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

import java.io.IOException;

import javax.annotation.Nullable;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import org.apache.lucene.index.PostingsEnum;

import org.apache.lucene.search.DocIdSetIterator;

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

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

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;

/\*\*

\* A posting list intended for low-df terms, terms that have a small number of postings.

\*

\* The postings (docs and positions) are stored in PackedInts, packed based on the largest docId

\* and position across all low-df terms in a field.

\*

\* All docIds are packed together in their own PackedInts, and all positions are stored together

\* in their own PackedInts.

\* - A docId is stored for every single posting, that is if a doc has a frequency of N, it will be

\* stored N times.

\* - For fields that omitPositions, positions are not stored at all.

\*

\* Example:

\* Postings in the form (docId, position):

\* (1, 0), (1, 1), (2, 1), (2, 3), (2, 5), (4, 0), (5, 0)

\* Will be stored as:

\* packedDocIds: [1, 1, 2, 2, 2, 4, 5]

\* packedPositions: [0, 1, 1, 3, 5, 0, 0]

\*/

public class LowDFPackedIntsPostingLists extends OptimizedPostingLists {

private static final SearchCounter GETTING\_POSITIONS\_WITH\_OMIT\_POSITIONS =

SearchCounter.export("low\_df\_packed\_ints\_posting\_list\_getting\_positions\_with\_omit\_positions");

/\*\*

\* Internal class for hiding PackedInts Readers and Writers. A Mutable instance of PackedInts is

\* only required when we're optimizing a new index.

\* For the read side, we only need a PackedInts.Reader.

\* For loaded indexes, we also only need a PackedInts.Reader.

\*/

private static final class PackedIntsWrapper {

// Will be null if we are operating on a loaded in read-only index.

@Nullable

private final PackedInts.Mutable mutablePackedInts;

private final PackedInts.Reader readerPackedInts;

private PackedIntsWrapper(PackedInts.Mutable mutablePackedInts) {

this.mutablePackedInts = Preconditions.checkNotNull(mutablePackedInts);

this.readerPackedInts = mutablePackedInts;

}

private PackedIntsWrapper(PackedInts.Reader readerPackedInts) {

this.mutablePackedInts = null;

this.readerPackedInts = readerPackedInts;

}

public int size() {

return readerPackedInts.size();

}

public PackedInts.Reader getReader() {

return readerPackedInts;

}

public void set(int index, long value) {

this.mutablePackedInts.set(index, value);

}

}

private final PackedIntsWrapper packedDocIds;

/\*\*

\* Will be null for fields that omitPositions.

\*/

@Nullable

private final PackedIntsWrapper packedPositions;

private final boolean omitPositions;

private final int totalPostingsAcrossTerms;

private final int maxPosition;

private int currentPackedIntsPosition;

/\*\*

\* Creates a new LowDFPackedIntsPostingLists.

\* @param omitPositions whether positions should be omitted or not.

\* @param totalPostingsAcrossTerms how many postings across all terms this field has.

\* @param maxPosition the largest position used in all the postings for this field.

\*/

public LowDFPackedIntsPostingLists(

boolean omitPositions,

int totalPostingsAcrossTerms,

int maxPosition) {

this(

new PackedIntsWrapper(PackedInts.getMutable(

totalPostingsAcrossTerms,

PackedInts.bitsRequired(MAX\_DOC\_ID),

PackedInts.DEFAULT)),

omitPositions

? null

: new PackedIntsWrapper(PackedInts.getMutable(

totalPostingsAcrossTerms,

PackedInts.bitsRequired(maxPosition),

PackedInts.DEFAULT)),

omitPositions,

totalPostingsAcrossTerms,

maxPosition);

}

private LowDFPackedIntsPostingLists(

PackedIntsWrapper packedDocIds,

@Nullable

PackedIntsWrapper packedPositions,

boolean omitPositions,

int totalPostingsAcrossTerms,

int maxPosition) {

this.packedDocIds = packedDocIds;

this.packedPositions = packedPositions;

this.omitPositions = omitPositions;

this.totalPostingsAcrossTerms = totalPostingsAcrossTerms;

this.maxPosition = maxPosition;

this.currentPackedIntsPosition = 0;

}

@Override

public int copyPostingList(PostingsEnum postingsEnum, int numPostings) throws IOException {

int pointer = currentPackedIntsPosition;

int docId;

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

assert docId <= MAX\_DOC\_ID;

int freq = postingsEnum.freq();

assert freq <= numPostings;

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

packedDocIds.set(currentPackedIntsPosition, docId);

if (packedPositions != null) {

int position = postingsEnum.nextPosition();

assert position <= maxPosition;

packedPositions.set(currentPackedIntsPosition, position);

}

currentPackedIntsPosition++;

}

}

return pointer;

}

@Override

public EarlybirdPostingsEnum postings(

int postingListPointer,

int numPostings,

int flags) throws IOException {

if (PostingsEnum.featureRequested(flags, PostingsEnum.POSITIONS) && !omitPositions) {

assert packedPositions != null;

return new LowDFPackedIntsPostingsEnum(

packedDocIds.getReader(),

packedPositions.getReader(),

postingListPointer,

numPostings);

} else {

if (PostingsEnum.featureRequested(flags, PostingsEnum.POSITIONS) && omitPositions) {

GETTING\_POSITIONS\_WITH\_OMIT\_POSITIONS.increment();

}

return new LowDFPackedIntsPostingsEnum(

packedDocIds.getReader(),

null, // no positions

postingListPointer,

numPostings);

}

}

@VisibleForTesting

int getPackedIntsSize() {

return packedDocIds.size();

}

@VisibleForTesting

int getMaxPosition() {

return maxPosition;

}

@VisibleForTesting

boolean isOmitPositions() {

return omitPositions;

}

@SuppressWarnings("unchecked")

@Override

public FlushHandler getFlushHandler() {

return new FlushHandler(this);

}

static class FlushHandler extends Flushable.Handler<LowDFPackedIntsPostingLists> {

private static final String OMIT\_POSITIONS\_PROP\_NAME = "omitPositions";

private static final String TOTAL\_POSTINGS\_PROP\_NAME = "totalPostingsAcrossTerms";

private static final String MAX\_POSITION\_PROP\_NAME = "maxPosition";

public FlushHandler() {

super();

}

public FlushHandler(LowDFPackedIntsPostingLists objectToFlush) {

super(objectToFlush);

}

@Override

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

LowDFPackedIntsPostingLists objectToFlush = getObjectToFlush();

flushInfo.addBooleanProperty(OMIT\_POSITIONS\_PROP\_NAME, objectToFlush.omitPositions);

flushInfo.addIntProperty(TOTAL\_POSTINGS\_PROP\_NAME, objectToFlush.totalPostingsAcrossTerms);

flushInfo.addIntProperty(MAX\_POSITION\_PROP\_NAME, objectToFlush.maxPosition);

out.writePackedInts(objectToFlush.packedDocIds.getReader());

if (!objectToFlush.omitPositions) {

assert objectToFlush.packedPositions != null;

out.writePackedInts(objectToFlush.packedPositions.getReader());

}

}

@Override

protected LowDFPackedIntsPostingLists doLoad(

FlushInfo flushInfo,

DataDeserializer in) throws IOException {

boolean omitPositions = flushInfo.getBooleanProperty(OMIT\_POSITIONS\_PROP\_NAME);

int totalPostingsAcrossTerms = flushInfo.getIntProperty(TOTAL\_POSTINGS\_PROP\_NAME);

int maxPosition = flushInfo.getIntProperty(MAX\_POSITION\_PROP\_NAME);

PackedIntsWrapper packedDocIds = new PackedIntsWrapper(in.readPackedInts());

PackedIntsWrapper packedPositions = null;

if (!omitPositions) {

packedPositions = new PackedIntsWrapper(in.readPackedInts());

}

return new LowDFPackedIntsPostingLists(

packedDocIds,

packedPositions,

omitPositions,

totalPostingsAcrossTerms,

maxPosition);

}

}

}