package com.twitter.search.earlybird.index;

import java.io.IOException;

import java.util.Arrays;

import com.twitter.search.common.partitioning.snowflakeparser.SnowflakeIdParser;

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;

public final class TweetIDToInternalIDMap implements Flushable {

private final int size;

private final int[] hash;

public final int halfSize;

private final int mask;

public int numMappings;

static final int PRIME\_NUMBER = 37;

// For FlushHandler.load() use only

private TweetIDToInternalIDMap(final int[] hash,

final int numMappings) {

this.hash = hash;

this.size = hash.length;

this.halfSize = size >> 1;

this.mask = size - 1;

this.numMappings = numMappings;

}

TweetIDToInternalIDMap(final int size) {

this.hash = new int[size];

Arrays.fill(hash, DocIDToTweetIDMapper.ID\_NOT\_FOUND);

this.size = size;

this.halfSize = size >> 1;

this.mask = size - 1;

this.numMappings = 0;

}

// Slightly different hash function from the one used to partition tweets to Earlybirds.

protected static int hashCode(final long tweetID) {

long timestamp = SnowflakeIdParser.getTimestampFromTweetId(tweetID);

int code = (int) ((timestamp - 1) ^ (timestamp >>> 32));

code = PRIME\_NUMBER \* (int) (tweetID & SnowflakeIdParser.RESERVED\_BITS\_MASK) + code;

return code;

}

protected static int incrementHashCode(int code) {

return ((code >> 8) + code) | 1;

}

private int hashPos(int code) {

return code & mask;

}

/\*\*

\* Associates the given tweet ID with the given internal doc ID.

\*

\* @param tweetID The tweet ID.

\* @param internalID The doc ID that should be associated with this tweet ID.

\* @param inverseMap The map that stores the doc ID to tweet ID associations.

\*/

public void add(final long tweetID, final int internalID, final long[] inverseMap) {

int code = hashCode(tweetID);

int hashPos = hashPos(code);

int value = hash[hashPos];

assert inverseMap[internalID] == tweetID;

if (value != DocIDToTweetIDMapper.ID\_NOT\_FOUND) {

final int inc = incrementHashCode(code);

do {

code += inc;

hashPos = hashPos(code);

value = hash[hashPos];

} while (value != DocIDToTweetIDMapper.ID\_NOT\_FOUND);

}

assert value == DocIDToTweetIDMapper.ID\_NOT\_FOUND;

hash[hashPos] = internalID;

numMappings++;

}

/\*\*

\* Returns the doc ID corresponding to the given tweet ID.

\*

\* @param tweetID The tweet ID.

\* @param inverseMap The map that stores the doc ID to tweet ID associations.

\* @return The doc ID corresponding to the given tweet ID.

\*/

public int get(long tweetID, final long[] inverseMap) {

int code = hashCode(tweetID);

int hashPos = hashPos(code);

int value = hash[hashPos];

if (value != DocIDToTweetIDMapper.ID\_NOT\_FOUND && inverseMap[value] != tweetID) {

final int inc = incrementHashCode(code);

do {

code += inc;

hashPos = hashPos(code);

value = hash[hashPos];

} while (value != DocIDToTweetIDMapper.ID\_NOT\_FOUND && inverseMap[value] != tweetID);

}

if (hashPos == -1) {

return DocIDToTweetIDMapper.ID\_NOT\_FOUND;

}

return hash[hashPos];

}

@Override

public TweetIDToInternalIDMap.FlushHandler getFlushHandler() {

return new FlushHandler(this);

}

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

public FlushHandler() {

super();

}

private static final String HASH\_ARRAY\_SIZE\_PROP\_NAME = "HashArraySize";

private static final String MASK\_PROP\_NAME = "Mask";

private static final String NUM\_MAPPINGS\_PROP\_NAME = "NumMappings";

public FlushHandler(TweetIDToInternalIDMap objectToFlush) {

super(objectToFlush);

}

@Override

protected void doFlush(FlushInfo flushInfo, DataSerializer out)

throws IOException {

TweetIDToInternalIDMap mapper = getObjectToFlush();

flushInfo

.addIntProperty(HASH\_ARRAY\_SIZE\_PROP\_NAME, mapper.hash.length)

.addIntProperty(MASK\_PROP\_NAME, mapper.mask)

.addIntProperty(NUM\_MAPPINGS\_PROP\_NAME, mapper.numMappings);

out.writeIntArray(mapper.hash);

}

@Override

protected TweetIDToInternalIDMap doLoad(FlushInfo flushInfo, DataDeserializer in)

throws IOException {

final int[] hash = in.readIntArray();

final int numMappings = flushInfo.getIntProperty(NUM\_MAPPINGS\_PROP\_NAME);

return new TweetIDToInternalIDMap(hash, numMappings);

}

}

}