HTCondor Design Document:

Cooperative Scheduling

# Introduction

While a pilot job presents itself as a batch job to HTCondor, it differs in that it may have an essentially unlimited amount of valid work. In batch systems, we tend to place runtime limits on user jobs for two reasons:

1. As a mechanism to enforce fairshare.
2. To prevent the user executable in an infinite loop from consuming unlimited resources.

For pilot jobs, (a) is still valid; (b) is no longer a concern.

When the batch system must preempt jobs, it may aim to minimize the number of hours of lost work. For a traditional batch job, we assume the lost work is equal to the job runtime. For a pilot job, which may run several user jobs in sequence, we currently

This document outlines some extensions to the HTCondor batch system to increase the lifetime of pilots and decrease the number of hours of wasted time.

# Architecture

We introduce the concept of *cooperative scheduling*. This has three pieces:

1. **Preemption Potential**: The site negotiator will inform the site schedd when the submitter of a running job has gone over its fairshare *and* another user’s job would have matched against claimed slot if it were idle.
   1. This should only be done if the other user has higher effective priority and, optionally, in a separate accounting group[[1]](#footnote-1).
2. **Uncommitted Work**: The pilot will inform the site schedd (and hence site negotiator) of the number of uncommitted[[2]](#footnote-2) hours.
3. **Cooperative Draining**: The pilot will inform the site schedd when it believes it will next commit its work. The pilot has the ability to query the starter for its current state.

Item (1) provides hints to the pilot system about whether it is over its fairshare at the site. Ideally, the pilot system should not start a new user job if it knows it is over its fairshare (and hence in danger of preemption).

Item (2) informs the site negotiator about the cost of preemption. The negotiator can rank potential candidates preemptions to minimize badput.

Item (3) informs the site HTCondor about the pilot-estimated costs of draining the slot and informs the pilot about whether its node should be drained.

# Implementation

## Potential Preemption

#### Condor

During the negotiation cycle, if a potential matched machine is running a job but PREEMPTION\_REQUIREMENTS evaluates to FALSE, nothing is currently done. Consider this case for the remainder of this section.

We introduce a new policy parameter, **RECORD\_POTENTIAL\_PREEMPTION**. If this evaluates to true, the potential matching resource and potential preempting submitter is placed on an internal, per-schedd, potential preemption list.

During the negotiation cycle, if the potential match is indeed preempted, it is removed from the potential preemption list. If there are other potentially preempting submitters, they are not added to the list.

At the end of the negotiation cycle, iterate through the potential preemption lists. For the running job on the potentially preempted slot, record:

* **LastPotentialPreemptionTime**: Time the negotiation cycle finished.
* **LastPotentialPreemptingUser**: The accounting group / user that was considered to preempt this job.

These attributes are cleared before the job is run if they were set on a previous run.

We will also add a new ephemeral attribute to the job ad, only present during the ranking portion of matchmaking – **QueueAge**. This is set to **time()-QDate**; while this is straightforward to calculate in the schedd, we use this as an ephemeral attribute to prevent auto-cluster from breaking. This should only be used during ranking as it can result in incorrect matching if used in the requirements.

#### GlideinWMS

Using the startd-cron mechanism, glideinWMS pilots can periodically query:

condor\_chirp get\_job\_attr\_delayed **LastPotentialPreemptionTime**

If this last potential preemption time is defined and within the last 20 minutes, the glideinWMS starter’s START expression will evaluate to FALSE.

### Example

Nebraska has two top-level accounting groups, *cms* and *other*. The *cms* accounting group should receive 90% of the resources and *other* 10%. Jobs in *cms* should preempt jobs in *other* if they have been in queue for 2 hours.

Within the *cms* group, there are *production* and *analysis* jobs. There is one *production* user and 10 analysis users (all from the same glideinWMS instance). Production and analysis both get 50% of the *cms* share. Jobs in the *cms* group should never preempt another job in the *cms* group; however, we would like to use cooperative fairshare to keep the sub-groups balanced.

They set:

PREEMPTION\_REQUIREMENTS = (SubmitterGroup isnt null) && regexp(“^cms\.”, SubmitterGroup)) && (RemoteGroup =?= “other”) && (QueueAge < 7200)

RECORD\_POTENTIAL\_PREEMPTION = SubmitterGroup =!= RemoteGroup

## Uncommitted Work

#### Condor

Via a new condor\_chirp verb (“condor\_chirp set\_last\_commit $UNIX\_TIMESTAMP”), we allow the following attribute to be set in the starter: **LastCommit**. By default, this is equal to the ClassAd expression **JobStart**.

The match’s **UncommittedTime** is calculated as:

(Negotiation cycle start time) – LastCommit

and added to the list of ephemeral attributes available during matchmaking for the slot.

A site can then add the following to their configuration:

NEGOTIATOR\_POST\_JOB\_RANK = -UncommittedTime

This will cause the negotiator to preferentially preempt slots with the least amount of uncommitted time.

This will also cause the starter to update the ExpectedMachineQuickDrainingBadput, which is used by condor\_defrag to estimate the cost of a quick drain.

#### glideinWMS

The glideinWMS startd will need to periodically set the LastCommit time via a startd-cron job.

## Cooperative Draining

#### Condor

Via a new condor\_chirp verb (“condor\_chirp set\_expected\_commit $UNIX\_TIMESTAMP”), we allow the site batch job set ExpectedMachineGracefulDrainingBadput. This is the attribute used by condor\_defrag to estimate graceful draining costs.

We will add another condor\_chirp verb (“condor\_chirp get\_starter\_attr $ATTRIBUTE”) to retrieve starter attributes. This will allow the pilot to query for the starter state.

#### glideinWMS

The glideinWMS startd will need to periodically set the ExpectedMachineGracefulDrainingBadputtime via a startd-cron job.

The glideinWMS startd will need to periodically query the site starter state to see if it is currently draining; if so, it should make its START expression evaluate to False.

1. We optionally only consider separate accounting groups because the same pilot system may be submitting as separate users for technical reasons. Preempting one pilot for another pilot from the same system is not helpful. [↑](#footnote-ref-1)
2. When work has been committed, the pilot may be killed without losing any progress. Typically, this means a user job has finished – but could also include any checkpointing activity. [↑](#footnote-ref-2)