrapper.sh

[BNL_CLOUD-ec2-spot]

wmsstatusplugin = CondorLocal
wmsqueue = BNL_CLOUD
batchstatusplugin = CondorEC2
batchsubmitplugin = CondorEC2
schedplugin = Ready,MaxPerCycle,MaxToRun
sched.maxpercycle.maximum = 100
sched.maxtorun.maximum = 5000

batchsubmit_condorec2.gridresource = https://ec2.amazonaws.com/
batchsubmit_condorec2.ami_id = ami-7a21bd13
batchsubmit_condorec2.instance_type = m1.xlarge
batchsubmit_condorec2.spot_price = 0.156
batchsubmit_condorec2.access_key_id = /home/apf/ec2-racf-cloud/access.key
batchsubmit_condorec2.secret_access_key = /home/apf/ec2-racf-cloud/secret.key