package com.twitter.search.earlybird.index;

import java.io.IOException;

import com.google.common.annotations.VisibleForTesting;

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;

import it.unimi.dsi.fastutil.ints.Int2IntMap;

import it.unimi.dsi.fastutil.ints.Int2IntOpenHashMap;

/\*\*

\* Maps 32-bit document IDs to seconds-since-epoch timestamps.

\*/

public class RealtimeTimeMapper extends AbstractInMemoryTimeMapper {

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

protected final Int2IntOpenHashMap timeMap;

private final int capacity;

public RealtimeTimeMapper(int capacity) {

super();

this.capacity = capacity;

timeMap = new Int2IntOpenHashMap(capacity);

timeMap.defaultReturnValue(ILLEGAL\_TIME);

}

@Override

public int getTime(int docID) {

return timeMap.get(docID);

}

@Override

protected void setTime(int docID, int timeSeconds) {

timeMap.put(docID, timeSeconds);

}

public final void addMapping(int docID, int timeSeconds) {

doAddMapping(docID, timeSeconds);

}

@Override

public TimeMapper optimize(DocIDToTweetIDMapper originalTweetIdMapper,

DocIDToTweetIDMapper optimizedTweetIdMapper) throws IOException {

return new OptimizedTimeMapper(this, originalTweetIdMapper, optimizedTweetIdMapper);

}

/\*\*

\* Evaluates whether two instances of RealtimeTimeMapper are equal by value. It is

\* slow because it has to check every tweet ID/timestamp in the map.

\*/

@VisibleForTesting

boolean verySlowEqualsForTests(RealtimeTimeMapper that) {

return reverseMapLastIndex == that.reverseMapLastIndex

&& reverseMapIds.verySlowEqualsForTests(that.reverseMapIds)

&& reverseMapTimes.verySlowEqualsForTests(that.reverseMapTimes)

&& capacity == that.capacity

&& timeMap.equals(that.timeMap);

}

private RealtimeTimeMapper(

int capacity,

int reverseMapLastIndex,

int[] docIds,

int[] timestamps,

IntBlockPool reverseMapTimes,

IntBlockPool reverseMapIds

) {

super(reverseMapLastIndex, reverseMapTimes, reverseMapIds);

this.capacity = capacity;

timeMap = new Int2IntOpenHashMap(capacity);

timeMap.defaultReturnValue(ILLEGAL\_TIME);

Preconditions.checkState(docIds.length == timestamps.length);

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

timeMap.put(docIds[i], timestamps[i]);

}

}

@Override

public RealtimeTimeMapper.FlushHandler getFlushHandler() {

return new RealtimeTimeMapper.FlushHandler(this);

}

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

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";

private static final String CAPACITY\_PROP = "capacity";

public FlushHandler() {

super();

}

public FlushHandler(RealtimeTimeMapper objectToFlush) {

super(objectToFlush);

}

@Override

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

RealtimeTimeMapper mapper = getObjectToFlush();

flushInfo.addIntProperty(CAPACITY\_PROP, mapper.capacity);

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

serializer.writeInt(mapper.timeMap.size());

for (Int2IntMap.Entry entry : mapper.timeMap.int2IntEntrySet()) {

serializer.writeInt(entry.getIntKey());

serializer.writeInt(entry.getIntValue());

}

mapper.reverseMapTimes.getFlushHandler().flush(

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

mapper.reverseMapIds.getFlushHandler().flush(

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

}

@Override

protected RealtimeTimeMapper doLoad(FlushInfo flushInfo, DataDeserializer in)

throws IOException {

int size = in.readInt();

int[] docIds = new int[size];

int[] timestamps = new int[size];

for (int i = 0; i < size; i++) {

docIds[i] = in.readInt();

timestamps[i] = in.readInt();

}

return new RealtimeTimeMapper(

flushInfo.getIntProperty(CAPACITY\_PROP),

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

docIds,

timestamps,

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

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

}

}

}