package com.twitter.search.earlybird.util;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.Date;

import java.util.List;

import java.util.Map;

import java.util.concurrent.TimeUnit;

import java.util.concurrent.atomic.AtomicInteger;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.collect.Maps;

import org.apache.commons.lang.mutable.MutableInt;

import org.apache.commons.lang.mutable.MutableLong;

import org.apache.lucene.index.IndexOptions;

import org.apache.lucene.index.PostingsEnum;

import org.apache.lucene.index.Terms;

import org.apache.lucene.index.TermsEnum;

import org.apache.lucene.search.DocIdSetIterator;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.collections.Pair;

import com.twitter.search.common.concurrent.ScheduledExecutorServiceFactory;

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

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

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

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

import com.twitter.search.common.partitioning.base.Segment;

import com.twitter.search.common.schema.base.ImmutableSchemaInterface;

import com.twitter.search.common.schema.base.Schema;

import com.twitter.search.core.earlybird.index.DocIDToTweetIDMapper;

import com.twitter.search.core.earlybird.index.EarlybirdIndexSegmentAtomicReader;

import com.twitter.search.core.earlybird.index.TimeMapper;

import com.twitter.search.earlybird.common.config.EarlybirdConfig;

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

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

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

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

/\*\*

\* A background task that periodically gets and exports the number of tweets per hour that are

\* indexed on this earlybird.

\* Specifically used for making sure that we are not missing data for any hours in the search

\* archives.

\* The task loops though all the segments that are indexed by this earlybird, and for each segment

\* looks at all the createdAt dates for all of the documents in that segment.

\*

\* Also keeps track off an exposes as a stat the number of hours that do not have any tweets in the

\* min/max range of data that IS indexed on this earlybird. i.e if we only have data for

\* 2006/01/01:02 and 2006/01/01:04, it will consider 2006/01/01:03 as a missing hour.

\* Hours before 2006/01/01:02 or after 2006/01/01:04 will not be considered as missing.

\*/

public class TweetCountMonitor extends OneTaskScheduledExecutorManager {

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

private static final String THREAD\_NAME\_FORMAT = "TweetCountMonitor-%d";

private static final boolean THREAD\_IS\_DAEMON = true;

public static final String RUN\_INTERVAL\_MINUTES\_CONFIG\_NAME =

"tweet\_count\_monitor\_run\_interval\_minutes";

public static final String START\_CHECK\_HOUR\_CONFIG\_NAME =

"tweet\_count\_monitor\_start\_check\_hour";

public static final String HOURLY\_MIN\_COUNT\_CONFIG\_NAME =

"tweet\_count\_monitor\_hourly\_min\_count";

public static final String DAILY\_MIN\_COUNT\_CONFIG\_NAME =

"tweet\_count\_monitor\_daily\_min\_count";

@VisibleForTesting

public static final AtomicInteger INSTANCE\_COUNTER = new AtomicInteger(0);

private static final long MILLIS\_IN\_A\_DAY = TimeUnit.DAYS.toMillis(1);

private final SegmentManager segmentManager;

private final SearchStatsReceiver searchStatsReceiver;

private final int instanceCounter;

// The first date in format "YYYYMMDDHH" that we want to check counts for.

private final int startCheckHour;

// The last date in format "YYYYMMDDHH" that we want to check counts for.

private final int endCheckHour;

//Smallest number of docs we expect to have for each day.

private final int dailyMinCount;

// Smallest number of docs we expect to have for each hour.

private final int hourlyMinCount;

// Binary stat, set to 0 when the monitor is running

private final SearchLongGauge isRunningStat;

// How long each iteration takes

private final SearchTimerStats checkTimeStat;

private final Map<String, FieldTermCounter> fieldTermCounters;

private final Map<String, SearchTimerStats> fieldCheckTimeStats;

/\*\*

\* Create a TweetCountMonitor to monitor all segments in the given segmentManager

\*/

public TweetCountMonitor(

SegmentManager segmentManager,

ScheduledExecutorServiceFactory executorServiceFactory,

long shutdownWaitDuration,

TimeUnit shutdownWaitUnit,

SearchStatsReceiver searchStatsReceiver,

CriticalExceptionHandler criticalExceptionHandler) {

this(segmentManager,

EarlybirdConfig.getInt(START\_CHECK\_HOUR\_CONFIG\_NAME, 0),

EarlybirdConfig.getInt(RUN\_INTERVAL\_MINUTES\_CONFIG\_NAME, -1),

EarlybirdConfig.getInt(HOURLY\_MIN\_COUNT\_CONFIG\_NAME, 0),

EarlybirdConfig.getInt(DAILY\_MIN\_COUNT\_CONFIG\_NAME, 0),

executorServiceFactory,

shutdownWaitDuration,

shutdownWaitUnit,

searchStatsReceiver,

criticalExceptionHandler);

}

@VisibleForTesting

TweetCountMonitor(

SegmentManager segmentManager,

int startCheckHourFromConfig,

int schedulePeriodMinutes,

int hourlyMinCount,

int dailyMinCount,

ScheduledExecutorServiceFactory executorServiceFactory,

long shutdownWaitDuration,

TimeUnit shutdownWaitUnit,

SearchStatsReceiver searchStatsReceiver,

CriticalExceptionHandler criticalExceptionHandler) {

super(

executorServiceFactory,

THREAD\_NAME\_FORMAT,

THREAD\_IS\_DAEMON,

PeriodicActionParams.atFixedRate(

schedulePeriodMinutes,

TimeUnit.MINUTES

),

new ShutdownWaitTimeParams(

shutdownWaitDuration,

shutdownWaitUnit

),

searchStatsReceiver,

criticalExceptionHandler);

this.segmentManager = segmentManager;

this.searchStatsReceiver = searchStatsReceiver;

this.instanceCounter = INSTANCE\_COUNTER.incrementAndGet();

this.hourlyMinCount = hourlyMinCount;

this.dailyMinCount = dailyMinCount;

String isRunningStatName = "tweet\_count\_monitor\_is\_running\_v\_" + this.instanceCounter;

this.isRunningStat = SearchLongGauge.export(isRunningStatName);

String checkTimeStatName = "tweet\_count\_monitor\_check\_time\_v\_" + this.instanceCounter;

this.checkTimeStat = SearchTimerStats.export(checkTimeStatName, TimeUnit.MILLISECONDS, true);

this.startCheckHour = Math.max(

startCheckHourFromConfig,

dateToHourValue(segmentManager.getPartitionConfig().getTierStartDate()));

this.endCheckHour = dateToHourValue(segmentManager.getPartitionConfig().getTierEndDate());

this.fieldTermCounters = Maps.newHashMap();

this.fieldTermCounters.put(

FieldTermCounter.TWEET\_COUNT\_KEY,

new FieldTermCounter(

FieldTermCounter.TWEET\_COUNT\_KEY,

instanceCounter,

startCheckHour,

endCheckHour,

hourlyMinCount,

dailyMinCount));

this.fieldCheckTimeStats = Maps.newHashMap();

}

private int dateToHourValue(Date date) {

Calendar cal = Calendar.getInstance(FieldTermCounter.TIME\_ZONE);

cal.setTime(date);

return FieldTermCounter.getHourValue(cal);

}

private void updateHourlyCounts() {

// Iterate the current index to count all tweets anf field hits.

Map<String, Map<Integer, MutableInt>> newCountMap = getNewTweetCountMap();

for (Map.Entry<String, Map<Integer, MutableInt>> newCounts : newCountMap.entrySet()) {

final String fieldName = newCounts.getKey();

FieldTermCounter termCounter = fieldTermCounters.get(fieldName);

if (termCounter == null) {

termCounter = new FieldTermCounter(

fieldName,

instanceCounter,

startCheckHour,

endCheckHour,

hourlyMinCount,

dailyMinCount);

fieldTermCounters.put(fieldName, termCounter);

}

termCounter.runWithNewCounts(newCounts.getValue());

}

}

/\*\*

\* Loops through all segments, and all documents in each segment, and for each document

\* gets the createdAt timestamp (in seconds) from the TimeMapper.

\* Based on that, returns a map with the count of:

\* . the number of tweets for each hour

\* . the number of tweets corresponding to each field for each hour

\*/

private Map<String, Map<Integer, MutableInt>> getNewTweetCountMap() {

Iterable<SegmentInfo> segmentInfos = segmentManager.getSegmentInfos(

SegmentManager.Filter.Enabled, SegmentManager.Order.NEW\_TO\_OLD);

Map<String, Map<Integer, MutableInt>> newCountMap = Maps.newHashMap();

Map<Integer, MutableInt> newCounts = Maps.newHashMap();

newCountMap.put(FieldTermCounter.TWEET\_COUNT\_KEY, newCounts);

ImmutableSchemaInterface schemaSnapshot =

segmentManager.getEarlybirdIndexConfig().getSchema().getSchemaSnapshot();

Calendar cal = Calendar.getInstance(FieldTermCounter.TIME\_ZONE);

for (SegmentInfo segmentInfo : segmentInfos) {

try {

EarlybirdSingleSegmentSearcher searcher = segmentManager.getSearcher(

segmentInfo.getTimeSliceID(), schemaSnapshot);

if (searcher != null) {

EarlybirdIndexSegmentAtomicReader reader = searcher.getTwitterIndexReader();

TimeMapper timeMapper = reader.getSegmentData().getTimeMapper();

List<Pair<String, Integer>> outsideEndDateRangeDocList = new ArrayList<>();

// Get the number of tweets for each hour.

int docsOutsideEndDateRange = getNewTweetCountsForSegment(

segmentInfo, reader, timeMapper, cal, newCounts);

if (docsOutsideEndDateRange > 0) {

outsideEndDateRangeDocList.add(new Pair<>(

FieldTermCounter.TWEET\_COUNT\_KEY, docsOutsideEndDateRange));

}

// Get the number of tweets with corresponding field for each hour.

for (Schema.FieldInfo fieldInfo : schemaSnapshot.getFieldInfos()) {

if (fieldInfo.getFieldType().indexOptions() == IndexOptions.NONE) {

continue;

}

String fieldName = fieldInfo.getName();

docsOutsideEndDateRange = getNewFieldTweetCountsForSegment(

segmentInfo, reader, timeMapper, cal, fieldName, newCountMap);

if (docsOutsideEndDateRange > 0) {

outsideEndDateRangeDocList.add(new Pair<>(fieldName, docsOutsideEndDateRange));

}

}

LOG.info("Inspected segment: " + segmentInfo + " found "

+ outsideEndDateRangeDocList.size()

+ " fields with documents outside of segment end date.");

for (Pair<String, Integer> outsideEndRange : outsideEndDateRangeDocList) {

LOG.info(" outside end date range - segment: " + segmentInfo.getSegmentName()

+ " field: " + outsideEndRange.toString());

}

}

} catch (IOException e) {

LOG.error("Exception getting daily tweet counts for timeslice: " + segmentInfo, e);

}

}

return newCountMap;

}

private void incrementNumDocsWithIllegalTimeCounter(String segmentName, String fieldSuffix) {

String statName = String.format(

"num\_docs\_with\_illegal\_time\_for\_segment\_%s%s\_counter", segmentName, fieldSuffix);

SearchCounter counter = SearchCounter.export(statName);

counter.increment();

}

private int getNewTweetCountsForSegment(

SegmentInfo segmentInfo,

EarlybirdIndexSegmentAtomicReader reader,

TimeMapper timeMapper,

Calendar cal,

Map<Integer, MutableInt> newTweetCounts) {

DocIDToTweetIDMapper tweetIdMapper = reader.getSegmentData().getDocIDToTweetIDMapper();

long dataEndTimeExclusiveMillis = getDataEndTimeExclusiveMillis(segmentInfo);

int docsOutsideEndDateRange = 0;

int docId = Integer.MIN\_VALUE;

while ((docId = tweetIdMapper.getNextDocID(docId)) != DocIDToTweetIDMapper.ID\_NOT\_FOUND) {

UpdateCountType updateCountType =

updateTweetCount(timeMapper, docId, dataEndTimeExclusiveMillis, cal, newTweetCounts);

if (updateCountType == UpdateCountType.ILLEGAL\_TIME) {

incrementNumDocsWithIllegalTimeCounter(segmentInfo.getSegmentName(), "");

} else if (updateCountType == UpdateCountType.OUT\_OF\_RANGE\_TIME) {

docsOutsideEndDateRange++;

}

}

return docsOutsideEndDateRange;

}

private int getNewFieldTweetCountsForSegment(

SegmentInfo segmentInfo,

EarlybirdIndexSegmentAtomicReader reader,

TimeMapper timeMapper,

Calendar cal,

String field,

Map<String, Map<Integer, MutableInt>> newCountMap) throws IOException {

int docsOutsideEndDateRange = 0;

Map<Integer, MutableInt> fieldTweetCounts =

newCountMap.computeIfAbsent(field, k -> Maps.newHashMap());

Terms terms = reader.terms(field);

if (terms == null) {

LOG.warn("Field <" + field + "> is missing terms in segment: "

+ segmentInfo.getSegmentName());

return 0;

}

long startTimeMillis = System.currentTimeMillis();

long dataEndTimeExclusiveMillis = getDataEndTimeExclusiveMillis(segmentInfo);

for (TermsEnum termsEnum = terms.iterator(); termsEnum.next() != null;) {

DocIdSetIterator docsIterator = termsEnum.postings(null, PostingsEnum.NONE);

for (int docId = docsIterator.nextDoc();

docId != DocIdSetIterator.NO\_MORE\_DOCS; docId = docsIterator.nextDoc()) {

UpdateCountType updateCountType = updateTweetCount(

timeMapper, docId, dataEndTimeExclusiveMillis, cal, fieldTweetCounts);

if (updateCountType == UpdateCountType.ILLEGAL\_TIME) {

incrementNumDocsWithIllegalTimeCounter(

segmentInfo.getSegmentName(), "\_and\_field\_" + field);

} else if (updateCountType == UpdateCountType.OUT\_OF\_RANGE\_TIME) {

docsOutsideEndDateRange++;

}

}

}

updateFieldRunTimeStats(field, System.currentTimeMillis() - startTimeMillis);

return docsOutsideEndDateRange;

}

private enum UpdateCountType {

OK\_TIME,

ILLEGAL\_TIME,

OUT\_OF\_RANGE\_TIME,

}

private static UpdateCountType updateTweetCount(

TimeMapper timeMapper,

int docId,

long dataEndTimeExclusiveMillis,

Calendar cal,

Map<Integer, MutableInt> newTweetCounts) {

int timeSecs = timeMapper.getTime(docId);

if (timeSecs == TimeMapper.ILLEGAL\_TIME) {

return UpdateCountType.ILLEGAL\_TIME;

}

if (dataEndTimeExclusiveMillis == Segment.NO\_DATA\_END\_TIME

|| timeSecs \* 1000L < dataEndTimeExclusiveMillis) {

Integer hourlyValue = FieldTermCounter.getHourValue(cal, timeSecs);

MutableInt count = newTweetCounts.get(hourlyValue);

if (count == null) {

count = new MutableInt(0);

newTweetCounts.put(hourlyValue, count);

}

count.increment();

return UpdateCountType.OK\_TIME;

} else {

return UpdateCountType.OUT\_OF\_RANGE\_TIME;

}

}

/\*\*

\* If a segment has an end date, return the last timestamp (exclusive, and in millis) for which

\* we expect it to have data.

\* @return Segment.NO\_DATA\_END\_TIME if the segment does not have an end date.

\*/

private long getDataEndTimeExclusiveMillis(SegmentInfo segmentInfo) {

long dataEndDate = segmentInfo.getSegment().getDataEndDateInclusiveMillis();

if (dataEndDate == Segment.NO\_DATA\_END\_TIME) {

return Segment.NO\_DATA\_END\_TIME;

} else {

return dataEndDate + MILLIS\_IN\_A\_DAY;

}

}

private void updateFieldRunTimeStats(String fieldName, long runTimeMs) {

SearchTimerStats timerStats = fieldCheckTimeStats.get(fieldName);

if (timerStats == null) {

final String statName = "tweet\_count\_monitor\_check\_time\_field\_" + fieldName;

timerStats = searchStatsReceiver.getTimerStats(

statName, TimeUnit.MILLISECONDS, false, false, false);

fieldCheckTimeStats.put(fieldName, timerStats);

}

timerStats.timerIncrement(runTimeMs);

}

@VisibleForTesting

String getStatName(String fieldName, Integer date) {

return FieldTermCounter.getStatName(fieldName, instanceCounter, date);

}

@VisibleForTesting

Map<Integer, AtomicInteger> getExportedCounts(String fieldName) {

if (fieldTermCounters.get(fieldName) == null) {

return null;

} else {

return fieldTermCounters.get(fieldName).getExportedCounts();

}

}

@VisibleForTesting

Map<Integer, MutableLong> getDailyCounts(String fieldName) {

if (fieldTermCounters.get(fieldName) == null) {

return null;

} else {

return fieldTermCounters.get(fieldName).getDailyCounts();

}

}

@VisibleForTesting

long getHoursWithNoTweets(String fieldName) {

return fieldTermCounters.get(fieldName).getHoursWithNoTweets();

}

@VisibleForTesting

long getDaysWithNoTweets(String fieldName) {

return fieldTermCounters.get(fieldName).getDaysWithNoTweets();

}

@VisibleForTesting

Map<String, SearchLongGauge> getExportedHourlyCountStats(String fieldName) {

return fieldTermCounters.get(fieldName).getExportedHourlyCountStats();

}

@Override

protected void runOneIteration() {

LOG.info("Starting to get hourly tweet counts");

final long startTimeMillis = System.currentTimeMillis();

isRunningStat.set(1);

try {

updateHourlyCounts();

} catch (Exception ex) {

LOG.error("Unexpected exception while getting hourly tweet counts", ex);

} finally {

isRunningStat.set(0);

long elapsedTimeMillis = System.currentTimeMillis() - startTimeMillis;

checkTimeStat.timerIncrement(elapsedTimeMillis);

LOG.info("Done getting daily tweet counts. Hours without tweets: "

+ getHoursWithNoTweets(FieldTermCounter.TWEET\_COUNT\_KEY));

LOG.info("Updating tweet count takes " + (elapsedTimeMillis / 1000) + " secs.");

}

}

}