Historical Design Doc: Auto-clustering

Aug 2013

[As a *historical design doc*, this document describes from an architecture/developer perspective an already implemented mechanism in HTCondor]

Motivation

Previous to auto-clustering back in 2005 or so, we noticed that the negotiator spent the majority of its time considering job ClassAds that did not result in a match. On average over a sample six day period in the CS Pool, 10.0% of the ads considered during a cycle resulted in a match, meaning that nearly 90% of the negotiator's time was being spent doing work that did not produce a tangible result (i.e. a match). The numbers in the other pools we profiled, such as the GLOW pool, were similar --- in GLOW during the same period, 15.9% of the ads considered resulted in a match.

The goal was to minimize presenting job ads to the negotiator that do not result in a match.

Approach

The approach of auto-clustering is to assign each job *auto-cluster id*, or signature. By definition, if a given job ClassAd with auto-cluster id X could not be matched, then none of the other jobs with auto-cluster id X could be matched.

The effectiveness of this approach is dependent upon common usage of HTCondor. It was our hypothesis that when a typical HTCondor user submits large groups of jobs, these jobs would normally differ in ways that are not significant for purposes of matching against a machine. In other words, we hoped that most job ClassAds from a given user differ in terms of input files, command line arguments, and output files, and generally have very little difference in terms of requirements or preferences upon a given machine.

Our algorithm for automatic determination of an auto-cluster id is as follows:

1. At the start of a negotiation cycle, the negotiator will inspect each machine ClassAd and create a list of attribute names that cannot be resolved within the scope of the machine ad itself. In bilateral matching, it can be assumed that these undefined attributes will likely need to come from a job ClassAd. We call these undefined attributes *significant attributes*.
2. The negotiator adds to the set of significant attributes any unresolved attributes contained in negotiator policy expressions (e.g. PREEMPTION_REQUIREMENTS) when these expressions are evaluated within the context of typical machine ClassAd.
3. The negotiator passes the union of all significant attributes it found to each schedd.
4. For each job, the schedd augments the significant attributes received from the negotiator by adding the job's *Requirements* and *Rank* attributes, as well as any job ClassAd attribute that is referenced by the job's own Requirements or Rank expression. The value of all the job's significant attributes and their corresponding values are placed into a string and hashed. The hash value is said to be the job's *matchmaking signature*.

5. All jobs with the same matchmaking signature are assigned the same unique auto-cluster id.

In an attempt to minimize the cost associated with performing auto-clustering, the auto-cluster id for each job is cached by placing the id as an attribute into the job ClassAd itself. During negotiation, the auto-cluster is only recomputed if this attribute is missing. The cached value is discarded and thus recomputed if i) the union of the list of significant attributes received from all negotiator(s) change (this could occur if a site administrator changes machine Requirements or Rank expressions, for instance), or ii) if any of the job attributes used to compute the signature change. To implement (ii), the list of job significant attributes is also inserted as an attribute into the job ad itself. When the queue management *SetAttribute()* method in the schedd is invoked, it will remove the cached auto-cluster id attribute if the attribute being updated is present in the list of significant attributes.

We next modified the negotiation process so the schedd will ignore for the remainder of the negotiation cycle any job ClassAd that has the same auto-cluster id as a job for which the negotiator failed to find a match.

In practice, some numeric significant job attributes can cause the number of auto-clusters increase unnecessarily. For example, it is common for machines to require that the ImageSize of a job be less than available Memory, but it is undesirable to consider jobs that differ in image size by 3 bytes to be negotiated separately. Therefore, we changed the schedd to quantize job attributes dealing with image size and disk usage for purposes of matchmaking by rounding up 20% by default.

Performance

Some performance results of this work are presented in Todd Tannenbaum's CS 739 project writeup. Refer to the paper for methodology and details, but a couple relevant quotations:

- "With the introduction of auto clustering, the average percent of job ads considered by the negotiator that resulted in a match increased from 10.0% to 64.3% with a six day sample of Comp Sci Pool trace data, and similarly went from 15.9% to 78.3% in the GLOW Pool."
- "In the course of the six days within the Comp Sci pool, an average of 85 active users had 5831 jobs submitted. These 5831 job ClassAds were distilled down into an average of just 372 autoclusters (and thus 372 resource request ads) — an approximate reduction by 15x the number of ads the negotiator needed to consider during the matchmaking cycle. It took the schedd an average of 4.7 seconds to perform the autoclustering on all 5831 (on average) job ads, but recall the autocluster information is cached. Furthermore, the autoclustering load will be partitioned across multiple schedds."

Future Work

The schedd should collect metrics over time (incl average, stddev) about the current number of active auto-clusters and ideally the amount of time spent computing auto-clusters IDs.

Perhaps machine ads could similarly be auto-clustered (see gt #2130). An obvious application of this mechanism would be to speed up matchmaking within the negotiator, but perhaps a more interesting application would be for the negotiator to send a condensed auto-clustered list of available machines down to the schedd instead of the schedd sending resource requests up to the negotiator.

Currently the schedd uses the union of significant attributes sent to it from all negotiators that it flocks with. This likely results in more auto-clusters than required for any one pool. Instead, there could be a separate set of auto-cluster IDs/significant attributes per negotiator.

When the negotiator fails to find a match for a given job ad, it sends back a match failure reason (e.g. “No machine found that matches job requirements”) which the schedd then inserts into the job ad. However, with auto-clustering, all jobs in the same auto-cluster are skipped after a failure to match. The end result is currently *condor_q -analyze* misleadingly reports for many jobs “Request has not yet been considered by the matchmaker” because in fact they have been skipped due to auto-clustering. Some job attributes, like matchmaker failure reason, should be shared across all jobs in an autocluster.

The heuristic of quantizing certain scalar job attributes like ImageSize feels clumsy and imprecise; maybe something better could be done.