Vac and Vcycle

Andrew McNab

University of Manchester
LHCb and GridPP

Overview

- Vac and Vcycle
- Pilot VM lifecycle
- IaaS Cloud vs IaaC Vacuum models
- Applying Vacuum ideas back to IaaS
- user_data templates and boot images
- Target shares
- Accounting
- Admin-friendly philosophy
- Multiprocessor VMs
- Status at sites

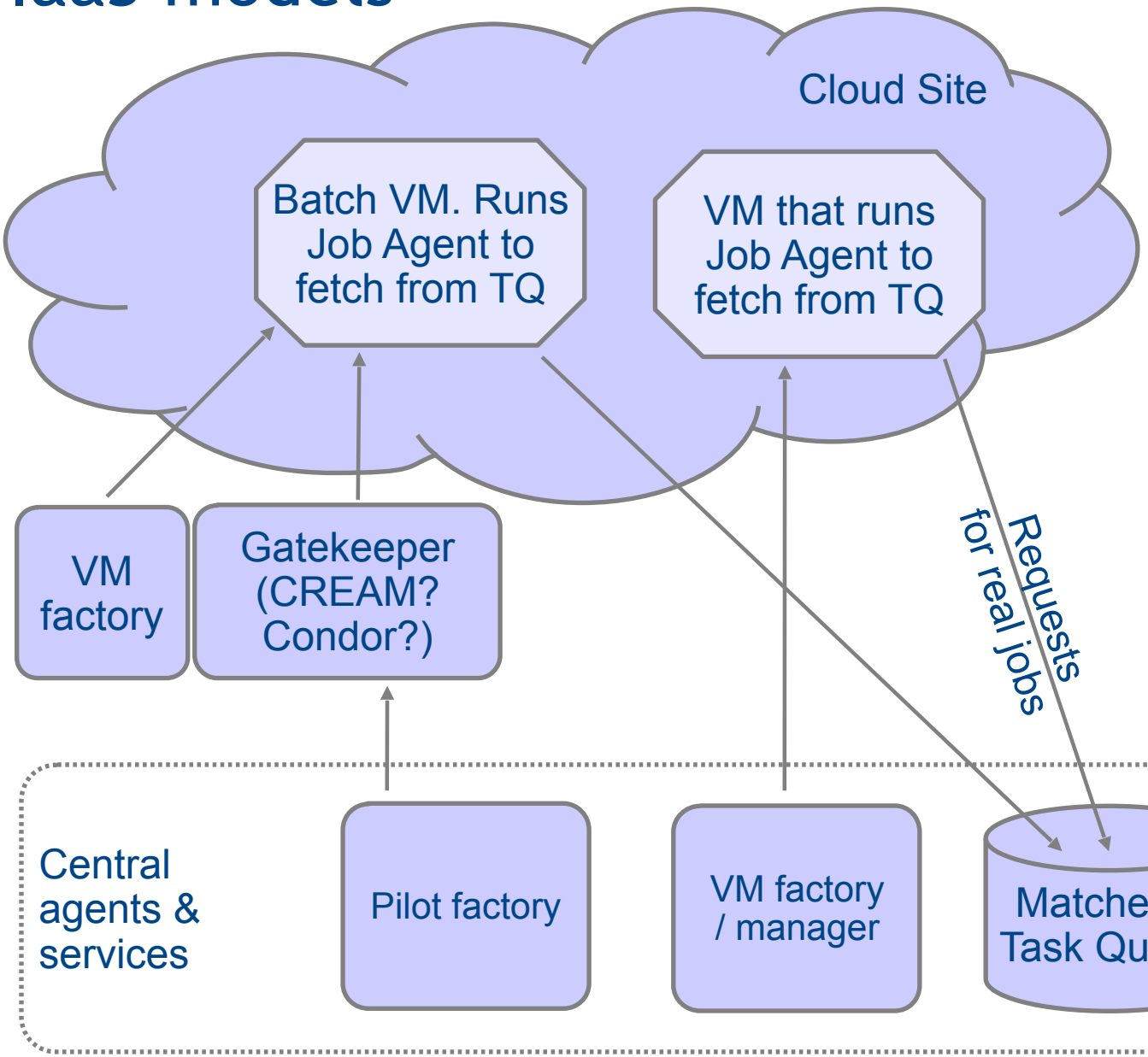
Vac and Vcycle

- 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, Birmingham, LHCb and ATLAS
- Both make very similar assumptions about how the VMs behave
  - The same LHCb, ATLAS, CMS VMs working in production on Vac and Vcycle
  - Compatible GridPP DIRAC VMs available too
- See this afternoon's talk about VMs in WLCG for "Pilot VM" details
- **Vac/Vcycle not tied to CernVM but uses it at all stages in practice**

“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 (somehow)
  - shutdown_message file can be used to say why the VM shut down
- Experiments’ VMs are a lot simpler for the site to handle than WNs
  - **Largely due to internal reliance on CernVM-FS**

IaaS models



Several ways of using IaaS clouds to get pilot clients running

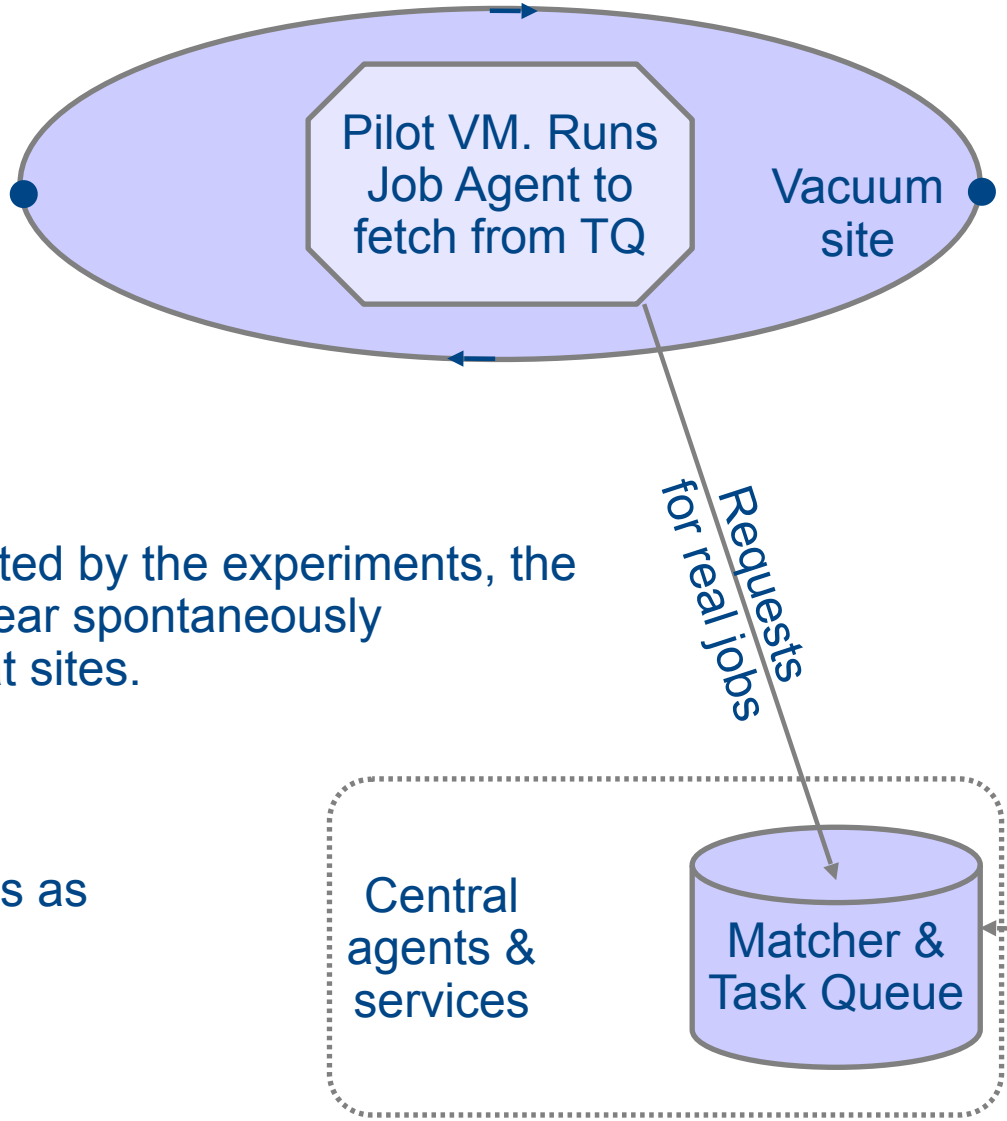
Fetch payload jobs from task queues as we do with Grid+Batch

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)

Vac

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
- Following 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 resource provider 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
- CernVM-FS and pilots mean a small user_data file is all the site needs
  - Experiments can provide a template to create the site-specific user_data

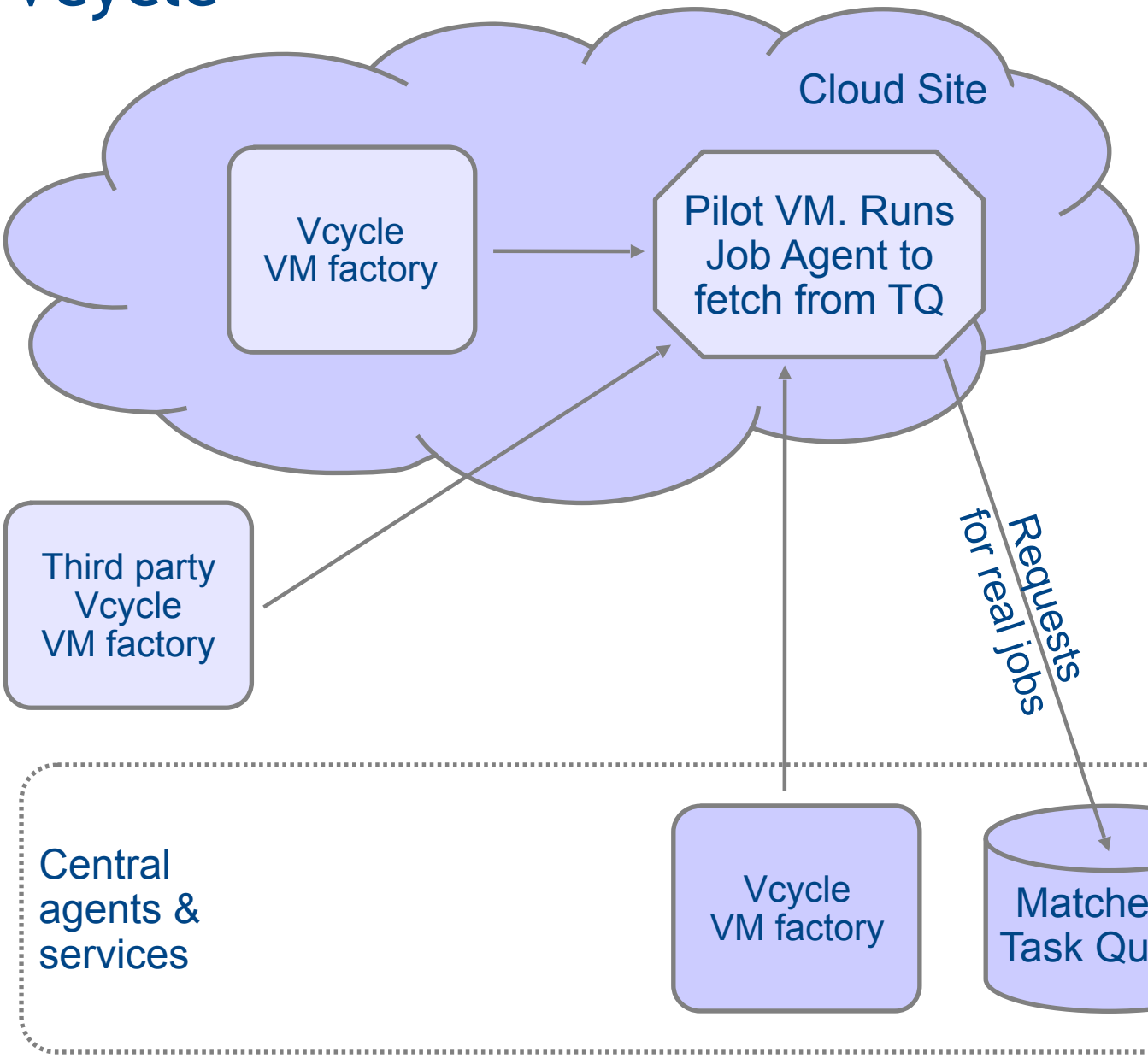
Vac implementation

- 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 same /etc/vac.d/*.conf files; managed by Puppet, Chef, Cfengine, ...
- Factories communicate load info with each other via UDP
- Natively supports CernVM 3 (~SL6) and CernVM 2
- 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
  - (Could also kill stalled/idle VMs based on CPU usage)

Vcycle implementation

- Apply 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 REST API directly
  - OCCI fork has been done by WLCG team at CERN (Luis Villazon Esteban)
  - OCCI plugin using OCCI REST API being written; EC2 to follow.

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

VMs started and monitored by Vcycle, but not managed in detail ("black boxes")

user_data templates and images

- Resource provider can create user_data file, or let Vac/Vcycle do it from a template.
- Templates can be given as URLs, on experiment's web server
  - Refetched each time since very small
- Experiment can also supply desired uCernVM 3 boot images
  - Since so small (20MB) again can be specified as URLs
  - Cached and rechecked each time with If-Modified-Since
  - Vcycle uploads new VM images to OpenStack automatically
- Experiments can update user_data and images everywhere without having to contact site admins
- Easy to add new experiment to a Vac or Vcycle installation

Example of configuration

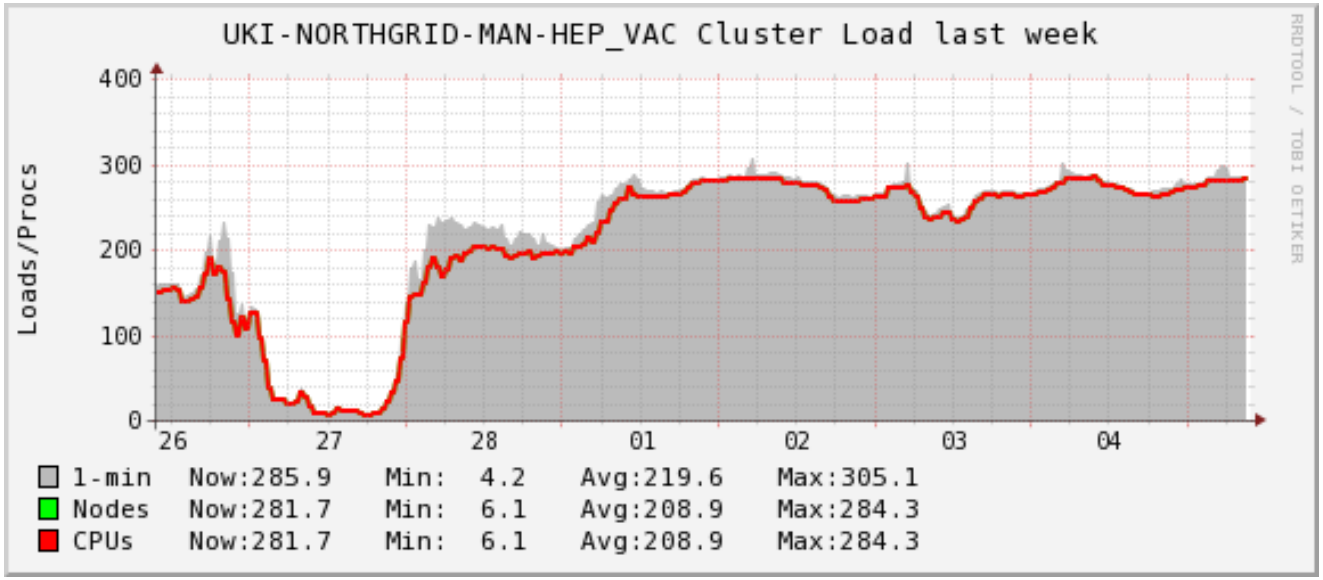
- Section of vac.conf used to enable LHCb VMs at Manchester
- They just need this and to create a hostcert/key.pem
- (vcycle.conf configuration is very similar)
- Compare what YAIM has to do to add a VO to a CE/Batch site

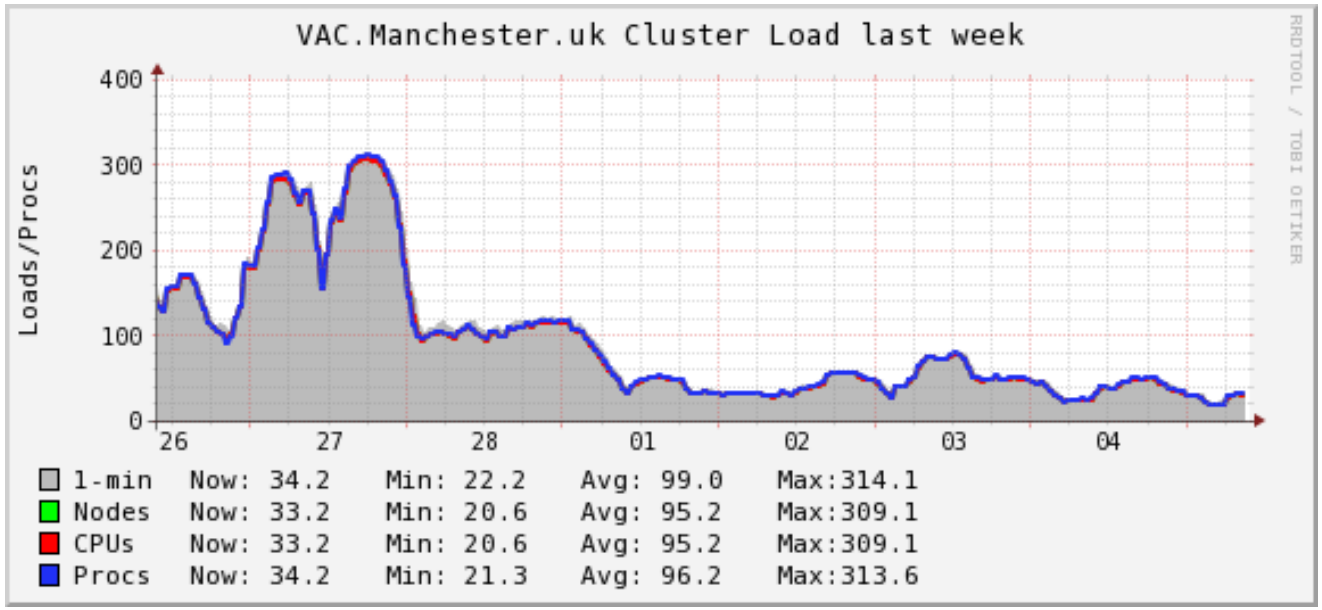
```
[vmtype lhcbprod]
vm_model = cernvm3
root_image = https://lhcbproject.web.cern.ch/lhcbproject/Operations/VM/cernvm3.iso
rootpublickey = /root/.ssh/id_rsa.pub
backoff_seconds = 600
fizzle_seconds = 600
max_wallclock_seconds = 172800
log_machineoutputs = True
accounting_fqan=/lhcb/Role=NULL/Capability=NULL
heartbeat_file = vm-heartbeat
heartbeat_seconds = 600
user_data = https://lhcbproject.web.cern.ch/lhcbproject/Operations/VM/user_data
user_data_option_dirac_site = VAC.Manchester.uk
user_data_option_cvmfs_proxy = http://squid-cache.tier2.hep.manchester.ac.uk:3128
user_data_file_hostcert = hostcert.pem
user_data_file_hostkey = hostkey.pem
```

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: ATLAS vs LHCb (vs CMS)





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 and shares 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 work as with Vac. Counts in units of HS06/VM.
- Target shares/limits are at the level of Projects/Tenancies
  - Vcycle instances can support multiple tenancies, perhaps at multiple remote sites
  - Different target share pools in the different tenancies
  - eg Vcycle instance at Manchester manages ATLAS, CMS, and LHCb VMs at IC and CERN separately.
- The target share model is instantaneous, as with Vac

Accounting with Vac and Vcycle

- 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 and Vcycle write out APEL SSM record into a directory when each VM finishes
  - See this afternoon's VMs and WLCG talk for details
- These can be sent to the central APEL service by the standard APEL smssend tool (as ARC does)
- For Vcycle, this provides an alternative to installing accounting reporting tools for OpenStack etc at the site
- For Vac, it makes the accounting at the site completely self-contained at the level of the individual physical machine
  - No longer need to depend on a site APEL service

Vac installation and configuration

- RPMs are provided in a yum repository
  - Also possible to install from source from GitHub
- Only major dependency is libvirt, so very straightforward to install each factory machine
- The distribution includes a Vac Puppet module
  - Ensures vacd is running
  - Installs/updates /etc/vac.d/*.conf files
    - Vac merges installed files to create configuration
    - Puppet module allows different sections of configuration to be installed on different sets of machines
  - Optionally sets up Vac Nagios monitor
  - Optionally sets up APEL accounting publishing

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
- Vcycle has HEPSPEC06 per VM parameter which is the basis of target shares, and implicitly takes multiprocessor weighting into account

Vac and Vcycle at sites

- Manchester
  - Vac running ATLAS, CMS, LHCb VMs
    - In last year: 300,000 jobs, 196 CPU years
  - Vcycle instance managing Imperial (LHCb/ATLAS) and CERN development VMs
- Lancaster
  - Long running Vac site
  - Currently moving to an increased number of machines using nodes retired from batch
- Oxford
  - Vac running ATLAS, CMS, LHCb VMs
- Imperial
  - GridPP OpenStack tenancy has ATLAS, CMS, LHCb VMs
- Birmingham
  - Developing procedure to run Vac on interactive cluster machines overnight
- CERN
  - LHCb tenancy managed by Vcycle on LHCb vobox at CERN
  - Development tenancy with ATLAS, LHCb, CMS managed by Vcycle at Manchester
- Vac and Vcycle total resources each comparable to a typical Tier-2

Summary

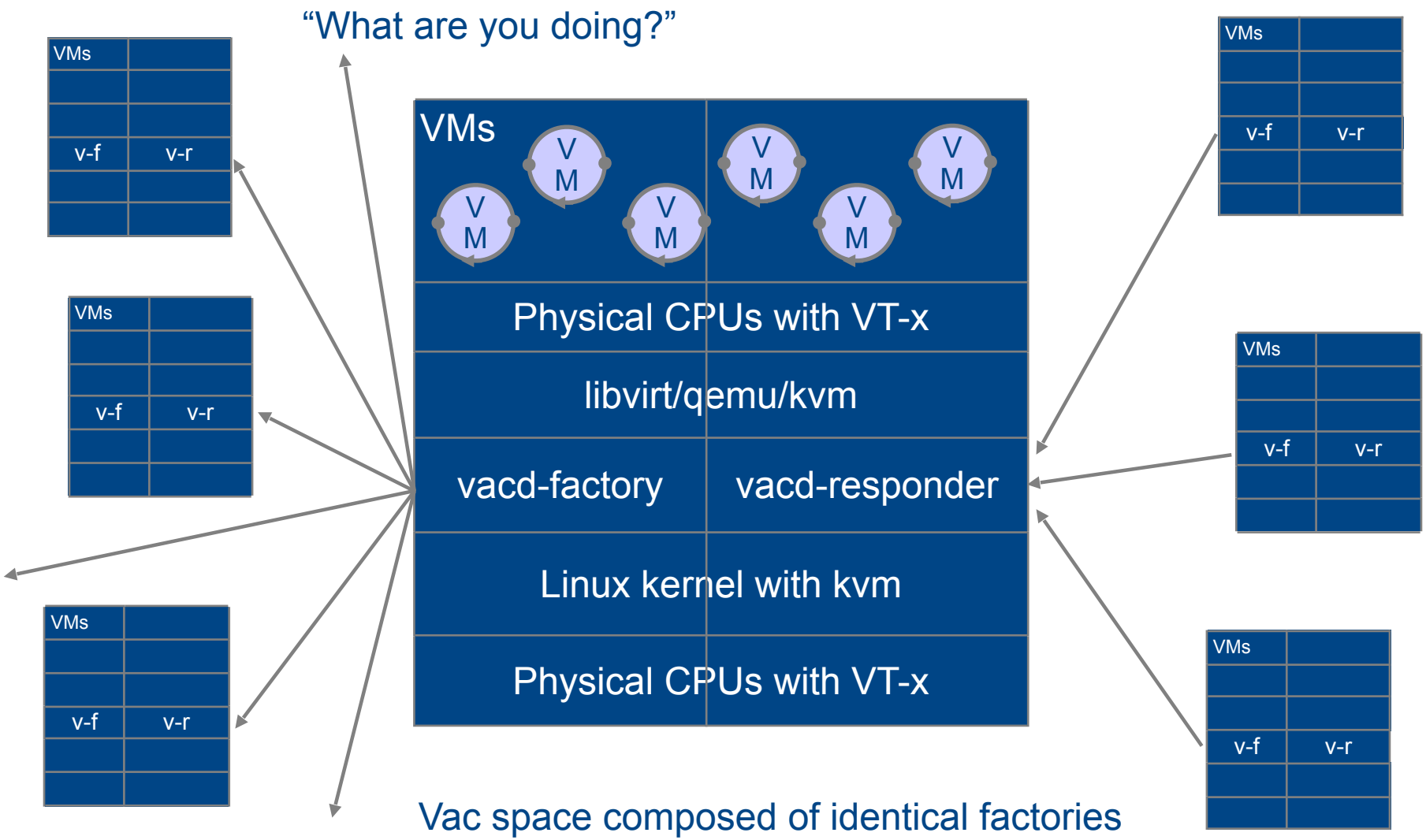
- Vac provides a simple way for sites to run VMs
- Vcycle provides a simple way to manage VMs on OpenStack
- “Simplicity is a feature”

- Both demonstrated with ATLAS, CMS, LHCb production jobs
  - Hundreds of thousands of jobs; hundreds of CPU years
- Aim has been to demonstrate production quality rather than roll out to a large number of sites so far
- For Vac http://www.gridpp.ac.uk/vac/ for RPMs, Yum repo, links to GitHub, docs, man pages etc
- For Vcycle http://www.gridpp.ac.uk/vcycle/ has links to source, man pages etc

- This afternoon’s talk on VMs in WLCG explains the VM internals

Extra slides

Vac factory node and site architecture

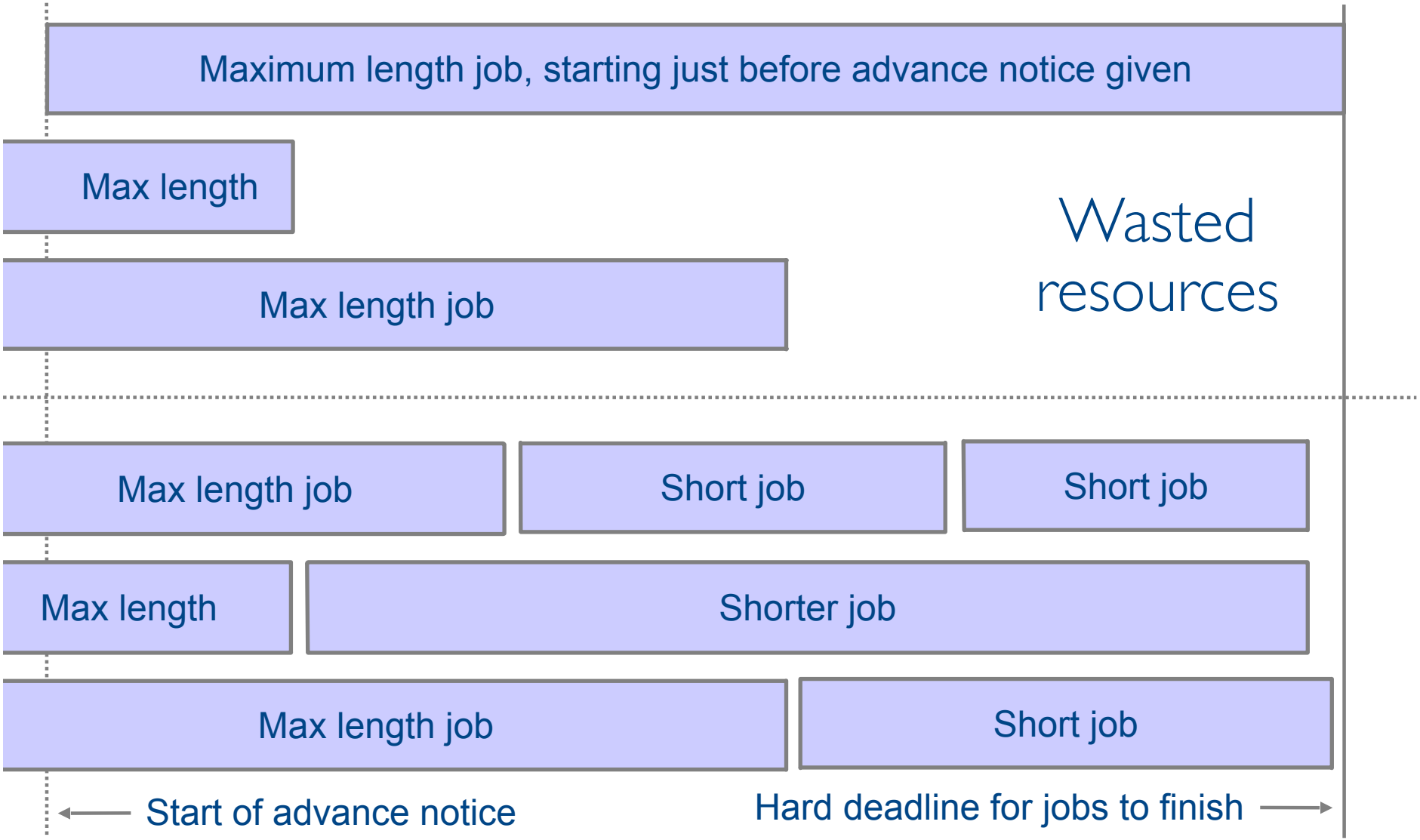


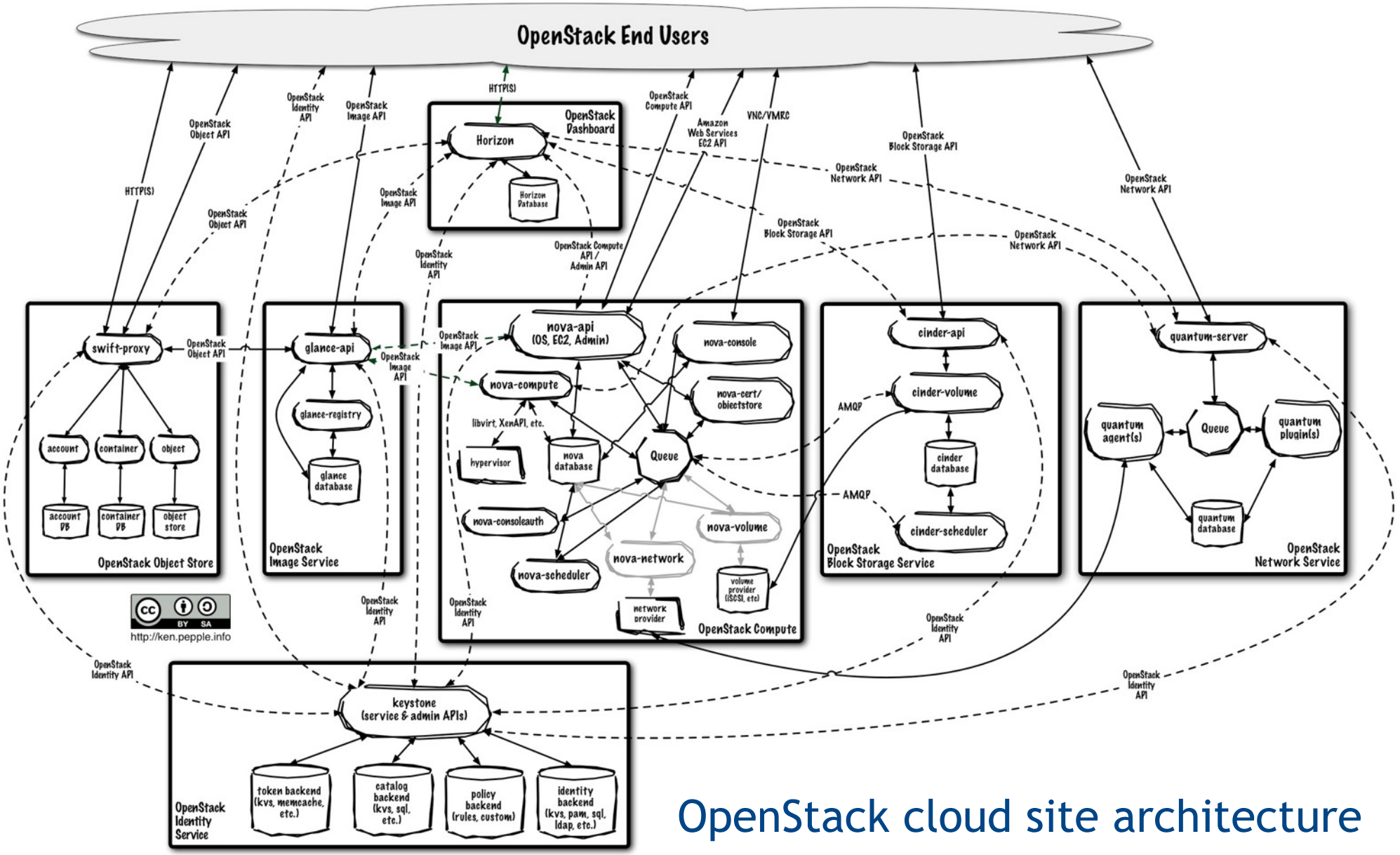
Vac “Back Off” procedure

- 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

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