package com.twitter.search.earlybird.partition;

import java.io.IOException;

import java.util.Iterator;

import scala.runtime.BoxedUnit;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import com.google.common.base.Stopwatch;

import com.google.common.base.Verify;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.search.common.config.Config;

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

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

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

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

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

import com.twitter.search.common.partitioning.snowflakeparser.SnowflakeIdParser;

import com.twitter.search.common.util.GCUtil;

import com.twitter.search.earlybird.EarlybirdStatus;

import com.twitter.search.earlybird.common.CaughtUpMonitor;

import com.twitter.search.earlybird.exception.CriticalExceptionHandler;

import com.twitter.search.earlybird.index.OutOfOrderRealtimeTweetIDMapper;

import com.twitter.search.earlybird.querycache.QueryCacheManager;

import com.twitter.search.earlybird.util.CoordinatedEarlybirdActionInterface;

import com.twitter.util.Await;

import com.twitter.util.Duration;

import com.twitter.util.Future;

import com.twitter.util.TimeoutException;

/\*\*

\* This class handles incoming new Tweets. It is responsible for creating segments for the incoming

\* Tweets when necessary, triggering optimization on those segments, and writing Tweets to the

\* correct segment.

\*/

public class TweetCreateHandler {

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

public static final long LATE\_TWEET\_TIME\_BUFFER\_MS = Duration.fromMinutes(1).inMilliseconds();

private static final String STATS\_PREFIX = "tweet\_create\_handler\_";

// To get a better idea of which of these succeeded and so on, see stats in SegmentManager.

private IndexingResultCounts indexingResultCounts;

private static final SearchRateCounter TWEETS\_IN\_WRONG\_SEGMENT =

SearchRateCounter.export(STATS\_PREFIX + "tweets\_in\_wrong\_segment");

private static final SearchRateCounter SEGMENTS\_CLOSED\_EARLY =

SearchRateCounter.export(STATS\_PREFIX + "segments\_closed\_early");

private static final SearchRateCounter INSERTED\_IN\_CURRENT\_SEGMENT =

SearchRateCounter.export(STATS\_PREFIX + "inserted\_in\_current\_segment");

private static final SearchRateCounter INSERTED\_IN\_PREVIOUS\_SEGMENT =

SearchRateCounter.export(STATS\_PREFIX + "inserted\_in\_previous\_segment");

private static final NewSegmentStats NEW\_SEGMENT\_STATS = new NewSegmentStats();

private static final SearchCounter CREATED\_SEGMENTS =

SearchCounter.export(STATS\_PREFIX + "created\_segments");

private static final SearchRateCounter INCOMING\_TWEETS =

SearchRateCounter.export(STATS\_PREFIX + "incoming\_tweets");

private static final SearchRateCounter INDEXING\_SUCCESS =

SearchRateCounter.export(STATS\_PREFIX + "indexing\_success");

private static final SearchRateCounter INDEXING\_FAILURE =

SearchRateCounter.export(STATS\_PREFIX + "indexing\_failure");

// Various stats and logging around creation of new segments, put in this

// class so that the code is not watered down too much by this.

private static class NewSegmentStats {

private static final String NEW\_SEGMENT\_STATS\_PREFIX =

STATS\_PREFIX + "new\_segment\_";

private static final SearchCounter START\_NEW\_AFTER\_REACHING\_LIMIT =

SearchCounter.export(NEW\_SEGMENT\_STATS\_PREFIX + "start\_after\_reaching\_limit");

private static final SearchCounter START\_NEW\_AFTER\_EXCEEDING\_MAX\_ID =

SearchCounter.export(NEW\_SEGMENT\_STATS\_PREFIX + "start\_after\_exceeding\_max\_id");

private static final SearchCounter TIMESLICE\_SET\_TO\_CURRENT\_ID =

SearchCounter.export(NEW\_SEGMENT\_STATS\_PREFIX + "timeslice\_set\_to\_current\_id");

private static final SearchCounter TIMESLICE\_SET\_TO\_MAX\_ID =

SearchCounter.export(NEW\_SEGMENT\_STATS\_PREFIX + "timeslice\_set\_to\_max\_id");

private static final SearchLongGauge TIMESPAN\_BETWEEN\_MAX\_AND\_CURRENT =

SearchLongGauge.export(NEW\_SEGMENT\_STATS\_PREFIX + "timespan\_between\_id\_and\_max");

void recordCreateNewSegment() {

CREATED\_SEGMENTS.increment();

}

void recordStartAfterReachingTweetsLimit(int numDocs, int numDocsCutoff,

int maxSegmentSize, int lateTweetBuffer) {

START\_NEW\_AFTER\_REACHING\_LIMIT.increment();

LOG.info(String.format(

"Will create new segment: numDocs=%,d, numDocsCutoff=%,d"

+ " | maxSegmentSize=%,d, lateTweetBuffer=%,d",

numDocs, numDocsCutoff, maxSegmentSize, lateTweetBuffer));

}

void recordStartAfterExceedingLargestValidTweetId(long tweetId, long largestValidTweetId) {

START\_NEW\_AFTER\_EXCEEDING\_MAX\_ID.increment();

LOG.info(String.format(

"Will create new segment: tweetDd=%,d, largestValidTweetID for segment=%,d",

tweetId, largestValidTweetId));

}

void recordSettingTimesliceToCurrentTweet(long tweetID) {

TIMESLICE\_SET\_TO\_CURRENT\_ID.increment();

LOG.info("Creating new segment: tweet that triggered it has the largest id we've seen. "

+ " id={}", tweetID);

}

void recordSettingTimesliceToMaxTweetId(long tweetID, long maxTweetID) {

TIMESLICE\_SET\_TO\_MAX\_ID.increment();

LOG.info("Creating new segment: tweet that triggered it doesn't have the largest id"

+ " we've seen. tweetId={}, maxTweetId={}",

tweetID, maxTweetID);

long timeDifference =

SnowflakeIdParser.getTimeDifferenceBetweenTweetIDs(maxTweetID, tweetID);

LOG.info("Time difference between max seen and last seen: {} ms", timeDifference);

TIMESPAN\_BETWEEN\_MAX\_AND\_CURRENT.set(timeDifference);

}

void wrapNewSegmentCreation(long tweetID, long maxTweetID,

long currentSegmentTimesliceBoundary,

long largestValidTweetIDForCurrentSegment) {

long timeDifferenceStartToMax = SnowflakeIdParser.getTimeDifferenceBetweenTweetIDs(

largestValidTweetIDForCurrentSegment,

currentSegmentTimesliceBoundary);

LOG.info("Time between timeslice boundary and largest valid tweet id: {} ms",

timeDifferenceStartToMax);

LOG.info("Created new segment: (tweetId={}, maxTweetId={}, maxTweetId-tweetId={} "

+ " | currentSegmentTimesliceBoundary={}, largestValidTweetIDForSegment={})",

tweetID, maxTweetID, maxTweetID - tweetID, currentSegmentTimesliceBoundary,

largestValidTweetIDForCurrentSegment);

}

}

private final SegmentManager segmentManager;

private final MultiSegmentTermDictionaryManager multiSegmentTermDictionaryManager;

private final int maxSegmentSize;

private final int lateTweetBuffer;

private long maxTweetID = Long.MIN\_VALUE;

private long largestValidTweetIDForCurrentSegment;

private long currentSegmentTimesliceBoundary;

private OptimizingSegmentWriter currentSegment;

private OptimizingSegmentWriter previousSegment;

private final QueryCacheManager queryCacheManager;

private final CriticalExceptionHandler criticalExceptionHandler;

private final SearchIndexingMetricSet searchIndexingMetricSet;

private final CoordinatedEarlybirdActionInterface postOptimizationRebuildsAction;

private final CoordinatedEarlybirdActionInterface gcAction;

private final CaughtUpMonitor indexCaughtUpMonitor;

private final OptimizationAndFlushingCoordinationLock optimizationAndFlushingCoordinationLock;

public TweetCreateHandler(

SegmentManager segmentManager,

SearchIndexingMetricSet searchIndexingMetricSet,

CriticalExceptionHandler criticalExceptionHandler,

MultiSegmentTermDictionaryManager multiSegmentTermDictionaryManager,

QueryCacheManager queryCacheManager,

CoordinatedEarlybirdActionInterface postOptimizationRebuildsAction,

CoordinatedEarlybirdActionInterface gcAction,

int lateTweetBuffer,

int maxSegmentSize,

CaughtUpMonitor indexCaughtUpMonitor,

OptimizationAndFlushingCoordinationLock optimizationAndFlushingCoordinationLock

) {

this.segmentManager = segmentManager;

this.criticalExceptionHandler = criticalExceptionHandler;

this.multiSegmentTermDictionaryManager = multiSegmentTermDictionaryManager;

this.queryCacheManager = queryCacheManager;

this.indexingResultCounts = new IndexingResultCounts();

this.searchIndexingMetricSet = searchIndexingMetricSet;

this.postOptimizationRebuildsAction = postOptimizationRebuildsAction;

this.gcAction = gcAction;

this.indexCaughtUpMonitor = indexCaughtUpMonitor;

Preconditions.checkState(lateTweetBuffer < maxSegmentSize);

this.lateTweetBuffer = lateTweetBuffer;

this.maxSegmentSize = maxSegmentSize;

this.optimizationAndFlushingCoordinationLock = optimizationAndFlushingCoordinationLock;

}

void prepareAfterStartingWithIndex(long maxIndexedTweetId) {

LOG.info("Preparing after starting with an index.");

Iterator<SegmentInfo> segmentInfosIterator =

segmentManager

.getSegmentInfos(SegmentManager.Filter.All, SegmentManager.Order.NEW\_TO\_OLD)

.iterator();

// Setup the last segment.

Verify.verify(segmentInfosIterator.hasNext(), "at least one segment expected");

ISegmentWriter lastWriter = segmentManager.getSegmentWriterForID(

segmentInfosIterator.next().getTimeSliceID());

Verify.verify(lastWriter != null);

LOG.info("TweetCreateHandler found last writer: {}", lastWriter.getSegmentInfo().toString());

this.currentSegmentTimesliceBoundary = lastWriter.getSegmentInfo().getTimeSliceID();

this.largestValidTweetIDForCurrentSegment =

OutOfOrderRealtimeTweetIDMapper.calculateMaxTweetID(currentSegmentTimesliceBoundary);

this.currentSegment = (OptimizingSegmentWriter) lastWriter;

if (maxIndexedTweetId == -1) {

maxTweetID = lastWriter.getSegmentInfo().getIndexSegment().getMaxTweetId();

LOG.info("Max tweet id = {}", maxTweetID);

} else {

// See SEARCH-31032

maxTweetID = maxIndexedTweetId;

}

// If we have a previous segment that's not optimized, set it up too, we still need to pick

// it up for optimization and we might still be able to add tweets to it.

if (segmentInfosIterator.hasNext()) {

SegmentInfo previousSegmentInfo = segmentInfosIterator.next();

if (!previousSegmentInfo.isOptimized()) {

ISegmentWriter previousSegmentWriter = segmentManager.getSegmentWriterForID(

previousSegmentInfo.getTimeSliceID());

if (previousSegmentWriter != null) {

LOG.info("Picked previous segment");

this.previousSegment = (OptimizingSegmentWriter) previousSegmentWriter;

} else {

// Should not happen.

LOG.error("Not found previous segment writer");

}

} else {

LOG.info("Previous segment info is optimized");

}

} else {

LOG.info("Previous segment info not found, we only have one segment");

}

}

private void updateIndexFreshness() {

searchIndexingMetricSet.highestStatusId.set(maxTweetID);

long tweetTimestamp = SnowflakeIdParser.getTimestampFromTweetId(

searchIndexingMetricSet.highestStatusId.get());

searchIndexingMetricSet.freshestTweetTimeMillis.set(tweetTimestamp);

}

/\*\*

\* Index a new TVE representing a Tweet create event.

\*/

public void handleTweetCreate(ThriftVersionedEvents tve) throws IOException {

INCOMING\_TWEETS.increment();

long id = tve.getId();

maxTweetID = Math.max(id, maxTweetID);

updateIndexFreshness();

boolean shouldCreateNewSegment = false;

if (currentSegment == null) {

shouldCreateNewSegment = true;

LOG.info("Will create new segment: current segment is null");

} else {

int numDocs = currentSegment.getSegmentInfo().getIndexSegment().getNumDocs();

int numDocsCutoff = maxSegmentSize - lateTweetBuffer;

if (numDocs >= numDocsCutoff) {

NEW\_SEGMENT\_STATS.recordStartAfterReachingTweetsLimit(numDocs, numDocsCutoff,

maxSegmentSize, lateTweetBuffer);

shouldCreateNewSegment = true;

} else if (id > largestValidTweetIDForCurrentSegment) {

NEW\_SEGMENT\_STATS.recordStartAfterExceedingLargestValidTweetId(id,

largestValidTweetIDForCurrentSegment);

shouldCreateNewSegment = true;

}

}

if (shouldCreateNewSegment) {

createNewSegment(id);

}

if (previousSegment != null) {

// Inserts and some updates can't be applied to an optimized segment, so we want to wait at

// least LATE\_TWEET\_TIME\_BUFFER between when we created the new segment and when we optimize

// the previous segment, in case there are late tweets.

// We leave a large (150k, typically) buffer in the segment so that we don't have to close

// the previousSegment before LATE\_TWEET\_TIME\_BUFFER has passed, but if we index

// lateTweetBuffer Tweets before optimizing, then we must optimize,

// so that we don't insert more than max segment size tweets into the previous segment.

long relativeTweetAgeMs =

SnowflakeIdParser.getTimeDifferenceBetweenTweetIDs(id, currentSegmentTimesliceBoundary);

boolean needToOptimize = false;

int numDocs = previousSegment.getSegmentInfo().getIndexSegment().getNumDocs();

String previousSegmentName = previousSegment.getSegmentInfo().getSegmentName();

if (numDocs >= maxSegmentSize) {

LOG.info(String.format("Previous segment (%s) reached maxSegmentSize, need to optimize it."

+ " numDocs=%,d, maxSegmentSize=%,d", previousSegmentName, numDocs, maxSegmentSize));

needToOptimize = true;

} else if (relativeTweetAgeMs > LATE\_TWEET\_TIME\_BUFFER\_MS) {

LOG.info(String.format("Previous segment (%s) is old enough, we can optimize it."

+ " Got tweet past time buffer of %,d ms by: %,d ms", previousSegmentName,

LATE\_TWEET\_TIME\_BUFFER\_MS, relativeTweetAgeMs - LATE\_TWEET\_TIME\_BUFFER\_MS));

needToOptimize = true;

}

if (needToOptimize) {

optimizePreviousSegment();

}

}

ISegmentWriter segmentWriter;

if (id >= currentSegmentTimesliceBoundary) {

INSERTED\_IN\_CURRENT\_SEGMENT.increment();

segmentWriter = currentSegment;

} else if (previousSegment != null) {

INSERTED\_IN\_PREVIOUS\_SEGMENT.increment();

segmentWriter = previousSegment;

} else {

TWEETS\_IN\_WRONG\_SEGMENT.increment();

LOG.info("Inserting TVE ({}) into the current segment ({}) even though it should have gone "

+ "in a previous segment.", id, currentSegmentTimesliceBoundary);

segmentWriter = currentSegment;

}

SearchTimer timer = searchIndexingMetricSet.statusStats.startNewTimer();

ISegmentWriter.Result result = segmentWriter.indexThriftVersionedEvents(tve);

searchIndexingMetricSet.statusStats.stopTimerAndIncrement(timer);

if (result == ISegmentWriter.Result.SUCCESS) {

INDEXING\_SUCCESS.increment();

} else {

INDEXING\_FAILURE.increment();

}

indexingResultCounts.countResult(result);

}

/\*\*

\* Many tests need to verify behavior with segments optimized & unoptimized, so we need to expose

\* this.

\*/

@VisibleForTesting

public Future<SegmentInfo> optimizePreviousSegment() {

String segmentName = previousSegment.getSegmentInfo().getSegmentName();

previousSegment.getSegmentInfo().setIndexing(false);

LOG.info("Optimizing previous segment: {}", segmentName);

segmentManager.logState("Starting optimization for segment: " + segmentName);

Future<SegmentInfo> future = previousSegment

.startOptimization(gcAction, optimizationAndFlushingCoordinationLock)

.map(this::postOptimizationSteps)

.onFailure(t -> {

criticalExceptionHandler.handle(this, t);

return BoxedUnit.UNIT;

});

waitForOptimizationIfInTest(future);

previousSegment = null;

return future;

}

/\*\*

\* In tests, it's easier if when a segment starts optimizing, we know that it will finish

\* optimizing. This way we have no race condition where we're surprised that something that

\* started optimizing is not ready.

\*

\* In prod we don't have this problem. Segments run for 10 hours and optimization is 20 minutes

\* so there's no need for extra synchronization.

\*/

private void waitForOptimizationIfInTest(Future<SegmentInfo> future) {

if (Config.environmentIsTest()) {

try {

Await.ready(future);

LOG.info("Optimizing is done");

} catch (InterruptedException | TimeoutException ex) {

LOG.info("Exception while optimizing", ex);

}

}

}

private SegmentInfo postOptimizationSteps(SegmentInfo optimizedSegmentInfo) {

segmentManager.updateStats();

// See SEARCH-32175

optimizedSegmentInfo.setComplete(true);

String segmentName = optimizedSegmentInfo.getSegmentName();

LOG.info("Finished optimization for segment: " + segmentName);

segmentManager.logState(

"Finished optimization for segment: " + segmentName);

/\*

\* Building the multi segment term dictionary causes GC pauses. The reason for this is because

\* it's pretty big (possible ~15GB). When it's allocated, we have to copy a lot of data from

\* survivor space to old gen. That causes several GC pauses. See SEARCH-33544

\*

\* GC pauses are in general not fatal, but since all instances finish a segment at roughly the

\* same time, they might happen at the same time and then it's a problem.

\*

\* Some possible solutions to this problem would be to build this dictionary in some data

\* structures that are pre-allocated or to build only the part for the last segment, as

\* everything else doesn't change. These solutions are a bit difficult to implement and this

\* here is an easy workaround.

\*

\* Note that we might finish optimizing a segment and then it might take ~60+ minutes until it's

\* a particular Earlybird's turn to run this code. The effect of this is going to be that we

\* are not going to use the multi segment dictionary for the last two segments, one of which is

\* still pretty small. That's not terrible, since right before optimization we're not using

\* the dictionary for the last segment anyways, since it's still not optimized.

\*/

try {

LOG.info("Acquire coordination lock before beginning post\_optimization\_rebuilds action.");

optimizationAndFlushingCoordinationLock.lock();

LOG.info("Successfully acquired coordination lock for post\_optimization\_rebuilds action.");

postOptimizationRebuildsAction.retryActionUntilRan(

"post optimization rebuilds", () -> {

Stopwatch stopwatch = Stopwatch.createStarted();

LOG.info("Starting to build multi term dictionary for {}", segmentName);

boolean result = multiSegmentTermDictionaryManager.buildDictionary();

LOG.info("Done building multi term dictionary for {} in {}, result: {}",

segmentName, stopwatch, result);

queryCacheManager.rebuildQueryCachesAfterSegmentOptimization(

optimizedSegmentInfo);

// This is a serial full GC and it defragments the memory so things can run smoothly

// until the next segment rolls. What we have observed is that if we don't do that

// later on some earlybirds can have promotion failures on an old gen that hasn't

// reached the initiating occupancy limit and these promotions failures can trigger a

// long (1.5 min) full GC. That usually happens because of fragmentation issues.

GCUtil.runGC();

// Wait for indexing to catch up before rejoining the serverset. We only need to do

// this if the host has already finished startup.

if (EarlybirdStatus.hasStarted()) {

indexCaughtUpMonitor.resetAndWaitUntilCaughtUp();

}

});

} finally {

LOG.info("Finished post\_optimization\_rebuilds action. Releasing coordination lock.");

optimizationAndFlushingCoordinationLock.unlock();

}

return optimizedSegmentInfo;

}

/\*\*

\* Many tests rely on precise segment boundaries, so we expose this to allow them to create a

\* particular segment.

\*/

@VisibleForTesting

public void createNewSegment(long tweetID) throws IOException {

NEW\_SEGMENT\_STATS.recordCreateNewSegment();

if (previousSegment != null) {

// We shouldn't have more than one unoptimized segment, so if we get to this point and the

// previousSegment has not been optimized and set to null, start optimizing it before

// creating the next one. Note that this is a weird case and would only happen if we get

// Tweets with drastically different IDs than we expect, or there is a large amount of time

// where no Tweets are created in this partition.

LOG.error("Creating new segment for Tweet {} when the previous segment {} was not sealed. "

+ "Current segment: {}. Documents: {}. largestValidTweetIDForSegment: {}.",

tweetID,

previousSegment.getSegmentInfo().getTimeSliceID(),

currentSegment.getSegmentInfo().getTimeSliceID(),

currentSegment.getSegmentInfo().getIndexSegment().getNumDocs(),

largestValidTweetIDForCurrentSegment);

optimizePreviousSegment();

SEGMENTS\_CLOSED\_EARLY.increment();

}

previousSegment = currentSegment;

// We have two cases:

//

// Case 1:

// If the greatest Tweet ID we have seen is tweetID, then when we want to create a new segment

// with that ID, so the Tweet being processed goes into the new segment.

//

// Case 2:

// If the tweetID is bigger than the max tweetID, then this method is being called directly from

// tests, so we didn't update the maxTweetID, so we can create a new segment with the new

// Tweet ID.

//

// Case 3:

// If it's not the greatest Tweet ID we have seen, then we don't want to create a

// segment boundary that is lower than any Tweet IDs in the current segment, because then

// some tweets from the previous segment would be in the wrong segment, so create a segment

// that has a greater ID than any Tweets that we have seen.

//

// Example:

// - We have seen tweets 3, 10, 5, 6.

// - We now see tweet 7 and we decide it's time to create a new segment.

// - The new segment will start at tweet 11. It can't start at tweet 7, because

// tweet 10 will be in the wrong segment.

// - Tweet 7 that we just saw will end up in the previous segment.

if (maxTweetID <= tweetID) {

currentSegmentTimesliceBoundary = tweetID;

NEW\_SEGMENT\_STATS.recordSettingTimesliceToCurrentTweet(tweetID);

} else {

currentSegmentTimesliceBoundary = maxTweetID + 1;

NEW\_SEGMENT\_STATS.recordSettingTimesliceToMaxTweetId(tweetID, maxTweetID);

}

currentSegment = segmentManager.createAndPutOptimizingSegmentWriter(

currentSegmentTimesliceBoundary);

currentSegment.getSegmentInfo().setIndexing(true);

largestValidTweetIDForCurrentSegment =

OutOfOrderRealtimeTweetIDMapper.calculateMaxTweetID(currentSegmentTimesliceBoundary);

NEW\_SEGMENT\_STATS.wrapNewSegmentCreation(tweetID, maxTweetID,

currentSegmentTimesliceBoundary, largestValidTweetIDForCurrentSegment);

segmentManager.removeExcessSegments();

}

void logState() {

LOG.info("TweetCreateHandler:");

LOG.info(String.format(" tweets sent for indexing: %,d",

indexingResultCounts.getIndexingCalls()));

LOG.info(String.format(" non-retriable failure: %,d",

indexingResultCounts.getFailureNotRetriable()));

LOG.info(String.format(" retriable failure: %,d",

indexingResultCounts.getFailureRetriable()));

LOG.info(String.format(" successfully indexed: %,d",

indexingResultCounts.getIndexingSuccess()));

LOG.info(String.format(" tweets in wrong segment: %,d", TWEETS\_IN\_WRONG\_SEGMENT.getCount()));

LOG.info(String.format(" segments closed early: %,d", SEGMENTS\_CLOSED\_EARLY.getCount()));

}

}