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

import java.util.Arrays;

import java.util.HashMap;

import java.util.List;

import java.util.OptionalInt;

import java.util.concurrent.TimeUnit;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Stopwatch;

import com.google.common.collect.ImmutableList;

import com.google.common.collect.Maps;

import org.apache.lucene.util.BytesRef;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.search.common.metrics.SearchTimerStats;

import com.twitter.search.common.util.LogFormatUtil;

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

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

/\*\*

\* This implementation took MultiSegmentTermDictionaryWithMap and replaced some of the

\* data structures with fastutil equivalents and it also uses a more memory efficient way to

\* store the precomputed data.

\*

\* This implementation has a requirement that each term per field needs to be present at

\* most once per document, since we only have space to index 2^24 terms and we have 2^23

\* documents as of now in realtime earlybirds.

\*

\* See UserIdMultiSegmentQuery class comment for more information on how this is used.

\*/

public class MultiSegmentTermDictionaryWithFastutil implements MultiSegmentTermDictionary {

private static final Logger LOG = LoggerFactory.getLogger(

MultiSegmentTermDictionaryWithFastutil.class);

@VisibleForTesting

public static final SearchTimerStats TERM\_DICTIONARY\_CREATION\_STATS =

SearchTimerStats.export("multi\_segment\_term\_dictionary\_with\_fastutil\_creation",

TimeUnit.MILLISECONDS, false);

private static final int MAX\_TERM\_ID\_BITS = 24;

private static final int TERM\_ID\_MASK = (1 << MAX\_TERM\_ID\_BITS) - 1; // First 24 bits.

private static final int MAX\_SEGMENT\_SIZE = 1 << (MAX\_TERM\_ID\_BITS - 1);

private final ImmutableList<OptimizedMemoryIndex> indexes;

// For each term, a list of (index id, term id) packed into an integer.

// The integer contains:

// byte 0: index (segment id). Since we have ~20 segments, this fits into a byte.

// bytes [1-3]: term id. The terms we're building this dictionary for are user ids

// associated with a tweet - from\_user\_id and in\_reply\_to\_user\_id. Since we have

// at most 2\*\*23 tweets in realtime, we'll have at most 2\*\*23 unique terms per

// segments. The term ids post optimization are consecutive numbers, so they will

// fit in 24 bits. We don't use the term dictionary in archive, which has more

// tweets per segment.

//

// To verify the maximum amount of tweets in a segment, see max\_segment\_size in

// earlybird-config.yml.

private final HashMap<BytesRef, IntArrayList> termsMap;

private final int numTerms;

private final int numTermEntries;

int encodeIndexAndTermId(int indexId, int termId) {

// Push the index id to the left and use the other 24 bits for the term id.

return (indexId << MAX\_TERM\_ID\_BITS) | termId;

}

void decodeIndexAndTermId(int[] arr, int packed) {

arr[packed >> MAX\_TERM\_ID\_BITS] = packed & TERM\_ID\_MASK;

}

/\*\*

\* Creates a new multi-segment term dictionary backed by a regular java map.

\*/

public MultiSegmentTermDictionaryWithFastutil(

String field,

List<OptimizedMemoryIndex> indexes) {

this.indexes = ImmutableList.copyOf(indexes);

// Pre-size the map with estimate of max number of terms. It should be at least that big.

OptionalInt optionalMax = indexes.stream().mapToInt(OptimizedMemoryIndex::getNumTerms).max();

int maxNumTerms = optionalMax.orElse(0);

this.termsMap = Maps.newHashMapWithExpectedSize(maxNumTerms);

LOG.info("About to merge {} indexes for field {}, estimated {} terms",

indexes.size(), field, LogFormatUtil.formatInt(maxNumTerms));

Stopwatch stopwatch = Stopwatch.createStarted();

BytesRef termBytesRef = new BytesRef();

for (int indexId = 0; indexId < indexes.size(); indexId++) {

// The inverted index for this field.

OptimizedMemoryIndex index = indexes.get(indexId);

int indexNumTerms = index.getNumTerms();

if (indexNumTerms > MAX\_SEGMENT\_SIZE) {

throw new IllegalStateException("too many terms: " + indexNumTerms);

}

for (int termId = 0; termId < indexNumTerms; termId++) {

index.getTerm(termId, termBytesRef);

IntArrayList indexTerms = termsMap.get(termBytesRef);

if (indexTerms == null) {

BytesRef term = BytesRef.deepCopyOf(termBytesRef);

indexTerms = new IntArrayList();

termsMap.put(term, indexTerms);

}

indexTerms.add(encodeIndexAndTermId(indexId, termId));

}

}

this.numTerms = termsMap.size();

this.numTermEntries = indexes.stream().mapToInt(OptimizedMemoryIndex::getNumTerms).sum();

TERM\_DICTIONARY\_CREATION\_STATS.timerIncrement(stopwatch.elapsed(TimeUnit.MILLISECONDS));

LOG.info("Done merging {} segments for field {} in {} - "

+ "num terms: {}, num term entries: {}.",

indexes.size(), field, stopwatch,

LogFormatUtil.formatInt(this.numTerms),

LogFormatUtil.formatInt(this.numTermEntries));

}

@Override

public int[] lookupTermIds(BytesRef term) {

int[] termIds = new int[indexes.size()];

Arrays.fill(termIds, EarlybirdIndexSegmentAtomicReader.TERM\_NOT\_FOUND);

IntArrayList indexTerms = termsMap.get(term);

if (indexTerms != null) {

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

decodeIndexAndTermId(termIds, indexTerms.getInt(i));

}

}

return termIds;

}

@Override

public ImmutableList<? extends InvertedIndex> getSegmentIndexes() {

return indexes;

}

@Override

public int getNumTerms() {

return this.numTerms;

}

@Override

public int getNumTermEntries() {

return this.numTermEntries;

}

}