package com.twitter.search.earlybird.partition.freshstartup;

import java.io.IOException;

import java.time.Duration;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Map;

import java.util.concurrent.TimeUnit;

import com.google.common.base.Stopwatch;

import com.google.common.collect.ImmutableList;

import org.apache.kafka.clients.consumer.ConsumerRecord;

import org.apache.kafka.clients.consumer.ConsumerRecords;

import org.apache.kafka.clients.consumer.KafkaConsumer;

import org.apache.kafka.common.TopicPartition;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.search.common.indexing.thriftjava.ThriftVersionedEvents;

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

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

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

import com.twitter.search.earlybird.factory.EarlybirdKafkaConsumersFactory;

import com.twitter.search.earlybird.partition.IndexingResultCounts;

/\*\*

\* Indexes updates for all segments after they have been optimized. Some of the updates have been

\* indexed before in the PreOptimizationSegmentIndexer, but the rest are indexed here.

\*/

class PostOptimizationUpdatesIndexer {

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

private static final String STAT\_PREFIX = "post\_optimization\_";

private static final String READ\_STAT\_PREFIX = STAT\_PREFIX + "read\_updates\_for\_segment\_";

private static final String APPLIED\_STAT\_PREFIX = STAT\_PREFIX + "applied\_updates\_for\_segment\_";

private final ArrayList<SegmentBuildInfo> segmentBuildInfos;

private final EarlybirdKafkaConsumersFactory earlybirdKafkaConsumersFactory;

private final TopicPartition updateTopic;

PostOptimizationUpdatesIndexer(

ArrayList<SegmentBuildInfo> segmentBuildInfos,

EarlybirdKafkaConsumersFactory earlybirdKafkaConsumersFactory,

TopicPartition updateTopic) {

this.segmentBuildInfos = segmentBuildInfos;

this.earlybirdKafkaConsumersFactory = earlybirdKafkaConsumersFactory;

this.updateTopic = updateTopic;

}

void indexRestOfUpdates() throws IOException {

LOG.info("Indexing rest of updates.");

long updatesStartOffset = segmentBuildInfos.get(0)

.getUpdateKafkaOffsetPair().getBeginOffset();

long updatesEndOffset = segmentBuildInfos.get(segmentBuildInfos.size() - 1)

.getUpdateKafkaOffsetPair().getEndOffset();

LOG.info(String.format("Total updates to go through: %,d",

updatesEndOffset - updatesStartOffset + 1));

KafkaConsumer<Long, ThriftVersionedEvents> kafkaConsumer =

earlybirdKafkaConsumersFactory.createKafkaConsumer("index\_rest\_of\_updates");

kafkaConsumer.assign(ImmutableList.of(updateTopic));

kafkaConsumer.seek(updateTopic, updatesStartOffset);

long readEvents = 0;

long foundSegment = 0;

long applied = 0;

Map<Integer, SearchRateCounter> perSegmentReadUpdates = new HashMap<>();

Map<Integer, SearchRateCounter> perSegmentAppliedUpdates = new HashMap<>();

Map<Integer, IndexingResultCounts> perSegmentIndexingResultCounts = new HashMap<>();

for (int i = 0; i < segmentBuildInfos.size(); i++) {

perSegmentReadUpdates.put(i, SearchRateCounter.export(READ\_STAT\_PREFIX + i));

perSegmentAppliedUpdates.put(i, SearchRateCounter.export(APPLIED\_STAT\_PREFIX + i));

perSegmentIndexingResultCounts.put(i, new IndexingResultCounts());

}

SearchTimerStats pollStats = SearchTimerStats.export(

"final\_pass\_polls", TimeUnit.NANOSECONDS, false);

SearchTimerStats indexStats = SearchTimerStats.export(

"final\_pass\_index", TimeUnit.NANOSECONDS, false);

Stopwatch totalTime = Stopwatch.createStarted();

boolean done = false;

do {

// Poll events.

SearchTimer pt = pollStats.startNewTimer();

ConsumerRecords<Long, ThriftVersionedEvents> records =

kafkaConsumer.poll(Duration.ofSeconds(1));

pollStats.stopTimerAndIncrement(pt);

// Index events.

SearchTimer it = indexStats.startNewTimer();

for (ConsumerRecord<Long, ThriftVersionedEvents> record : records) {

if (record.offset() >= updatesEndOffset) {

done = true;

}

readEvents++;

ThriftVersionedEvents tve = record.value();

long tweetId = tve.getId();

// Find segment to apply to. If we can't find a segment, this is an

// update for an old tweet that's not in the index.

int segmentIndex = -1;

for (int i = segmentBuildInfos.size() - 1; i >= 0; i--) {

if (segmentBuildInfos.get(i).getStartTweetId() <= tweetId) {

segmentIndex = i;

foundSegment++;

break;

}

}

if (segmentIndex != -1) {

SegmentBuildInfo segmentBuildInfo = segmentBuildInfos.get(segmentIndex);

perSegmentReadUpdates.get(segmentIndex).increment();

// Not already applied?

if (!segmentBuildInfo.getUpdateKafkaOffsetPair().includes(record.offset())) {

applied++;

// Index the update.

//

// IMPORTANT: Note that there you'll see about 2-3% of updates that

// fail as "retryable". This type of failure happens when the update is

// for a tweet that's not found in the index. We found out that we are

// receiving some updates for protected tweets and these are not in the

// realtime index - they are the source of this error.

perSegmentIndexingResultCounts.get(segmentIndex).countResult(

segmentBuildInfo.getSegmentWriter().indexThriftVersionedEvents(tve)

);

perSegmentAppliedUpdates.get(segmentIndex).increment();

}

}

if (record.offset() >= updatesEndOffset) {

break;

}

}

indexStats.stopTimerAndIncrement(it);

} while (!done);

LOG.info(String.format("Done in: %s, read %,d events, found segment for %,d, applied %,d",

totalTime, readEvents, foundSegment, applied));

LOG.info("Indexing time: {}", indexStats.getElapsedTimeAsString());

LOG.info("Polling time: {}", pollStats.getElapsedTimeAsString());

LOG.info("Per segment indexing result counts:");

for (int i = 0; i < segmentBuildInfos.size(); i++) {

LOG.info("{} : {}", i, perSegmentIndexingResultCounts.get(i));

}

LOG.info("Found and applied per segment:");

for (int i = 0; i < segmentBuildInfos.size(); i++) {

LOG.info("{}: found: {}, applied: {}",

i,

perSegmentReadUpdates.get(i).getCount(),

perSegmentAppliedUpdates.get(i).getCount());

}

}

}