## Defragmentation Daemon Design Document

### Overview

The partitionable slot defragmentation daemon (condor\_defrag) is responsible for scheduling drainage of partitionable slots so that partitioned resources can be regrouped into chunks big enough to satisfy large jobs (e.g. whole machine jobs). The goal is to support a mixture of large and small jobs with automated policies that allocate resources between these different types of jobs.

### Drainage Rate Policy

One simple policy is to drain whole machines at a fixed rate. This ensures a steady supply of new whole machines, in case they are needed. Additional constraints may further limit the draining rate: a limit on the total number of draining machines and a limit on the total number of defragmented machines.

Simplifications: Initially, no attempt will be made to auto-adjust the rate based on demand. An external process could take this into account and update the configuration of this daemon. Worst case: if the whole machines are not needed, it is assumed that they will be partitioned again when matched to small jobs. Which machines to drain will also be chosen without regard to job requirements. Also, no attempt will initially be made to reserve the freshly drained machines for large jobs. If large jobs are starving for resources, it is likely they will be matched to the drained machines before small jobs. In the future, further controls may be added to improve upon these basic mechanisms.

The parameters governing the drainage are the following:

* DEFRAG\_DRAINING\_MACHINES\_PER\_HOUR – maximum number of machines to initiate draining per hour
* DEFRAG\_MAX\_CONCURRENT\_DRAINING – maximum number of machines in draining state at one time
* DEFRAG\_MAX\_WHOLE\_MACHINES – if there are this many or more defragmented machines, do not drain any more
* DEFRAG\_WHOLE\_MACHINE\_EXPR – expression that identifies which machines are whole-machines (should not need to change default if using standard setup where one partitionable slot represents whole machine)
* DEFRAG\_REQUIREMENTS – machines to be drained must match this expression (i.e. the partitionable slot matches)
* DEFRAG\_RANK – expression that determines which machines are more desirable to drain
* DEFRAG\_DRAINING\_SCHEDULE – graceful, quick, fast
* DEFRAG\_INTERVAL – polling period of condor\_defrag

The daemon will periodically fetch ads from the collector and analyze the pool with respect to its policy. It will compute how many new machines to drain for the next polling interval. (It never tries to make up for past shortfall or excess. It only looks at current conditions, current configuration, and then plans for the next step in time.) It will then choose machines by selecting from among those that have one or more slots matching DEFRAG\_REQUIREMENTS and no slots matching DEFRAG\_WHOLE\_MACHINE\_EXPR. If there are more than enough candidates, it will rank the slot ads using DEFRAG\_RANK and will choose the machine with the highest ranked slot. In case of a tie, a random choice will be made. If any machines were chosen, it will then initiate drainage.

The draining rate is expressed as machines per hour. This is converted internally into machines per polling period (DEFRAG\_INTERVAL). The integer truncation error in the number of machines per period is corrected with additional draining once per hour (i.e. in the first evaluation cycle in the hour).  The remaining error is corrected once per day (i.e. in the first evaluation cycle of the day).  The lowest rate supported by this algorithm is 1 machine/day or 1 machine/polling period, whichever is lower.

## Draining Request

The defragmentation daemon will send the draining request with the following parameters:

* schedule: (graceful, quick, fast)
* staleness tolerance expression that must be true or request is rejected
  + for now, require expected badput to be within 25% plus some small amount that is considered negligible in case the expected badput was zero

## Monitoring

condor\_defrag will report draining-related statistics in its daemon ad:

* recent and total successful machines drained
* recent and total failures to drain machines
* recent and max concurrent machines draining
* recent and max whole machines
* fraction of time that was badput caused by draining over life of startds in pool
* fraction of time that was unclaimed time caused by draining over life of startds in pool

## Failures

Frequent restart of the defrag daemon could lead to a draining rate that exceeds the configured maximum unless steps are taken to avoid this. To protect against this, the daemon will save a timestamp of the last time it evaluated its policy. On restart, the daemon will use this saved timestamp to schedule the next evaluation. This timestamp is also used to know whether the hourly and/or daily corrections to the draining rate have already been applied or not. Failure to save the timestamp will cause the daemon to abort policy evaluation.

An alternative to saving a timestamp would be to have the startds advertise when they were last drained. condor\_defrag could use this information to figure out how many machines were recently drained. However, this is more suitable to a backward-looking policy than the forward-looking policy that we have thus far defined. In the forward-looking case, one issue is that the daemon does not know if it had the same configuration in the past as it does now. It does not know if it was even running in the past. It does not know whether draining was limited in the past due to the maximum number of concurrent draining machines and/or the maximum number of concurrent whole machines. For now, we are sticking with the forward-looking policy and are therefore using a saved timestamp rather than a survey of the pool.

Another type of failure is failure to query the collector. This will cause policy evaluation to be aborted, which may result in a draining rate that is lower than the configured target.

When assessing the current state of the pool, condor\_defrag will rely on the advertised state of machines in the collector. A consequence of this is that in the event of stale information in the collector, the policy may not be correctly enforced: the maximum number of concurrent draining machines and/or the maximum number of whole-machines may be exceeded.

If the request to drain a machine is not successful, condor\_defrag will log the failure and move on to the next candidate. This could result in less desirable machines being selected for draining.

## Development Plan for Pool Defragmentation

* [1 day] Create new condor\_defrag deamon, using rooster as a template. Get basic loop working in which ads are fetched. For testing purposes, just have it blindly initiate draining on all slots.
* [3 days] Implement draining policy evaluation.
* [1 day] Update documentation.
* [2 days] Implement automated tests:
  + MAX\_CONCURRENT\_DRAINING is not exceeded
  + Draining is initiated for machines that match DRAINING\_REQUIREMENTS
  + Draining is not initiated for machines that do not match DRAINING\_REQUIREMENTS
  + Draining is initiated for machine with highest DRAINING\_RANK