package com.twitter.search.earlybird.index;

import java.io.IOException;

import java.util.Arrays;

import com.google.common.base.Preconditions;

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 com.twitter.search.core.earlybird.index.TimeMapper;

import com.twitter.search.core.earlybird.index.inverted.IntBlockPool;

/\*\*

\* A TimeMapper implementation that stores the timestamps associated with the doc IDs in an array.

\*/

public class OptimizedTimeMapper extends AbstractInMemoryTimeMapper implements Flushable {

// Doc id to timestamp map. Timestamps that are negative are out-of-order.

protected final int[] timeMap;

// Size must be greater than the max doc ID stored in the optimized tweet ID mapper.

public OptimizedTimeMapper(RealtimeTimeMapper realtimeTimeMapper,

DocIDToTweetIDMapper originalTweetIdMapper,

DocIDToTweetIDMapper optimizedTweetIdMapper) throws IOException {

super();

int maxDocId = optimizedTweetIdMapper.getPreviousDocID(Integer.MAX\_VALUE);

timeMap = new int[maxDocId + 1];

Arrays.fill(timeMap, ILLEGAL\_TIME);

int docId = maxDocId;

while (docId != DocIDToTweetIDMapper.ID\_NOT\_FOUND) {

int originalDocId = originalTweetIdMapper.getDocID(optimizedTweetIdMapper.getTweetID(docId));

Preconditions.checkState(originalDocId != DocIDToTweetIDMapper.ID\_NOT\_FOUND);

int docIdTimestamp = realtimeTimeMapper.getTime(originalDocId);

Preconditions.checkState(docIdTimestamp != TimeMapper.ILLEGAL\_TIME);

doAddMapping(docId, docIdTimestamp);

docId = optimizedTweetIdMapper.getPreviousDocID(docId);

}

}

private OptimizedTimeMapper(int[] timeMap,

int reverseMapLastIndex,

IntBlockPool reverseMapTimes,

IntBlockPool reverseMapIds) {

super(reverseMapLastIndex, reverseMapTimes, reverseMapIds);

this.timeMap = timeMap;

}

@Override

public int getTime(int docID) {

return timeMap[docID];

}

@Override

protected void setTime(int docID, int timeSeconds) {

timeMap[docID] = timeSeconds;

}

@Override

public FlushHandler getFlushHandler() {

return new FlushHandler(this);

}

public static final class FlushHandler extends Flushable.Handler<OptimizedTimeMapper> {

private static final String REVERSE\_MAP\_LAST\_INDEX\_PROP = "reverseMapLastIndex";

private static final String TIMES\_SUB\_PROP = "times";

private static final String IDS\_SUB\_PROP = "ids";

public FlushHandler() {

super();

}

public FlushHandler(OptimizedTimeMapper objectToFlush) {

super(objectToFlush);

}

@Override

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

OptimizedTimeMapper mapper = getObjectToFlush();

out.writeIntArray(mapper.timeMap);

flushInfo.addIntProperty(REVERSE\_MAP\_LAST\_INDEX\_PROP, mapper.reverseMapLastIndex);

mapper.reverseMapTimes.getFlushHandler().flush(

flushInfo.newSubProperties(TIMES\_SUB\_PROP), out);

mapper.reverseMapIds.getFlushHandler().flush(

flushInfo.newSubProperties(IDS\_SUB\_PROP), out);

}

@Override

protected OptimizedTimeMapper doLoad(FlushInfo flushInfo, DataDeserializer in)

throws IOException {

return new OptimizedTimeMapper(

in.readIntArray(),

flushInfo.getIntProperty(REVERSE\_MAP\_LAST\_INDEX\_PROP),

new IntBlockPool.FlushHandler().load(flushInfo.getSubProperties(TIMES\_SUB\_PROP), in),

new IntBlockPool.FlushHandler().load(flushInfo.getSubProperties(IDS\_SUB\_PROP), in));

}

}

@Override

public TimeMapper optimize(DocIDToTweetIDMapper originalTweetIdMapper,

DocIDToTweetIDMapper optimizedTweetIdMapper) {

throw new UnsupportedOperationException("OptimizedTimeMapper instances are already optimized.");

}

}