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

import java.io.IOException;

import java.util.Arrays;

import org.apache.lucene.util.ArrayUtil;

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;

/\*\*

\* TermsArray provides information on each term in the posting list.

\*

\* It does not provide any concurrency guarantees. The writer must ensure that all updates are

\* visible to readers with an external memory barrier.

\*/

public class TermsArray implements Flushable {

private static final int BYTES\_PER\_POSTING = 5 \* Integer.BYTES;

public static final int INVALID = -1;

private final int size;

public final int[] termPointers;

private final int[] postingsPointers;

// Derived data. Not atomic and not reliable.

public final int[] largestPostings;

public final int[] documentFrequency;

public final int[] offensiveCounters;

TermsArray(int size, boolean useOffensiveCounters) {

this.size = size;

termPointers = new int[size];

postingsPointers = new int[size];

largestPostings = new int[size];

documentFrequency = new int[size];

if (useOffensiveCounters) {

offensiveCounters = new int[size];

} else {

offensiveCounters = null;

}

Arrays.fill(postingsPointers, INVALID);

Arrays.fill(largestPostings, INVALID);

}

private TermsArray(TermsArray oldArray, int newSize) {

this(newSize, oldArray.offensiveCounters != null);

copyFrom(oldArray);

}

private TermsArray(

int size,

int[] termPointers,

int[] postingsPointers,

int[] largestPostings,

int[] documentFrequency,

int[] offensiveCounters) {

this.size = size;

this.termPointers = termPointers;

this.postingsPointers = postingsPointers;

this.largestPostings = largestPostings;

this.documentFrequency = documentFrequency;

this.offensiveCounters = offensiveCounters;

}

TermsArray grow() {

int newSize = ArrayUtil.oversize(size + 1, BYTES\_PER\_POSTING);

return new TermsArray(this, newSize);

}

private void copyFrom(TermsArray from) {

copy(from.termPointers, termPointers);

copy(from.postingsPointers, postingsPointers);

copy(from.largestPostings, largestPostings);

copy(from.documentFrequency, documentFrequency);

if (from.offensiveCounters != null) {

copy(from.offensiveCounters, offensiveCounters);

}

}

private void copy(int[] from, int[] to) {

System.arraycopy(from, 0, to, 0, from.length);

}

/\*\*

\* Returns the size of this array.

\*/

public int getSize() {

return size;

}

/\*\*

\* Write side operation for updating the pointer to the last posting for a given term.

\*/

public void updatePostingsPointer(int termID, int newPointer) {

postingsPointers[termID] = newPointer;

}

/\*\*

\* The returned pointer is guaranteed to be memory safe to follow to its target. The data

\* structure it points to will be consistent and safe to traverse. The posting list may contain

\* doc IDs that the current reader should not see, and the reader should skip over these doc IDs

\* to ensure that the readers provide an immutable view of the doc IDs in a posting list.

\*/

public int getPostingsPointer(int termID) {

return postingsPointers[termID];

}

public int[] getDocumentFrequency() {

return documentFrequency;

}

/\*\*

\* Gets the array containing the first posting for each indexed term.

\*/

public int[] getLargestPostings() {

return largestPostings;

}

@SuppressWarnings("unchecked")

@Override

public FlushHandler getFlushHandler() {

return new FlushHandler(this);

}

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

private static final String SIZE\_PROP\_NAME = "size";

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

public FlushHandler(TermsArray objectToFlush) {

super(objectToFlush);

}

public FlushHandler() {

}

@Override

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

TermsArray objectToFlush = getObjectToFlush();

flushInfo.addIntProperty(SIZE\_PROP\_NAME, objectToFlush.size);

boolean hasOffensiveCounters = objectToFlush.offensiveCounters != null;

flushInfo.addBooleanProperty(HAS\_OFFENSIVE\_COUNTERS\_PROP\_NAME, hasOffensiveCounters);

out.writeIntArray(objectToFlush.termPointers);

out.writeIntArray(objectToFlush.postingsPointers);

out.writeIntArray(objectToFlush.largestPostings);

out.writeIntArray(objectToFlush.documentFrequency);

if (hasOffensiveCounters) {

out.writeIntArray(objectToFlush.offensiveCounters);

}

}

@Override

protected TermsArray doLoad(

FlushInfo flushInfo, DataDeserializer in) throws IOException {

int size = flushInfo.getIntProperty(SIZE\_PROP\_NAME);

boolean hasOffensiveCounters = flushInfo.getBooleanProperty(HAS\_OFFENSIVE\_COUNTERS\_PROP\_NAME);

int[] termPointers = in.readIntArray();

int[] postingsPointers = in.readIntArray();

int[] largestPostings = in.readIntArray();

int[] documentFrequency = in.readIntArray();

int[] offensiveCounters = hasOffensiveCounters ? in.readIntArray() : null;

return new TermsArray(

size,

termPointers,

postingsPointers,

largestPostings,

documentFrequency,

offensiveCounters);

}

}

}