package com.twitter.search.ingester.pipeline.twitter;

import java.util.concurrent.BlockingQueue;

import java.util.concurrent.ExecutorService;

import javax.naming.NamingException;

import com.google.common.collect.Queues;

import org.apache.commons.pipeline.StageException;

import org.apache.commons.pipeline.validation.ConsumedTypes;

import org.apache.commons.pipeline.validation.ProducesConsumed;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

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

import com.twitter.search.common.relevance.entities.TwitterMessage;

import com.twitter.search.common.relevance.text.TweetParser;

import com.twitter.search.ingester.pipeline.util.PipelineStageRuntimeException;

@ConsumedTypes(TwitterMessage.class)

@ProducesConsumed

public class TextFeatureExtractionWorkersStage extends TwitterBaseStage

<TwitterMessage, TwitterMessage> {

private static final Logger LOG =

LoggerFactory.getLogger(TextFeatureExtractionWorkersStage.class);

private static final int NUM\_THREADS = 5;

private static final int MAX\_QUEUE\_SIZE = 100;

private static final long SLOW\_TWEET\_TIME\_MILLIS = 1000;

private ExecutorService executorService = null;

// define as static so that FeatureExtractorWorker thread can use it

private static SearchRateCounter slowTweetCounter;

private SearchRateCounter threadErrorCounter;

private SearchRateCounter threadInterruptionCounter;

private final BlockingQueue<TwitterMessage> messageQueue =

Queues.newLinkedBlockingQueue(MAX\_QUEUE\_SIZE);

private TweetParser tweetParser;

@Override

public void initStats() {

super.initStats();

innerSetupStats();

}

@Override

protected void innerSetupStats() {

slowTweetCounter = SearchRateCounter.export(

getStageNamePrefix() + "\_text\_feature\_extraction\_slow\_tweet\_count");

SearchCustomGauge.export(getStageNamePrefix() + "\_queue\_size",

messageQueue::size);

threadErrorCounter = SearchRateCounter.export(

getStageNamePrefix() + "\_text\_quality\_evaluation\_thread\_error");

threadInterruptionCounter = SearchRateCounter.export(

getStageNamePrefix() + "\_text\_quality\_evaluation\_thread\_interruption");

}

@Override

protected void doInnerPreprocess() throws StageException, NamingException {

innerSetup();

// anything threading related, we don't need in V2 as of yet.

executorService = wireModule.getThreadPool(NUM\_THREADS);

for (int i = 0; i < NUM\_THREADS; ++i) {

executorService.submit(new FeatureExtractorWorker());

}

LOG.info("Initialized {} parsers.", NUM\_THREADS);

}

@Override

protected void innerSetup() {

tweetParser = new TweetParser();

}

@Override

public void innerProcess(Object obj) throws StageException {

if (!(obj instanceof TwitterMessage)) {

LOG.error("Object is not a TwitterMessage object: {}", obj);

return;

}

TwitterMessage message = TwitterMessage.class.cast(obj);

try {

messageQueue.put(message);

} catch (InterruptedException ie) {

LOG.error("Interrupted exception adding to the queue", ie);

}

}

private boolean tryToParse(TwitterMessage message) {

boolean isAbleToParse = false;

long startTime = clock.nowMillis();

// Parse tweet and merge the parsed out features into what we already have in the message.

try {

synchronized (this) {

tweetParser.parseTweet(message, false, false);

}

// If parsing failed we don't need to pass the tweet down the pipeline.

isAbleToParse = true;

} catch (Exception e) {

threadErrorCounter.increment();

LOG.error("Uncaught exception from tweetParser.parseTweet()", e);

} finally {

long elapsedTime = clock.nowMillis() - startTime;

if (elapsedTime > SLOW\_TWEET\_TIME\_MILLIS) {

LOG.debug("Took {}ms to parse tweet {}: {}", elapsedTime, message.getId(), message);

slowTweetCounter.increment();

}

}

return isAbleToParse;

}

@Override

protected TwitterMessage innerRunStageV2(TwitterMessage message) {

if (!tryToParse(message)) {

throw new PipelineStageRuntimeException("Failed to parse, not passing to next stage.");

}

return message;

}

@Override

public void innerPostprocess() {

if (executorService != null) {

executorService.shutdownNow();

}

executorService = null;

}

private class FeatureExtractorWorker implements Runnable {

public void run() {

while (!Thread.currentThread().isInterrupted()) {

TwitterMessage message = null;

try {

message = messageQueue.take();

} catch (InterruptedException ie) {

threadInterruptionCounter.increment();

LOG.error("Interrupted exception polling from the queue", ie);

continue;

} finally {

if (tryToParse(message)) {

emitAndCount(message);

}

}

}

}

}

}