package com.twitter.search.earlybird.index;

import java.io.IOException;

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.DocIDToTweetIDMapper;

import it.unimi.dsi.fastutil.longs.Long2IntMap;

import it.unimi.dsi.fastutil.longs.Long2IntOpenHashMap;

import it.unimi.dsi.fastutil.longs.LongArrays;

/\*\*

\* After a segment is complete, we call {@link EarlybirdSegment#optimizeIndexes()} to compact the

\* doc IDs assigned to the tweets in this segment, so that we can do faster ceil and floor lookups.

\*/

public class OptimizedTweetIDMapper extends TweetIDMapper {

// Maps doc IDs to tweet IDs. Therefore, it should be sorted in descending order of tweet IDs.

protected final long[] inverseMap;

private final Long2IntMap tweetIdToDocIdMap;

private OptimizedTweetIDMapper(long[] inverseMap,

long minTweetID,

long maxTweetID,

int minDocID,

int maxDocID) {

super(minTweetID, maxTweetID, minDocID, maxDocID, inverseMap.length);

this.inverseMap = inverseMap;

this.tweetIdToDocIdMap = buildTweetIdToDocIdMap();

}

public OptimizedTweetIDMapper(OutOfOrderRealtimeTweetIDMapper source) throws IOException {

super(source.getMinTweetID(),

source.getMaxTweetID(),

0,

source.getNumDocs() - 1,

source.getNumDocs());

inverseMap = source.sortTweetIds();

tweetIdToDocIdMap = buildTweetIdToDocIdMap();

}

private Long2IntMap buildTweetIdToDocIdMap() {

int[] values = new int[inverseMap.length];

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

values[i] = i;

}

Long2IntMap map = new Long2IntOpenHashMap(inverseMap, values);

map.defaultReturnValue(-1);

return map;

}

@Override

public int getDocID(long tweetID) {

return tweetIdToDocIdMap.getOrDefault(tweetID, ID\_NOT\_FOUND);

}

@Override

protected int getNextDocIDInternal(int docID) {

// The doc IDs are consecutive and TweetIDMapper already checked the boundary conditions.

return docID + 1;

}

@Override

protected int getPreviousDocIDInternal(int docID) {

// The doc IDs are consecutive and TweetIDMapper already checked the boundary conditions.

return docID - 1;

}

@Override

public long getTweetID(int internalID) {

return inverseMap[internalID];

}

@Override

protected int findDocIDBoundInternal(long tweetID, boolean findMaxDocID) {

int docId = tweetIdToDocIdMap.get(tweetID);

if (docId >= 0) {

return docId;

}

int binarySearchResult =

LongArrays.binarySearch(inverseMap, tweetID, (k1, k2) -> -Long.compare(k1, k2));

// Since the tweet ID is not present in this mapper, the binary search should return a negative

// value (-insertionPoint - 1). And since TweetIDMapper.findDocIdBound() already verified that

// tweetID is not smaller than all tweet IDs in this mapper, and not larger than all tweet IDs

// in this mapper, the insertionPoint should never be 0 or inverseMap.length.

int insertionPoint = -binarySearchResult - 1;

// The insertion point is the index in the tweet array of the upper bound of the search, so if

// we want the lower bound, because doc IDs are dense, we subtract one.

return findMaxDocID ? insertionPoint : insertionPoint - 1;

}

@Override

protected final int addMappingInternal(final long tweetID) {

throw new UnsupportedOperationException("The OptimizedTweetIDMapper is immutable.");

}

@Override

public DocIDToTweetIDMapper optimize() {

throw new UnsupportedOperationException("OptimizedTweetIDMapper is already optimized.");

}

@Override

public FlushHandler getFlushHandler() {

return new FlushHandler(this);

}

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

private static final String MIN\_TWEET\_ID\_PROP\_NAME = "MinTweetID";

private static final String MAX\_TWEET\_ID\_PROP\_NAME = "MaxTweetID";

private static final String MIN\_DOC\_ID\_PROP\_NAME = "MinDocID";

private static final String MAX\_DOC\_ID\_PROP\_NAME = "MaxDocID";

public FlushHandler() {

super();

}

public FlushHandler(OptimizedTweetIDMapper objectToFlush) {

super(objectToFlush);

}

@Override

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

OptimizedTweetIDMapper objectToFlush = getObjectToFlush();

flushInfo.addLongProperty(MIN\_TWEET\_ID\_PROP\_NAME, objectToFlush.getMinTweetID());

flushInfo.addLongProperty(MAX\_TWEET\_ID\_PROP\_NAME, objectToFlush.getMaxTweetID());

flushInfo.addIntProperty(MIN\_DOC\_ID\_PROP\_NAME, objectToFlush.getMinDocID());

flushInfo.addIntProperty(MAX\_DOC\_ID\_PROP\_NAME, objectToFlush.getMaxDocID());

out.writeLongArray(objectToFlush.inverseMap);

}

@Override

protected OptimizedTweetIDMapper doLoad(FlushInfo flushInfo, DataDeserializer in)

throws IOException {

return new OptimizedTweetIDMapper(in.readLongArray(),

flushInfo.getLongProperty(MIN\_TWEET\_ID\_PROP\_NAME),

flushInfo.getLongProperty(MAX\_TWEET\_ID\_PROP\_NAME),

flushInfo.getIntProperty(MIN\_DOC\_ID\_PROP\_NAME),

flushInfo.getIntProperty(MAX\_DOC\_ID\_PROP\_NAME));

}

}

}