Adapting HTCondor fairshare for mixed workloads

by S. Dal Pra,
C. Pellegrino

ISGC WS, 2022, Mar 25

Email: dalpra@infn.it

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

CNAF, located in Bologna, hosts the Italian Tier-1 of WLCG, operated by INFN.

- $(6+1) \times$ HTC-CE, $(1+1) \times$ CM, $875 \times$ WN, (~ 40000 CPU cores, ~ 435 KHS06)
- $1 \times$ SN for Remote Submission (from local UI, auth via `FS_REMOTE`)
- HTC 9.0.11 (latest Long Term Release version), HTC-CE, 5.1.3
- ~ 50 User groups: 26 Grid VOs, ~ 25 local user groups.
- Every user group yearly negotiates Computing Power **quotas**, which are then translated to percentual **shares** of the *expected* Total Computing Power of the centre.

Fairshare

- A fairshare policy is designed to provide active users with an average amount of CPU power proportional with their shares on the available resources.

Possible problems

Starvation. A job is never dispatched because there always are higher priority ones.

Underusage. Computing resources remain unused because constraints on job requirements never meet cachine capabilities.

Typical workload at INFN-T1

- Saturation: whenever a computing slot becomes available, there always is a pending job who could be started there.
- Heterogeneity: user groups have independent and almost unpredictable submission patterns (duration, memory and I/O needs).
- Job size: variable mix of multicore (8-core) and singlecore, even from the same user.

Limits of Fairshare

There are cases where a particular group can work far from its expected average quota. These depend on several conditions. Two notable ones are:

1. Multicore availability vs Multicore demand

- demand for multicore slots can grow and drop quickly and frequently.
- Multicore provisioning takes time: almost no Compute Node normally has 8 free slots, thus a draining policy is needed. This of course implies resources underusage during the draining time.

2. Different Node Computing Power

- Different Compute Node (aka WN) models have different power (in HS06).
- However the fairshare implementation does not consider specific core power.

Common strategies to provide Multicore resources

Use DEFRAG. HTCondor daemon configured to implement a draining policy

Cons: does not cope well with dynamic multicore demand.

Static provisioning. a set of nodes is dedicated to only accept multicore jobs

Cons: Risk of machine underusage or group overquota

Use both. Useful to model a fixed base load plus dynamic pattern.

Cons: better, but no static tuning fits all submission patterns.

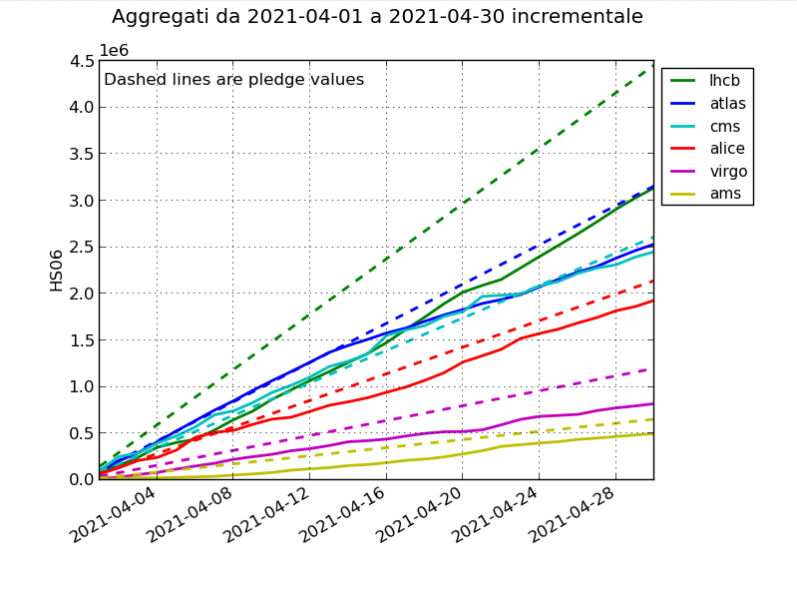
- All the above "classic methods", in order, were adopted at CNAF.
- A new strategy was defined, implemented and adopted starting from Jul. 2021. We compare results and explain the method.

Problems to address

Note. In the following: multicore $\longrightarrow$ mc , singlecore $\longrightarrow$ sc.

1. Unused slots on mc submit 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. mc 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.

Cumulative work vs pledges (classic methods vs new method)



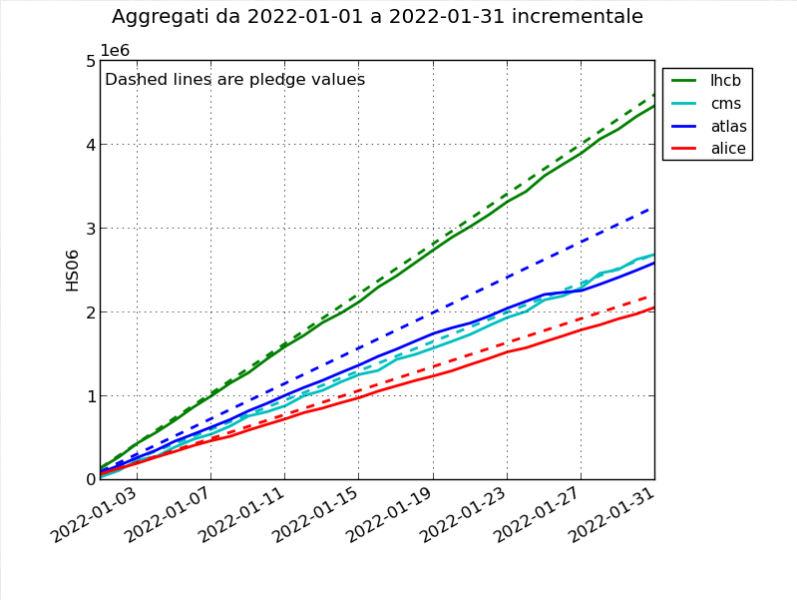


Figure 1. Left: April 2021. LHCb (sc only) under quota, ALICE (sc only as of April) almost ok. **Right: January 2022**. All groups reasonably near to target, except for ATLAS (mostly because of discontinuous submission flow)

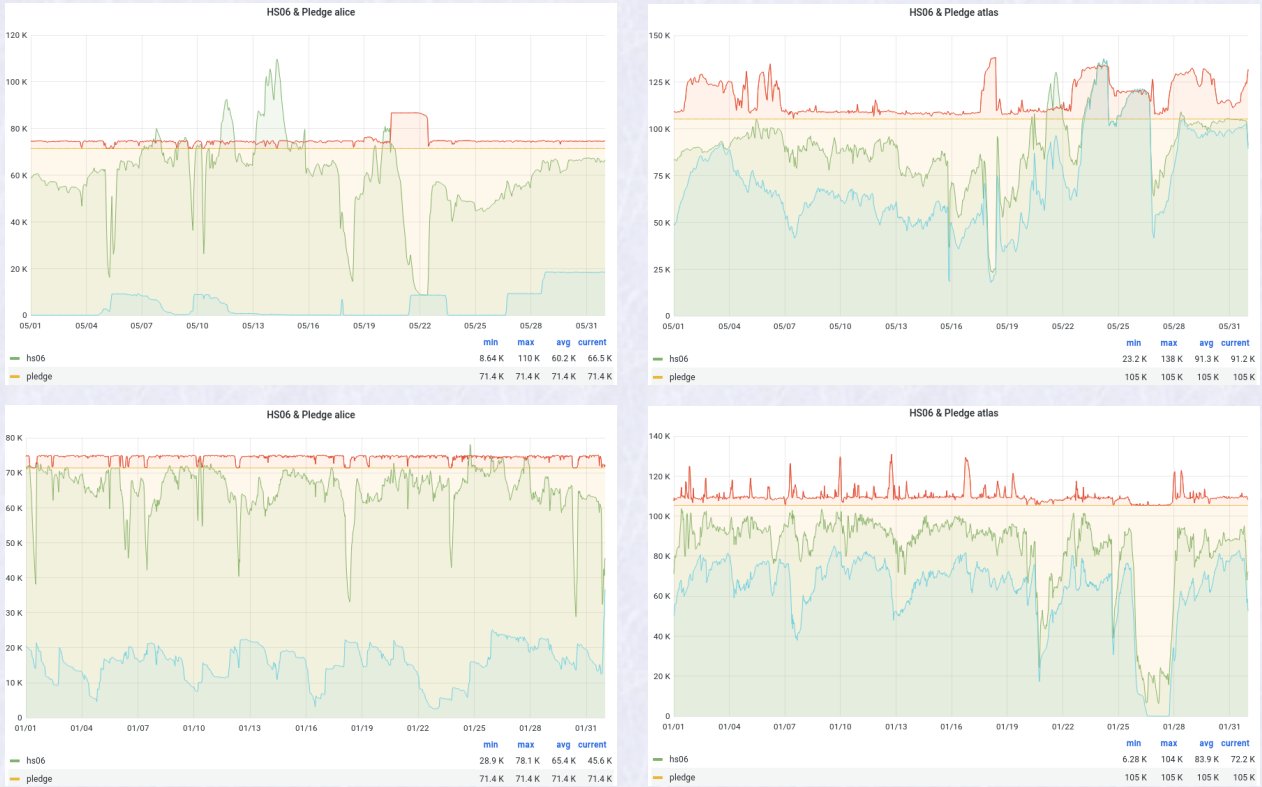


Figure 2. ALICE, ATLAS, 30 days, old (upper) vs new method (lower)

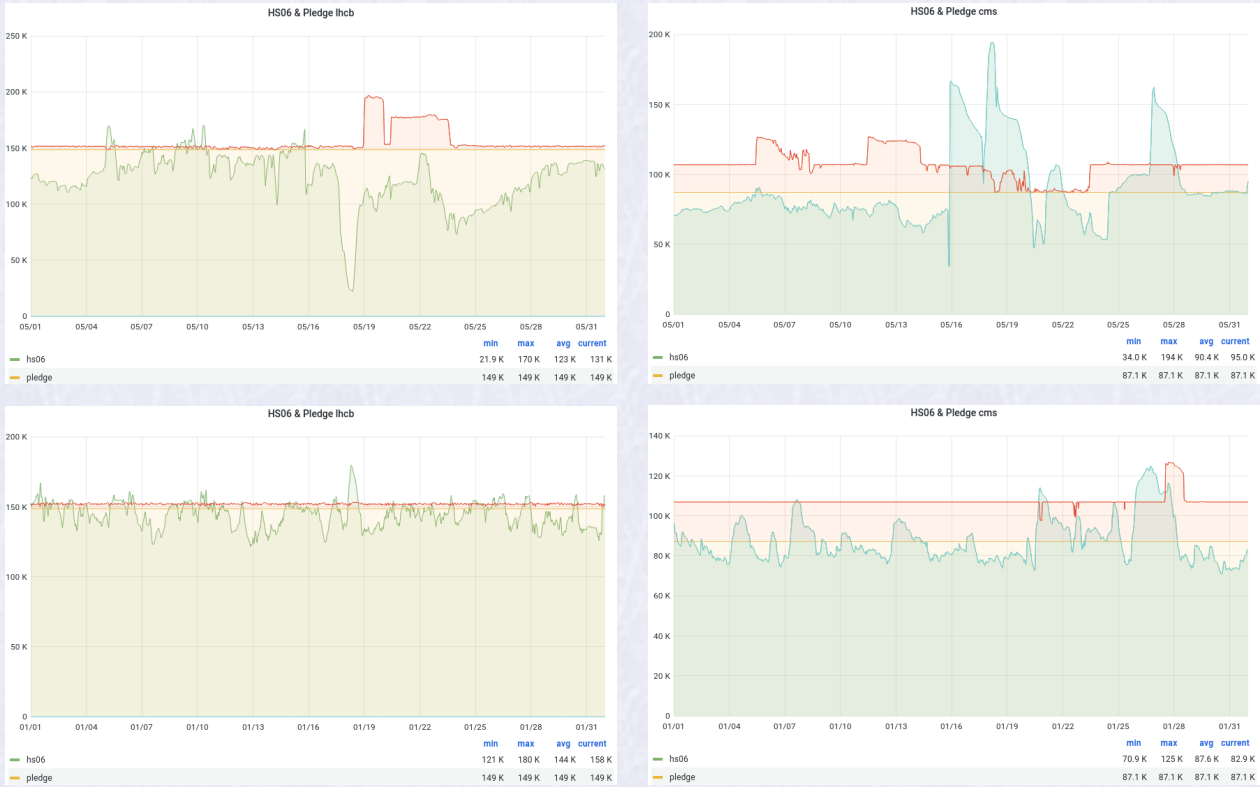
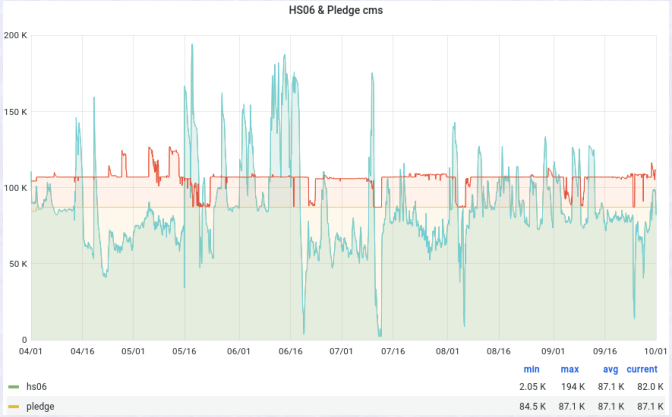
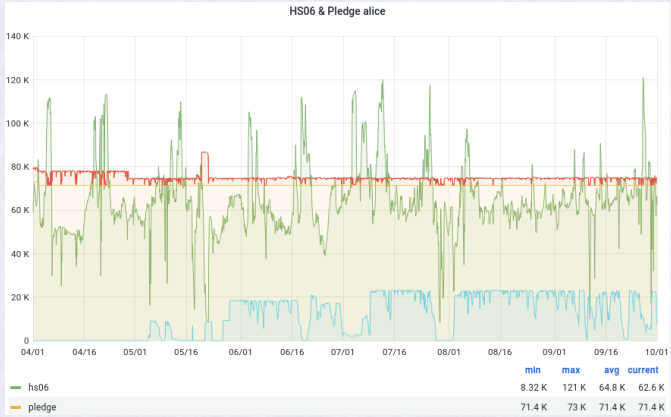
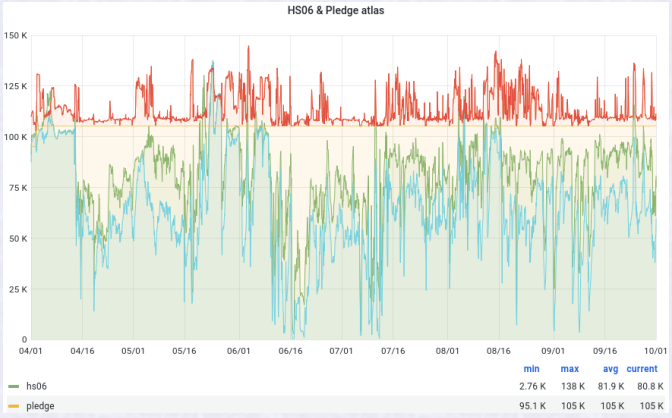


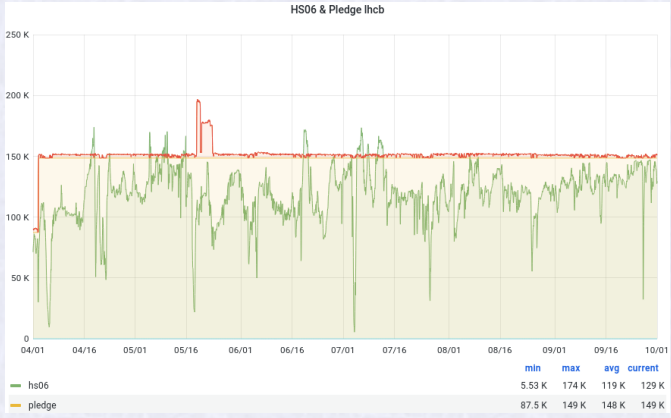
Figure 3. LHCb, CMS, 30 days, old (upper) vs new method (lower)

LHC VOs jobs, over 6 months (Apr. to Sep 2021)









The new method

Key idea: Dynamically adjust WN "preferences" to better meet current needs.

1. Grace time: when a mc job ends, only accept another mc for some time.
2. For each group g, having a target quota $\bar{q}_g$, consider the errors $e_g(t) = q_g(t) - \bar{q}_g$
3. Compute $g_r = \mathrm{Arg\,max}\{e_g\}$, $g_p = \mathrm{Arg\,min}\{e_g\}$ for groups with pending jobs.
4. Add constraints on a set of WN accept jobs from g_p and limit those from g_r.

Implementation

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

Collecting data (Warning: gory details ahead!)

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 attribute, 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 is under quota ($e_{\text{cms}} < 0$) and ALICE is over quota, we set a 8-core penalty for 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:

```
~$ 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 42 cms 8
...
```

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

```
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 set `t1_Target*` accordingly

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

`StartJobs` is a boolean expression to test job requirements against machine capabilities.

```
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
```

Observations

- This setup can be active on a subset ($\sim 50\%$ or more) of total computing power.
- The HTCondor `DEFRAG` daemon was stopped a few days after
- The `Errors` used to compute `sharectl.log` are in `HS06` units.
- The actual scheduling is decided by the HTCondor fairshare algorithm. This method is only intended to reduce imbalances experienced when using the classic methods.

Possible improvements and future plans

- For every user group g, having a target quota $\bar{q}_g$, the control policy only consider the latest value for the error $\boxed{e_g(t) = q_g(t) - \bar{q}_g}$, i.e. current - target quota. We expect better results by averaging it with past values (More in general: realizing a PID controller or other classic control theory approach).
- Since a rich dataset is being collected from our monitoring tools, we can also consider the training of a ML/DL model (reinforcement learning)