Dealing with dynamic and mixed workloads

BY STEFANO DAL PRA

HTCondor WS, 2021, Sep 20

Email: `dalpra@infn.it`

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

- $(6+1)\times$ HTC-CE, $1\times$ CM, $875\times$ WN, (40000 Cpus, ~ 435 KHS06)
- $1\times$ SN for Remote Submission (from local UI, auth via `FS_REMOTE`)
- HTC 8.8.15, HTC-CE, 3.4.1
- We plan upgrading to HTC 9.0.5, HTC-CE 5.1 in the next days

From last year talk about LSF→HTC migration

- 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.

This talk is mostly about what has been done about the above bullet. A new policy is in use from Early July 2021.

Usergroups, workloads, pledges and shares

- ~ 50 User groups: 24 Grid VOs, ~ 25 local.
- LHC VOs are the 4 major players (total pledge sum up to $412\,\mathrm{KHS06}$)
- each group has `GROUP_QUOTA_DYNAMIC_<group>=`x_i and $\sum x_i = 1.0$

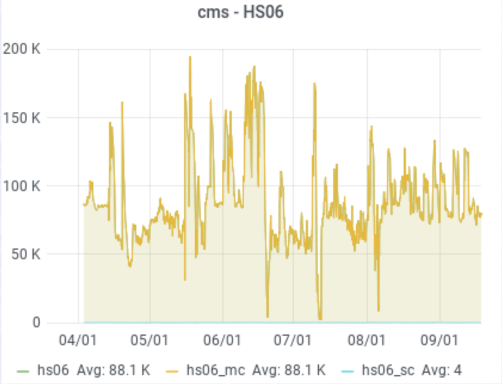
Multicore and singlecore

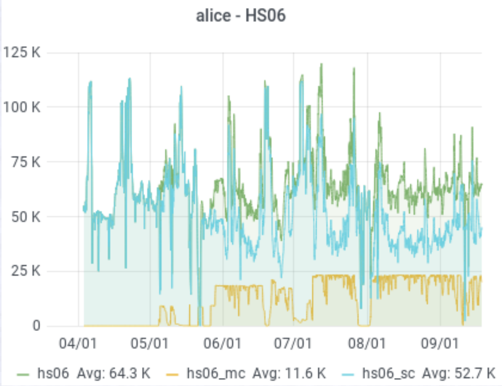
Note: in the following, sc → singlecore, mc or mcore → multicore.

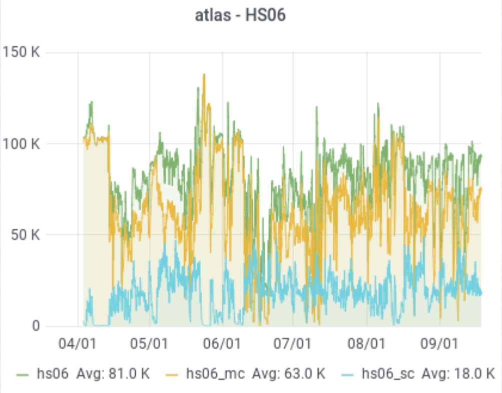
mc → `RequestCpus = 8` in `job.ad`

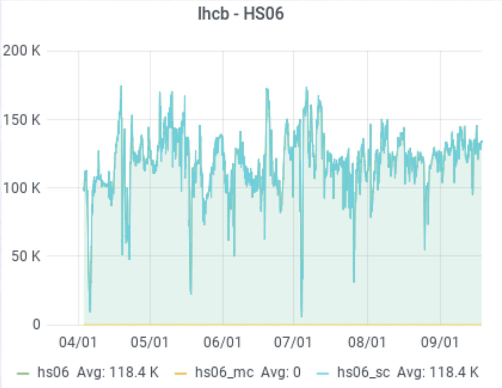
- Multicore jobs are $\sim 42\%$ (6 months average)
- 2,4,8 core (mostly 8; other size used by non–LHC groups)
- CMS → mc only, ATLAS → (quickly) variable mix
- ALICE → steady mix (from May '21), LHCb → sc only

LHC VOs jobs, last 6 months









Multicore provisioning

Initially: `DEFRAG` and recipe from the htcondor wiki

Later: also added a set of statically dedicated nodes.

Problems

1. Unused slots during mcore shortage, or overpledge of a group over another one
2. un-even opportunities: sc jobs tend to start sooner than mc of the same group
3. mcore of some groups last much longer than others (days vs hours)
4. fairshare makes no distinction on sc and mc and is unaware of different core power
5. whenever a mc job ends (and claim expired) one sc is likely to be the “next one” in queue and claim one of 8 just freed cores. Thus a new defrag time is needed.

Ideas for an alternative setup

1. Adding a grace time: when a mc job ends, only accept another mc for some time.
2. Order groups having pending jobs by expected HS06 usage - current HS06 usage (also consider size of pending jobs). First groups are "rich", last are "poor".
3. have (a subset of) machines imposing an upper limit to the rich, and accepting jobs from the poor group: mc or sc, depending on which has more pending.

Implementation

1. A `STARTD` cronjob defines a boolean machine classad `MC_GRACE` defined as `True` **if** 8 Cpus are free for less than a few minutes (currently: 8 mins).
2. This requires centrally collecting data for all running and pending jobs (see below)
3. Another `STARTD` cronjob (`JOB_CTL`) to set a few more custom machine classads

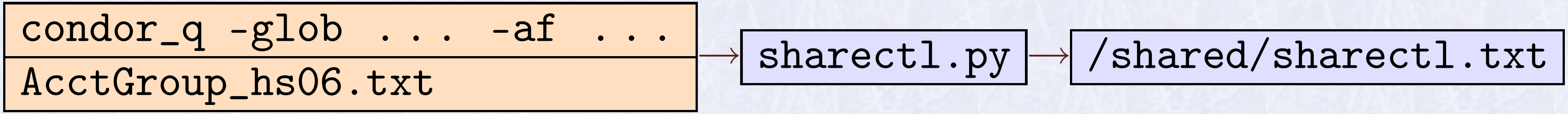
Collecting data Every 17 minutes we run:

```
condor_q -global -all -cons ’Member(JobStatus,{1,2}) && (JobUniverse != 7)’ \
-af JobStatus  ’split((RemoteHost ?: "u@PEND.t1"),"@.")[1]’ \
’split(AcctGroup,".")[0]’ ’time() - (JobStartDate ?: time())’ \
’CpusProvisioned ?: min({RequestCpus ?: 1,8})’ \
’((int(MATCH_t1_wn_hs06 ?: 400) + 0.0)/(MATCH_TotalSlotCpus ?: 40))’

# MATCH_t1_wn_hs06 = node power as machine classad, inherited by the job
# MATCH_TotalSlotCpus = same as NUM_CPUS, inherited by the job
# NOTE: We already collect elsewhere these data for monitoring
```

The output is worked on and appended to an auxiliary `shares_Error.log` file:

```
#         TO            VO  PLEDGE       DC       SC       MC      DHS     DPHS      PSC      PMC
1627101243             cms   87100     -542        0     4888    -3321   -29403        0    12784
1627101243           atlas  105300     -236     1608     4720     -489   -32021        4     2572
1627101243           alice   71400     1150     4170     1432    12594    -8785      242      410
```

Final result goes to a shared file:



```
condor_q -glob ... -af ...
AcctGroup_hs06.txt
```

→ `sharectl.py` → `/shared/sharectl.txt`

The sharectl file

Since CMS has less than expected and ALICE has more, we reduce by 8 the maximum number of allowed ALICE cores per machine. We do this according to the number of pending CMS jobs. We set this target into the shared file, which looks like this:

```
~$ cat /shared/sharectl.txt
cn-609-05-06 alice 56 cms 8
cn-610-02-03 alice 56 cms 8
cn-608-06-06 alice 56 cms 8
...
```

The JOB_CTL cronjob. Run by STARTD every 13 mins. It sets the following classads:

```
cn-609-05-06 ~]# condor_status -comp -af:ln t1_CurrentJobs t1_TargetGroups t1_Targetcores
t1_CurrentJobs = alice:64:lhcb:3:atlas:5
t1_TargetGroups = { "alice","cms" }
t1_Targetcores = { 56,8 } # { 0,0 } means no target
```

It checks for its hostname into `sharectl.txt` and sets `t1_Target*` accordingly

Now to the START expression (just a little bit cumbersome)

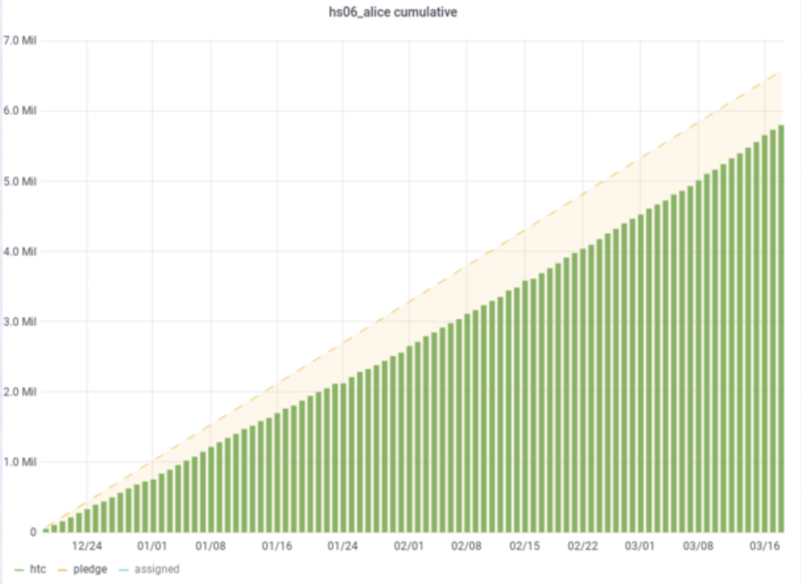
```
cn-609-05-06 ~]# ccv StartJobs
True && (!t1_overheat) && (t1_mc_grace) && t1_sharectl

#Prevent singlecore when MC_GRACE is True
cn-610-05-06 ~]# ccv t1_mc_grace
( (TARGET.RequestCpus > 1) || ((TARGET.RequestCpus == 1) && !(MC_GRACE ?: False)) )

cn-610-05-06 ~]# ccv t1_sharectl
( (t1_Targetcores[0] =?= 0) || \
(( split(AcctGroup,".")[0] =?= t1_TargetGroups[1] && RequestCpus =?= t1_Targetcores[1] ) || \
( AcctGroup =?= t1_TargetGroups[0] && \
(t1_Targetcores[0] ?: 0) > int(split(t1_CurrentJobs ?: "none:0",":")[1]))))
```

```
cn-609-05-06 ~]# condor_status -comp -af:ln t1_CurrentJobs t1_TargetGroups t1_Targetcores
t1_CurrentJobs = alice:64:lhcb:3:atlas:5
t1_TargetGroups = { "alice","cms" }
t1_Targetcores = { 56,8 } # { 0,0 } means no target
```

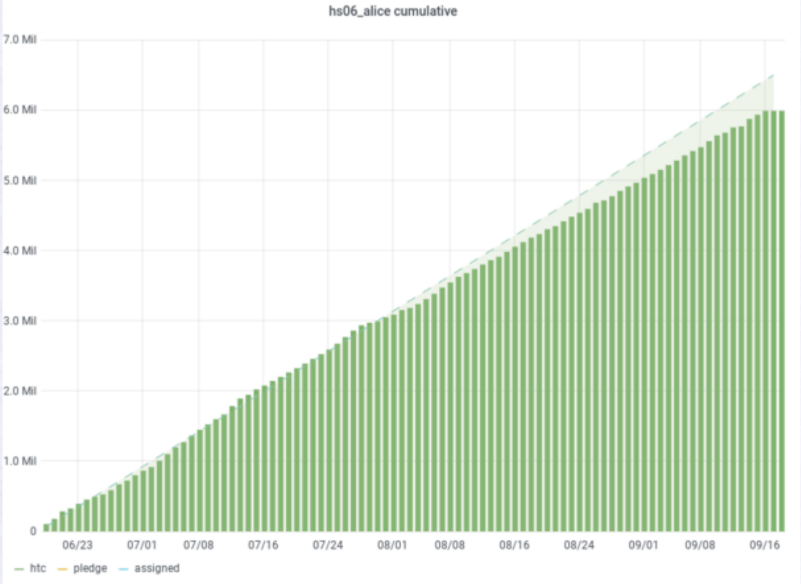
How does it work?

Started on July 2021, some stop (kernel upgrade and reboot). Steady from August.

ALICE, cumulative accounting (HS06_h). 3 months, before and after



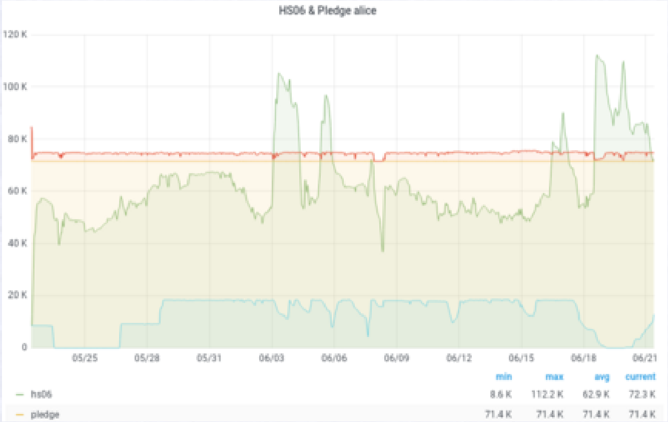
December 17 to March 16



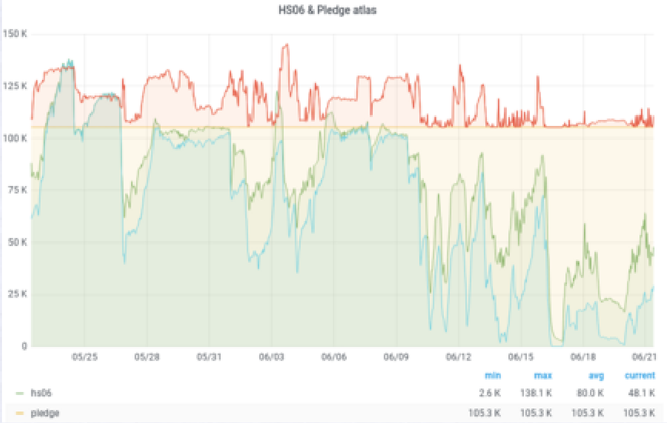
June 16 to September 16

dashed line is target value.

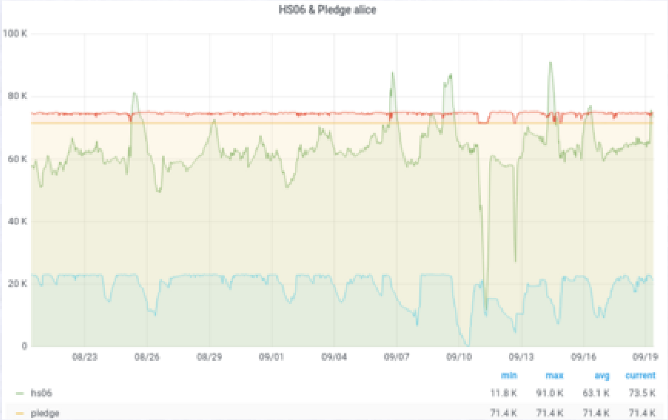
ALICE, ATLAS, 30 days, old setup vs new



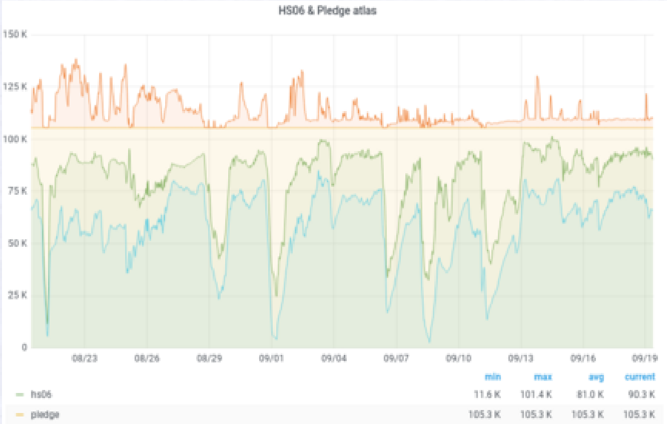
	min	max	avg	current
hs06	8.6 K	112.2 K	62.9 K	72.3 K
pledge	71.4 K	71.4 K	71.4 K	71.4 K



	min	max	avg	current
hs06	2.6 K	138.1 K	80.0 K	48.1 K
pledge	105.3 K	105.3 K	105.3 K	105.3 K

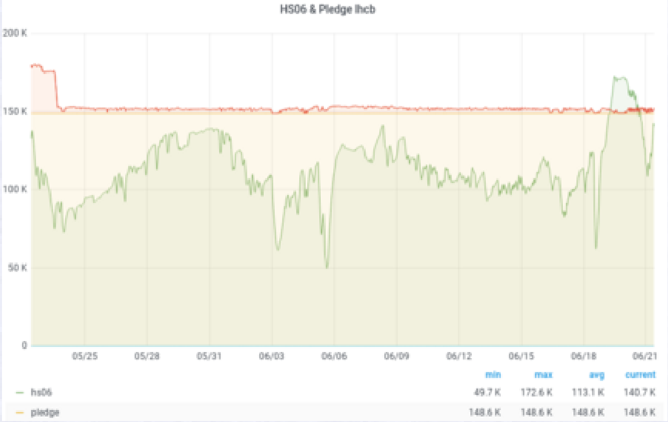


	min	max	avg	current
hs06	11.8 K	91.0 K	63.1 K	73.5 K
pledge	71.4 K	71.4 K	71.4 K	71.4 K

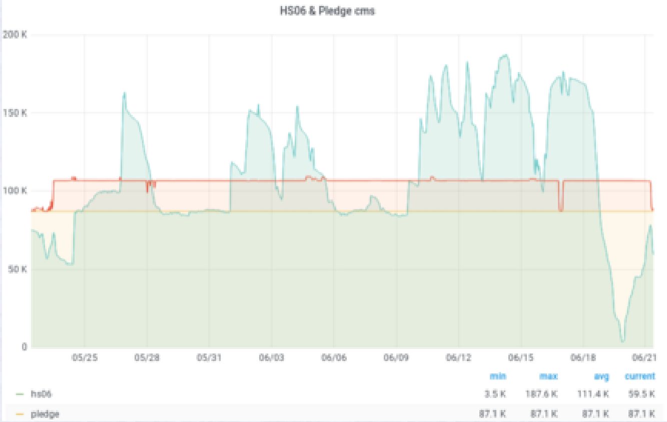


	min	max	avg	current
hs06	11.6 K	101.4 K	81.0 K	90.3 K
pledge	105.3 K	105.3 K	105.3 K	105.3 K

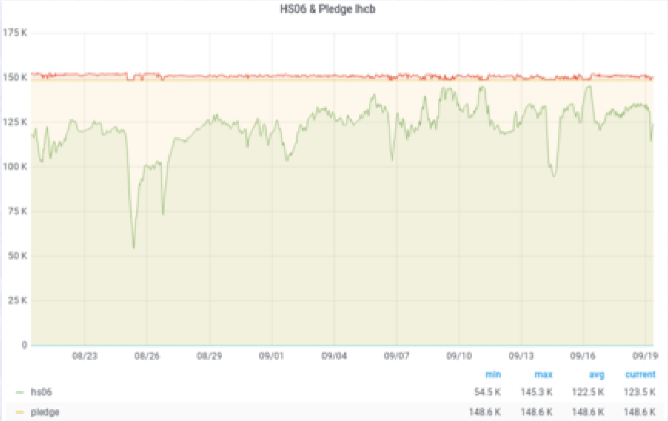
LHCb, CMS, 30 days, old setup vs new



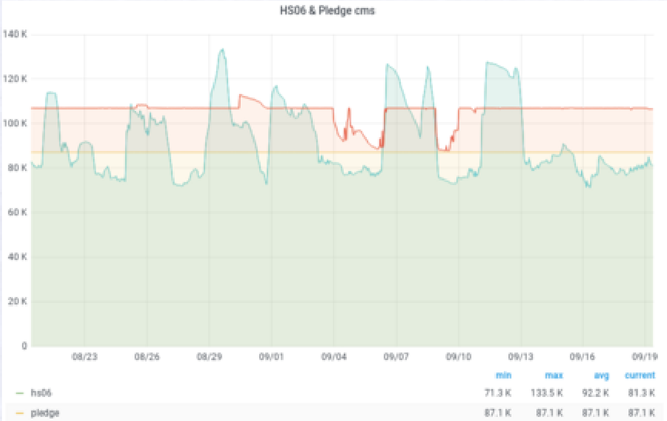
	min	max	avg	current
— hs06	49.7 K	172.6 K	113.1 K	140.7 K
— pledge	148.6 K	148.6 K	148.6 K	148.6 K



	min	max	avg	current
— hs06	3.5 K	187.6 K	111.4 K	59.5 K
— pledge	87.1 K	87.1 K	87.1 K	87.1 K



	min	max	avg	current
— hs06	54.5 K	145.3 K	122.5 K	123.5 K
— pledge	148.6 K	148.6 K	148.6 K	148.6 K



	min	max	avg	current
— hs06	71.3 K	133.5 K	92.2 K	81.3 K
— pledge	87.1 K	87.1 K	87.1 K	87.1 K

Observations

- This setup is active on a subset ($\sim 50\%$) of all computing power.
- The `DEFRAG` daemon was stopped a few days after
- The `Errors` used to compute `sharectl.log` are in `HS06` units.

Possible improvements

- The control policy only takes the latest value for $e_g(t) =$ current - expected quota. We expect better results by averaging it with past values.
- The same should hold for pending jobs, to ensure that we consider groups having a regular submission rate