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

import java.io.IOException;

import java.util.Comparator;

import java.util.Map;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

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.apache.lucene.util.BytesRef;

import org.apache.lucene.util.packed.PackedInts;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

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

import com.twitter.search.common.util.hash.BDZAlgorithm;

import com.twitter.search.common.util.hash.BDZAlgorithm.MPHFNotFoundException;

import com.twitter.search.common.util.hash.KeysSource;

import com.twitter.search.common.util.io.flushable.DataDeserializer;

import com.twitter.search.common.util.io.flushable.DataSerializer;

import com.twitter.search.common.util.io.flushable.FlushInfo;

import com.twitter.search.common.util.io.flushable.Flushable;

import com.twitter.search.core.earlybird.facets.FacetIDMap.FacetField;

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

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

public class OptimizedMemoryIndex extends InvertedIndex implements Flushable {

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

private static final Comparator<BytesRef> BYTES\_REF\_COMPARATOR = Comparator.naturalOrder();

private static final SearchCounter MPH\_NOT\_FOUND\_COUNT =

SearchCounter.export("twitter\_optimized\_index\_mph\_not\_found\_count");

private final PackedInts.Reader numPostings;

private final PackedInts.Reader postingListPointers;

private final PackedInts.Reader offensiveCounters;

private final MultiPostingLists postingLists;

private final TermDictionary dictionary;

private final int numDocs;

private final int sumTotalTermFreq;

private final int sumTermDocFreq;

private OptimizedMemoryIndex(EarlybirdFieldType fieldType,

int numDocs,

int sumTermDocFreq,

int sumTotalTermFreq,

PackedInts.Reader numPostings,

PackedInts.Reader postingListPointers,

PackedInts.Reader offensiveCounters,

MultiPostingLists postingLists,

TermDictionary dictionary) {

super(fieldType);

this.numDocs = numDocs;

this.sumTermDocFreq = sumTermDocFreq;

this.sumTotalTermFreq = sumTotalTermFreq;

this.numPostings = numPostings;

this.postingListPointers = postingListPointers;

this.offensiveCounters = offensiveCounters;

this.postingLists = postingLists;

this.dictionary = dictionary;

}

public OptimizedMemoryIndex(

EarlybirdFieldType fieldType,

String field,

InvertedRealtimeIndex source,

Map<Integer, int[]> termIDMapper,

FacetField facetField,

DocIDToTweetIDMapper originalTweetIdMapper,

DocIDToTweetIDMapper optimizedTweetIdMapper) throws IOException {

super(fieldType);

numDocs = source.getNumDocs();

sumTermDocFreq = source.getSumTermDocFreq();

sumTotalTermFreq = source.getSumTotalTermFreq();

Preconditions.checkNotNull(originalTweetIdMapper, "The segment must have a tweet ID mapper.");

Preconditions.checkNotNull(optimizedTweetIdMapper,

"The optimized tweet ID mapper cannot be null.");

// We rely on the fact that new terms always have a greater term ID. We ignore all terms that

// are equal to or greater than numTerms, as they may be incompletely applied. If new terms are

// added while optimizing, they will be re-added when we re-apply updates.

final KeysSource termsIterator = source.getKeysSource();

int numTerms = termsIterator.getNumberOfKeys();

int maxPublishedPointer = source.getMaxPublishedPointer();

int[] tempPostingListPointers = new int[numTerms];

BDZAlgorithm termsHashFunction = null;

final boolean supportTermTextLookup = facetField != null || fieldType.isSupportTermTextLookup();

if (supportTermTextLookup) {

try {

termsHashFunction = new BDZAlgorithm(termsIterator);

} catch (MPHFNotFoundException e) {

// we couldn't find a mphf for this field

// no problem, this can happen for very small fields

// - just use the fst in that case

LOG.warn("Unable to build MPH for field: {}", field);

MPH\_NOT\_FOUND\_COUNT.increment();

}

}

// Make sure to only call the expensive computeNumPostings() once.

int[] numPostingsSource = computeNumPostings(source, numTerms, maxPublishedPointer);

// The BDZ Algorithm returns a function from bytesref to term ID. However, these term IDs are

// different than the original term IDs (it's a hash function, not a hash \_table\_), so we have

// to remap the term IDs to match the ones generated by BDZ. We track that using the termIDMap.

int[] termIDMap = null;

if (termsHashFunction != null) {

termsIterator.rewind();

termIDMap = BDZAlgorithm.createIdMap(termsHashFunction, termsIterator);

if (facetField != null) {

termIDMapper.put(facetField.getFacetId(), termIDMap);

}

PackedInts.Reader termPointers = getPackedInts(source.getTermPointers(), termIDMap);

this.numPostings = getPackedInts(numPostingsSource, termIDMap);

this.offensiveCounters = source.getOffensiveCounters() == null ? null

: getPackedInts(source.getOffensiveCounters(), termIDMap);

this.dictionary = new MPHTermDictionary(

numTerms,

termsHashFunction,

termPointers,

source.getTermPool(),

TermPointerEncoding.DEFAULT\_ENCODING);

} else {

this.dictionary = FSTTermDictionary.buildFST(

source.getTermPool(),

source.getTermPointers(),

numTerms,

BYTES\_REF\_COMPARATOR,

supportTermTextLookup,

TermPointerEncoding.DEFAULT\_ENCODING);

this.numPostings = getPackedInts(numPostingsSource);

this.offensiveCounters = source.getOffensiveCounters() == null ? null

: getPackedInts(source.getOffensiveCounters());

}

TermsEnum allTerms = source.createTermsEnum(maxPublishedPointer);

this.postingLists = new MultiPostingLists(

!fieldType.hasPositions(),

numPostingsSource,

source.getMaxPosition());

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

allTerms.seekExact(termID);

PostingsEnum postingsEnum = new OptimizingPostingsEnumWrapper(

allTerms.postings(null), originalTweetIdMapper, optimizedTweetIdMapper);

int mappedTermID = termIDMap != null ? termIDMap[termID] : termID;

tempPostingListPointers[mappedTermID] =

postingLists.copyPostingList(postingsEnum, numPostingsSource[termID]);

}

this.postingListPointers = getPackedInts(tempPostingListPointers);

}

private static int[] map(int[] source, int[] map) {

int[] target = new int[map.length];

for (int i = 0; i < map.length; i++) {

target[map[i]] = source[i];

}

return target;

}

static PackedInts.Reader getPackedInts(int[] values) {

return getPackedInts(values, null);

}

private static PackedInts.Reader getPackedInts(int[] values, int[] map) {

int[] mappedValues = values;

if (map != null) {

mappedValues = map(mappedValues, map);

}

// first determine max value

long maxValue = Long.MIN\_VALUE;

for (int value : mappedValues) {

if (value > maxValue) {

maxValue = value;

}

}

PackedInts.Mutable packed =

PackedInts.getMutable(mappedValues.length, PackedInts.bitsRequired(maxValue),

PackedInts.DEFAULT);

for (int i = 0; i < mappedValues.length; i++) {

packed.set(i, mappedValues[i]);

}

return packed;

}

/\*\*

\* Returns per-term array containing the number of posting in this index for each term.

\* This call is extremely slow.

\*/

private static int[] computeNumPostings(

InvertedRealtimeIndex source,

int numTerms,

int maxPublishedPointer

) throws IOException {

int[] numPostings = new int[numTerms];

TermsEnum allTerms = source.createTermsEnum(maxPublishedPointer);

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

allTerms.seekExact(termID);

PostingsEnum docsEnum = allTerms.postings(null);

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

numPostings[termID] += docsEnum.freq();

}

}

return numPostings;

}

@Override

public int getNumDocs() {

return numDocs;

}

@Override

public int getSumTotalTermFreq() {

return sumTotalTermFreq;

}

@Override

public int getSumTermDocFreq() {

return sumTermDocFreq;

}

public OptimizedPostingLists getPostingLists() {

Preconditions.checkState(hasPostingLists());

return postingLists;

}

int getPostingListPointer(int termID) {

Preconditions.checkState(hasPostingLists());

return (int) postingListPointers.get(termID);

}

int getNumPostings(int termID) {

Preconditions.checkState(hasPostingLists());

return (int) numPostings.get(termID);

}

public boolean getTerm(int termID, BytesRef text, BytesRef termPayload) {

return dictionary.getTerm(termID, text, termPayload);

}

@Override

public FacetLabelAccessor getLabelAccessor() {

return new FacetLabelAccessor() {

@Override

protected boolean seek(long termID) {

if (termID != EarlybirdIndexSegmentAtomicReader.TERM\_NOT\_FOUND) {

hasTermPayload = getTerm((int) termID, termRef, termPayload);

offensiveCount = offensiveCounters != null

? (int) offensiveCounters.get((int) termID) : 0;

return true;

} else {

return false;

}

}

};

}

@Override

public Terms createTerms(int maxPublishedPointer) {

return new OptimizedIndexTerms(this);

}

@Override

public TermsEnum createTermsEnum(int maxPublishedPointer) {

return dictionary.createTermsEnum(this);

}

@Override

public int lookupTerm(BytesRef term) throws IOException {

return dictionary.lookupTerm(term);

}

@Override

public int getLargestDocIDForTerm(int termID) throws IOException {

Preconditions.checkState(hasPostingLists());

if (termID == EarlybirdIndexSegmentAtomicReader.TERM\_NOT\_FOUND) {

return EarlybirdIndexSegmentAtomicReader.TERM\_NOT\_FOUND;

} else {

return postingLists.getLargestDocID((int) postingListPointers.get(termID),

(int) numPostings.get(termID));

}

}

@Override

public int getDF(int termID) {

return (int) numPostings.get(termID);

}

@Override

public int getNumTerms() {

return dictionary.getNumTerms();

}

@Override

public void getTerm(int termID, BytesRef text) {

dictionary.getTerm(termID, text, null);

}

@VisibleForTesting TermDictionary getTermDictionary() {

return dictionary;

}

@Override

public FlushHandler getFlushHandler() {

return new FlushHandler(this);

}

public boolean hasPostingLists() {

return postingListPointers != null

&& postingLists != null

&& numPostings != null;

}

@VisibleForTesting

OptimizedPostingLists getOptimizedPostingLists() {

return postingLists;

}

public static class FlushHandler extends Flushable.Handler<OptimizedMemoryIndex> {

private static final String NUM\_DOCS\_PROP\_NAME = "numDocs";

private static final String SUM\_TOTAL\_TERM\_FREQ\_PROP\_NAME = "sumTotalTermFreq";

private static final String SUM\_TERM\_DOC\_FREQ\_PROP\_NAME = "sumTermDocFreq";

private static final String USE\_MIN\_PERFECT\_HASH\_PROP\_NAME = "useMinimumPerfectHashFunction";

private static final String SKIP\_POSTING\_LIST\_PROP\_NAME = "skipPostingLists";

private static final String HAS\_OFFENSIVE\_COUNTERS\_PROP\_NAME = "hasOffensiveCounters";

public static final String IS\_OPTIMIZED\_PROP\_NAME = "isOptimized";

private final EarlybirdFieldType fieldType;

public FlushHandler(EarlybirdFieldType fieldType) {

super();

this.fieldType = fieldType;

}

public FlushHandler(OptimizedMemoryIndex objectToFlush) {

super(objectToFlush);

fieldType = objectToFlush.fieldType;

}

@Override

protected void doFlush(FlushInfo flushInfo, DataSerializer out) throws IOException {

long startTime = getClock().nowMillis();

OptimizedMemoryIndex objectToFlush = getObjectToFlush();

boolean useHashFunction = objectToFlush.dictionary instanceof MPHTermDictionary;

boolean skipPostingLists = !objectToFlush.hasPostingLists();

flushInfo.addIntProperty(NUM\_DOCS\_PROP\_NAME, objectToFlush.numDocs);

flushInfo.addIntProperty(SUM\_TERM\_DOC\_FREQ\_PROP\_NAME, objectToFlush.sumTermDocFreq);

flushInfo.addIntProperty(SUM\_TOTAL\_TERM\_FREQ\_PROP\_NAME, objectToFlush.sumTotalTermFreq);

flushInfo.addBooleanProperty(USE\_MIN\_PERFECT\_HASH\_PROP\_NAME, useHashFunction);

flushInfo.addBooleanProperty(SKIP\_POSTING\_LIST\_PROP\_NAME, skipPostingLists);

flushInfo.addBooleanProperty(HAS\_OFFENSIVE\_COUNTERS\_PROP\_NAME,

objectToFlush.offensiveCounters != null);

flushInfo.addBooleanProperty(IS\_OPTIMIZED\_PROP\_NAME, true);

if (!skipPostingLists) {

out.writePackedInts(objectToFlush.postingListPointers);

out.writePackedInts(objectToFlush.numPostings);

}

if (objectToFlush.offensiveCounters != null) {

out.writePackedInts(objectToFlush.offensiveCounters);

}

if (!skipPostingLists) {

objectToFlush.postingLists.getFlushHandler().flush(

flushInfo.newSubProperties("postingLists"), out);

}

objectToFlush.dictionary.getFlushHandler().flush(flushInfo.newSubProperties("dictionary"),

out);

getFlushTimerStats().timerIncrement(getClock().nowMillis() - startTime);

}

@Override

protected OptimizedMemoryIndex doLoad(

FlushInfo flushInfo, DataDeserializer in) throws IOException {

long startTime = getClock().nowMillis();

boolean useHashFunction = flushInfo.getBooleanProperty(USE\_MIN\_PERFECT\_HASH\_PROP\_NAME);

boolean skipPostingLists = flushInfo.getBooleanProperty(SKIP\_POSTING\_LIST\_PROP\_NAME);

PackedInts.Reader postingListPointers = skipPostingLists ? null : in.readPackedInts();

PackedInts.Reader numPostings = skipPostingLists ? null : in.readPackedInts();

PackedInts.Reader offensiveCounters =

flushInfo.getBooleanProperty(HAS\_OFFENSIVE\_COUNTERS\_PROP\_NAME)

? in.readPackedInts() : null;

MultiPostingLists postingLists = skipPostingLists ? null

: (new MultiPostingLists.FlushHandler())

.load(flushInfo.getSubProperties("postingLists"), in);

TermDictionary dictionary;

if (useHashFunction) {

dictionary = (new MPHTermDictionary.FlushHandler(TermPointerEncoding.DEFAULT\_ENCODING))

.load(flushInfo.getSubProperties("dictionary"), in);

} else {

dictionary = (new FSTTermDictionary.FlushHandler(TermPointerEncoding.DEFAULT\_ENCODING))

.load(flushInfo.getSubProperties("dictionary"), in);

}

getLoadTimerStats().timerIncrement(getClock().nowMillis() - startTime);

return new OptimizedMemoryIndex(fieldType,

flushInfo.getIntProperty(NUM\_DOCS\_PROP\_NAME),

flushInfo.getIntProperty(SUM\_TERM\_DOC\_FREQ\_PROP\_NAME),

flushInfo.getIntProperty(SUM\_TOTAL\_TERM\_FREQ\_PROP\_NAME),

numPostings,

postingListPointers,

offensiveCounters,

postingLists,

dictionary);

}

}

}