package com.twitter.search.earlybird.common;

import java.util.concurrent.atomic.AtomicBoolean;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.search.common.metrics.SearchCustomGauge;

/\*\*

\* A monitor which enforces the condition that a single thread's work is caught up, and allows

\* other threads to wait to be notified when the work is complete. An AtomicBoolean ensures the

\* current status is visible to all threads.

\*/

public class CaughtUpMonitor {

private static final Logger LOG = LoggerFactory.getLogger(CaughtUpMonitor.class);

protected final AtomicBoolean isCaughtUp = new AtomicBoolean(false);

public CaughtUpMonitor(String statPrefix) {

SearchCustomGauge.export(statPrefix + "\_is\_caught\_up", () -> isCaughtUp() ? 1 : 0);

}

public boolean isCaughtUp() {

return isCaughtUp.get();

}

/\*\*

\* Set caught up state, and notify waiting threads if caught up.

\*/

public synchronized void setAndNotify(boolean caughtUp) {

isCaughtUp.set(caughtUp);

if (caughtUp) {

// Readers are caught up, notify waiting threads

notifyAll();

}

}

/\*\*

\* Wait using Object.wait() until caught up or until thread is interrupted.

\*/

public synchronized void resetAndWaitUntilCaughtUp() {

LOG.info("Waiting to catch up.");

// Explicitly set isCaughtUp to false before waiting

isCaughtUp.set(false);

try {

while (!isCaughtUp()) {

wait();

}

} catch (InterruptedException e) {

LOG.error("{} was interrupted while waiting to catch up", Thread.currentThread());

}

LOG.info("Caught up.");

}

}