package com.twitter.search.core.earlybird.index.inverted;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import com.google.common.base.Preconditions;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.apache.lucene.index.PostingsEnum;

import org.apache.lucene.index.TermsEnum;

import org.apache.lucene.search.DocIdSetIterator;

import org.apache.lucene.util.BytesRef;

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

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

import com.twitter.search.core.earlybird.facets.AbstractFacetCountingArray;

import com.twitter.search.core.earlybird.facets.FacetLabelProvider;

import com.twitter.search.core.earlybird.facets.FacetUtil;

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

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

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

import com.twitter.search.core.earlybird.index.column.DocValuesManager;

public final class IndexOptimizer {

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

private IndexOptimizer() {

}

/\*\*

\* Optimizes this in-memory index segment.

\*/

public static EarlybirdRealtimeIndexSegmentData optimize(

EarlybirdRealtimeIndexSegmentData source) throws IOException {

LOG.info("Starting index optimizing.");

ConcurrentHashMap<String, InvertedIndex> targetMap = new ConcurrentHashMap<>();

LOG.info(String.format(

"Source PerFieldMap size is %d", source.getPerFieldMap().size()));

LOG.info("Optimize doc id mapper.");

// Optimize the doc ID mapper first.

DocIDToTweetIDMapper originalTweetIdMapper = source.getDocIDToTweetIDMapper();

DocIDToTweetIDMapper optimizedTweetIdMapper = originalTweetIdMapper.optimize();

TimeMapper optimizedTimeMapper =

source.getTimeMapper() != null

? source.getTimeMapper().optimize(originalTweetIdMapper, optimizedTweetIdMapper)

: null;

// Some fields have their terms rewritten to support the minimal perfect hash function we use

// (note that it's a minimal perfect hash function, not a minimal perfect hash \_table\_).

// The FacetCountingArray stores term IDs. This is a map from the facet field ID to a map from

// original term ID to the new, MPH term IDs.

Map<Integer, int[]> termIDMapper = new HashMap<>();

LOG.info("Optimize inverted indexes.");

optimizeInvertedIndexes(

source, targetMap, originalTweetIdMapper, optimizedTweetIdMapper, termIDMapper);

LOG.info("Rewrite and map ids in facet counting array.");

AbstractFacetCountingArray facetCountingArray = source.getFacetCountingArray().rewriteAndMapIDs(

termIDMapper, originalTweetIdMapper, optimizedTweetIdMapper);

Map<String, FacetLabelProvider> facetLabelProviders =

FacetUtil.getFacetLabelProviders(source.getSchema(), targetMap);

LOG.info("Optimize doc values manager.");

DocValuesManager optimizedDocValuesManager =

source.getDocValuesManager().optimize(originalTweetIdMapper, optimizedTweetIdMapper);

LOG.info("Optimize deleted docs.");

DeletedDocs optimizedDeletedDocs =

source.getDeletedDocs().optimize(originalTweetIdMapper, optimizedTweetIdMapper);

final boolean isOptimized = true;

return new EarlybirdRealtimeIndexSegmentData(

source.getMaxSegmentSize(),

source.getTimeSliceID(),

source.getSchema(),

isOptimized,

optimizedTweetIdMapper.getNextDocID(Integer.MIN\_VALUE),

targetMap,

facetCountingArray,

optimizedDocValuesManager,

facetLabelProviders,

source.getFacetIDMap(),

optimizedDeletedDocs,

optimizedTweetIdMapper,

optimizedTimeMapper,

source.getIndexExtensionsData());

}

private static void optimizeInvertedIndexes(

EarlybirdRealtimeIndexSegmentData source,

ConcurrentHashMap<String, InvertedIndex> targetMap,

DocIDToTweetIDMapper originalTweetIdMapper,

DocIDToTweetIDMapper optimizedTweetIdMapper,

Map<Integer, int[]> termIDMapper

) throws IOException {

for (Map.Entry<String, InvertedIndex> entry : source.getPerFieldMap().entrySet()) {

String fieldName = entry.getKey();

Preconditions.checkState(entry.getValue() instanceof InvertedRealtimeIndex);

InvertedRealtimeIndex sourceIndex = (InvertedRealtimeIndex) entry.getValue();

EarlybirdFieldType fieldType = source.getSchema().getFieldInfo(fieldName).getFieldType();

InvertedIndex newIndex;

if (fieldType.becomesImmutable() && sourceIndex.getNumTerms() > 0) {

Schema.FieldInfo facetField = source.getSchema().getFacetFieldByFieldName(fieldName);

newIndex = new OptimizedMemoryIndex(

fieldType,

fieldName,

sourceIndex,

termIDMapper,

source.getFacetIDMap().getFacetField(facetField),

originalTweetIdMapper,

optimizedTweetIdMapper);

} else {

newIndex = optimizeMutableIndex(

fieldType,

fieldName,

sourceIndex,

originalTweetIdMapper,

optimizedTweetIdMapper);

}

targetMap.put(fieldName, newIndex);

}

}

/\*\*

\* Optimize a mutable index.

\*/

private static InvertedIndex optimizeMutableIndex(

EarlybirdFieldType fieldType,

String fieldName,

InvertedRealtimeIndex originalIndex,

DocIDToTweetIDMapper originalMapper,

DocIDToTweetIDMapper optimizedMapper

) throws IOException {

Preconditions.checkState(!fieldType.isStorePerPositionPayloads());

TermsEnum allTerms = originalIndex.createTermsEnum(originalIndex.getMaxPublishedPointer());

int numTerms = originalIndex.getNumTerms();

InvertedRealtimeIndex index = new InvertedRealtimeIndex(

fieldType,

TermPointerEncoding.DEFAULT\_ENCODING,

fieldName);

index.setNumDocs(originalIndex.getNumDocs());

for (int termID = 0; termID < numTerms; termID++) {

allTerms.seekExact(termID);

PostingsEnum postingsEnum = new OptimizingPostingsEnumWrapper(

allTerms.postings(null), originalMapper, optimizedMapper);

BytesRef termPayload = originalIndex.getLabelAccessor().getTermPayload(termID);

copyPostingList(index, postingsEnum, termID, allTerms.term(), termPayload);

}

return index;

}

/\*\*

\* Copies the given posting list into these posting lists.

\*

\* @param postingsEnum enumerator of the posting list that needs to be copied

\*/

private static void copyPostingList(

InvertedRealtimeIndex index,

PostingsEnum postingsEnum,

int termID,

BytesRef term,

BytesRef termPayload

) throws IOException {

int docId;

while ((docId = postingsEnum.nextDoc()) != DocIdSetIterator.NO\_MORE\_DOCS) {

index.incrementSumTermDocFreq();

for (int i = 0; i < postingsEnum.freq(); i++) {

index.incrementSumTotalTermFreq();

int position = postingsEnum.nextPosition();

int newTermID = InvertedRealtimeIndexWriter.indexTerm(

index,

term,

docId,

position,

termPayload,

null, // We know that fields that remain mutable never have a posting payload.

TermPointerEncoding.DEFAULT\_ENCODING);

// Our term lookups are very slow, so we cache term dictionaries for some fields across many

// segments, so we must keep the term IDs the same while remapping.

Preconditions.checkState(newTermID == termID);

}

}

}

}