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

import com.google.common.base.Preconditions;

import org.apache.lucene.index.PostingsEnum;

import org.apache.lucene.util.BytesRef;

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

import static com.twitter.search.core.earlybird.index.inverted.SkipListContainer.INVALID\_POSITION;

/\*\*

\* TermDocs enumerator used by {@link SkipListPostingList}.

\*/

public class SkipListPostingsEnum extends PostingsEnum {

/\*\* Initialize cur doc ID and frequency. \*/

private int curDoc = TermsArray.INVALID;

private int curFreq = 0;

private final int postingPointer;

private final int cost;

/\*\*

\* maxPublishedPointer exists to prevent us from returning documents that are partially indexed.

\* These pointers are safe to follow, but the documents should not be returned. See

\* {@link EarlybirdRealtimeIndexSegmentData#getSyncData()} ()}.

\*/

private final int maxPublishedPointer;

/\*\* Skip list info and search key \*/

private final SkipListContainer<SkipListPostingList.Key> skiplist;

private final SkipListPostingList.Key key = new SkipListPostingList.Key();

/\*\*

\* Pointer/posting/docID of next posting in the skip list.

\* Notice the next here is relative to last posting with curDoc ID.

\*/

private int nextPostingPointer;

private int nextPostingDocID;

/\*\*

\* We save the positionPointer because we must walk the posting list to obtain term frequency

\* before we can start iterating through document positions. To do that walk, we increment

\* postingsPointer until it points to the first posting for the next doc, so postingsPointer is no

\* longer what we want to use as the start of the position list. The position pointer starts out

\* pointing to the first posting with that doc ID value. There can be duplicate doc ID values with

\* different positions. To find subsequent positions, we simply walk the posting list using this

\* pointer.

\*/

private int positionPointer = -1;

/\*\*

\* The payloadPointer should only be called after calling nextPosition, as it points to a payload

\* for each position. It is not updated unless nextPosition is called.

\*/

private int payloadPointer = -1;

/\*\* Search finger used in advance method. \*/

private final SkipListSearchFinger advanceSearchFinger;

/\*\*

\* A new {@link PostingsEnum} for a real-time skip list-based posting list.

\*/

public SkipListPostingsEnum(

int postingPointer,

int docFreq,

int maxPublishedPointer,

SkipListContainer<SkipListPostingList.Key> skiplist) {

this.postingPointer = postingPointer;

this.skiplist = skiplist;

this.advanceSearchFinger = this.skiplist.buildSearchFinger();

this.maxPublishedPointer = maxPublishedPointer;

this.nextPostingPointer = postingPointer;

// WARNING:

// docFreq is approximate and may not be the true document frequency of the posting list.

this.cost = docFreq;

if (postingPointer != -1) {

// Because the posting pointer is not negative 1, we know it's valid.

readNextPosting();

}

advanceSearchFinger.reset();

}

@Override

public final int nextDoc() {

// Notice if skip list is exhausted nextPostingPointer will point back to postingPointer since

// skip list is circle linked.

if (nextPostingPointer == postingPointer) {

// Skip list is exhausted.

curDoc = NO\_MORE\_DOCS;

curFreq = 0;

} else {

// Skip list is not exhausted.

curDoc = nextPostingDocID;

curFreq = 1;

positionPointer = nextPostingPointer;

// Keep reading all the posting with the same doc ID.

// Notice:

// - posting with the same doc ID will be stored consecutively

// since the skip list is sorted.

// - if skip list is exhausted, nextPostingPointer will become postingPointer

// since skip list is circle linked.

readNextPosting();

while (nextPostingPointer != postingPointer && nextPostingDocID == curDoc) {

curFreq++;

readNextPosting();

}

}

// Returned updated curDoc.

return curDoc;

}

/\*\*

\* Moves the enumerator forward by one element, then reads the information at that position.

\* \*/

private void readNextPosting() {

// Move search finger forward at lowest level.

advanceSearchFinger.setPointer(0, nextPostingPointer);

// Read next posting pointer.

nextPostingPointer = skiplist.getNextPointer(nextPostingPointer);

// Read the new posting positioned under nextPostingPointer into the nextPostingDocID.

readNextPostingInfo();

}

private boolean isPointerPublished(int pointer) {

return pointer <= maxPublishedPointer;

}

/\*\* Read next posting and doc id encoded in next posting. \*/

private void readNextPostingInfo() {

// We need to skip over every pointer that has not been published to this Enum, otherwise the

// searcher will see unpublished documents. We also end termination if we reach

// nextPostingPointer == postingPointer, because that means we have reached the end of the

// skiplist.

while (!isPointerPublished(nextPostingPointer) && nextPostingPointer != postingPointer) {

// Move search finger forward at lowest level.

advanceSearchFinger.setPointer(0, nextPostingPointer);

// Read next posting pointer.

nextPostingPointer = skiplist.getNextPointer(nextPostingPointer);

}

// Notice if skip list is exhausted, nextPostingPointer will be postingPointer

// since skip list is circle linked.

if (nextPostingPointer != postingPointer) {

nextPostingDocID = skiplist.getValue(nextPostingPointer);

} else {

nextPostingDocID = NO\_MORE\_DOCS;

}

}

/\*\*

\* Jump to the target, then use {@link #nextDoc()} to collect nextDoc info.

\* Notice target might be smaller than curDoc or smallestDocID.

\*/

@Override

public final int advance(int target) {

if (target == NO\_MORE\_DOCS) {

// Exhaust the posting list, so that future calls to docID() always return NO\_MORE\_DOCS.

nextPostingPointer = postingPointer;

}

if (nextPostingPointer == postingPointer) {

// Call nextDoc to ensure that all values are updated and we don't have to duplicate that

// here.

return nextDoc();

}

// Jump to target if target is bigger.

if (target >= curDoc && target >= nextPostingDocID) {

jumpToTarget(target);

}

// Retrieve next doc.

return nextDoc();

}

/\*\*

\* Set the next posting pointer (and info) to the first posting

\* with doc ID equal to or larger than the target.

\*

\* Notice this method does not set curDoc or curFreq.

\*/

private void jumpToTarget(int target) {

// Do a ceil search.

nextPostingPointer = skiplist.searchCeil(

key.withDocAndPosition(target, INVALID\_POSITION), postingPointer, advanceSearchFinger);

// Read next posting information.

readNextPostingInfo();

}

@Override

public int nextPosition() {

// If doc ID is equal to no more docs than we are past the end of the posting list. If doc ID

// is invalid, then we have not called nextDoc yet, and we should not return a real position.

// If the position pointer is past the current doc ID, then we should not return a position

// until nextDoc is called again (we don't want to return positions for a different doc).

if (docID() == NO\_MORE\_DOCS

|| docID() == TermsArray.INVALID

|| skiplist.getValue(positionPointer) != docID()) {

return INVALID\_POSITION;

}

payloadPointer = positionPointer;

int position = skiplist.getPosition(positionPointer);

do {

positionPointer = skiplist.getNextPointer(positionPointer);

} while (!isPointerPublished(positionPointer) && positionPointer != postingPointer);

return position;

}

@Override

public BytesRef getPayload() {

if (skiplist.getHasPayloads() == SkipListContainer.HasPayloads.NO) {

return null;

}

int pointer = skiplist.getPayloadPointer(this.payloadPointer);

Preconditions.checkState(pointer > 0);

return PayloadUtil.decodePayload(skiplist.getBlockPool(), pointer);

}

@Override

public int startOffset() {

return -1;

}

@Override

public int endOffset() {

return -1;

}

@Override

public final int docID() {

return curDoc;

}

@Override

public final int freq() {

return curFreq;

}

@Override

public long cost() {

return cost;

}

}