Vac and Vcycle

Andrew McNab

University of Manchester
LHCb and GridPP

Overview

- Vac and Vcycle
- Pilot VM lifecycle
- IaaS Cloud and Vacuum models
- Machine/job features
- Target shares
- Accounting
- GridPP DIRAC and VMs
- Admin-friendly philosophy
- Multiprocessor support
- Future plans

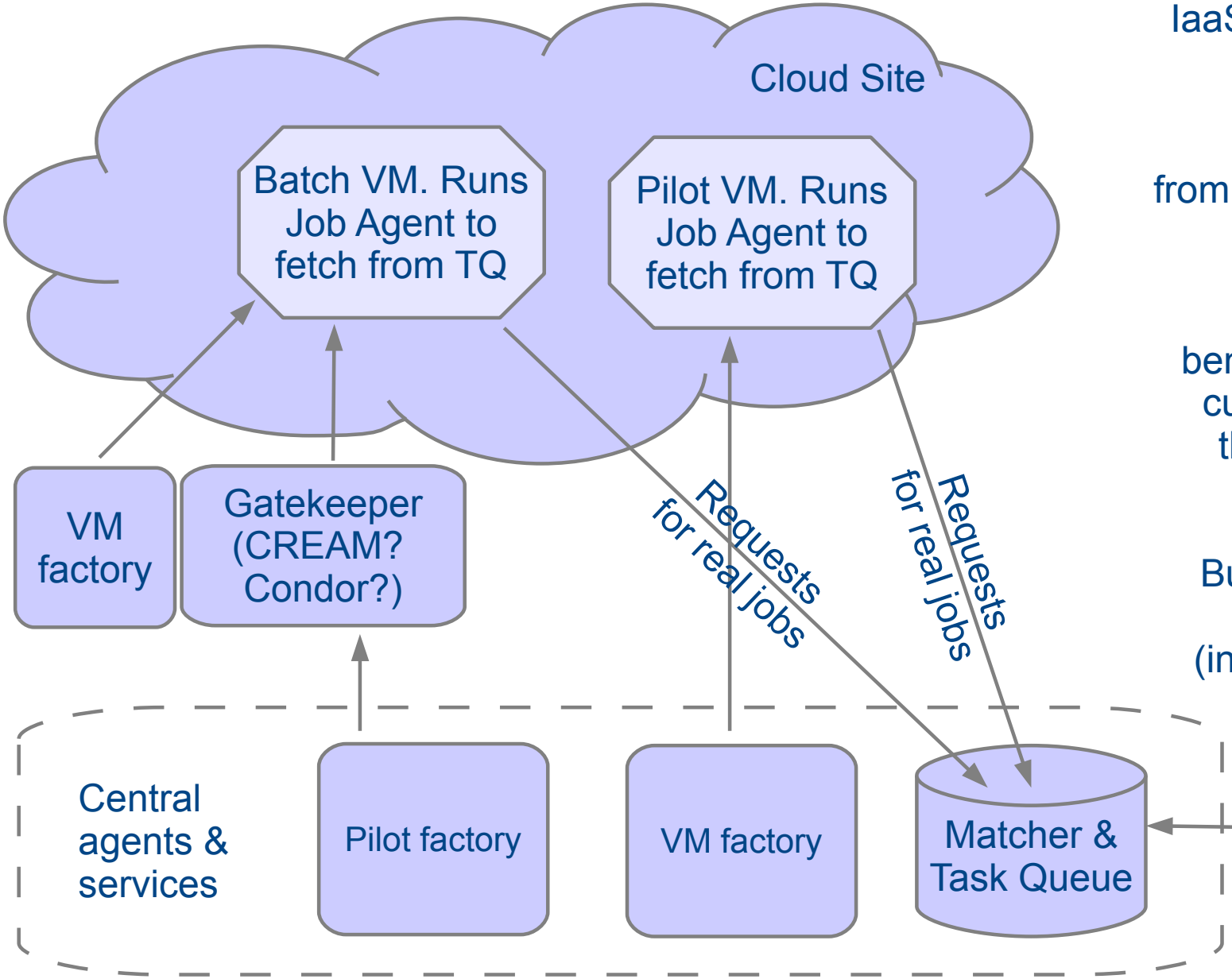
Vac and Vcycle

- Both VM Lifecycle Managers
- Vac is a standalone daemon you run on each worker node machine to create its VMs
- Vcycle manages VMs on IaaS Clouds like OpenStack
  - Can be run at the site, by the experiment, or by regional groups like GridPP
- Both developed at Manchester as part of GridPP Clouds/VMs effort
  - With help from Lancaster, Oxford, IC, CERN, LHCb and ATLAS
- Both make very similar assumptions about how the VMs behave
  - The same LHCb and ATLAS VMs working in production on Vac and Vcycle
  - Compatible GridPP DIRAC VMs available too
- (See my GridPP32 Pitlochry talk about VMs themselves)

“Pilot VM” lifecycle

- Vac and Vcycle assume the VMs have a defined lifecycle
- Need a boot image and user_data file with contextualisation
  - Experiment provides procedure to make a site-wide user_data file
- Virtual disks and boot media defined and VM started
- machinefeatures and jobfeatures directories may be used by the VM to get wall time limits, number of CPUs etc
- The VM runs and its state is monitored
- VM executes shutdown -h when finished or if no more work available
  - Maybe also update a heartbeat file and so stalled or overruning VMs are killed
- Log files to /etc/machineoutputs which are saved
  - shutdown_message file can be used to say why the VM shut down

IaaS models



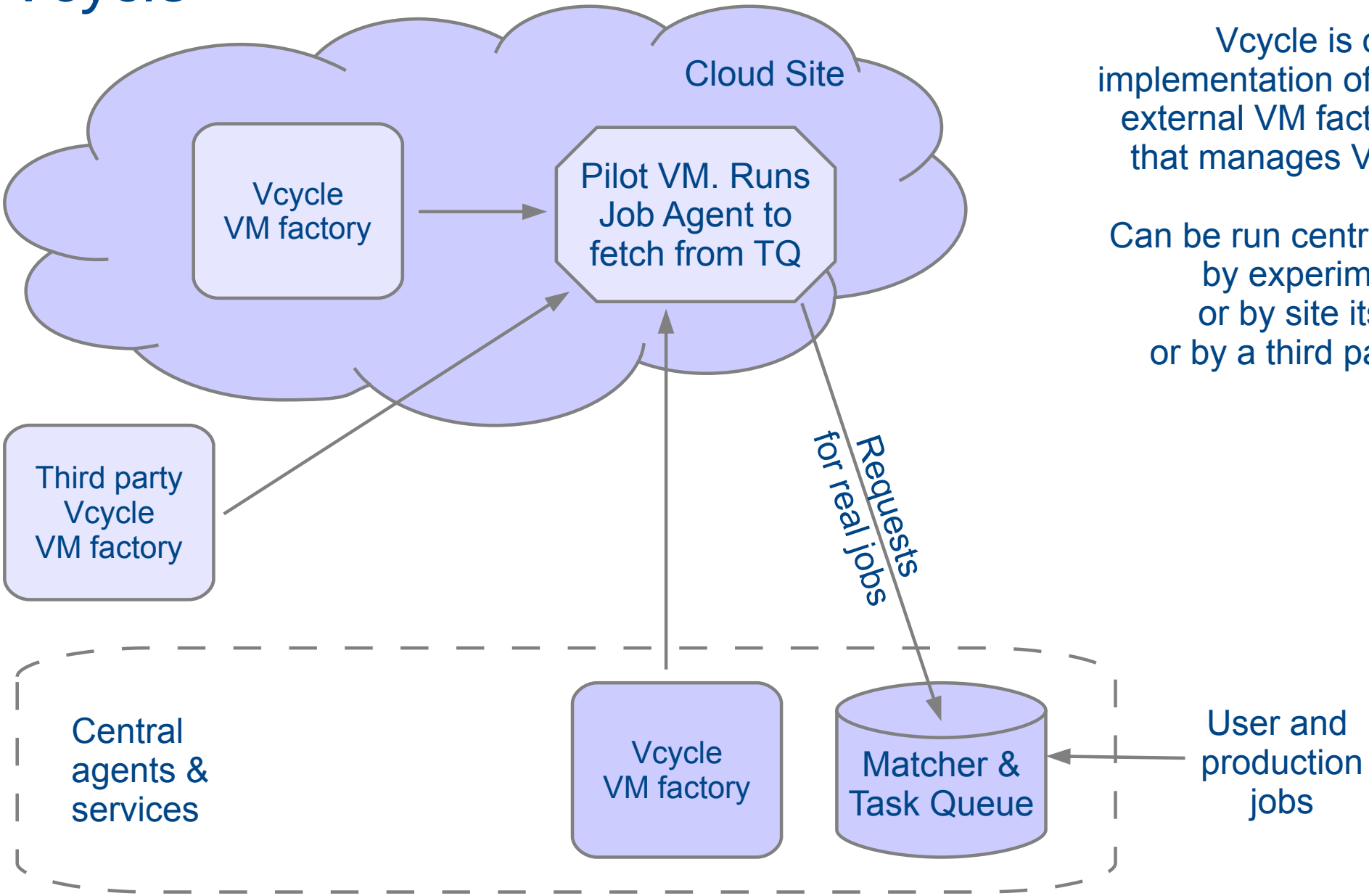
Several ways of using IaaS clouds to get pilot clients running

Fetch payload jobs from task queues as we do with Grid+Batch

These give you the benefits of using VMs: custom environment, that's maintained by experiments

But lots of layers and multiple queues (including inside IaaS provisioning)

Vcycle

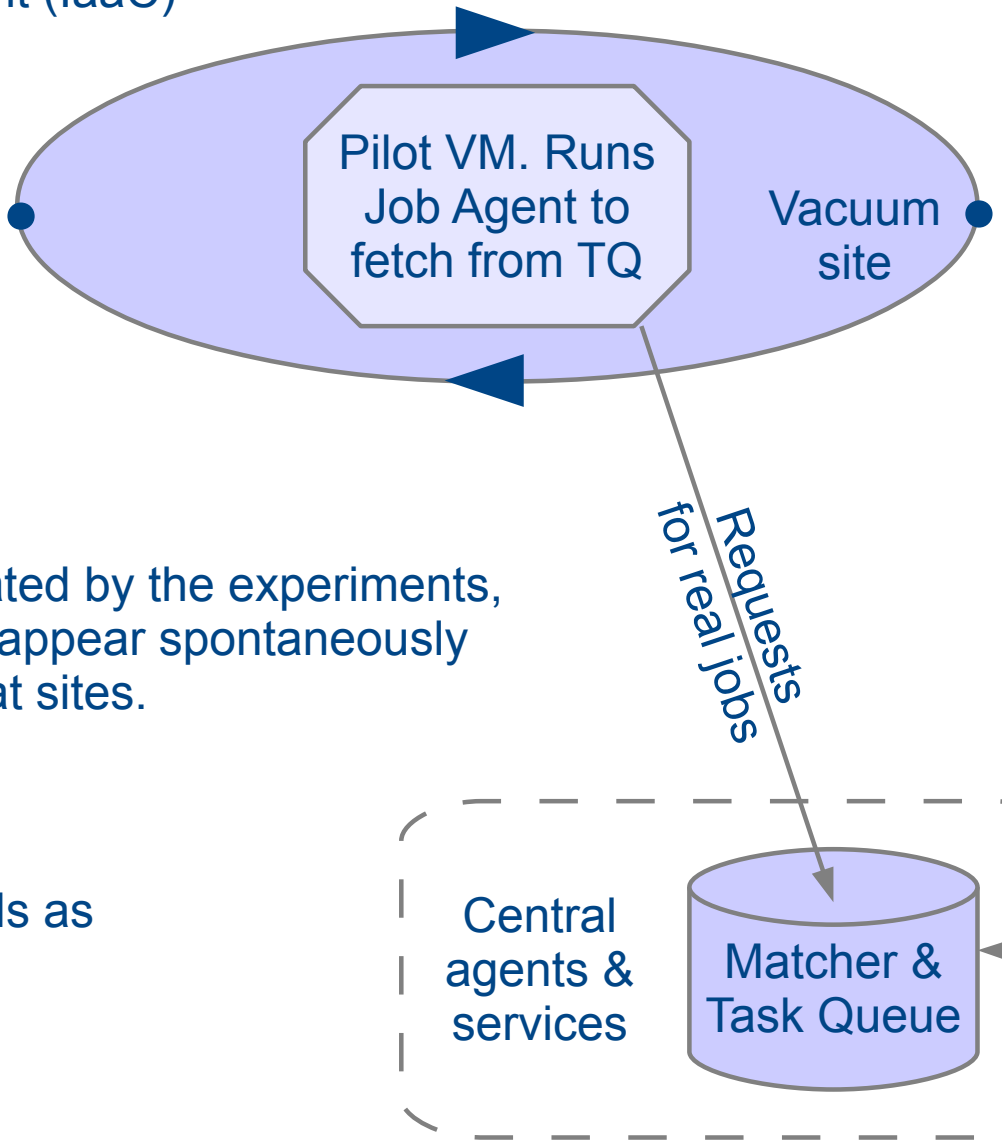


Vcycle is one implementation of an external VM factory that manages VMs

Can be run centrally by experiment or by site itself or by a third party

Vac's Vacuum Model

Infrastructure-as-a-Client (IaaC)



Since we have the pilot framework, we can do something much simpler.

Strip the system right down and have each physical host at the site create the VMs itself.

Instead of being created by the experiments, the virtual machines appear spontaneously "out of the vacuum" at sites.

Ideally use same VMs as with IaaS clouds

Vacuum model

- For the experiments, VMs appear by “spontaneous production in the vacuum”
  - Like virtual particles in the physical vacuum: they appear, potentially interact, and then disappear
- From the CHEP 2013 paper:
  - *“The Vacuum model can be defined as a scenario in which virtual machines are created and contextualized for experiments by the site itself. The contextualization procedures are supplied in advance by the experiments and launch clients within the virtual machines to obtain work from the experiments' central queue of tasks.”*
- At many sites, 90% of the work is done by 2 or 3 experiments
  - So a simple, reliable way of running their “baseload” of jobs is worthwhile
- cvmfs and pilots mean a small user_data file is all the site needs
  - Experiments can provide a script to create the site's user_data

Vac implementation

- On each physical node, Vac VM factory daemon runs to create and supply contextualization user_data to transient VMs
- Multiple VM flavours (“VM types”) are supported, ~1 per experiment
- Each site or Vac “space” is composed of autonomous factory nodes
  - All using the same /etc/vac.d/*.conf files; supplied by Puppet, Chef, Cfengine, ...
- Factories communicate load info with each other via UDP
- Natively supports CernVM 2 (~SL5) and CernVM 3 (~SL6)
  - Also provides a logical partition to the VM to use as fast workspace
- VMs on a NAT network, with the factory node at 169.254.169.254
- Vac assumes the VM will shut itself down if it has no work
  - Vac can also check a heartbeat file and destroy stalled/idle VMs
  - May add optional check on CPU usage too

Vcycle implementation

- Applies Vac ideas to OpenStack etc IaaS resources
- Experiment-neutral, and can be run by experiment or site or 3rd party
- Vcycle daemon creates VMs using user_data file
- Watches what they do
- Backs off if they are failing to stay running
  - No work? Fatal errors?
  - Can also use shutdown messages to make better decisions
- Provides machine/job features via HTTP
  - Also used to collect log files, shutdown messages, and heartbeat updates
- Currently uses OpenStack native nova API
  - OCCI fork has been done by WLCG team at CERN (Luis Villazon Esteban)
  - Reintegration and EC2 support in progress

Machine/job features

- Proposed HEPiX protocol and current WLCG task force
- Allows site/host to communicate details of machine and the job slot to the job or VM
  - HS06, shutdown time for VM/host, CPU and memory limits, ...
- One key file per key/value pair, in one of two directories
- The Vac factory node offers these directories to its VMs via NFS over its internal private network
  - Also a writeable NFS directory for log files, shutdown reason, heartbeat files etc
- Vcycle does this use HTTP(S) web server on the Vcycle machine
- The basis for several scenarios for telling VMs and payload jobs what resources they have and how long they can run for
  - Want graceful termination of VMs to avoid disrupting payload jobs
  - /etc/machinefeatures/shutdowntime always set using max_wallclock_seconds

Target shares with Vac

- When a VM slot becomes available, the node decides how to fill it.
- The other Vac nodes are queried via UDP to discover what they are running, in units of HS06
- The node bases decision on its list of target shares for each VM type
  - The VM type with the least slots, weighted by target share, gets the free slot
  - Uses a site-wide back-off procedure to veto VM types that don't have any work
- This approach is very simple, means factory nodes are autonomous
  - Avoids a central management daemon which would be a single point of failure
- The target shares are instantaneous
  - They are fair, in that if all experiments submit lots of jobs, the site shares out the capacity according to the stated shares
  - But quiet periods aren't credited and carried forward, so may need to adjust targetshares to achieve annual shares, say (as many batch sites do...)

Target shares with Vcycle

- When a VM slot becomes available, Vcycle decides how to fill it
- Go through each VM type seeing if it is suitable to be created
- Limits for each VM type are taken into account
  - Uses a site-wide back-off procedure to veto VM types that don't have any work
  - Can mix VMs for different experiments in the same Project/Tenancy
  - Target shares are currenly 1:1 if limits haven't been reached, but changing to Vac model to allow X:Y ratios between VM types
- Target shares/limits are at the level of Projects/Tenancies
  - Vcycle instances can support multiple projects, perhaps at multiple remote sites
  - Different target share pools in the different projects
  - eg Vcycle instance at Manchester manages ATLAS and LHCb VMs at IC, ATLAS VMs at CERN
- The target share model is instantaneous, as with Vac

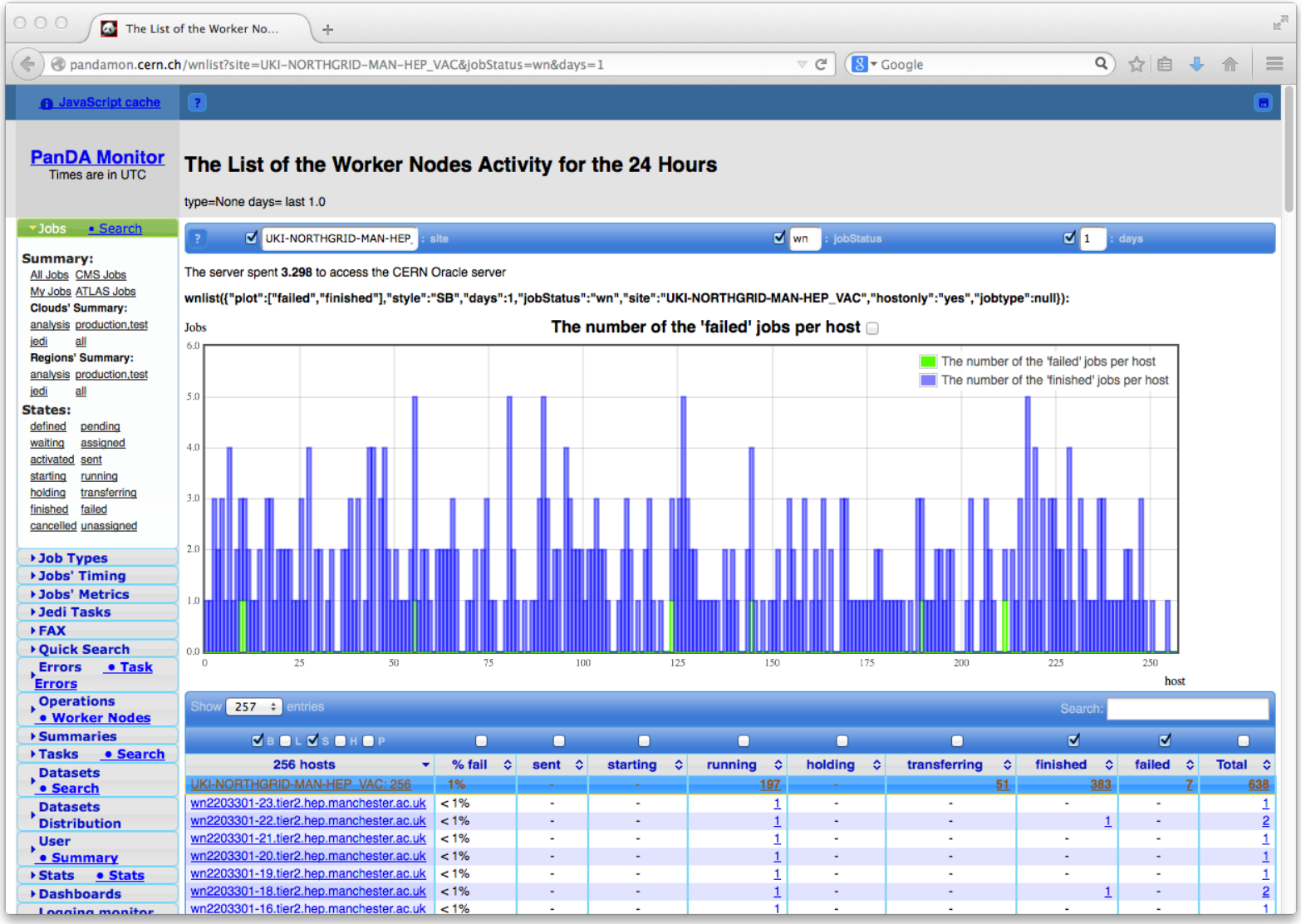
Accounting with Vac

- Many sites need to report usage through APEL and EGI Accounting
- Well established for conventional gatekeeper+batch grid sites
  - There is also an EGI Cloud accounting activity
- Vac records data by updating PBS/BLAH format accounting log files on each factory node when each VM finishes
- These are consumed by the normal APEL PBS log file analyzer and published in the same way as gatekeeper+batch resources
- APEL 3 parser / local DB verified to be working properly with Vac
- In the future, intend to support direct reporting to central APEL service using ssm (as ARC does)
- Intend to support this in Vcycle too, as one option for sites to do Cloud accounting
  - Especially useful if cannot add EGI accounting to the OpenStack resources

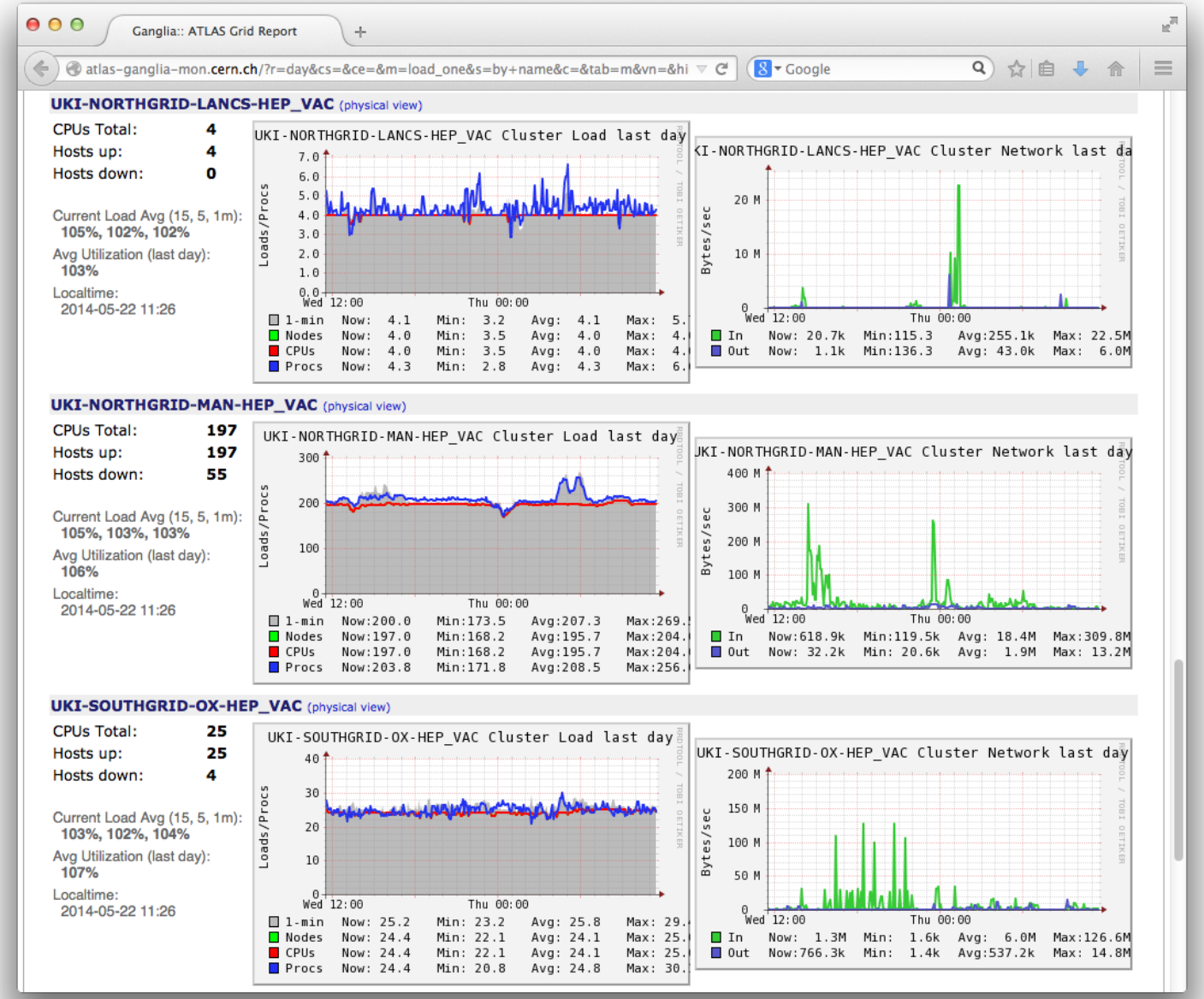
Vac and Vcycle at sites

- Manchester
  - Vac running ATLAS and LHCb VMs
  - Vcycle instance managing Imperial (LHCb/ATLAS) and CERN (ATLAS) VMs
- Lancaster
  - Vac running ATLAS and LHCb VMs
  - Vac being added to old farm now
- Oxford
  - Vac running ATLAS and LHCb VMs
- Imperial
  - GridPP OpenStack tenancy “gridpp-vcycle” has LHCb and ATLAS VMs
- CERN
  - LHCb tenancy managed by Vcycle on LHCb vobox at CERN
  - (A few) ATLAS VMs managed by Vcycle at Manchester

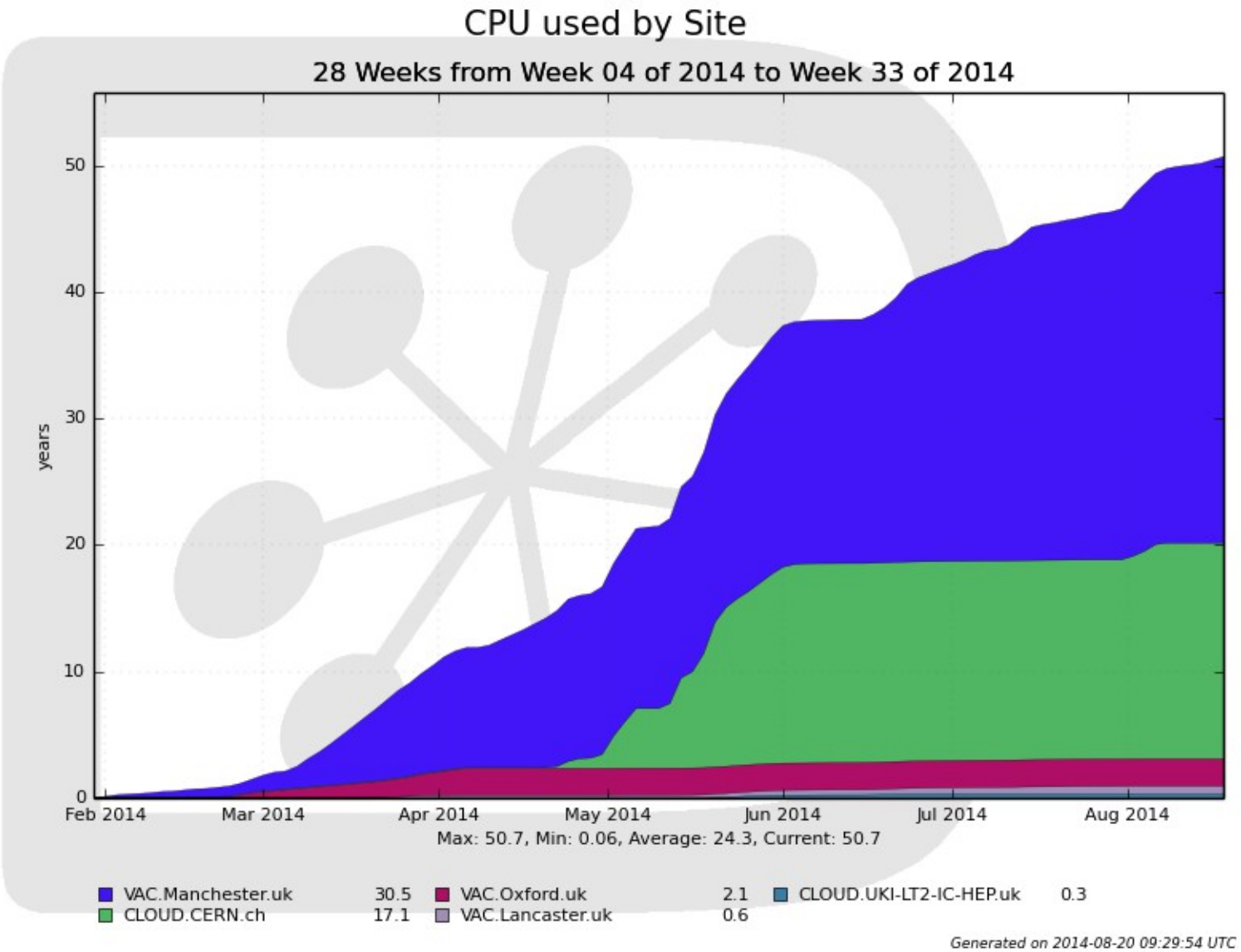
ATLAS Panda jobs running on Vac VMs



Ganglia monitoring of ATLAS Vac sites



LHCb CPU time from Vac and Vcycle VMs



GridPP DIRAC and VMs

- GridPP is the grid consortium of UK particle physics institutes (~20)
- Runs a DIRAC service at Imperial, to support smaller/newer experiments (eg NA62) and to provide catch-all VO (gridpp)
  - Running DIRAC pilots and payloads at sites via CREAM / ARC, plus some Vac
- Last month GridPP agreed the aim to get as many sites as possible to run DIRAC jobs in VMs
  - Either using Vac, or IaaS Cloud (OpenStack, …) plus something (Vcycle)
- Starting to monitor DIRAC job execution at sites with SAM jobs to verify the ongoing status, so can report in weekly ops meetings
- Is then easy to add LHCb and ATLAS VMs running production jobs
  - Also intend to support existing CMS VMs, so non-CMS sites can run as Tier-3s
- **So a lot more sites in the UK are likely to be running jobs in VMs**

Vac's admin-friendly philosophy

- Vac re-reads configuration and rebuilds its view of what the VMs are doing at the start of each cycle (60secs)
  - Do not need to restart the daemon when changing things (eg via Puppet)
  - Do not need to worry about daemon vs VM inconsistent states
  - We frequently do RPM updates of Vac without disrupting production VMs
- Simple so reliable: boot failure rate is << 1/1000
- Proper man pages for vac command, vacd, vac.conf etc
- Admin Guide with examples and help with “gotchas”
- Sanity checks (eg NAT iptables set up?) and log file warnings
- Nagios monitor provided
- Factory nodes autonomous so can easily take sets of machines down
  - Or deal with losing the power on one rack without this disturbing the others!
- Aim to be as simple to manage as using Apache to serve static files

Multiprocessor VMs

- These are supported by Vac as a parameter in the node's configuration file
- Each factory node has one current value in force
- Can be changed at any time, and new VMs will be created using it
- Vac keeps track of existing VMs' geometry to avoid overcommitting
- VM processor count taken into account by targetshares mechanism
- This system is designed to allow dynamic repartitioning of a Vac space into, say, single processor and 8-processor VM subspaces
  - Just update node's configuration parameter in Puppet etc and wait
- This parallels the dynamic partitioning models being evaluated by the WLCG Multicore Task Force for conventional batch systems
- Similar counting procedure needs to be added to Vcycle too

Vcycle work at CERN

- Laurence, Luis, and others looking at using Vcycle to run VMs in tenancies available through EGI
- Thinking of the problem in terms of wanting to provide a “steady pressure” of job slots as long as there are payloads waiting to be run
- Vcycle is one way of doing this, as it monitors whether VMs find payloads to run or not
- Have produced a version of Vcycle that uses OCCI API rather than OpenStack's Nova API
- Using this for ATLAS and looking at CMS. Would be interested in collaboration with ALICE too (LHCb already known to work.)
- As with grid, EGI aims to provide a common interface for all participating sites, the same images available, and to take care of APEL accounting at the site end

Future plans

- Recruit more sites – you?
- See if other experiments' VMs need alternative VM models
- Support Cloud Init contextualization and EC2 metadata in Vac
- Direct APEL 3 reporting using ssm for both
- Use GridPP DIRAC to roll out IaaC/IaaS VMs to more sites
- Reusable Puppet module for Vac
- Finish integrating OCCI and EC2 support in Vcycle

- But avoid “feature creep”, since simplicity is a feature in itself

Summary

- Vac provides a simple way for sites to run VMs
- Vcycle provides a simple way to manage VMs on OpenStack
- Both demonstrated with ATLAS and LHCb production jobs
  - Measured VM efficiency for MC is very good now

- GridPP is in a very good position in this field
- We are able to run production work in VMs if/when the experiments or projects start pushing in that direction

- See http://www.gridpp.ac.uk/vac/ for RPMs, Yum repo, links to GitHub, docs, man pages etc

Extra slides

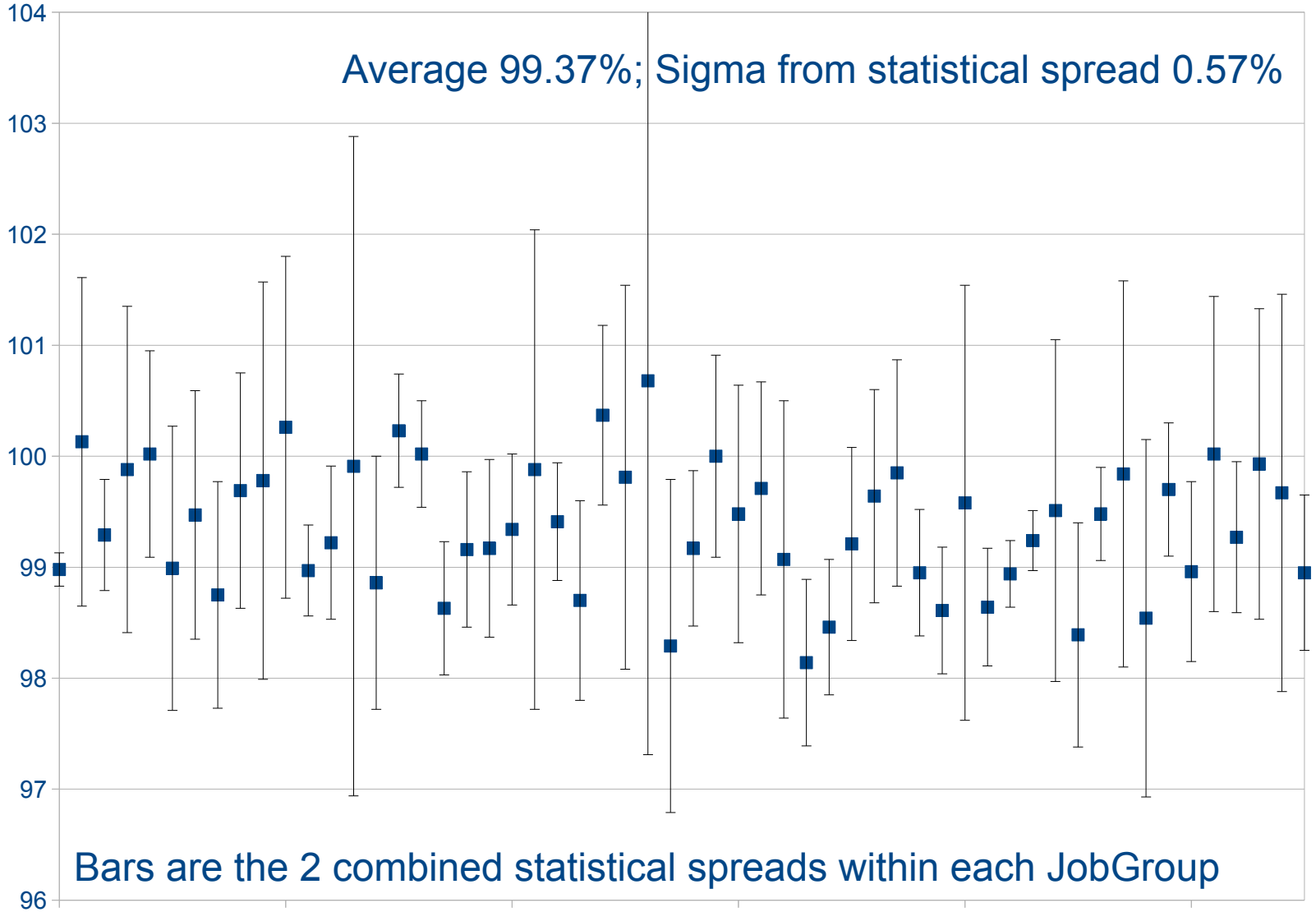
Efficiency measurements for HEPiX in May

- What is today's intrinsic cost of using VMs rather than real machines?
  - Lots of hard work done by many people: VT-x, kvm, qemu-kvm, ksm, virtio
- Can get overall cpu/wall efficiency measurements from accounting
- At Manchester, these are comparable for Batch vs VMs
  - So for April/May 2014, the weighted average ATLAS+LHCb efficiencies for batch and Vac VMs were both 92.0%
  - That doesn't distinguish between setup CPU, overhead CPU, and payload CPU
- People have made direct benchmark comparisons or done test runs
  - But can we measure this with routine workloads too?
- We have machines from the same 2010 purchase running both
  - 288 VM slots on Vac; and ~1000 PBS batch job slots
- So can do direct comparison of efficiency of production payload jobs

VM vs Batch efficiency with LHCb MC

- LHCb Monte Carlo production is organised as JobGroups
  - Each job in a JobGroup is part of one production, with the same parameters
  - Expect similar amount of time in event generator, detector physics, reco etc
- We run enough jobs at Manchester that jobs from the same JobGroup will be run on both Batch and VM
  - Sample was 5534 (3216+2318) jobs in 56 JobGroups, 15-22 May 2014
- Using LHCb DIRAC monitoring of payload jobs, can calculate the CPU/Walltime efficiencies for the Batch jobs and for the VM jobs within the same JobGroup separately
- Can then take quotient to give VM/Batch efficiency ratio
  - Also calculate statistical spreads within each group
- The machines we used are dual 6-core Westmere from 2010, with HT enabled, 2GB/slot, and running 24 batch jobs or 24 VMs

VM eff. / Batch eff., per LHCb MC JobGroup



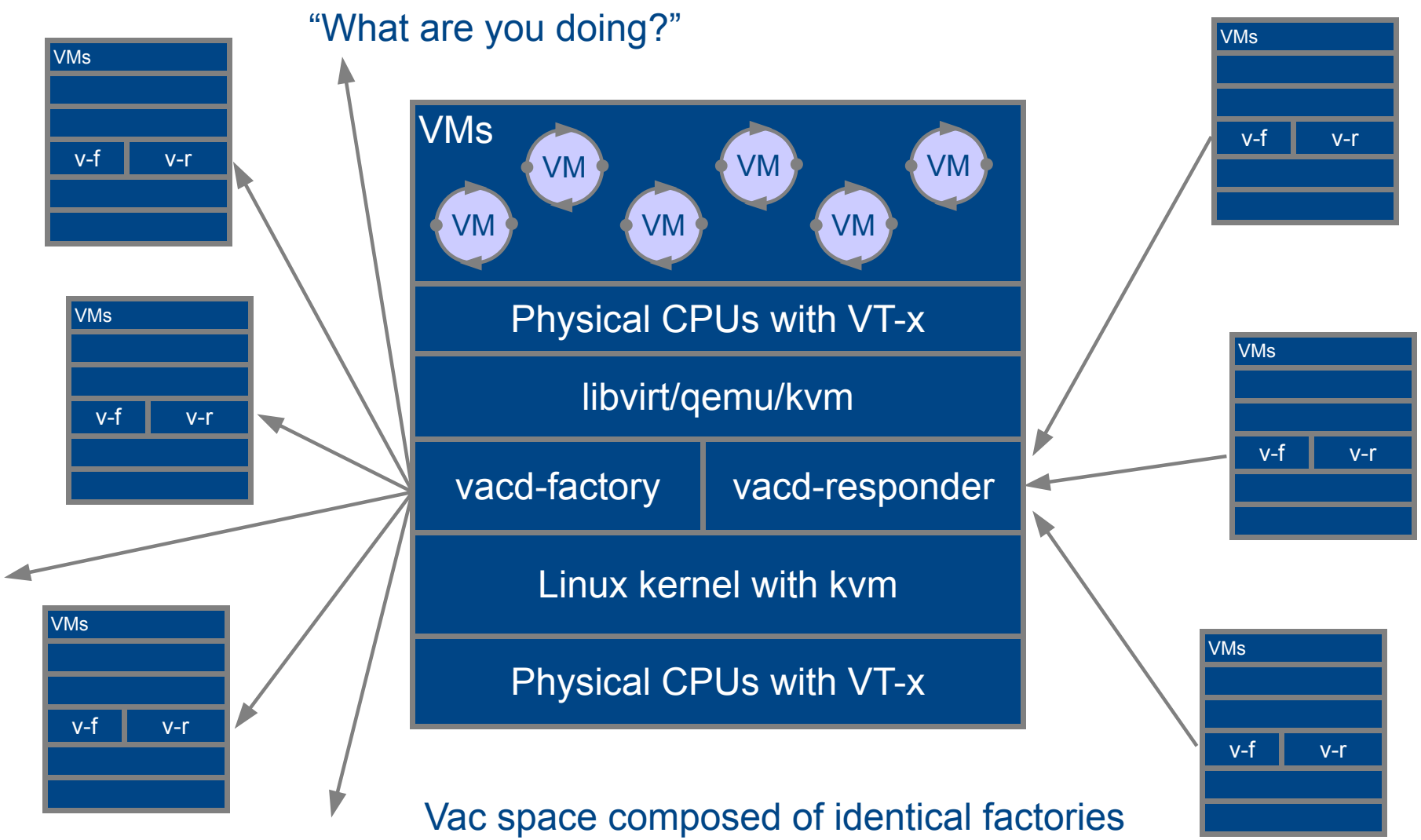
15-22 May 2014

56 LHCb MC JobGroups; 5534 jobs

3216 Batch jobs; 2318 VM jobs

Machines: 12-core Westmere, HT enabled, 2GB/slot, 24 job or VM slots

Vac factory node and site architecture

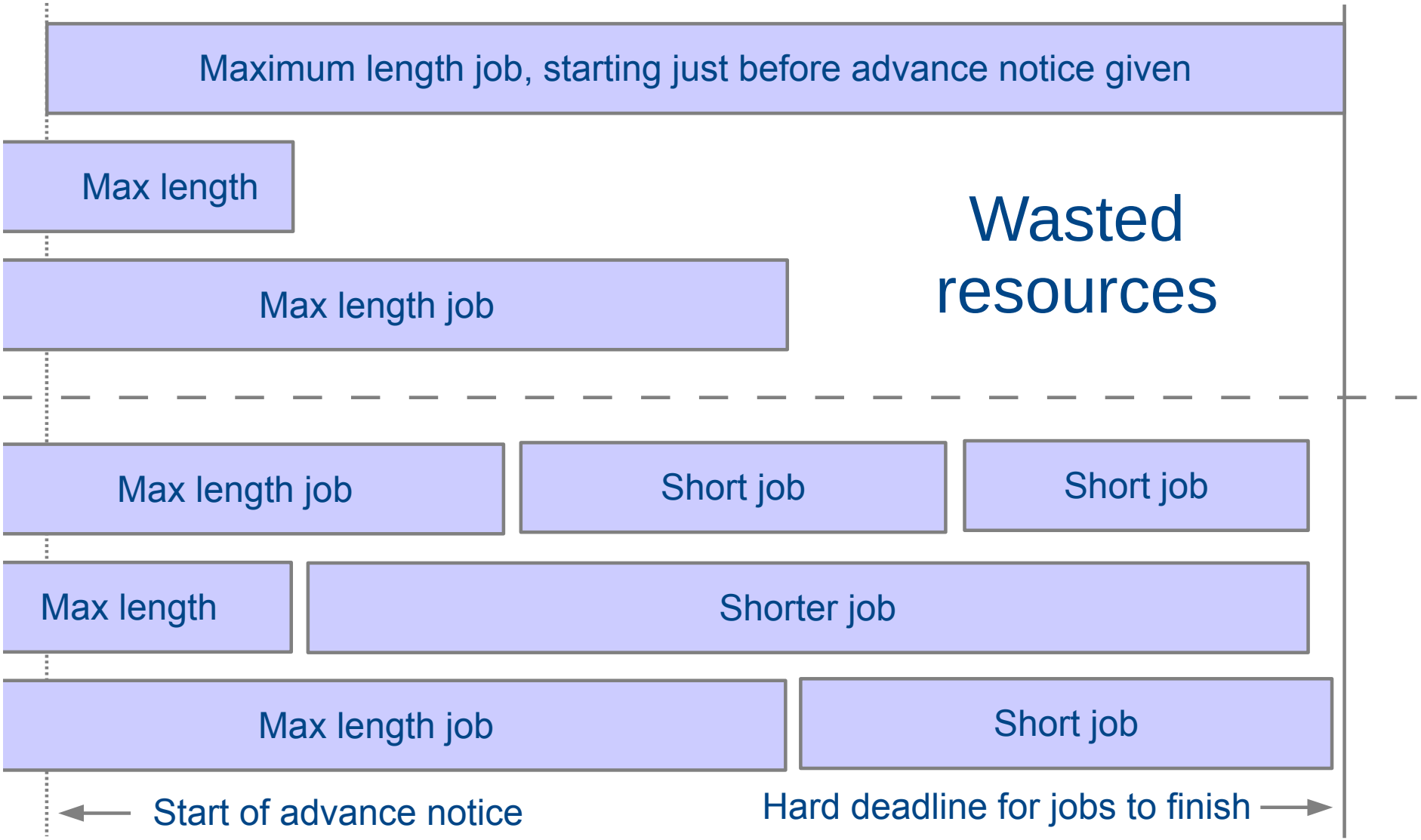


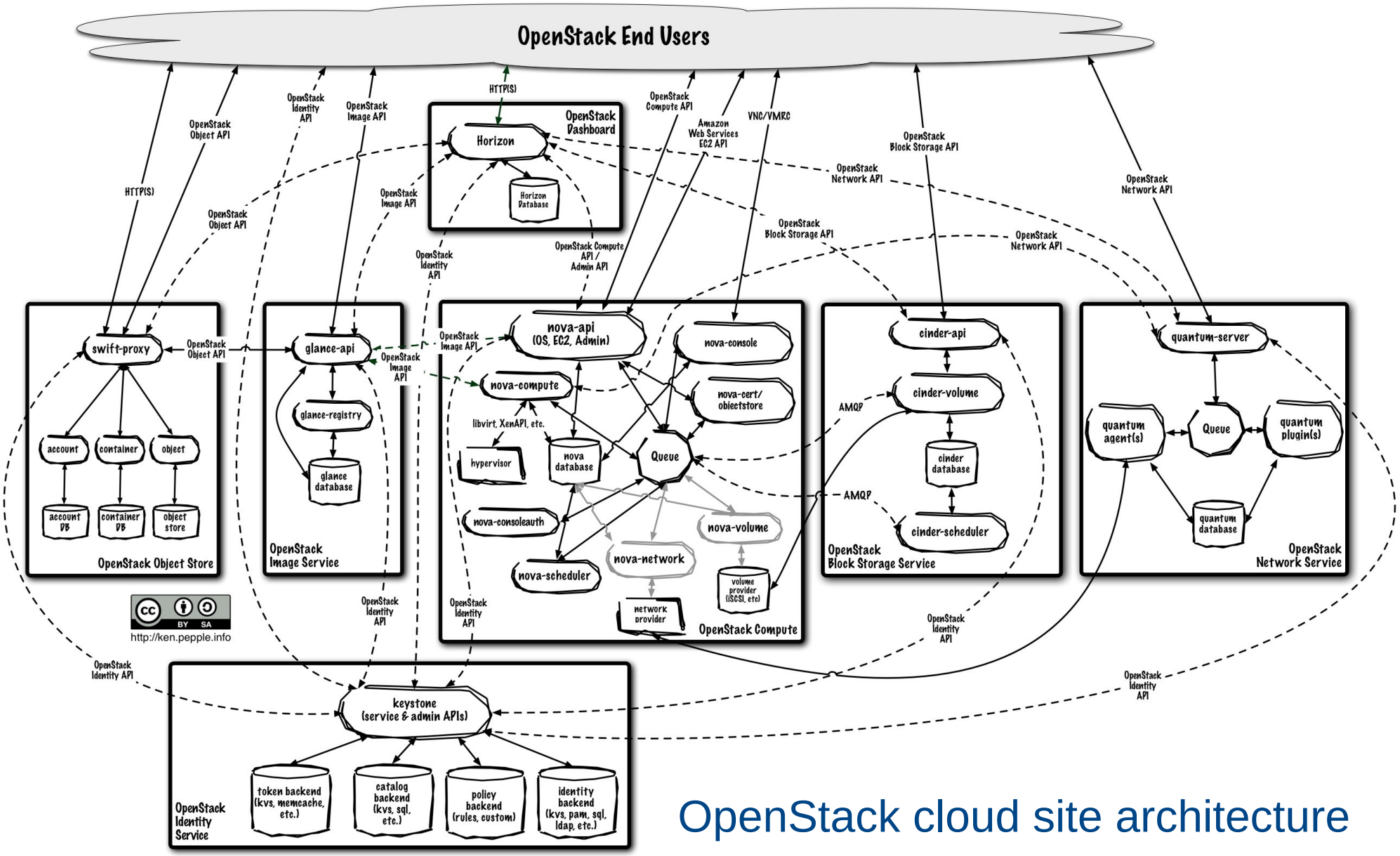
Vac “Back Off” procedure

- To avoid overloading Matcher/TaskQueue, Vac implements “back off”
- If a VM finishes with “no work” / “banned” / “site misconfigured” outcomes then it counts as an abort
  - If no outcome given, then if a VM finishes after less than fizzle_seconds (600sec?) then it counts as an abort
- For a VM type (~experiment), if an abort has happened on **any** factory in the last backoff_seconds (600 sec?), then no more VMs of that type will be started
- After that, if an abort happened in the last backoff_seconds + fizzle_seconds and any new VMs have run for less than fizzle_seconds, then no more VMs of that type will be started
  - ie try to run one or two test VMs to see if ok now
- If backoff_seconds + fizzle_seconds have passed without more aborts, then can start VMs again as fast as slots become available

The Masonry Problem...

The Masonry Problem





OpenStack cloud site architecture