package com.twitter.search.earlybird.util;

import java.util.Collections;

import java.util.HashMap;

import java.util.Map;

import java.util.Set;

import java.util.concurrent.TimeUnit;

import java.util.concurrent.atomic.AtomicLong;

import java.util.function.Function;

import java.util.stream.Collectors;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

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

import org.apache.lucene.index.IndexOptions;

import org.apache.lucene.index.Terms;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

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

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

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

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

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

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

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

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 terms per field that are

\* indexed on this earlybird, averaged over all segments.

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

\* archives.

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

\* looks at the term counts for all fields in that segment.

\*

\* Also keeps track of the number of fields that do not have any term counts (or below the specified

\* threshold) in the data that is indexed on this earlybird.

\*/

public class TermCountMonitor extends OneTaskScheduledExecutorManager {

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

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

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

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

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

private static Function<String, String> termStatNameFunc =

field -> "term\_count\_on\_field\_" + field;

private static Function<String, String> tokenStatNameFunc =

field -> "token\_count\_on\_field\_" + field;

private static Function<String, String> missingFieldStatNameFunc =

field -> "term\_count\_monitor\_missing\_field\_" + field;

private static class RawFieldCounter {

private MutableLong numTerms = new MutableLong(0L);

private MutableLong numTokens = new MutableLong(0L);

}

@VisibleForTesting

static class ExportedFieldCounter {

private final AtomicLong numTerms;

private final AtomicLong numTokens;

ExportedFieldCounter(RawFieldCounter rawCounter) {

this.numTerms = new AtomicLong(rawCounter.numTerms.longValue());

this.numTokens = new AtomicLong(rawCounter.numTokens.longValue());

}

ExportedFieldCounter(long numInitialTerms, long numInitialTokens) {

this.numTerms = new AtomicLong(numInitialTerms);

this.numTokens = new AtomicLong(numInitialTokens);

}

@VisibleForTesting

long getNumTerms() {

return numTerms.longValue();

}

@VisibleForTesting

long getNumTokens() {

return numTokens.longValue();

}

}

private final int fieldMinTermCount =

EarlybirdConfig.getInt("term\_count\_monitor\_min\_count", 0);

private final SegmentManager segmentManager;

private final Map<String, SearchLongGauge> missingFields;

private final Map<String, SearchLongGauge> termStats;

private final Map<String, SearchLongGauge> tokenStats;

private final Map<String, ExportedFieldCounter> exportedCounts;

private final SearchLongGauge termCountOnAllFields;

private final SearchLongGauge tokenCountOnAllFields;

private final SearchLongGauge fieldsWithNoTermCountStat;

private final SearchLongGauge isRunningStat;

private final SearchTimerStats checkTimeStat;

@Override

protected void runOneIteration() {

LOG.info("Starting to get per-field term counts");

isRunningStat.set(1);

final SearchTimer timer = checkTimeStat.startNewTimer();

try {

updateFieldTermCounts();

} catch (Exception ex) {

LOG.error("Unexpected exception while getting per-field term counts", ex);

} finally {

LOG.info(

"Done getting per-field term counts. Fields with low term counts: {}",

getFieldsWithLowTermCount());

isRunningStat.set(0);

checkTimeStat.stopTimerAndIncrement(timer);

}

}

/\*\*

\* Create a term count monitor which monitors the number of terms in segments

\* managed by the given segment manager.

\*/

public TermCountMonitor(

SegmentManager segmentManager,

ScheduledExecutorServiceFactory executorServiceFactory,

long shutdownWaitDuration,

TimeUnit shutdownWaitUnit,

SearchStatsReceiver searchStatsReceiver,

CriticalExceptionHandler criticalExceptionHandler) {

super(

executorServiceFactory,

THREAD\_NAME\_FORMAT,

THREAD\_IS\_DAEMON,

PeriodicActionParams.atFixedRate(

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

TimeUnit.MINUTES),

new ShutdownWaitTimeParams(

shutdownWaitDuration,

shutdownWaitUnit

),

searchStatsReceiver,

criticalExceptionHandler);

this.segmentManager = segmentManager;

this.missingFields = new HashMap<>();

this.termStats = new HashMap<>();

this.tokenStats = new HashMap<>();

this.exportedCounts = new HashMap<>();

this.termCountOnAllFields = getSearchStatsReceiver().getLongGauge("term\_count\_on\_all\_fields");

this.tokenCountOnAllFields = getSearchStatsReceiver().getLongGauge("token\_count\_on\_all\_fields");

this.fieldsWithNoTermCountStat =

getSearchStatsReceiver().getLongGauge("fields\_with\_low\_term\_counts");

this.isRunningStat =

getSearchStatsReceiver().getLongGauge("term\_count\_monitor\_is\_running");

this.checkTimeStat =

getSearchStatsReceiver().getTimerStats(

"term\_count\_monitor\_check\_time", TimeUnit.MILLISECONDS, true, true, false);

}

private SearchLongGauge getOrCreateLongGauge(

Map<String, SearchLongGauge> gauges, String field, Function<String, String> nameSupplier) {

SearchLongGauge stat = gauges.get(field);

if (stat == null) {

stat = getSearchStatsReceiver().getLongGauge(nameSupplier.apply(field));

gauges.put(field, stat);

}

return stat;

}

private void updateFieldTermCounts() {

// 0. Get the current per-field term counts

Map<String, RawFieldCounter> newCounts = getFieldStats();

LOG.info("Computed field stats for all segments");

// 1. Update all existing keys

for (Map.Entry<String, ExportedFieldCounter> exportedCount : exportedCounts.entrySet()) {

String field = exportedCount.getKey();

ExportedFieldCounter exportedCountValue = exportedCount.getValue();

RawFieldCounter newCount = newCounts.get(field);

if (newCount == null) {

exportedCountValue.numTerms.set(0L);

exportedCountValue.numTokens.set(0L);

} else {

exportedCountValue.numTerms.set(newCount.numTerms.longValue());

exportedCountValue.numTokens.set(newCount.numTokens.longValue());

// clean up so that we don't check this field again when we look for new field

newCounts.remove(field);

}

}

// 2. Add and export all new fields' term counts

for (Map.Entry<String, RawFieldCounter> newCount: newCounts.entrySet()) {

String field = newCount.getKey();

Preconditions.checkState(!exportedCounts.containsKey(field),

"Should have already processed and removed existing fields: " + field);

ExportedFieldCounter newStat = new ExportedFieldCounter(newCount.getValue());

exportedCounts.put(field, newStat);

}

// 3. Export as a stat the term counts for all the known fields.

for (Map.Entry<String, ExportedFieldCounter> exportedCount : exportedCounts.entrySet()) {

String field = exportedCount.getKey();

ExportedFieldCounter counter = exportedCount.getValue();

getOrCreateLongGauge(termStats, field, termStatNameFunc).set(counter.numTerms.get());

getOrCreateLongGauge(tokenStats, field, tokenStatNameFunc).set(counter.numTokens.get());

}

// 4. Export as a stat, number of fields not having enough term counts (i.e. <= 0)

int fieldsWithNoTermCounts = 0;

for (Map.Entry<String, ExportedFieldCounter> fieldTermCount : exportedCounts.entrySet()) {

String field = fieldTermCount.getKey();

AtomicLong exportedCountValue = fieldTermCount.getValue().numTerms;

if (exportedCountValue.get() <= fieldMinTermCount) {

LOG.warn(

"Found a field with too few term counts. Field: {} count: {}",

field, exportedCountValue);

fieldsWithNoTermCounts++;

}

}

this.fieldsWithNoTermCountStat.set(fieldsWithNoTermCounts);

}

/\*\*

\* Loops through all segments, and for each field gets the average term/token count.

\* Based on that, returns a map from each field to its term/token count (average per segment).

\*/

private Map<String, RawFieldCounter> getFieldStats() {

Iterable<SegmentInfo> segmentInfos = segmentManager.getSegmentInfos(

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

Map<String, RawFieldCounter> rawCounts = new HashMap<>();

ImmutableSchemaInterface schemaSnapshot =

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

Set<String> missingFieldsCandidates = schemaSnapshot

.getFieldInfos()

.stream()

.filter(fieldInfo -> fieldInfo.getFieldType().indexOptions() != IndexOptions.NONE)

.map(Schema.FieldInfo::getName)

.collect(Collectors.toSet());

int segmentCount = 0;

for (SegmentInfo segmentInfo : segmentInfos) {

segmentCount++;

try {

EarlybirdSingleSegmentSearcher searcher = segmentManager.getSearcher(

segmentInfo.getTimeSliceID(), schemaSnapshot);

if (searcher != null) {

EarlybirdIndexSegmentAtomicReader reader = searcher.getTwitterIndexReader();

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

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

continue;

}

String fieldName = fieldInfo.getName();

RawFieldCounter count = rawCounts.get(fieldName);

if (count == null) {

count = new RawFieldCounter();

rawCounts.put(fieldName, count);

}

Terms terms = reader.terms(fieldName);

if (terms != null) {

missingFieldsCandidates.remove(fieldName);

count.numTerms.add(terms.size());

long sumTotalTermFreq = terms.getSumTotalTermFreq();

if (sumTotalTermFreq != -1) {

count.numTokens.add(sumTotalTermFreq);

}

}

}

}

} catch (Exception e) {

LOG.error("Exception getting average term count per field: " + segmentInfo, e);

}

}

// Update missing fields stats.

missingFieldsCandidates.forEach(

field -> getOrCreateLongGauge(missingFields, field, missingFieldStatNameFunc).set(1));

missingFields.keySet().stream()

.filter(

field -> !missingFieldsCandidates.contains(field))

.forEach(

field -> getOrCreateLongGauge(missingFields, field, missingFieldStatNameFunc).set(0));

long totalTermCount = 0;

long totalTokenCount = 0;

if (segmentCount == 0) {

LOG.error("No segments are found to calculate per-field term counts.");

} else {

LOG.debug("TermCountMonitor.getPerFieldTermCount.segmentCount = {}", segmentCount);

LOG.debug(" field: term count (average per segment)");

for (Map.Entry<String, RawFieldCounter> entry : rawCounts.entrySet()) {

String field = entry.getKey();

final long averageTermCount = entry.getValue().numTerms.longValue() / segmentCount;

final long averageTokenCount = entry.getValue().numTokens.longValue() / segmentCount;

totalTermCount += entry.getValue().numTerms.longValue();

totalTokenCount += entry.getValue().numTokens.longValue();

LOG.debug(" '{} term': {}", field, averageTermCount);

LOG.debug(" '{} token': {}", field, averageTokenCount);

entry.getValue().numTerms.setValue(averageTermCount);

entry.getValue().numTokens.setValue(averageTokenCount);

}

}

LOG.info("Total term count: {}", totalTermCount);

LOG.info("Total token count: {}", totalTokenCount);

this.termCountOnAllFields.set(totalTermCount);

this.tokenCountOnAllFields.set(totalTokenCount);

return rawCounts;

}

@VisibleForTesting

Map<String, ExportedFieldCounter> getExportedCounts() {

return Collections.unmodifiableMap(this.exportedCounts);

}

@VisibleForTesting

long getFieldsWithLowTermCount() {

return fieldsWithNoTermCountStat.get();

}

@VisibleForTesting

Map<String, SearchLongGauge> getMissingFields() {

return missingFields;

}

}