Design Document: Non-blocking Queue Responses

Author: Brian Bockelman

# Introduction

When responding to ‘condor\_q’, the schedd currently forks a child process to handle the response. This allows the client to see a consistent view of the queue without blocking the schedd. Because of Unix’s copy-on-write behavior, the memory use by the child should be minimal (assuming not many memory pages of the parent are written to while the child exists).

However:

* **Blocking behavior can still occur**. If the schedd hits the maximum number of concurrent children, the parent will perform a blocking response. Because the sysadmin has no way to predict load and frequency of client requests, it is effectively impossible to prevent blocking.
* **Children processes can use significant resources**. On busy schedds, the parent’s memory changes rapidly enough that the child process sees significant memory use. We have measured approximately 500 MB/s increase of RSS for a single child on the busiest schedds.
  + This is not an issue we can reasonably solve while still forking children – we have little ability to control or measure this growth.

We propose to implement a new query protocol that allows the schedd to respond from the parent process without blocking.

# Architecture

We plan a non-blocking callback mechanism for writing responding to job queries.

We will add a new command handler solely for job queries. This must be separate from QMGMT because QMGMT contains a global state, preventing multiple concurrent queries. We instead will add a stable iterator class that can iterate through a ClassAd collection (and multiple iterators can exist at once).

Responses will be sent to the client until the ReliSock would block; at that point, state will be kept in an object on the heap, a callback will be registered for when the ReliSock can be written into, and control will be returned to DaemonCore. When the ReliSock is ready for writes, the callback will be invoked on the state object and iteration through the collection will continue.

In addition to blocking on the TCP socket, blocking can also be caused by selective filters. For example, ‘condor\_q –const false’ will result in iteration through the entire queue before returning to DaemonCore – possibly blocking a schedd with a large queue for several seconds. Accordingly, once every 1,000 jobs considered, control will be returned to DaemonCore.

Note this new query protocol will have slightly different semantics compared to the existing protocol. The client will not receive a snapshot of the queue from a single point in time; if jobs are being added or removed while the queue is running, these jobs may or may not be reflected in the response. However, we are able to guarantee that jobs will never be sent twice in the response.

# Implementation

To implement a non-blocking condor\_q protocol, we need the following three preliminary pieces:

1. **Register sockets for writing in DaemonCore**. DaemonCore does not have the ability to perform callbacks when a socket is ready for writing. We will add the ability to register a socket for writing (similar to how pipes are handled).
2. **Non-blocking writes for CEDAR**. CEDAR will be extended to buffer responses in memory when a write will block. We will add the following methods to ReliSock:
   1. Allow the schedd to enable non-blocking operation. This will be done with an object that automatically returns to blocking mode when it goes out of scope (this is done to prevent the schedd from inadvertently keeping the socket in non-blocking mode).
   2. Allowing the schedd to determine when buffering occurs.
   3. Further, as the EOM method must never block in the current ReliSock semantics, we will add a new EOM\_nonblocking method that may not necessarily finish the EOM protocol.
   4. We will also add a EOM\_finish method which attempts to finish a prior EOM\_nonblocking call.
3. **Stable iterators for the ClassAd collection**. A new iterator class will be added for HashTable. This iterator will be stable – that is, adding or removing entries to the HashTable will not invalidate the iterator or cause it to repeat table entries. On top of this iterator, we will add a specialized iterator for the ClassAd collection that filters out ads according to a ClassAd expression.

Once these are in place, we will add a new DC command to the schedd and a new class to track the state of the response to the client. Support for the new query protocol will be added to condor\_q and the python bindings. For backward compatibility, the client will use the old protocol for older schedds. Additionally, we will provide a client parameter that disables the old protocol.