HTCondor workflows at Utility Supercomputing Scale: How?

Ian D. Alderman
Cycle Computing

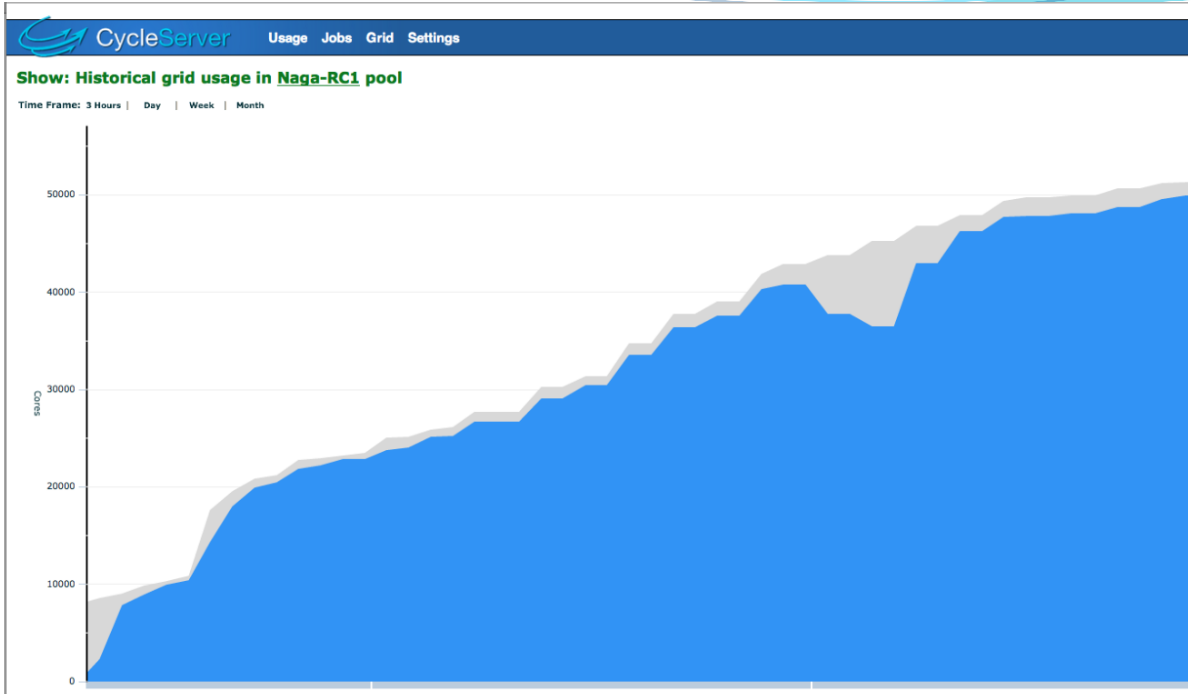
CYCLECOMPUTING

Thundering Herd Problem

CYCLECOMPUTING

Thundering Herd Problem

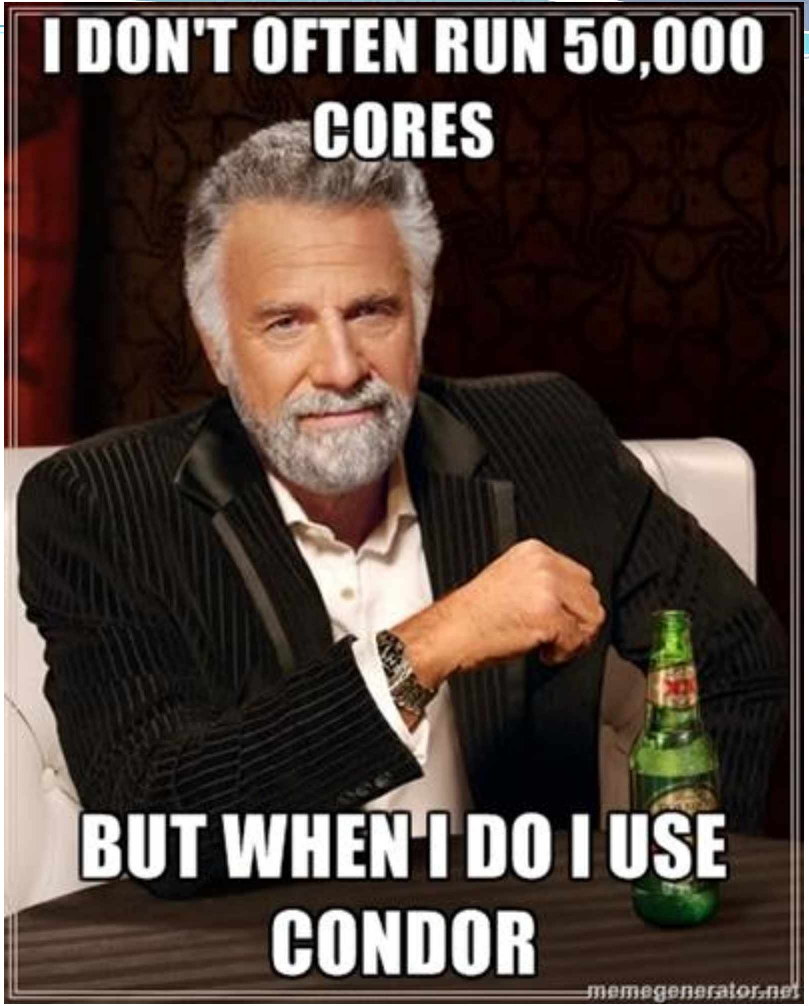
- Classical OS problem: multiple processes are waiting for the same event, but only one can respond at a time.
- In the cloud, what happens to the (underlying) infrastructure when you start 10k servers is someone else's problem.
- What happens at the platform and application level is your problem
- Experience is helpful.



Show: Historical grid usage in Naga-RC1 pool

Time Frame: 3 Hours | Day | Week | Month

Ramping up to 50,000 cores



CYCLECOMPUTING

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.

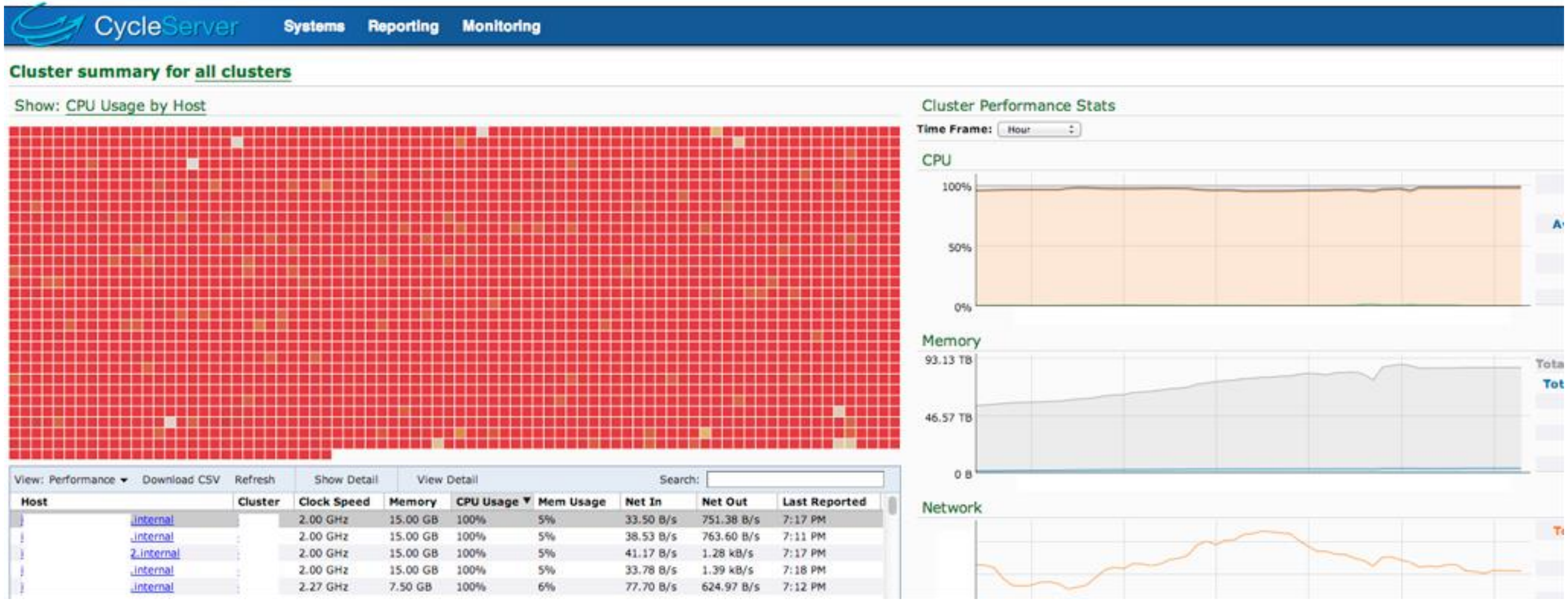
Message in a bottleneck?

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

Host	Cluster	Clock Speed	Memory	CPU Usage ▼	Mem Usage	Net In	Net Out	Last Reported
.internal		2.00 GHz	15.00 GB	100%	5%	33.50 B/s	751.38 B/s	7:17 PM
.internal		2.00 GHz	15.00 GB	100%	5%	38.53 B/s	763.60 B/s	7:11 PM
2.internal		2.00 GHz	15.00 GB	100%	5%	41.17 B/s	1.28 kB/s	7:17 PM
.internal		2.00 GHz	15.00 GB	100%	5%	33.78 B/s	1.39 kB/s	7:18 PM
.internal		2.27 GHz	7.50 GB	100%	6%	77.70 B/s	624.97 B/s	7:12 PM

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.

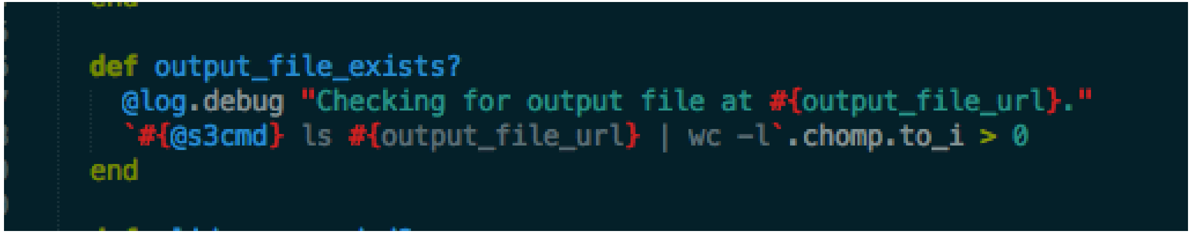
- No file transfer
- No input
- No results
- No output, error or log
- So how does the job do anything?

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.

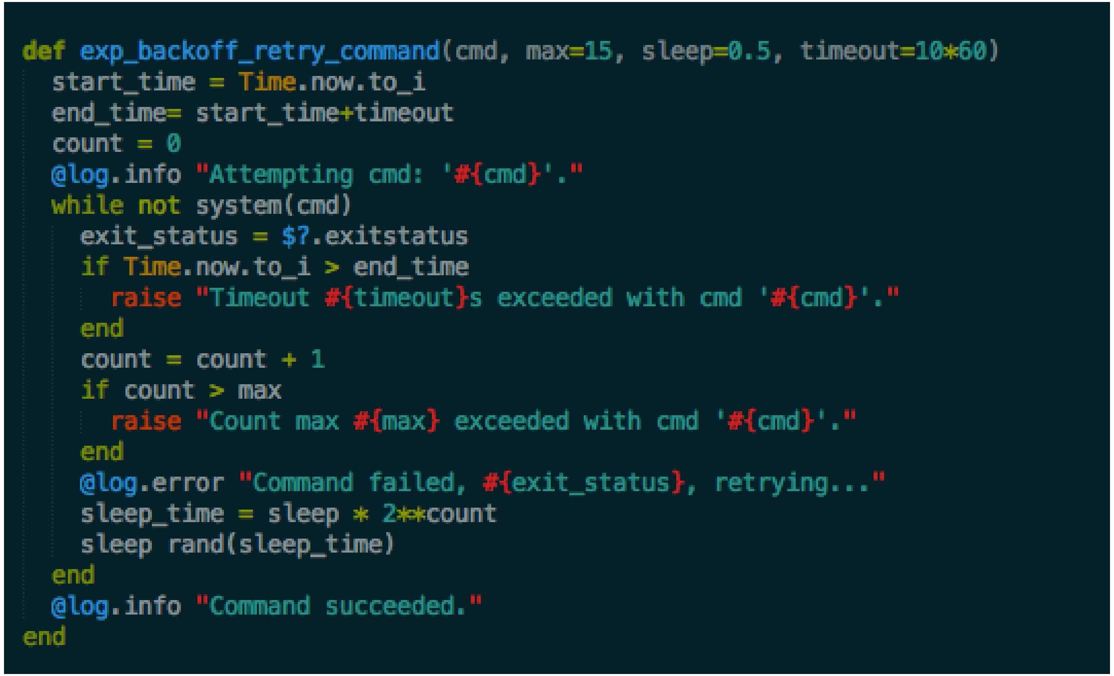
```
opts = GetoptLong.new( [ '--verbose', '-v', GetoptLong::NO_ARGUMENT ],
                       [ '--overwrite', '-o', GetoptLong::NO_ARGUMENT ],
                       [ '--stdout', '-s', GetoptLong::NO_ARGUMENT ],
                       [ '--config', '-c', GetoptLong::REQUIRED_ARGUMENT] )

param = Hash.new
param[:verbose]   = false
param[:stdout]    = false
param[:overwrite] = false

opts.each do |opt, arg|
  case opt
  when '--verbose'
    param[:verbose] = true
  when "--stdout"
    param[:stdout] = true
  when "--overwrite"
    param[:overwrite] = true
  when "--config"
    param[:config] = arg
  end
end

class GlideJobWrapper
  def initialize(a, p)
    @verbose   = p[:verbose]
    @stdout    = p[:stdout]
    @overwrite = p[:overwrite]

    @config = {
```

Don’t overwrite results



```ruby
def output_file_exists?
  @log.debug "Checking for output file at #{output_file_url}."
  `#{@s3cmd} ls #{output_file_url} | wc -l`.chomp.to_i > 0
end
```

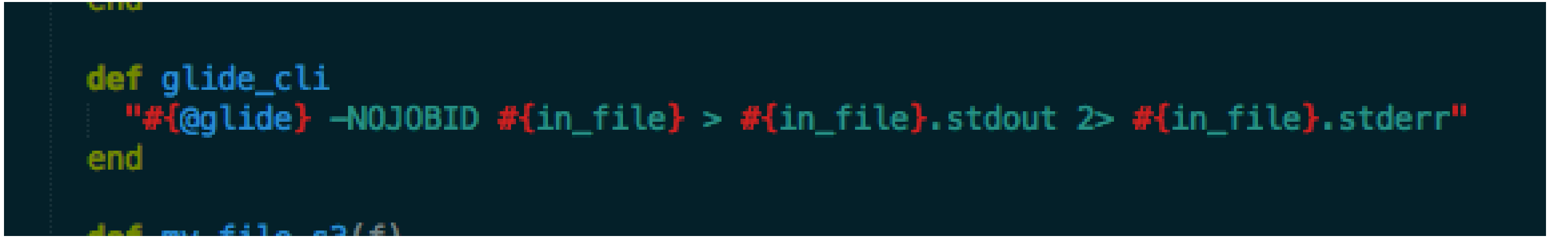
Actual check to see if results are there already



```ruby
def exp_backoff_retry_command(cmd, max=15, sleep=0.5, timeout=10*60)
  start_time = Time.now.to_i
  end_time= start_time+timeout
  count = 0
  @log.info "Attempting cmd: '#{cmd}'."
  while not system(cmd)
    exit_status = $?.exitstatus
    if Time.now.to_i > end_time
      raise "Timeout #{timeout}s exceeded with cmd '#{cmd}'."
    end
    count = count + 1
    if count > max
      raise "Count max #{max} exceeded with cmd '#{cmd}'."
    end
    @log.error "Command failed, #{exit_status}, retrying..."
    sleep_time = sleep * 2**count
    sleep rand(sleep_time)
  end
  @log.info "Command succeeded."
end
```

Exponential back-off for data transfer

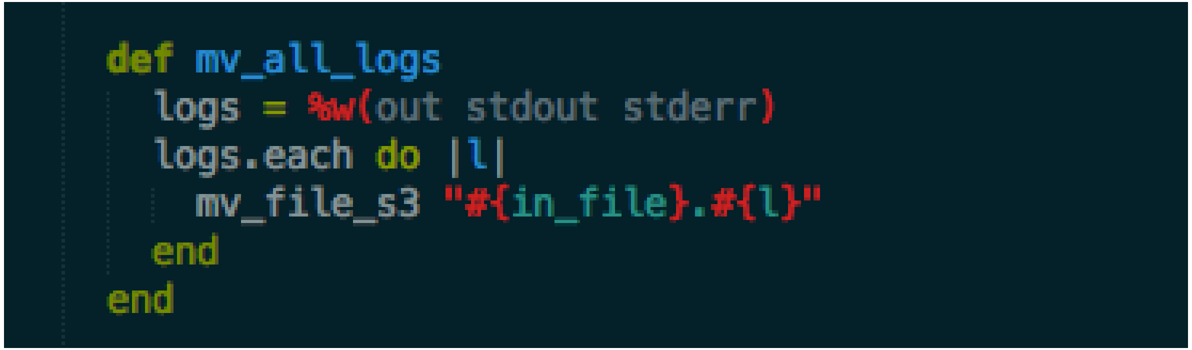
CYCLECOMPUTING



```
def glide_cli
  "#{@glide} -NOJOBID #{in_file} > #{in_file}.stdout 2> #{in_file}.stderr"
end
```

Actual command line captures stdout and stderr

CYCLECOMPUTING

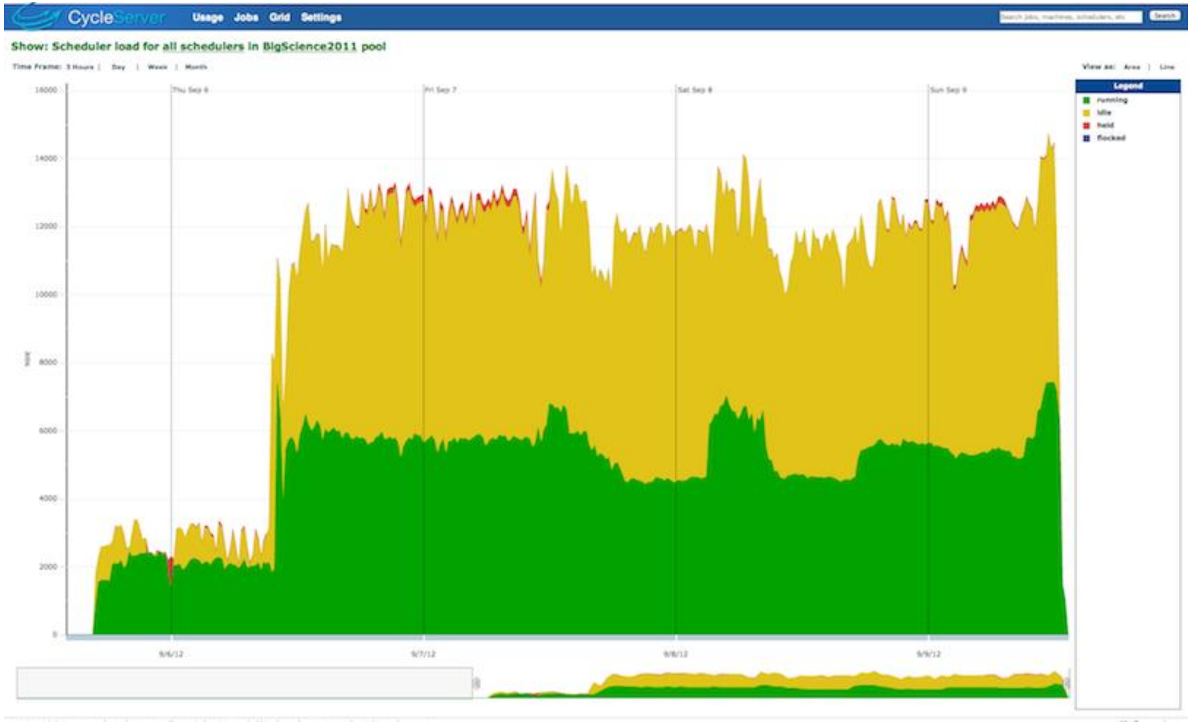


```
def mv_all_logs
  logs = %w(out stdout stderr)
  logs.each do |l|
    mv_file_s3 "#{in_file}.#{l}"
  end
end
```

If command succeeds, save stdout and stderr

Actual submit file

```
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
```



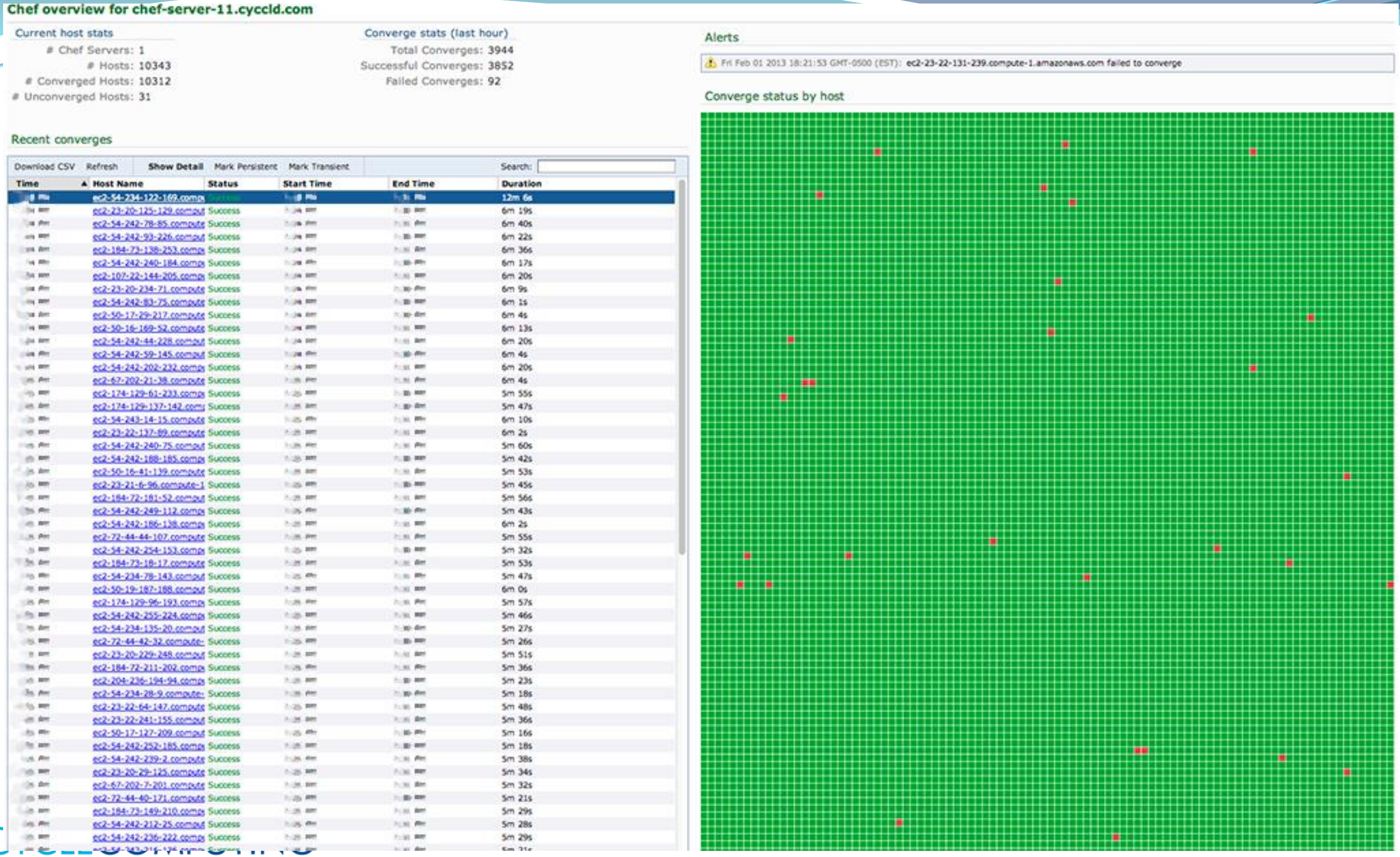
Show: Scheduler load for all schedulers in BigScience2011 pool

DAGMan is your friend

CYCLECOMPUTING

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.cycccld.com

Current host stats

\# Chef Servers: 1
\# Hosts: 10343
\# Converged Hosts: 10312
\# Unconverged Hosts: 31

Converge stats (last hour)

Total Converges: 3944
Successful Converges: 3852
Failed Converges: 92

Alerts

⚠ Fri Feb 01 2013 18:21:53 GMT-0500 (EST): ec2-23-22-131-239.compute-1.amazonaws.com failed to converge

Converge status by host

Recent converges

Time ▲	Host Name	Status	Start Time	End Time	Duration
[redacted]	ec2-54-234-122-169.comp	Success	[redacted]	[redacted]	12m 6s
[redacted]	ec2-23-20-125-129.comput	Success	[redacted]	[redacted]	6m 19s
[redacted]	ec2-54-242-78-85.compute	Success	[redacted]	[redacted]	6m 40s
[redacted]	ec2-54-242-93-226.comput	Success	[redacted]	[redacted]	6m 22s
[redacted]	ec2-184-73-138-253.comp	Success	[redacted]	[redacted]	6m 36s
[redacted]	ec2-54-242-240-184.comp	Success	[redacted]	[redacted]	6m 17s
[redacted]	ec2-107-22-144-205.comp	Success	[redacted]	[redacted]	6m 20s
[redacted]	ec2-23-20-234-71.compute	Success	[redacted]	[redacted]	6m 9s
[redacted]	ec2-54-242-83-75.compute	Success	[redacted]	[redacted]	6m 1s
[redacted]	ec2-50-17-29-217.compute	Success	[redacted]	[redacted]	6m 4s
[redacted]	ec2-50-16-169-52.compute	Success	[redacted]	[redacted]	6m 13s
[redacted]	ec2-54-242-44-228.comput	Success	[redacted]	[redacted]	6m 20s
[redacted]	ec2-54-242-59-145.comput	Success	[redacted]	[redacted]	6m 4s
[redacted]	ec2-54-242-202-232.comp	Success	[redacted]	[redacted]	6m 20s
[redacted]	ec2-67-202-21-38.compute	Success	[redacted]	[redacted]	6m 4s
[redacted]	ec2-174-129-61-233.comp	Success	[redacted]	[redacted]	5m 55s
[redacted]	ec2-174-129-137-142.com	Success	[redacted]	[redacted]	5m 47s
[redacted]	ec2-54-243-14-15.compute	Success	[redacted]	[redacted]	6m 10s
[redacted]	ec2-23-22-137-89.compute	Success	[redacted]	[redacted]	6m 2s
[redacted]	ec2-54-242-240-75.comput	Success	[redacted]	[redacted]	5m 60s
[redacted]	ec2-54-242-188-185.comp	Success	[redacted]	[redacted]	5m 42s
[redacted]	ec2-50-16-41-139.compute	Success	[redacted]	[redacted]	5m 53s
[redacted]	ec2-23-21-6-96.compute-1	Success	[redacted]	[redacted]	5m 45s
[redacted]	ec2-184-72-181-52.comput	Success	[redacted]	[redacted]	5m 56s
[redacted]	ec2-54-242-249-112.comp	Success	[redacted]	[redacted]	5m 43s
[redacted]	ec2-54-242-186-138.comp	Success	[redacted]	[redacted]	6m 2s
[redacted]	ec2-72-44-44-107.compute	Success	[redacted]	[redacted]	5m 55s
[redacted]	ec2-54-242-254-153.comp	Success	[redacted]	[redacted]	5m 32s
[redacted]	ec2-184-73-18-17.compute	Success	[redacted]	[redacted]	5m 53s
[redacted]	ec2-54-234-78-143.comput	Success	[redacted]	[redacted]	5m 47s
[redacted]	ec2-50-19-187-188.comput	Success	[redacted]	[redacted]	6m 0s
[redacted]	ec2-174-129-96-193.comp	Success	[redacted]	[redacted]	5m 57s
[redacted]	ec2-54-242-255-224.comp	Success	[redacted]	[redacted]	5m 46s
[redacted]	ec2-54-234-135-20.comput	Success	[redacted]	[redacted]	5m 27s
[redacted]	ec2-72-44-42-32.compute-	Success	[redacted]	[redacted]	5m 26s
[redacted]	ec2-23-20-229-248.comput	Success	[redacted]	[redacted]	5m 51s
[redacted]	ec2-184-72-211-202.comp	Success	[redacted]	[redacted]	5m 36s
[redacted]	ec2-204-236-194-94.comp	Success	[redacted]	[redacted]	5m 23s
[redacted]	ec2-54-234-28-9.compute-	Success	[redacted]	[redacted]	5m 18s
[redacted]	ec2-23-22-64-147.compute	Success	[redacted]	[redacted]	5m 48s
[redacted]	ec2-23-22-241-155.comput	Success	[redacted]	[redacted]	5m 36s
[redacted]	ec2-50-17-127-209.comput	Success	[redacted]	[redacted]	5m 16s
[redacted]	ec2-54-242-252-185.comp	Success	[redacted]	[redacted]	5m 18s
[redacted]	ec2-54-242-239-2.compute	Success	[redacted]	[redacted]	5m 38s
[redacted]	ec2-23-20-29-125.compute	Success	[redacted]	[redacted]	5m 34s
[redacted]	ec2-67-202-7-201.compute	Success	[redacted]	[redacted]	5m 32s
[redacted]	ec2-72-44-40-171.compute	Success	[redacted]	[redacted]	5m 21s
[redacted]	ec2-184-73-149-210.comp	Success	[redacted]	[redacted]	5m 29s
[redacted]	ec2-54-242-212-25.comput	Success	[redacted]	[redacted]	5m 28s
[redacted]	ec2-54-242-236-222.comp	Success	[redacted]	[redacted]	5m 29s

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

CYCLECOMPUTING