package com.twitter.search.earlybird.partition;

import java.util.concurrent.ConcurrentLinkedDeque;

import java.util.concurrent.atomic.AtomicLong;

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

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

/\*\*

\* A queue with metrics on size, enqueue rate and dequeue rate.

\*/

public class InstrumentedQueue<T> {

private final SearchRateCounter enqueueRate;

private final SearchRateCounter dequeueRate;

private final AtomicLong queueSize = new AtomicLong();

private final ConcurrentLinkedDeque<T> queue;

public InstrumentedQueue(String statsPrefix) {

SearchLongGauge.export(statsPrefix + "\_size", queueSize);

enqueueRate = SearchRateCounter.export(statsPrefix + "\_enqueue");

dequeueRate = SearchRateCounter.export(statsPrefix + "\_dequeue");

queue = new ConcurrentLinkedDeque<>();

}

/\*\*

\* Adds a new element to the queue.

\*/

public void add(T tve) {

queue.add(tve);

enqueueRate.increment();

queueSize.incrementAndGet();

}

/\*\*

\* Returns the first element in the queue. If the queue is empty, {@code null} is returned.

\*/

public T poll() {

T tve = queue.poll();

if (tve != null) {

dequeueRate.increment();

queueSize.decrementAndGet();

}

return tve;

}

public long getQueueSize() {

return queueSize.get();

}

}