Large-scale Cloud-based clusters using Boxgrinder, Condor, Panda, and APF

John Hover

OSG All-Hands Meeting 2013

Indianapolis, Indiana

Outline

Rationale

- In general...
- OSG-specific

Dependencies/Limitations

Current Status

- VMs with Boxgrinder
- AutoPyFactory (APF) and Panda
- Condor Scaling Work
- EC2 Spot, Openstack

Next Steps and Plans

Discussion

A Reminder

Rationale

Why Cloud interfaces rather than Globus?

Common interface for end-user virtualization management, thus..

Easy expansion to external cloud resources-- same workflow to expand to:

- Local Openstack resources.
- Commercial and academic cloud resources.
- Future OSG and DOE site cloud resources.

Includes all benefits of non-Cloud virtualization: customized OS environments for reliable opportunistic usage.

Flexible facility management:

- Reboot host nodes without draining queues.
- Move running VMs to other hosts.

Flexible VO usage:

- Rapid prototyping and testing of platforms for experiments.

OSG Rationale

Why are we talking about this at an OSG meeting?

- OSG VOs are interested in cloud usage, both local, remote and commercial.
- The new OSG CE (HTCondor-based) could easily provide an interface to local or remote Cloud-based resources, while performing authentication/authorization.
- OSG itself may consider offering a central, transparent gateway to external cloud resources. (Mentioned in Ruth's talk regarding commerical partnerships for CPU and storage.)

This work addresses the ease, flexibility, and scalability of cloud-based clusters.

This talk is a technical overview of an end-to-end modular approach.

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).
- When starting large numbers of VMs, sometimes a few enter ERROR state, requiring removal (Openstack)
- Boxgrinder requires patches for mixed libs, and SL5/EC2.
- EC2 offers public IPs, Openstack nodes often 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.

VM Authoring

Programmatic Worker Node VM creation using Boxgrinder:

- http://boxgrinder.org/
- http://svn.usatlas.bnl.gov/svn/griddev/boxgrinder/

Notable features:

- Modular appliance inheritance. The wn-atlas definition inherits the wn-osg profile, which in turn inherits from base.
- Connects back to static Condor schedd for jobs.
- BG creates images dynamically for kvm/libvirt, EC2, virtualbox, vmware via 'platform plugins'.
- BG can upload built images automatically to Openstack (v3), EC2, **libvirt**, or local directory via 'delivery plugins'.

Important for OSG: Easy to test on your workstation!

- OSG could provide pre-built VMs (would need contextualization) or
- OSG could provide extensible templates for VOs.

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 sshd on"
    - "/sbin/chkconfig ntpd on"
```

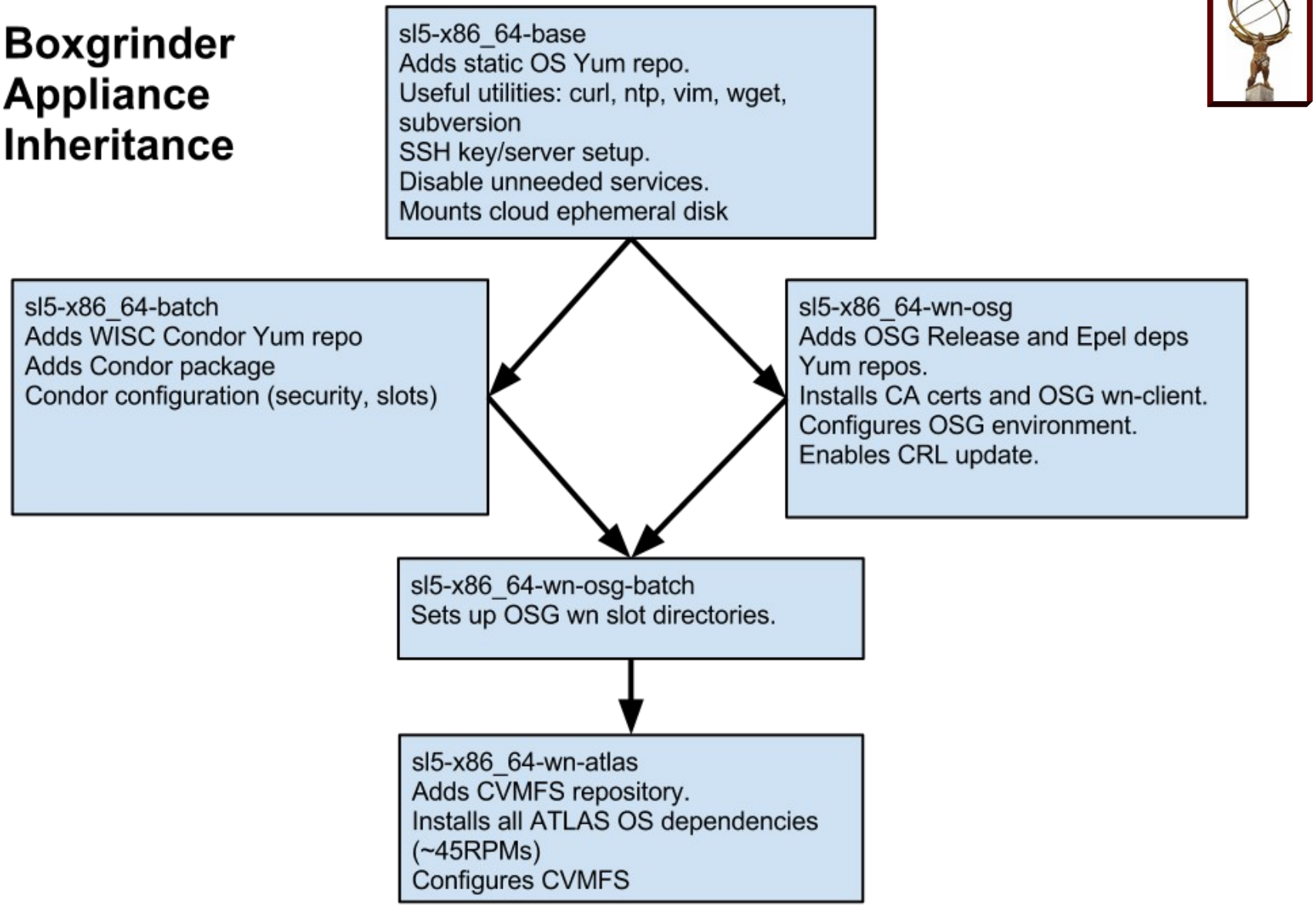
Boxgrinder Child Appliance

```
name: sl5-x86_64-batch
appliances:
  - sl5-x86_64-base
packages:
  - condor
repos:
  - name: "htcondor-stable"
    baseurl:
"http://research.cs.wisc.edu/htcondor/yum/stable/rhel5"

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"
    baseurl: "http://dev.racf.bnl.gov/yum/snapshots/rhel5/osg-release-
2012-07-10/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



WN Deployment Recipe

Build and upload VM:

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

<Add your condor_password file>

<Edit COLLECTOR_HOST to point to your collector>

boxgrinder-build -f boxgrinder/sl5-x86_64-wn-atlas.appl -p ec2 -d ami

boxgrinder-build -f boxgrinder/sl5-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: XXXXXXXXX
    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
```

Elastic Cluster: Components

Static HTCondor central manager

- Standalone, used only for Cloud work.

AutoPyFactory (APF) configured with two queues

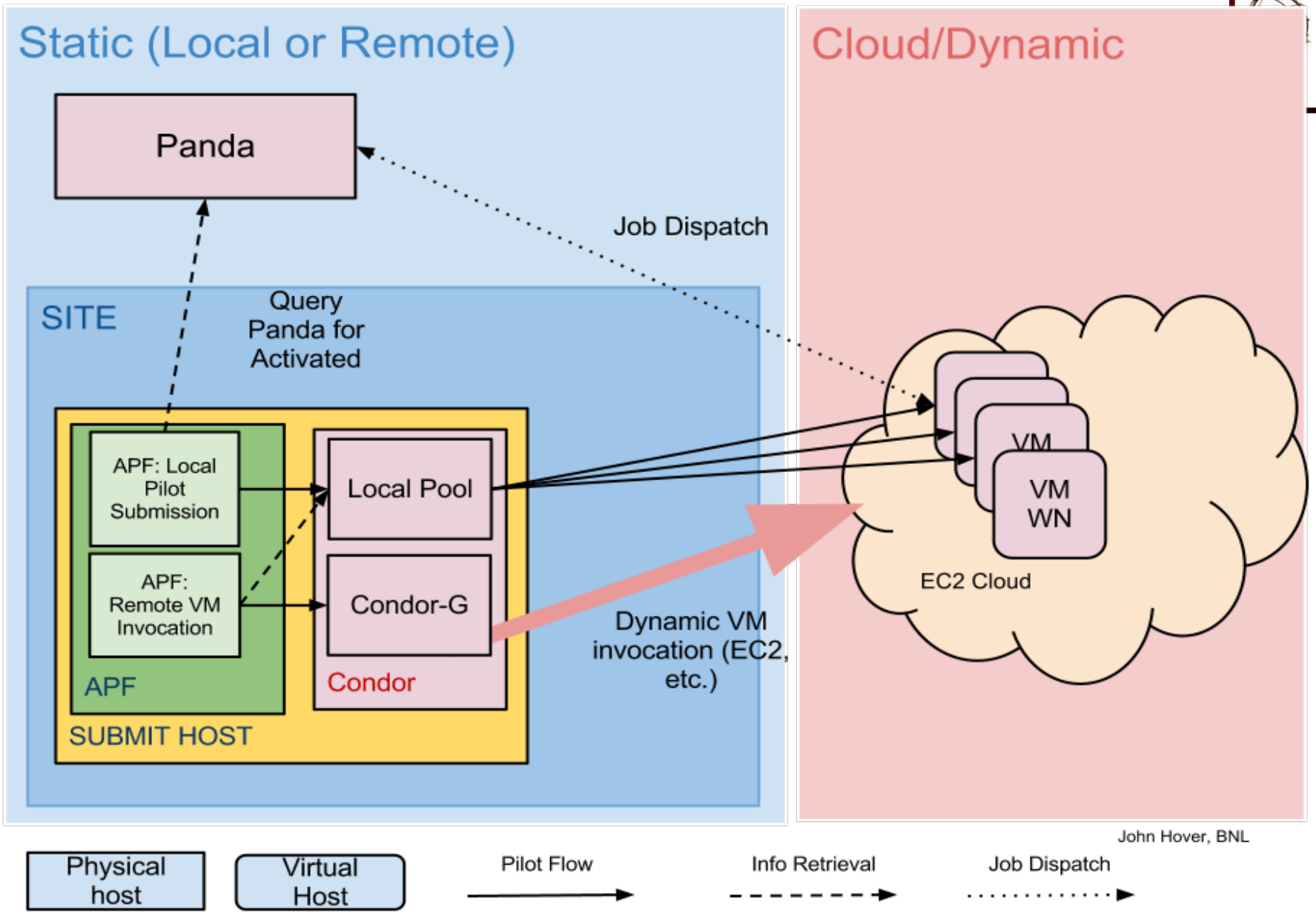
- One observes a Panda queue, when jobs are activated, submits pilots to local cluster Condor queue.
- Another observes the local Condor pool. When jobs are Idle, submits WN VMs to IaaS (up to some limit). When WNs are Unclaimed, shuts them down.

Worker Node VMs

- Generic Condor startds associated connect back to local Condor cluster. All VMs are identical, don't need public IPs, and don't need to know about each other.
- CVMFS software access.

Panda site

- Associated with static BNL SE, LFC, etc.


Static (Local or Remote)
Cloud/Dynamic
Panda
Job Dispatch
SITE
Query Panda for Activated
APF: Local Pilot Submission
APF: Remote VM Invocation
APF
Local Pool
Condor-G
Condor
SUBMIT HOST
Dynamic VM invocation (EC2, etc.)
VM
VM WN
EC2 Cloud
John Hover, BNL
Physical host
Virtual Host
Pilot Flow
Info Retrieval
Job Dispatch

```
#/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
```

Elastic Cluster Components

Condor scaling test used manually started EC2/Openstack VMs. Now we want APF to manage this:

2 AutoPyFactory (APF) Queues

- First (standard) observes a Panda queue, submits pilots to local Condor pool.
- Second observes a local Condor pool, when jobs are Idle, submits WN VMs to IaaS (up to some limit).

Worker Node VMs

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

Panda site (BNL_CLOUD)

- Associated with BNL SE, LFC, CVMFS-based releases.
- But no site-internal configuration (NFS, file transfer, etc).

VM Lifecycle Management

Current status:

- **Automatic ramp-up working properly.**
- Submits properly to EC2 and Openstack via separate APF queues.
- Passive draining when Panda queue work completes.
- Out-of-band shutdown and termination via command line tool:
- Required configuration to allow APF user to retire nodes.
  (_condor_PASSWORD_FILE).

Next steps:

- Active ramp-down via retirement from within APF.
- Adds in tricky issue of “un-retirement” during alternation between ramp-up and ramp-down.
- APF issues *condor_off -peaceful -daemon startd -name <host>*
- APF uses *condor_q* and *condor_status* to associate startds with VM jobs.
  Adds in startd status to VM job info and aggregate statistics.

Ultimate Capabilities

APF's intrinsic queue/plugin architecture, and code in development, will allow:

- Multiple targets
  - E.g., EC2 us-east-1, us-west-1, us-west-2 all submitted equally (1/3).
- Cascading targets, e.g.:
  - We can preferentially utilize free site clouds (e.g. local Openstack or other academic clouds)
  - Once that is full we submit to EC2 spot-priced nodes.
  - During particularly high demand, submit EC on-demand nodes.
  - Retire and terminate in reverse order.

The various pieces exist and have been tested, but final integration in APF is in progress.

Condor Scaling 1

RACF recieved a $50K grant from Amazon: Great opportunity to test:

- Condor scaling to thousands of nodes over WAN
- Empirically determine costs

Naive Approach:

- Single Condor host (schedd, collector, etc.)
- Single process for each daemon
- Password authentication
- Condor Connection Broker (CCB)

Result: **Maxed out at ~3000 nodes**

- Collector load causing timeouts of schedd daemon.
- CCB overload?
- Network connections exceeding open file limits
- Collector duty cycle -> .99.

Condor Scaling 2

Refined approach:

- Tune OS limits: 1M open files, 65K max processes.
- Split schedd from (collector,negotiator,CCB)
- Run 20 collector processes. Startds randomly choose one. Enable collector reporting, sub-collectors report to non-public collector
- Enable shared port daemon on all nodes: multiplexes TCP connections. Results in dozens of connections rather than thousands.
- Enable session auth, so that connections after the first bypass password auth check.

Result:

- Smooth operation up to 5000 startds, even with large bursts.
- No disruption of schedd operation on other host.
- Collector duty cycle ~.35. Substantial headroom left. Switching to 7-slot startds would get us to ~35000 slots, with marginal additional load.

Condor Scaling 3

Overall results:

- Ran ~5000 nodes for several weeks.
- Production simulation jobs. Stageout to BNL.
- Spent approximately $13K. Only $750 was for data transfer.
- Moderate failure rate due to spot terminations.
- Actual spot price paid very close to baseline, e.g. still less than .$.01/hr for m1.small.
- No solid statistics on efficiency/cost yet, beyond a rough appearance of “competitive.”

EC2 Spot Pricing and Condor

On-demand vs. Spot

- On-Demand: You pay standard price. Never terminates.
- Spot: You declare *maximum* price. You pay current,variable spot price. If/when spot price exceeds your maximum, instance is terminated without warning. Note: NOT like priceline.com, where you pay what you bid.

Problems:

- Memory provided in units of 1.7GB, less than ATLAS standard.
- More memory than needed per “virtual core”
- NOTE: On our private Openstack, we created a 1-core, 2GB RAM instance type--avoiding this problem.

Condor now supports submission of spot-price instance jobs.

- Handles it by making one-time spot request, then cancelling it when fulfilled.

EC2 Types

Type	Memory	VCores	"CUs"	CU/Core	$Spot/hr Typical	$On-Demand/hr	Slots?
m1.small	1.7G	1	1	1	.007	.06	-
m1.medium	3.75G	1	2	2	.013	.12	1
m1.large	7.5G	2	4	2	.026	.24	3
m1.xlarge	15G	4	8	2	.052	.48	7

Issues/Observations:

- We currently bid 3 * <baseline>. Is this optimal?
- 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.**
- Amazon provides 1.7G memory per CU, not "CPU". Insufficient for ATLAS work (tested).
- Do 7 slots on m1.xlarge perform economically?

EC2 Spot Considerations

Service and Pricing

- Nodes terminated without warning. (No signal.)
- Partial hours are *not charged*.

Therefore, VOs 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-hour units of work, VOs could get significant free time!

Programmatic Repeatability, Extensibility

The ***key*** feature of our work has been to make ***all*** our process and configs general and public, so others can use it. Except for pilot submission (AutoPyFactory), we have used only ***standard***, widely used technology (RHEL/SL, Condor, Boxgrinder, Openstack).

- Boxgrinder appliance definitions are published, and modular for re-use by OSG and/or other ATLAS groups.
- All source repositories are public and usable over the internet,e.g.:
  - Snapshots of external repositories, for consistent builds:
    - **http://dev.racf.bnl.gov/yum/snapshots/**
    - **http://dev.racf.bnl.gov/yum/grid/osg-epel-deps/**
  - Custom repo:
    - **http://dev.racf.bnl.gov/yum/grid/testing**
- Our Openstack host configuration Puppet manifests are published and will be made generic enough to be borrowed.

Conclusions

Acknowledgements

Jose Caballero: APF development

Xin Zhao: BNL Openstack deployment

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

Reminder

Tomorrow's Blueprint session:

- 11:00AM to 12:00
- Discussion and input rather than a talk.
- Bring your ideas, questions, requests.
- What next-generation technology or approaches do you think OSG should consider or support?

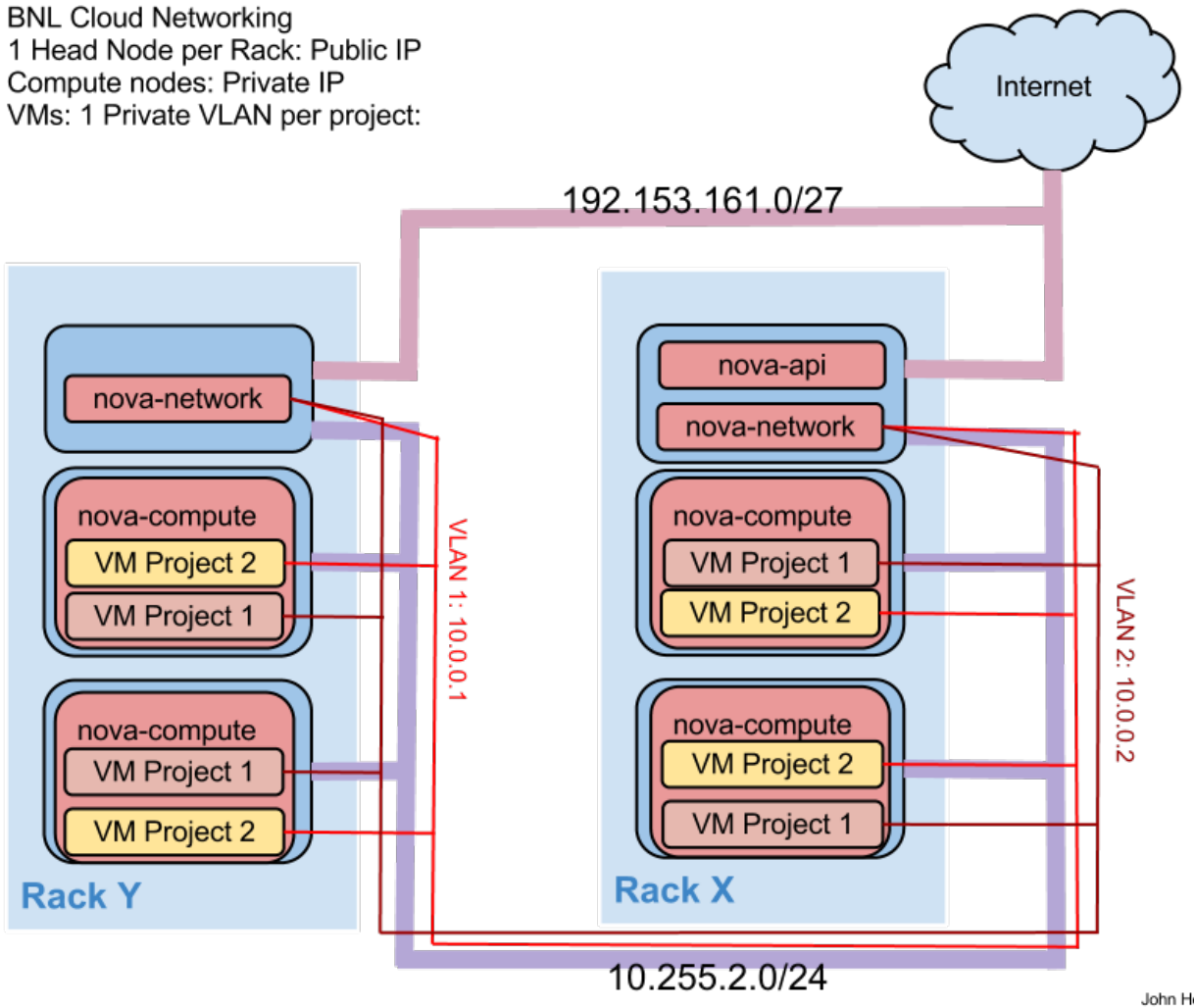
Extra Slides

BNL Openstack Cloud

Openstack 4.0 (Essex)

- 1 Controller, 100 execute hosts (~300 2GB VMs), fairly recent hardware (3 years), KVM virtualization w/ hardware support.
- Per-rack network partitioning (10Gb throughput shared)
- Provides EC2 (nova), S3 (swift), and an image service (glance).
- Essex adds keystone identity/auth service, Dashboard.
- Programmatically deployed, with configs publically available.
- Fully automated compute-node installation/setup (Puppet)
- Enables 'tenants'; partitions VMs into separate authentication groups, such that users cannot terminate (or see) each other's VMs. Three projects currently.
- Winning platform war--CERN switching to OpenStack
  - BNL sent 2 people to Openstack confernce, CERN attended.

BNL Openstack Layout



Next Steps/Plans

APF Development

- Complete APF VM Lifecycle Management feature.
- Simplify/refactor Condor-related plugins to reduce repeated code. Fault tolerance.
- Run multi-target workflows. Is more between-queue coordination is necessary in practice.

Controlled Performance/Efficiency/Cost Measurements

- Test m1.xlarge (4 “cores”, 15GB RAM) with 4, 5, 6, and 7 slots.
- Measure “goodput” under various spot pricing schemes. Is 3*<baseline> sensible?
- Google Compute Engine?

Other concerns

- Return to refinement of VM images. Contextualization.

Panda

BNL_CLOUD

- Standard production Panda site.
- Configured to use wide-area stagein/out (SRM, LFC), so same cluster can be extended transparently to Amazon or other public academic clouds.
- Steadily running ~200 prod jobs on auto-built VMs for months. Bursts to 5000
- Very low job failure rate due to software.
- HC tests, auto-exclude enabled -- no problems so far.
- Performance actually better than main BNL prod site. (slide follows).
- Also ran hybrid Openstack/EC2 cluster with no problems.