package com.twitter.search.earlybird.partition;

import java.io.IOException;

import java.util.Collections;

import java.util.List;

import java.util.Map;

import java.util.concurrent.TimeUnit;

import javax.annotation.Nullable;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import com.google.common.collect.ImmutableList;

import com.google.common.collect.ImmutableMap;

import com.google.common.collect.Lists;

import com.google.common.collect.Maps;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.decider.Decider;

import com.twitter.search.common.decider.DeciderUtil;

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.schema.earlybird.EarlybirdCluster;

import com.twitter.search.core.earlybird.index.inverted.InvertedIndex;

import com.twitter.search.core.earlybird.index.inverted.MultiSegmentTermDictionary;

import com.twitter.search.core.earlybird.index.inverted.MultiSegmentTermDictionaryWithFastutil;

import com.twitter.search.core.earlybird.index.inverted.OptimizedMemoryIndex;

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

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

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

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

/\*\*

\* Manages MultiSegmentTermDictionary's for specific fields on this earlybird. Only manages them

\* for optimized segments, and should only regenerate new dictionaries when the list of optimized

\* segments changes. See SEARCH-10836

\*/

public class MultiSegmentTermDictionaryManager {

private static final Logger LOG =

LoggerFactory.getLogger(MultiSegmentTermDictionaryManager.class);

@VisibleForTesting

public static final SearchTimerStats TERM\_DICTIONARY\_CREATION\_STATS =

SearchTimerStats.export("multi\_segment\_term\_dictionary\_manager\_build\_dictionary",

TimeUnit.MILLISECONDS, false);

public static final MultiSegmentTermDictionaryManager NOOP\_INSTANCE =

new MultiSegmentTermDictionaryManager(

new Config(Collections.emptyList()), null, null, null, null) {

@Override

public boolean buildDictionary() {

return false;

}

};

private static final String MANAGER\_DISABLED\_DECIDER\_KEY\_PREFIX =

"multi\_segment\_term\_dictionary\_manager\_disabled\_in\_";

public static class Config {

private final ImmutableList<String> fieldNames;

public Config(List<String> fieldNames) {

Preconditions.checkNotNull(fieldNames);

this.fieldNames = ImmutableList.copyOf(fieldNames);

}

public List<String> managedFieldNames() {

return fieldNames;

}

public boolean isEnabled() {

return EarlybirdConfig.getBool("multi\_segment\_term\_dictionary\_enabled", false);

}

}

@VisibleForTesting

public static String getManagerDisabledDeciderName(EarlybirdCluster earlybirdCluster) {

return MANAGER\_DISABLED\_DECIDER\_KEY\_PREFIX + earlybirdCluster.name().toLowerCase();

}

private static final class FieldStats {

private final SearchTimerStats buildTime;

private final SearchLongGauge numTerms;

private final SearchLongGauge numTermEntries;

private FieldStats(SearchStatsReceiver statsReceiver, String fieldName) {

Preconditions.checkNotNull(fieldName);

Preconditions.checkNotNull(statsReceiver);

String timerName = String.format(

"multi\_segment\_term\_dictionary\_manager\_field\_%s\_build\_dictionary", fieldName);

this.buildTime = statsReceiver.getTimerStats(

timerName, TimeUnit.MILLISECONDS, false, false, false);

String numTermsName = String.format(

"multi\_segment\_term\_dictionary\_manager\_field\_%s\_num\_terms", fieldName);

this.numTerms = statsReceiver.getLongGauge(numTermsName);

String numTermEntriesName = String.format(

"multi\_segment\_term\_dictionary\_manager\_field\_%s\_num\_term\_entries", fieldName);

this.numTermEntries = statsReceiver.getLongGauge(numTermEntriesName);

}

}

private final Config config;

@Nullable private final SegmentManager segmentManager;

@Nullable private final Decider decider;

@Nullable private final EarlybirdCluster earlybirdCluster;

private final ImmutableMap<String, FieldStats> fieldTimerStats;

// A per-field map of multi-segment term dictionaries. Each key is a field. The values are the

// multi-segment term dictionaries for that field.

private volatile ImmutableMap<String, MultiSegmentTermDictionary> multiSegmentTermDictionaryMap;

private List<SegmentInfo> previousSegmentsToMerge;

public MultiSegmentTermDictionaryManager(

Config config,

SegmentManager segmentManager,

SearchStatsReceiver statsReceiver,

Decider decider,

EarlybirdCluster earlybirdCluster) {

this.config = config;

this.segmentManager = segmentManager;

this.decider = decider;

this.earlybirdCluster = earlybirdCluster;

this.multiSegmentTermDictionaryMap = ImmutableMap.of();

this.previousSegmentsToMerge = Lists.newArrayList();

ImmutableMap.Builder<String, FieldStats> builder = ImmutableMap.builder();

if (statsReceiver != null) {

for (String fieldName : config.managedFieldNames()) {

builder.put(fieldName, new FieldStats(statsReceiver, fieldName));

}

}

this.fieldTimerStats = builder.build();

}

/\*\*

\* Return the most recently built MultiSegmentTermDictionary for the given field.

\* Will return null if the field is not supported by this manager.

\*/

@Nullable

public MultiSegmentTermDictionary getMultiSegmentTermDictionary(String fieldName) {

return this.multiSegmentTermDictionaryMap.get(fieldName);

}

/\*\*

\* Build new versions of multi-segment term dictionaries if the manager is enabled, and new

\* segments are available.

\* @return true if the manager actually ran, and generated new versions of multi-segment term

\* dictionaries.

\*

\* We synchronize this method because it would be a logic error to modify the variables from

\* multiple threads simultaneously, and it is possible for two segments to finish optimizing at

\* the same time and try to run it.

\*/

public synchronized boolean buildDictionary() {

if (!config.isEnabled()) {

return false;

}

Preconditions.checkNotNull(decider);

Preconditions.checkNotNull(earlybirdCluster);

if (DeciderUtil.isAvailableForRandomRecipient(decider,

getManagerDisabledDeciderName(earlybirdCluster))) {

LOG.info("Multi segment term dictionary manager is disabled via decider for cluster {}.",

earlybirdCluster);

this.multiSegmentTermDictionaryMap = ImmutableMap.of();

this.previousSegmentsToMerge = Lists.newArrayList();

return false;

}

List<SegmentInfo> segmentsToMerge = getSegmentsToMerge();

if (differentFromPreviousList(segmentsToMerge)) {

long start = System.currentTimeMillis();

try {

this.multiSegmentTermDictionaryMap = createNewDictionaries(segmentsToMerge);

this.previousSegmentsToMerge = segmentsToMerge;

return true;

} catch (IOException e) {

LOG.error("Unable to build multi segment term dictionaries", e);

return false;

} finally {

long elapsed = System.currentTimeMillis() - start;

TERM\_DICTIONARY\_CREATION\_STATS.timerIncrement(elapsed);

}

} else {

LOG.warn("No-op for buildDictionary()");

return false;

}

}

/\*\*

\* Only merge terms from enabled and optimized segments. No need to look at non-enabled segments,

\* and we also don't want to use un-optimized segments as their term dictionaries are still

\* changing.

\*/

private List<SegmentInfo> getSegmentsToMerge() {

Iterable<SegmentInfo> segmentInfos =

segmentManager.getSegmentInfos(Filter.Enabled, Order.OLD\_TO\_NEW);

List<SegmentInfo> segmentsToMerge = Lists.newArrayList();

for (SegmentInfo segmentInfo : segmentInfos) {

if (segmentInfo.getIndexSegment().isOptimized()) {

segmentsToMerge.add(segmentInfo);

}

}

return segmentsToMerge;

}

private boolean differentFromPreviousList(List<SegmentInfo> segmentsToMerge) {

// there is a potentially different approach here to only check if the

// segmentsToMerge is subsumed by the previousSegmentsToMerge list, and not recompute

// the multi segment term dictionary if so.

// There is a case where a new segment is added, the previously current segment is not yet

// optimized, but the oldest segment is dropped. With this impl, we will recompute to remove

// the dropped segment, however, we will recompute soon again when the

// "previously current segment" is actually optimized. We can potentially delay the first

// merging before the optimization.

if (this.previousSegmentsToMerge.size() == segmentsToMerge.size()) {

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

if (previousSegmentsToMerge.get(i).compareTo(segmentsToMerge.get(i)) != 0) {

return true;

}

}

return false;

}

return true;

}

/\*\*

\* Rebuild the term dictionaries from scratch for all the managed fields.

\* Returning a brand new map here with all the fields' term dictionaries so that we can isolate

\* failures to build, and only replace the entire map of all the fields are built successfully.

\*/

private ImmutableMap<String, MultiSegmentTermDictionary> createNewDictionaries(

List<SegmentInfo> segments) throws IOException {

Map<String, MultiSegmentTermDictionary> map = Maps.newHashMap();

for (String field : config.managedFieldNames()) {

LOG.info("Merging term dictionaries for field {}", field);

List<OptimizedMemoryIndex> indexesToMerge = findFieldIndexesToMerge(segments, field);

if (indexesToMerge.isEmpty()) {

LOG.info("No indexes to merge for field {}", field);

} else {

long start = System.currentTimeMillis();

MultiSegmentTermDictionary multiSegmentTermDictionary =

mergeDictionaries(field, indexesToMerge);

map.put(field, multiSegmentTermDictionary);

long elapsed = System.currentTimeMillis() - start;

LOG.info("Done merging term dictionary for field {}, for {} segments in {}ms",

field, indexesToMerge.size(), elapsed);

FieldStats fieldStats = fieldTimerStats.get(field);

fieldStats.buildTime.timerIncrement(elapsed);

fieldStats.numTerms.set(multiSegmentTermDictionary.getNumTerms());

fieldStats.numTermEntries.set(multiSegmentTermDictionary.getNumTermEntries());

}

}

return ImmutableMap.copyOf(map);

}

private List<OptimizedMemoryIndex> findFieldIndexesToMerge(

List<SegmentInfo> segments, String field) throws IOException {

List<OptimizedMemoryIndex> indexesToMerge = Lists.newArrayList();

for (SegmentInfo segment : segments) {

EarlybirdSegment indexSegment = segment.getIndexSegment();

Preconditions.checkState(indexSegment.isOptimized(),

"Expect segment to be optimized: %s", segment);

InvertedIndex fieldIndex = Preconditions.checkNotNull(indexSegment.getIndexReader())

.getSegmentData().getFieldIndex(field);

// See SEARCH-11952

// We will only have a InvertedIndex/OptimizedMemoryIndex here

// in the in-memory non-lucene-based indexes, and not in the archive. We can somewhat

// reasonably extend this to work with the archive by making the dictionaries work with

// TermsEnum's directly instead of OptimizedMemoryIndex's. Leaving this as a further

// extension for now.

if (fieldIndex != null) {

if (fieldIndex instanceof OptimizedMemoryIndex) {

indexesToMerge.add((OptimizedMemoryIndex) fieldIndex);

} else {

LOG.info("Found field index for field {} in segment {} of type {}",

field, segment, fieldIndex.getClass());

}

} else {

LOG.info("Found null field index for field {} in segment {}", field, segment);

}

}

LOG.info("Found good fields for {} out of {} segments", indexesToMerge.size(),

segments.size());

return indexesToMerge;

}

private MultiSegmentTermDictionary mergeDictionaries(

String field,

List<OptimizedMemoryIndex> indexes) {

// May change this if we get a better implementation in the future.

return new MultiSegmentTermDictionaryWithFastutil(field, indexes);

}

}