package com.twitter.search.earlybird.partition;

import java.util.concurrent.locks.ReentrantLock;

import com.google.common.annotations.VisibleForTesting;

/\*\*

\* Lock used to ensure that flushing does not occur concurrently with the gc\_before\_optimization

\* and post\_optimization\_rebuilds actions - see where we call the "lock" method of this class.

\*

\* Both coordinated actions include a full GC in them, for reasons described in that part

\* of the code. After the GC, they wait until indexing has caught up before rejoining the serverset.

\*

\* If we flush concurrently with these actions, we can pause indexing for a while and waiting

\* until we're caught up can take some time, which can affect the memory state negatively.

\* For example, the first GC (before optimization) we do so that we have a clean state of memory

\* before optimization.

\*

\* The other reason we lock before executing the actions is because if we have flushing that's

\* currently running, once it finishes, we will rejoin the serverset and that can be followed by

\* a stop-the-world GC from the actions, which will affect our success rate.

\*/

public class OptimizationAndFlushingCoordinationLock {

private final ReentrantLock lock;

public OptimizationAndFlushingCoordinationLock() {

this.lock = new ReentrantLock();

}

public void lock() {

lock.lock();

}

public void unlock() {

lock.unlock();

}

public boolean tryLock() {

return lock.tryLock();

}

@VisibleForTesting

public boolean hasQueuedThreads() {

return lock.hasQueuedThreads();

}

}