Replacing LSF with HTCondor: the INFN-T1 experience

BY INFN-CNAF

HTC WS, 2020, Sep 21

Email: dalpra@infn.it

INFN-T1, Former Status (CREAM-CE/LSF, 2018)

- $\sim$ 400 KHS06, 36500 slots, 850 physical hosts
- $(5+2) \times \mathrm{CREAM-CE/LSF}\ 9.1.3$
- ~ 40 User groups: 24 Grid VOs, ~ 25 local

INFN-T1, Current Status (HTC-CE/HTC, 2020)

- $(6+1) \times$ HTC-CE, $1 \times$ CM, $850 \times$ WN, (36500 Cpus, ~ 400 KHS06)
- One more WN, with $2 \times$ K-40 GPUs (Grid access via HTC-CE)
- $1 \times$ SN for Remote Submission (from local UI, auth via `FS_REMOTE`)

This talk is about our experience on the migration process from LSF to HTCondor, current status and work in progress.

Planning a migration: Requirements

Our initial requirements ("small reversible steps" approach)

1. The switch should be (at most) transparent for our users:
   - LHC VOs not an issue: ready to access local resources through a CE.
   - Local users: convert from using `bsub` to `condor_submit`
2. The impact on our existing site management tools should be reduced at most
3. The switch should be reversible (rollback to LSF possible as extreme ratio)
4. Need cohexistence of two distinct prod clusters for a while (allow their time to user groups)
5. Cluster management should remain "similar", to some extent

Starting a migration: actions, decisions and steps

Actual work started on March 2018 by setting up a

HTCondor Testbed

- $1\times$ SN $+1\times$ CM, $3\times$ WN, 16 slot each (Mar. 2018)
- Manual set up, individual host/daemons configuration
- $1\times$ HTC-CE, (May 2018)

Actions

- Practice with it
- Get some test pilots submitted by LHC experiments (Sep. 2018)
- writing scripts and management tools, adapt to work with existing facilities
- Plan: start a small Production Cluster, then gradually move WNs there

Configuring a HTC Cluster

LSF had two especially comfortable features:

- a small set of text files on a shared filesystem defines properties and behaviour of the whole cluster
- hostgroups: hierarchical sets of named hosts. Defined with simple syntax, can be flexibly combined with simple set operations (union and difference)

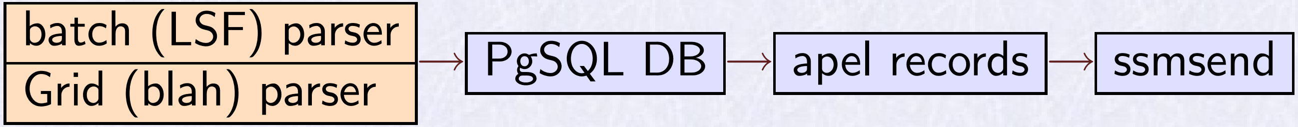
Example We adapted to HTC these two features this way:

- A node is member of the hostgroups: `wn`, `rack2`, `alice`
- The node has the script `htconf.py|` in its main configuration file, which output the content of the files: wn.conf, `rack2.conf`, `alice.conf`
- Lastly, the `<hostname>.conf` file is read, if present (it can always override).

Accounting

We used a custom accounting system for CREAM-CE/LSF (no APEL). We did some work to adapt it

Accounting with LSF

- 

- User DN and FQAN are the main Grid-side info collected from Blah.
- We collect a few more data for internal use: job exit status, WN name and HS06 power, job exit status,...
- We need to collect the same data from our Submit Nodes (SCHEDD), then we can re-use the other components.

Accounting with HTC

Initially: python bindings to query HTC for job history. It works, but a few timeouts were experienced. Defining `PER_JOB_HISTORY_DIR` turned out to be a safer choice.

- `PER_JOB_HISTORY_DIR=/var/lib/gratia/data/`
- One accounting text file per job, `history.<jobid>` with <key> = <value> pairs, one per line. Each file is *complete* (have both grid and batch data: no need for blah records, no need to lookup for matches between sets of grid and batch records).
- python: `jobfile2dict(fn)` read a history log file into a python dict. We take the wanted ones and INSERT INTO our accounting DB. We collect the same set of keys that the apel HTCondor parser collects, and a few more (hs06 node power and others, for internal use). Each SCHEDD push data to the DB every 3 minutes.
- After parse & insert, the file is archived to a backup directory (to prevent double counting, enable further and deeper inspection, in case of doubts).

Apel records obtained as a SQL VIEW:

```
acct=> SELECT * FROM apelhtjob WHERE "Processors"=8 LIMIT 1;
+-------------------------------------------------------------
Site                           | INFN-T1
SubmitHost                     | ce02-htc.cr.cnaf.infn.it#7737.0#1555345220
MachineName                    | htc-2.cr.cnaf.infn.it
Queue                          | cms
LocalJobId                     | 7737
LocalUserId                    | pilcms006
GlobalUserName                 | /[...]CN=cmspilot04/vocms080.cern.ch
FQAN                           | /cms/Role=pilot/Capability=NULL
VO                             | cms
VOGroup                        | /cms
VORole                         | Role=pilot
WallDuration                   | 41848
CpuDuration                    | 40549
Processors                     | 8
NodeCount                      | 1
StartTime                      | 1555345239
EndTime                        | 1555350470
InfrastructureDescription | APEL-HTC-CE
InfrastructureType             | grid
ServiceLevelType               | HEPSPEC
ServiceLevel                   | 10.832
```

Managing HTCondor

- LSF: configure everything on a small set of files
- Puppet + Foreman: provisioning and main setup. Good for semi-static *known to work* configurations. Not easy to achieve a desired level of flexibility (example: temporarily excluding a VO from working on an arbitrary set of WNs)
- `htconf.py`: simple tool to complement or override puppet settings, adding granularity. Currently it makes use of a shared filesystem across machines in the pool.

`/shared/fs/htconf.py|` declared in a main HTCondor config file. It injects a set of knobs to the machine running it, depending on the *role*, *group* and *name* of the machine.

- More similar to the way LSF is configured (a small set of config files)
- several different configurations can be tested and compared quickly
- Example: temporarily adding a few classAd to an arbitrary set of WNs is a matter of defining the hostgroup `<groupname>` and the classads into `<groupname>.conf`

Command line tools

- condor_status, condor_q extremely powerful to inspect job and pool status, yet easy to get cumbersome. Most frequent LSF commands have been emulated using python bindings:

LSF	HTC
bjobs	hjobs
bqueues	hqueues
bhosts	hhosts

```
[root@htc-2 ~]# hjobs.py
JobId   Owner fromhost JobStart Cpus Machine TotalCpus CPUsUsage
25571.0 pagnes sn-01 2019-10-31:03:32:29 1 wn-201-07-15-01-a 16.0 0.99
25764.0 pagnes sn-01 2019-10-31:03:42:55 1 wn-201-07-37-04-a 16.0 0.97
```

Once confident with `condor_* -af . . .` , these are being less frequently used.

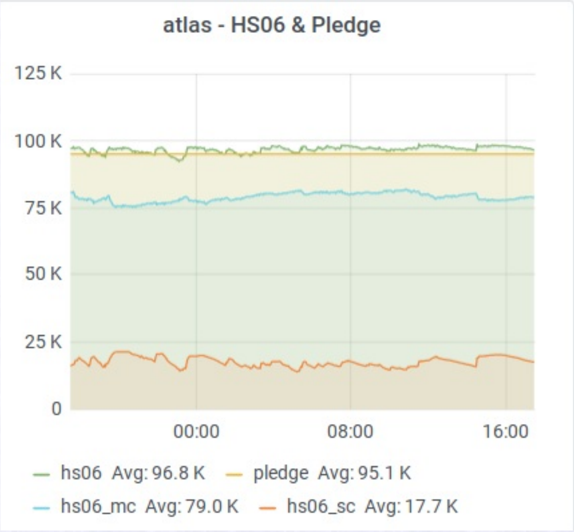
Other tools

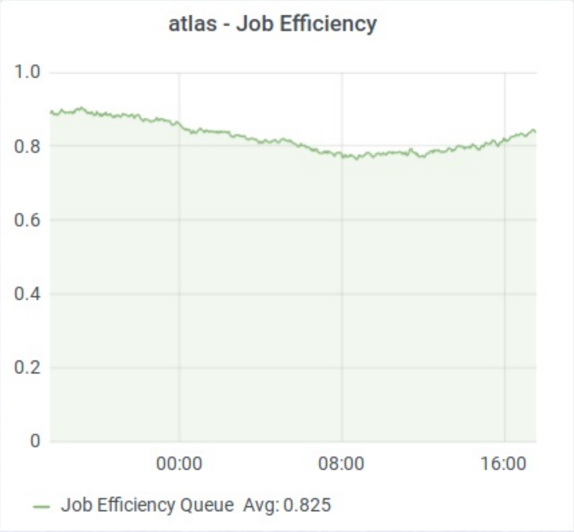
- dump_htc_shares.py injects GROUP_QUOTA_DYNAMIC_<name> values in the HTC conf. based on HS06 pledges of the user groups.
- dynup.py simple tool to ease WN upgrades requiring reboot (Kernel Upgrades)
- lostwn.py to report missing WNs (before learning of condor_status -absent)
- check_gpfs run by the STARTD as a condor CRON job, reports boolean health status of the gpfs filesystems in the machine, such as GPFS_<name> = True etc. This CRON feature quite improves things.
- job_mon.py a variant of hjobs.py used to feed our InfluxDB, accessed by graphana.

In progress: we consider writing new tools to using a straight condor_status or condor_q command to get access to the rich set of classAd functions. In particular we are practicing with ExperimentalCustomPrintFormats (https://rb.gy/bsumjq).

Monitoring and Reporting (From https://t1metria.cr.cnaf.infn.it)

The complete list of running jobs, with runtime, cputime, Cpus, HS06 in use etc. is collected every 3 mins and stored in a timeseries database, where grafana gets data to generate reports.





HS06 of Running and done jobs for a AcctGroup (queue, in LSF terms)

Coming soon: positive values from InfluxDB, negative ones from PostgreSQL

HS06 of Running and done jobs

Same as previous slide, for the whole pool

The Migration

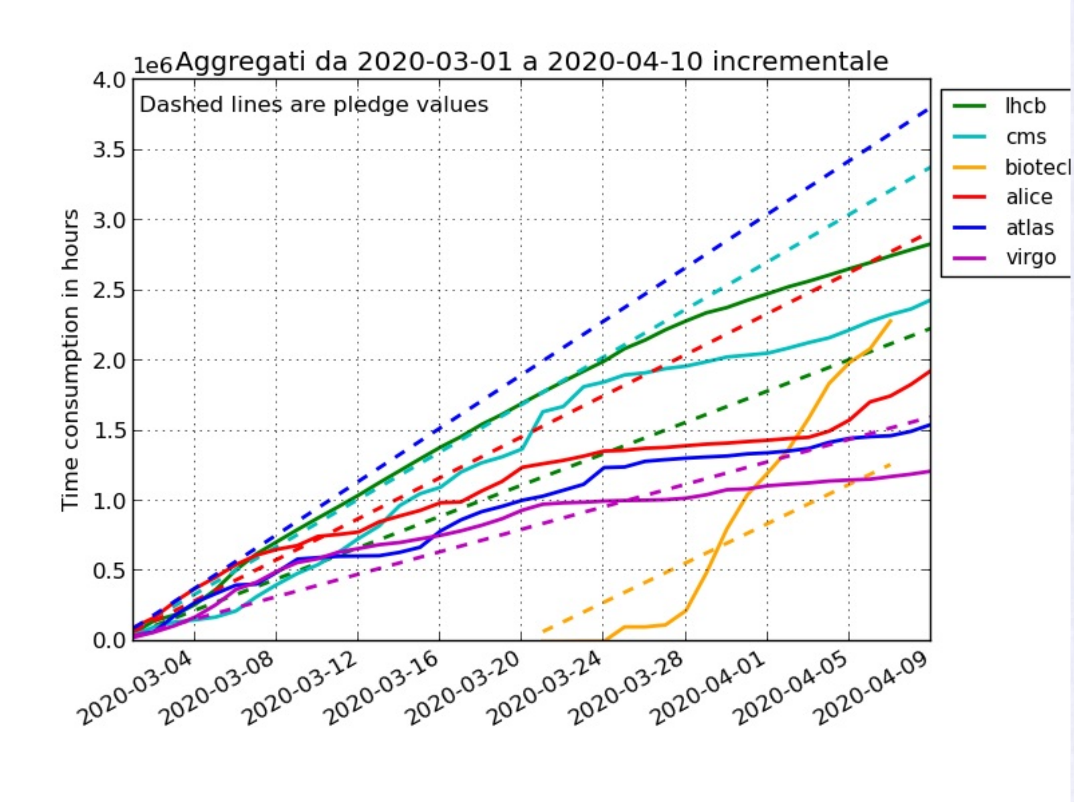
After having built a minimum set of tools and having accounting in place

- Clone the test cluster to a production one.
- Add small set of 16 WNs, have LHC VOs using it (ALICE, CMS by May 2019).
- Add a Schedd for local submitters, have local groups practice with it, with help from our "user support" team.
- Add more HTC-CEs, more WNs, Migrate VOs and local groups.
- Leave LSF with a small bunch of WNs for last "late users" (May to Jun 2020).
- Warning: during transition, providing correct shares to all is troublesome.

Migrating WNs

No need to drain the machine: LSF and HTC jobs can cohexist in the same WN, by reducing slots in LSF and setting `NUM_CPUS` accordingly at the STARTD.

Migration halfway: LSF and HTC both active, some VOs 100% on HTC, other half and half, plus an urgent "covid research project" served on demand. "rollback" was used to serve this.



Current status

- LSF phase out completed by Jun 2020
- Definitely an improvement (HTC cron jobs, DAG jobs, GPU provisioning to name a few). All LSF use cases ported successfully to HTC. New capabilities and features just waiting for us to take advantage of them.
- A number of issues emerged during or after migration. Most of them have been quickly assessed thanks to help from HTC team and community (thanks, once again!)

In progress (shortened list)

- Some tuning in progress (using `JOB_TRANSFORM` to enforce limits or prevent ill-formed submissions)
- Need (or wish) to improve our fairshare setup to consider different HS06 of WNs. Useful when job distribution of one AcctGroup is not homogeneus through the nodes.

Conclusion

- All the needed components of our farm have successfully been moved or adapted to work with HTCondor
- HTCondor is definitely a progress and an improvement for us.
- The initial testbed cluster was precious to validate operations or troubleshoot problems and gain earlier experience.
- Our learning is still in progress but we can deal well with ordinary troubleshooting.