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

import java.util.List;

import java.util.concurrent.BlockingQueue;

import java.util.concurrent.ExecutorService;

import javax.naming.NamingException;

import com.google.common.collect.ImmutableList;

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.common\_internal.text.version.PenguinVersion;

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

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

import com.twitter.search.common.relevance.classifiers.TweetEvaluator;

import com.twitter.search.common.relevance.classifiers.TweetOffensiveEvaluator;

import com.twitter.search.common.relevance.classifiers.TweetTextClassifier;

import com.twitter.search.common.relevance.classifiers.TweetTextEvaluator;

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

import com.twitter.search.common.relevance.scorers.TweetTextScorer;

@ConsumedTypes(TwitterMessage.class)

@ProducesConsumed

public class TextQualityEvaluationWorkerStage extends TwitterBaseStage

<TwitterMessage, TwitterMessage> {

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

private static final int NUM\_THREADS = 5;

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

// based on the batched branch 3 elements in the queue times 200 tweets per batch.

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

private final BlockingQueue<TwitterMessage> messages =

Queues.newLinkedBlockingQueue(MAX\_QUEUE\_SIZE);

private static final String DO\_TEXT\_QUALITY\_EVALUATION\_DECIDER\_KEY\_TEMPLATE =

"ingester\_%s\_do\_text\_quality\_evaluation";

private ExecutorService executorService = null;

private SearchRateCounter unscoredTweetCounter;

private TweetTextClassifier classifier;

private final TweetTextScorer scorer = new TweetTextScorer(null);

// Defined as static so that ClassifierWorker thread can use it

private static SearchRateCounter slowTweetCounter;

private SearchRateCounter threadErrorCounter;

private SearchRateCounter threadInterruptionCounter;

private String deciderKey;

@Override

public void initStats() {

super.initStats();

innerSetupStats();

}

public SearchRateCounter getUnscoredTweetCounter() {

return unscoredTweetCounter;

}

@Override

protected void innerSetupStats() {

threadErrorCounter = SearchRateCounter.export(

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

threadInterruptionCounter = SearchRateCounter.export(

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

unscoredTweetCounter = SearchRateCounter.export(

getStageNamePrefix() + "\_text\_quality\_evaluation\_tweets\_unscored\_count");

slowTweetCounter = SearchRateCounter.export(

getStageNamePrefix() + "\_text\_quality\_evaluation\_slow\_tweet\_count");

SearchCustomGauge.export(getStageNamePrefix() + "\_queue\_size", messages::size);

}

@Override

protected void doInnerPreprocess() throws StageException, NamingException {

innerSetup();

executorService = wireModule.getThreadPool(NUM\_THREADS);

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

executorService.submit(

new ClassifierWorker());

}

LOG.info("Initialized {} classfiers and scorers.", NUM\_THREADS);

}

@Override

protected void innerSetup() throws NamingException {

deciderKey = String.format(DO\_TEXT\_QUALITY\_EVALUATION\_DECIDER\_KEY\_TEMPLATE,

earlybirdCluster.getNameForStats());

List<PenguinVersion> supportedPenguinVersions = wireModule.getPenguinVersions();

TweetOffensiveEvaluator tweetOffensiveEvaluator = wireModule.getTweetOffensiveEvaluator();

ImmutableList<TweetEvaluator> evaluators =

ImmutableList.of(tweetOffensiveEvaluator, new TweetTextEvaluator());

classifier = new TweetTextClassifier(

evaluators,

wireModule.getServiceIdentifier(),

supportedPenguinVersions);

}

@Override

public void innerProcess(Object obj) throws StageException {

if (!(obj instanceof TwitterMessage)) {

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

return;

}

if (decider.isAvailable(deciderKey)) {

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

try {

messages.put(message);

} catch (InterruptedException ie) {

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

}

} else {

unscoredTweetCounter.increment();

emitAndCount(obj);

}

}

@Override

protected TwitterMessage innerRunStageV2(TwitterMessage message) {

if (decider.isAvailable(deciderKey)) {

classifyAndScore(message);

} else {

unscoredTweetCounter.increment();

}

return message;

}

private void classifyAndScore(TwitterMessage message) {

long startTime = clock.nowMillis();

try {

// The tweet signature computed here might not be correct, since we did not resolve the

// tweet URLs yet. This is why BasicIndexingConverter does not set the tweet signature

// feature on the event it builds.

//

// We correct the tweet signature later in the ComputeTweetSignatureStage, and

// DelayedIndexingConverter sets this feature on the URL update event it creates.

synchronized (this) {

scorer.classifyAndScoreTweet(classifier, message);

}

} catch (Exception e) {

threadErrorCounter.increment();

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

} finally {

long elapsedTime = clock.nowMillis() - startTime;

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

LOG.warn("Took {}ms to classify and score tweet {}: {}",

elapsedTime, message.getId(), message);

slowTweetCounter.increment();

}

}

}

@Override

public void innerPostprocess() {

if (executorService != null) {

executorService.shutdownNow();

}

executorService = null;

}

private class ClassifierWorker implements Runnable {

public void run() {

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

TwitterMessage message;

try {

message = messages.take();

} catch (InterruptedException ie) {

threadInterruptionCounter.increment();

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

continue;

}

// We want to emit even if we couldn't score the tweet.

classifyAndScore(message);

emitAndCount(message);

}

}

}

}