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

import java.io.IOException;

import org.apache.lucene.index.BaseTermsEnum;

import org.apache.lucene.index.ImpactsEnum;

import org.apache.lucene.index.PostingsEnum;

import org.apache.lucene.index.SlowImpactsEnum;

import org.apache.lucene.index.TermsEnum;

import org.apache.lucene.util.BytesRef;

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

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

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.index.EarlybirdIndexSegmentAtomicReader;

public class MPHTermDictionary implements TermDictionary, Flushable {

private final BDZAlgorithm termsHashFunction;

private final PackedInts.Reader termPointers;

private final ByteBlockPool termPool;

private final TermPointerEncoding termPointerEncoding;

private final int numTerms;

MPHTermDictionary(int numTerms, BDZAlgorithm termsHashFunction,

PackedInts.Reader termPointers, ByteBlockPool termPool,

TermPointerEncoding termPointerEncoding) {

this.numTerms = numTerms;

this.termsHashFunction = termsHashFunction;

this.termPointers = termPointers;

this.termPool = termPool;

this.termPointerEncoding = termPointerEncoding;

}

@Override

public int getNumTerms() {

return numTerms;

}

@Override

public int lookupTerm(BytesRef term) {

int termID = termsHashFunction.lookup(term);

if (termID >= getNumTerms() || termID < 0) {

return EarlybirdIndexSegmentAtomicReader.TERM\_NOT\_FOUND;

}

if (ByteTermUtils.postingEquals(termPool, termPointerEncoding

.getTextStart((int) termPointers.get(termID)), term)) {

return termID;

} else {

return EarlybirdIndexSegmentAtomicReader.TERM\_NOT\_FOUND;

}

}

@Override

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

int termPointer = (int) termPointers.get(termID);

boolean hasTermPayload = termPointerEncoding.hasPayload(termPointer);

int textStart = termPointerEncoding.getTextStart(termPointer);

// setBytesRef sets the passed in BytesRef "text" to the term in the termPool.

// As a side effect it returns the offset of the next entry in the pool after the term,

// which may optionally be used if this term has a payload.

int termPayloadStart = ByteTermUtils.setBytesRef(termPool, text, textStart);

if (termPayload != null && hasTermPayload) {

ByteTermUtils.setBytesRef(termPool, termPayload, termPayloadStart);

}

return hasTermPayload;

}

@Override

public TermsEnum createTermsEnum(OptimizedMemoryIndex index) {

return new MPHTermsEnum(index);

}

public static class MPHTermsEnum extends BaseTermsEnum {

private int termID;

private final BytesRef bytesRef = new BytesRef();

private final OptimizedMemoryIndex index;

MPHTermsEnum(OptimizedMemoryIndex index) {

this.index = index;

}

@Override

public int docFreq() {

return index.getDF(termID);

}

@Override

public PostingsEnum postings(PostingsEnum reuse, int flags) throws IOException {

int postingsPointer = index.getPostingListPointer(termID);

int numPostings = index.getNumPostings(termID);

return index.getPostingLists().postings(postingsPointer, numPostings, flags);

}

@Override

public ImpactsEnum impacts(int flags) throws IOException {

return new SlowImpactsEnum(postings(null, flags));

}

@Override

public SeekStatus seekCeil(BytesRef text) throws IOException {

termID = index.lookupTerm(text);

if (termID == -1) {

return SeekStatus.END;

} else {

return SeekStatus.FOUND;

}

}

@Override

public BytesRef next() {

return null;

}

@Override

public long ord() {

return termID;

}

@Override

public void seekExact(long ord) {

if (ord < index.getNumTerms()) {

termID = (int) ord;

index.getTerm(termID, bytesRef, null);

}

}

@Override

public BytesRef term() {

return bytesRef;

}

@Override

public long totalTermFreq() {

return docFreq();

}

}

@SuppressWarnings("unchecked")

@Override

public FlushHandler getFlushHandler() {

return new FlushHandler(this);

}

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

static final String NUM\_TERMS\_PROP\_NAME = "numTerms";

private final TermPointerEncoding termPointerEncoding;

public FlushHandler(TermPointerEncoding termPointerEncoding) {

super();

this.termPointerEncoding = termPointerEncoding;

}

public FlushHandler(MPHTermDictionary objectToFlush) {

super(objectToFlush);

this.termPointerEncoding = objectToFlush.termPointerEncoding;

}

@Override

protected void doFlush(FlushInfo flushInfo, DataSerializer out)

throws IOException {

MPHTermDictionary objectToFlush = getObjectToFlush();

flushInfo.addIntProperty(NUM\_TERMS\_PROP\_NAME, objectToFlush.getNumTerms());

out.writePackedInts(objectToFlush.termPointers);

objectToFlush.termPool.getFlushHandler().flush(flushInfo.newSubProperties("termPool"), out);

objectToFlush.termsHashFunction.getFlushHandler()

.flush(flushInfo.newSubProperties("termsHashFunction"), out);

}

@Override

protected MPHTermDictionary doLoad(FlushInfo flushInfo,

DataDeserializer in) throws IOException {

int numTerms = flushInfo.getIntProperty(NUM\_TERMS\_PROP\_NAME);

PackedInts.Reader termPointers = in.readPackedInts();

ByteBlockPool termPool = (new ByteBlockPool.FlushHandler()).load(

flushInfo.getSubProperties("termPool"), in);

BDZAlgorithm termsHashFunction = (new BDZAlgorithm.FlushHandler()).load(

flushInfo.getSubProperties("termsHashFunction"), in);

return new MPHTermDictionary(numTerms, termsHashFunction, termPointers,

termPool, termPointerEncoding);

}

}

}