Workload Management Delayed Move

Currently, the Workload management move trigger kills the query being moved to a different pool if destination pool does not have enough capacity. We propose a "delayed move" configuration which doesn't immediately kill the query when the destination pool is full but lets the query run in the source pool as long as possible. Workload manager will attempt the move to destination pool only when there is claim upon the source pool. We also introduce additional options for retrying delayed moves periodically and moving the session to the destination pool after a specified amount of time.

Current design:

1. The trigger validator thread checks for trigger violations at specified intervals.
2. If it finds that a move trigger has been violated, it creates a move session/event and notifies the workload management master thread.
3. It then waits for the move event to be processed by the master thread.
4. The master thread runs in a loop and keeps processing all the workload management related events- kill/move/get requests etc.
5. The master thread processes the move event - it removes the session from the source pool and moves it to the destination pool. It then sets the move event future so that the waiting trigger validator thread is unblocked.

Proposed changes:

1. Create a hive config to turn on the delayed move configuration- *hive.server2.wm.delayed.move*.
2. Create another config for delayed move expiration- *hive.server2.wm.delayed.move.timeout*.
3. Add a new field *delayedMoveSessions* to Pool to keep track of all the delayed move sessions, so that the session can be moved to the destination pool when the pool gets incoming requests. Incoming requests to take priority over delayed move sessions.
4. Add a new field s*tartTime* to *MoveSession* to keep track of when the move was created.
5. Add a flag *isDelayedMove* to *wmTezSession.*
6. When a move trigger is violated, create a move session/event and notify the master thread only if the *wmTezSession* is not marked as “delayed move”. This is to avoid creating duplicate move events.
7. Change the *handleMoveSessionOnMasterThread()* method which is invoked by the master thread to process move events - If destination pool is full and the delayed move config is set to true, instead of killing the query, add the move to the *delayedMoveSessions* queue of the source pool and mark the *wmtezSession* as “delayed move”.
8. Add another step to the master thread loop after all the incoming requests have been obtained to decide which delayed moves can be processed now, and then to process the selected delayed moves. Processing a delayed move means to pop the move out of the *delayedMoveSessions* queue of the pool and process the delayed move as a normal move session by invoking *handleMoveSessionOnMasterThread()*. We use the following criteria to decide whether a delayed move should be processed:
    a. If any delayed move session has expired - a delayed move has been running in source pool for longer than *hive.server2.wm.delayed.move.timeout*.

   b. If the destination pool capacity has freed up.
   c. If a pool has incoming requests and doesn't have enough capacity to accommodate all the incoming requests.
9. Create a thread – *DelayedMoveThread* which wakes up at specified intervals (configurable using *hive.server2.wm.delayed.move.validator.interval*) and in turn wakes up the master thread so that the master thread can process the delayed moves which have expired and for which destination pools have freed up. This thread is needed because the master thread is event-driven and in case there are no events, a delayed move can run indefinitely in the source pool. This thread wakes up the master thread in the same way as other threads - by invoking the function *notifyWmThreadUnderLock(*).