HTCondor & ARC CEs

Andrew Lahiff, Alastair Dewhurst,
John Kelly, Ian Collier

GridPP 32

25 March 2014, Pitlochry, Scotland

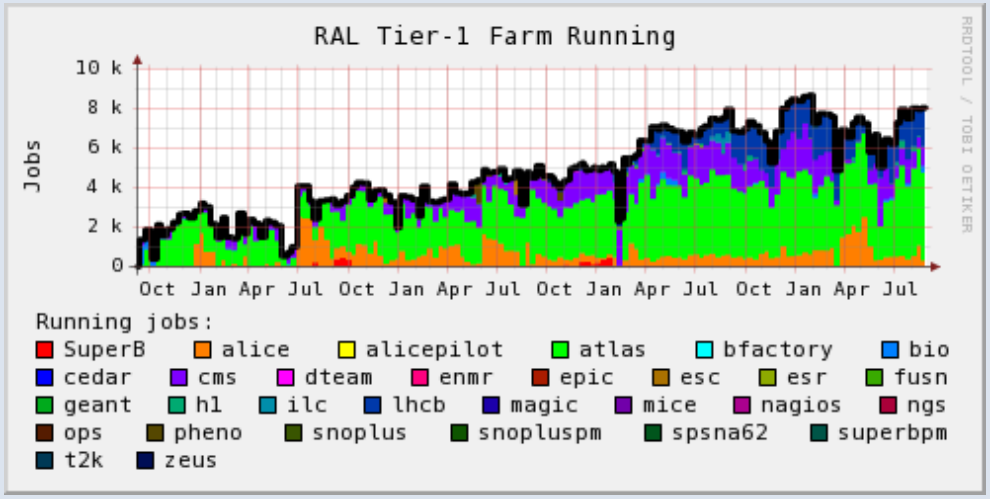
Outline

1. RAL batch system
   - Background
   - Why we migrated to HTCondor
   - Experience so far
   - Features
2. ARC CEs

HTCondor

RAL batch system

- Batch system at the RAL Tier-1
  - 656 worker nodes, 9312 slots, 93000 HEPSPEC06
  - Growing soon to beyond 12000 slots
- Torque + Maui had been used for many years
  - Had many issues
  - Severity & number of problems increased as size of farm
  - Increasing effort just to keep it working
- In August 2012 started looking for an alternative



Jobs running for the past 4 years

(as of Aug 2013)

Choosing a new batch system

- Considered, tested & eventually rejected the following technologies:
  - LSF, Univa Grid Engine
    - Avoid commercial products unless absolutely necessary
  - Open source Grid Engines
    - Competing products, not sure which has best long-term future
    - Communities appear less active than SLURM & HTCondor
    - Existing Tier-1s using Univa Grid Engine rather than open source
  - Torque 4 + Maui
    - Maui problematic
    - Torque 4 seems less scalable than alternatives
      - Easily able to cause problems with high job submission / query rates
  - SLURM
    - Carried out extensive testing and comparison with HTCondor
    - Found that for our use case:
      - Very fragile, easy to break
      - Unable to get to work reliably above 6000 jobs slots
- For more information, see

  http://indico.cern.ch/event/247864/session/5/contribution/21

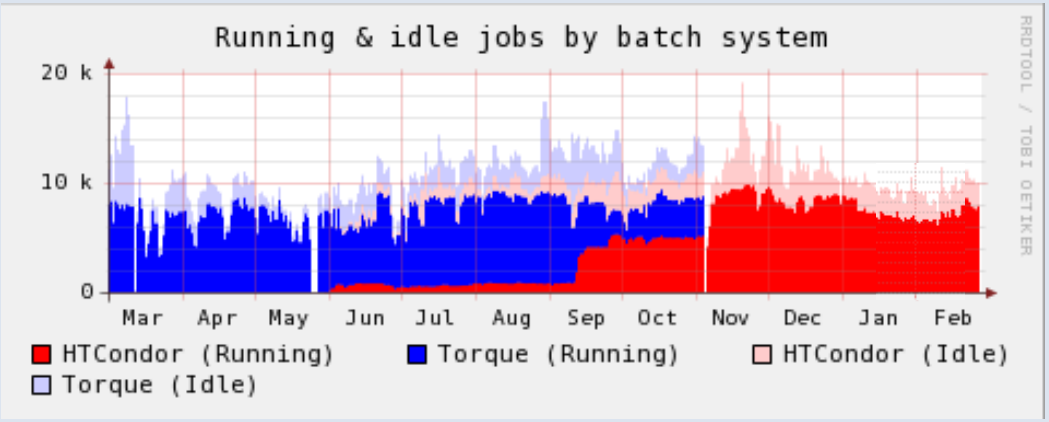
Choosing a new batch system

- HTCondor chosen as replacement for Torque + Maui
  - Has the features we require
  - Seems very stable
  - Easily able to run 16,000 simultaneous jobs
    - Prior to deployment into production we didn't try larger numbers of jobs
      - Didn't expect to exceed this number of slots within the next few years
      - See later slide
    - Didn't do any tuning – it "just worked"
  - Is more customizable than all other batch systems

Migration to HTCondor

- Timeline

2012 Aug	- Started evaluating alternatives to Torque/Maui
2013 June	- Began testing HTCondor with ATLAS & CMS
2013 Aug	- Choice of HTCondor approved by RAL Tier-1 management
2013 Sept	- Declared HTCondor as a production service
	- Moved 50% of pledged CPU resources to HTCondor
	(upgraded WNs to SL6 as well as migrating to HTCondor)
2013 Nov	- Migrated remaining resources to HTCondor



Experience so far

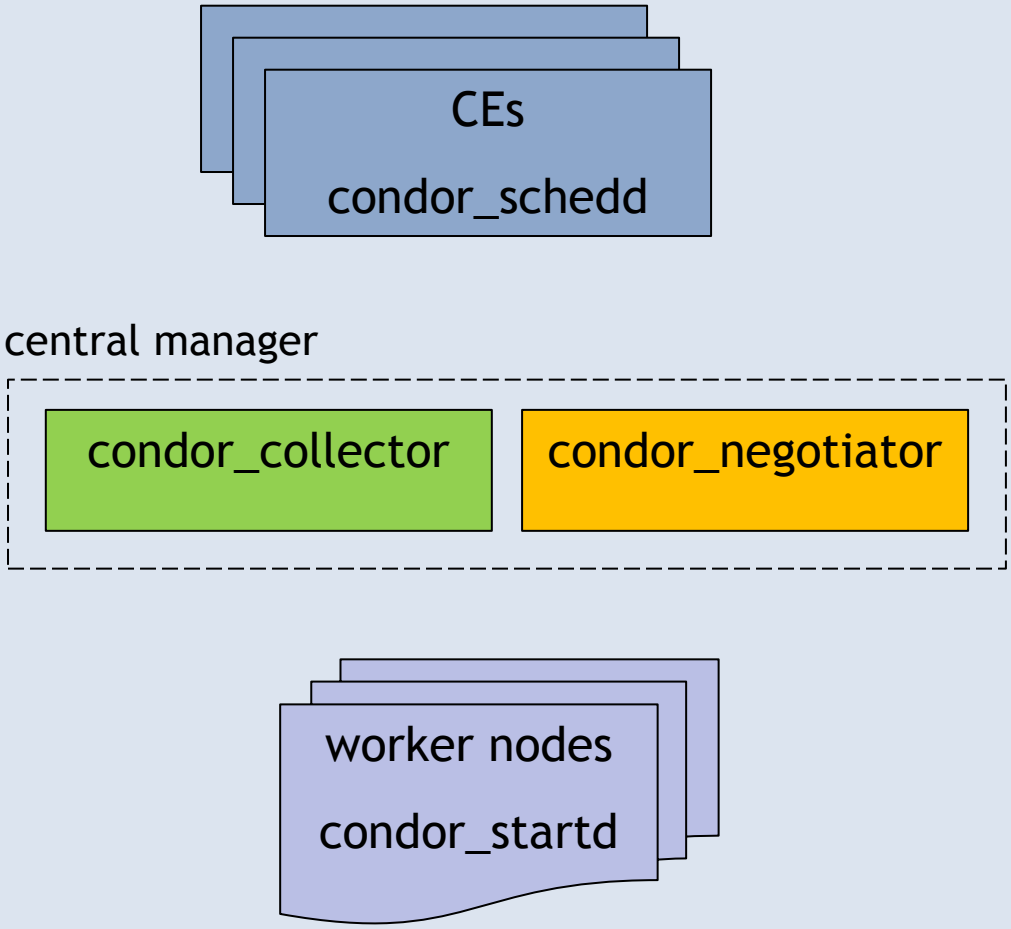
- Experience
  - Very stable operation, no crashes or memory leaks
  - Job start rate much higher than Torque + Maui, even when throttled
  - Staff able to spend time investigating improvements/new features, not just fire-fighting
- Problems
  - During early testing found job submission hung when 1 of a HA pair of central managers was switched off
    - Quickly fixed & released in 8.0.2
  - Experienced jobs dying 2 hours after network break between CEs and WNs
    - Quickly fixed & released in 8.1.4
  - Found problem affecting HTCondor-G job submission to ARC CE with HTCondor batch system
    - Quickly fixed & released in 8.0.5
  - ***i.e. support by HTCondor developers has been very good***
  - Self-inflicted problems
    - Accidently had Quattor configured to restart HTCondor on worker nodes when configuration changed, rather than running “condor_reconfig” – jobs were killed

HTCondor

- Somewhat different to Torque, SLURM, Grid Engine, LSF
  - Jobs state their needs (Requirements) & preferences (Rank)
  - Machines state their needs (Requirements) & preferences (Rank)
  - Represented as ClassAds
    - AttributeName = Value
    - AttributeName = Expression
  - The negotiator (match-maker) matches job ClassAds with machine ClassAds, taking into account
    - Requirements of the machine & the job
    - Rank of the machine & the job
    - Priorities
- Daemons
  - Startd: represents resource providers (machines)
  - Schedd: represents resource consumers (jobs)
  - Collector: stores job & machine ClassAds
  - Negotiator: matches jobs to machines
  - Master: watches over other daemons & restarts them if necessary

HTCondor

- Architecture



Initial install

- HTCondor is *not* difficult, but it's very configurable
  - "HTCondor... there's a knob for that"
  - Gives people the impression that it's complicated?
- Most basic install & configuration is trivial
  - Installs collector, negotiator, schedd and startd on the same machine

```
# yum install condor
# service condor start
```

- **Immediately** can run jobs

```
-bash-4.1$ condor_submit condor.sub
Submitting job(s).
1 job(s) submitted to cluster 1.
-bash-4.1$ condor_q

-- Submitter: lcg0732.gridpp.rl.ac.uk : <130.246.216.4:34655> : lcg0732.gridpp.rl.ac.uk
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
   1.0   alahiff         3/4  10:25   0+00:00:02 R  0   0.0  sleep 60

1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended
```

Pre-requisites

- A shared filesystem is not required
  - HTCondor can transfer files between WNs and CEs
    - Doesn't use scp, therefore no ssh configuration is required
    - Can use encryption
    - Can configure limits on concurrent uploads, downloads, file sizes, etc
- Resources
  - For "small" sites (including RAL Tier-1) central manager uses very little CPU & memory
  - CEs need lots of memory
    - condor_shadow process for each running job, requiring ~ 1 MB per slot
    - Use of 32-bit HTCondor rpms on CEs (only) reduces memory requirements
  - At RAL:
    - 5 CEs, with a schedd on each
    - Each CE has 20 GB RAM

Features

- Queues
  - HTCondor has no concept of queues in the Torque sense
    - We see no reason to have such queues
  - Jobs just request their requirements
    - Cpus, memory, disk, OS, …
- Fairshares
  - Hierarchical accounting groups
- Highly available central manager
  - Shared filesystem not required, HTCondor replicates the required data
- Python API
  - Useful for monitoring
- condor_gangliad
  - Generates lots of ganglia metrics for all HTCondor daemons

Job isolation

- Features we're using
  - Per-job PID namespaces
    - Jobs cannot interfere with or even see other processes running on the WN, even those by the same user
- Features we're testing
  - CPU affinity *(currently being tested on production WNs)*
    - Ensure jobs can only use the number of cores they requested
  - Cgroups
    - Makes it impossible for quickly-forking processes to escape control, more accurate memory accounting
    - CPU: ensure jobs use their share of cores, or more if others are idle
    - Memory: protect both the WN & jobs from jobs using too much memory
  - "Mount under scratch" (bind mounts)
    - Each job has it's own /tmp, /var/tmp
    - Haven't got working yet with glexec jobs (but should be possible)
- Features we haven't tried (yet)
  - Named chroots

WN health-check

- Implemented using a startd cron
  - Can put arbitrary information into ClassAds, including output from scripts
  - We put information about problems into the WN ClassAds
- Used to prevent new jobs from starting in the event of problems
    - Checks CVMFS, disk, swap, …
    - If problem with ATLAS CVMFS, only stops new ATLAS jobs from starting
  - Example query:

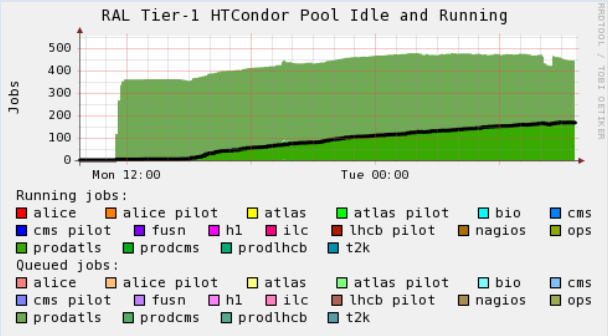
```
# condor_status -constraint 'NODE_STATUS =!= "All_OK" &&
  PartitionableSlot == True' -autoformat Machine NODE_STATUS
lcg0975.gridpp.rl.ac.uk Problem: CVMFS for atlas.cern.ch
lcg1120.gridpp.rl.ac.uk Problem: CVMFS for lhcb.cern.ch
lcg1125.gridpp.rl.ac.uk Problem: CVMFS for atlas.cern.ch
lcg1127.gridpp.rl.ac.uk Problem: CVMFS for atlas.cern.ch
lcg1406.gridpp.rl.ac.uk Problem: CVMFS for alice.cern.ch
lcg1423.gridpp.rl.ac.uk Problem: CVMFS for atlas.cern.ch
lcg1426.gridpp.rl.ac.uk Problem: CVMFS for atlas.cern.ch
lcg1520.gridpp.rl.ac.uk Problem: CVMFS for atlas.cern.ch
```

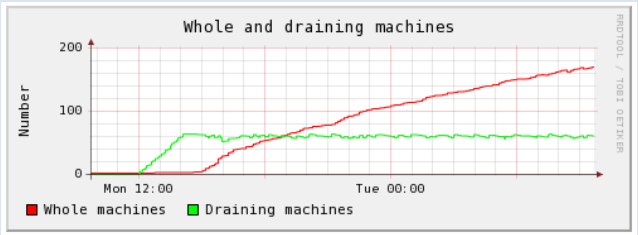
Scalability

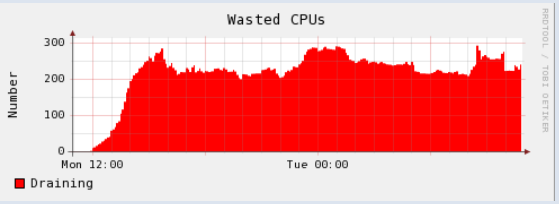
- Modular architecture helps with scalability
  - Can add additional schedds
  - Can setup multi-tier collectors
  - Can even use multiple negotiators
- Simultaneous numbers of running jobs
  - Prior to deployment into production
    - 110 8-core worker nodes, high number of slots each
    - Reached 16,000 running jobs without tuning
      - Didn't attempt any more than this at the time
  - More recent testing
    - 64 32-core worker nodes, high number of slots each
    - Exceeded 30,000 running jobs successfully
      - Didn't attempt any more than this

Multi-core jobs

- WN configuration
  - Partitionable slots: WN resources (CPU, memory, disk, …) divided up as necessary for jobs
- Partitioning of resources
  - We're using dynamic allocation of multi-core jobs
  - Could partition resources if we wanted
- condor_defrag daemon
  - Finds WNs to drain, drains them, then cancels draining when necessary
  - Works immediately out-of-the-box, but we're tuning it to:
    - Minimize wasted CPUs
    - Ensure start-up rate of multi-core jobs is adequate
    - Maintain required number of running multi-core jobs







Dynamic WNs

- HTCondor was designed to make use of opportunistic resources
  - No hard-wired list of WNs
  - No restarting of any services
  - No pre-configuration of potential WNs
    - WNs advertise themselves to the collector
    - With appropriate security permissions, can join the pool and run jobs
- Dynamic provisioning of virtual WNs
  - Can use existing power management functionality in HTCondor
  - condor_rooster
    - Designed to wake-up powered down physical WNs as needed
    - Can configure to run command to instantiate a VM
  - Easy to configure HTCondor on virtual WNs to drain then shutdown the VM after a certain time
  - Tested successfully at RAL (not yet in production)
    - CHEP 2013 http://indico.cern.ch/event/214784/session/9/contribution/205
    - HEPiX Fall 2013 http://indico.cern.ch/event/247864/session/4/contribution/53

Other features

- Some interesting features
  - Job hooks
    - Example: WNs pulling work, rather than work being pushed to WNs
  - Flocking
    - Redirect idle jobs to a different HTCondor pool
  - Job router
    - Transform jobs from one type to another
    - Redirect idle jobs to
      - Other batch systems (HTCondor, PBS, LSF, Grid Engine)
      - Other grid sites (ARC, CREAM, …)
      - Clouds (EC2)
  - Jobs can be virtual machines
    - HTCondor can manage virtual machines (VMware, Xen, KVM)
  - HTCondor-CE
    - CE which is a standard HTCondor installation with different config
    - OSG starting to phase this in
- Upcoming
  - Live job i/o statistics

ARC CEs

HTCondor & grid middleware

- EMI-3 CREAM CE
  - HTCondor not officially supported
    - BLAH supports HTCondor
      - Job submission works!
    - HTCondor support in YAIM doesn't exist in EMI-3
      - We modified the appropriate YAIM function so that the blah configuration file is generated correctly
    - Script for publishing dynamic information doesn't exist in EMI-3
      - Wrote our own based on the scripts in old CREAM CEs
      - Updated to support partitionable slots
    - APEL parser for HTCondor doesn't exist in EMI-3
      - Wrote a script which writes PBS style accounting records from condor history files, which are then read by PBS APEL parser
  - Relatively straightforward to get an EMI-3 CREAM CE working
    - We will make our scripts available to the community
      - Should eventually be in EMI
    - Milan Tier-2 also helpful

ARC CEs

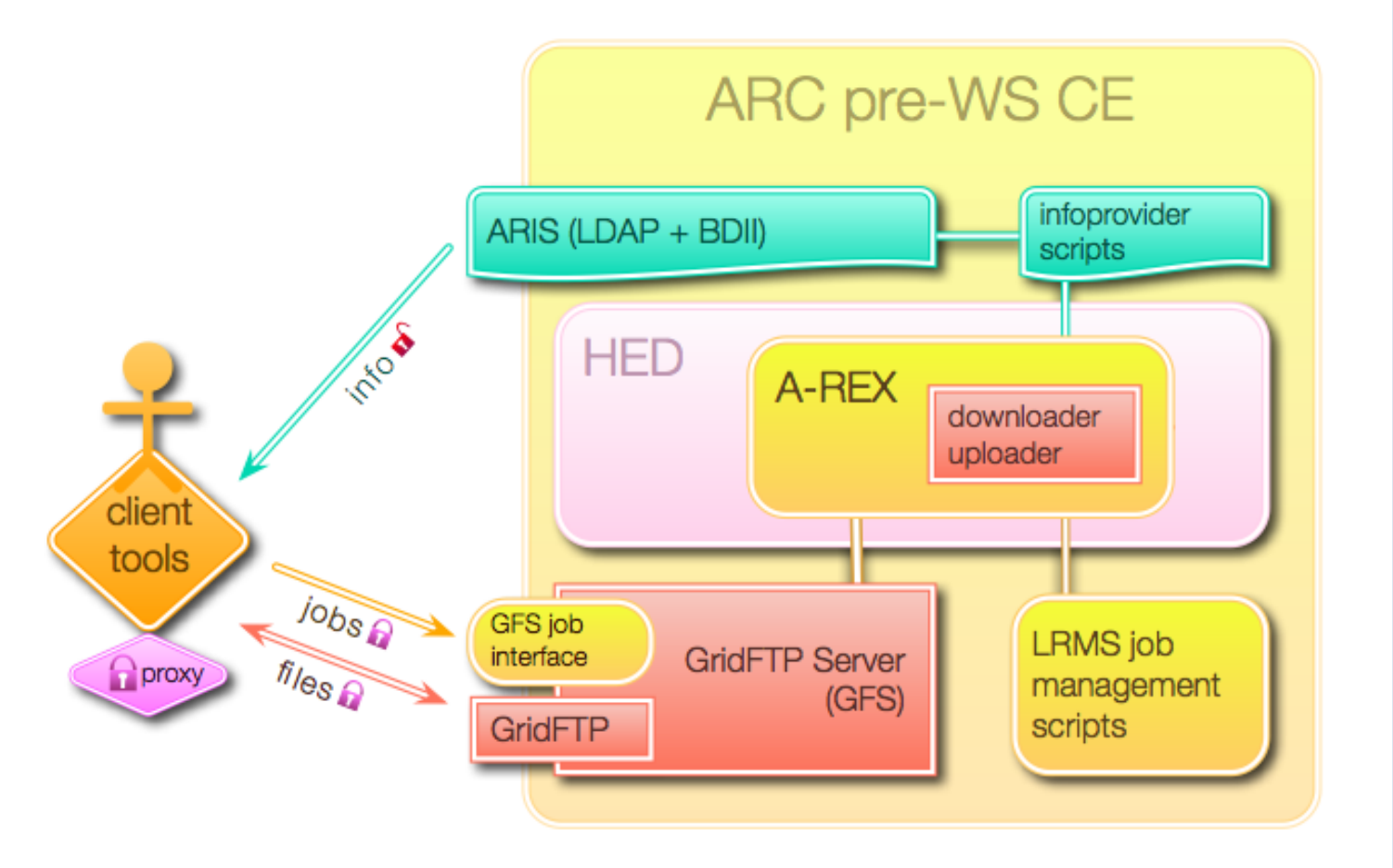
- ARC CEs
  - Successfully being used by some ATLAS & CMS Tier-2s outside of Nordugrid (with SLURM, Grid Engine, …)
    - Including LRZ-LMU (ATLAS), Estonia Tier 2 (CMS)
    - Glasgow and Imperial first UK sites to try ARC CEs
  - The LHC VOs and ARC CEs
    - ATLAS and CMS fine
      - At RAL ATLAS & CMS only have access to ARC CEs
    - LHCb added ability to DIRAC to submit to ARC CEs
      - Not yet at the point of LHCb only using ARC CEs
    - ALICE can't use them yet, but will work on this

Benefits

- Benefits of ARC CEs
  - Support HTCondor better than CREAM CEs do
    - Especially in the upcoming new release
  - Simpler than CREAM CEs
    - Just a single config file: /etc/arc.conf
    - No YAIM
    - No Tomcat
    - No MySQL
      - Information about jobs currently stored in files
  - ARC CE accounting publisher (JURA) can send accounting records directly to APEL using SSM
    - APEL publisher node not required

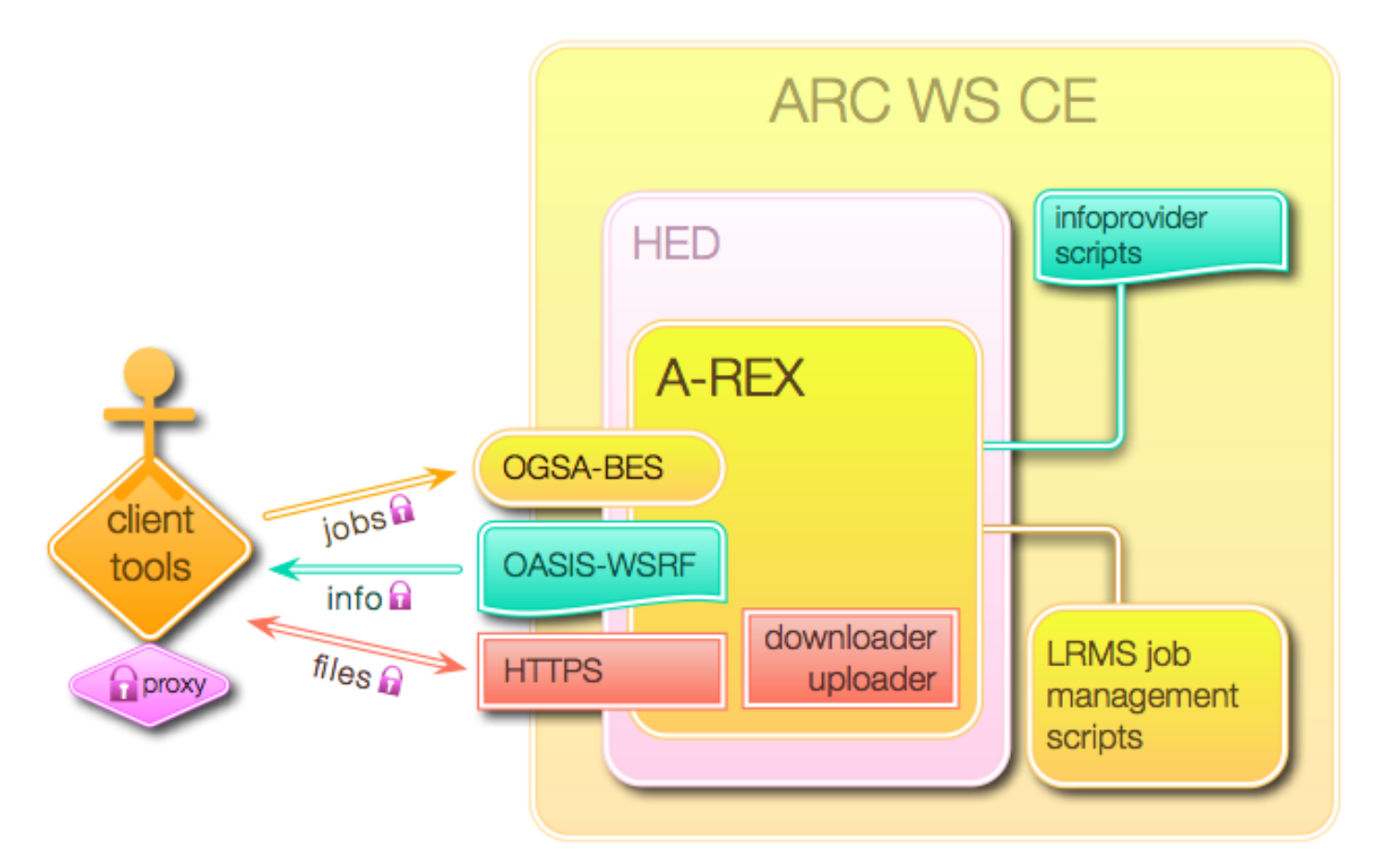
ARC CE architecture

- GridFTP + LDAP (pre-Web service)
  - Used by HTCondor-G (e.g. ATLAS & CMS pilot factories, glite-WMS)



ARC CE architecture

- Web service
  - Works with RFC proxies only; not used by HTCondor-G



HTCondor integration

- EMI-3
  - 3.0.3 ARC rpms
  - A number of modifications required to batch system backend scripts
    - Main issue: scripts were written before HTCondor had partitionable slots
- UMD-3
  - 4.0.0 ARC rpms
  - Better
- Nordugrid svn trunk
  - http://download.nordugrid.org/nightlies/packages/nordugrid-arc/trunk/
  - Contain 10 patches to HTCondor backend scripts submitted by RAL
    - We have write access to the Nordugrid subversion repository
  - There will be a new official release soon
    - As of 19th March, release candidate almost ready
- Future
  - New HTCondor job submission script written from scratch (RAL)
  - …

Experience so far

- No major problems
  - Have generally just ignored them
- Issues
  - October: script checking status of jobs started taking so long it could no longer keep up
    - Optimizations made (now in the nightly builds)
    - Problem hasn't happened again since
  - GGUS ticket open about DefaultSE in VO views
    - Should be easy to fix

Summary

- Due to scalability problems with Torque + Maui, migrated to HTCondor
- We are happy with the choice we made based on our requirements
  - Confident that the functionality & scalability of HTCondor will meet our needs for the foreseeable future
- We have both ARC & CREAM CEs working with HTCondor
  - Relatively easy to get working
  - Aim to phase-out CREAM CEs

Questions? Comments?