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.collect.ImmutableList;

import com.google.common.collect.Lists;

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;

/\*\*

\* A rather simple implementation of a MultiSegmentTermDictionary that just keeps all terms in a

\* java hash map, and all the termIds for a term in a java list.

\*

\* An alternate implementation could have an MPH for the map, and a IntBlockPool for storing

\* the term ids.

\*

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

\*/

public class MultiSegmentTermDictionaryWithMap implements MultiSegmentTermDictionary {

private static final Logger LOG = LoggerFactory.getLogger(

MultiSegmentTermDictionaryWithMap.class);

@VisibleForTesting

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

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

TimeUnit.MILLISECONDS, false);

private final ImmutableList<OptimizedMemoryIndex> indexes;

private final HashMap<BytesRef, List<IndexTerm>> termsMap;

private final int numTerms;

private final int numTermEntries;

private static class IndexTerm {

private int indexId;

private final int termId;

public IndexTerm(int indexId, int termId) {

this.indexId = indexId;

this.termId = termId;

}

}

/\*\*

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

\*/

public MultiSegmentTermDictionaryWithMap(

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));

long start = System.currentTimeMillis();

BytesRef termText = new BytesRef();

long copiedBytes = 0;

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

// The inverted index for this field.

OptimizedMemoryIndex index = indexes.get(indexId);

int indexNumTerms = index.getNumTerms();

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

index.getTerm(termId, termText);

// This copies the underlying array to a new array.

BytesRef term = BytesRef.deepCopyOf(termText);

copiedBytes += term.length;

List<IndexTerm> indexTerms = termsMap.computeIfAbsent(term, k -> Lists.newArrayList());

indexTerms.add(new IndexTerm(indexId, termId));

}

}

this.numTerms = termsMap.size();

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

long elapsed = System.currentTimeMillis() - start;

TERM\_DICTIONARY\_CREATION\_STATS.timerIncrement(elapsed);

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

+ "num terms: {}, num term entries: {}, copied bytes: {}",

indexes.size(), field, elapsed,

LogFormatUtil.formatInt(this.numTerms), LogFormatUtil.formatInt(this.numTermEntries),

LogFormatUtil.formatInt(copiedBytes));

}

@Override

public int[] lookupTermIds(BytesRef term) {

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

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

List<IndexTerm> indexTerms = termsMap.get(term);

if (indexTerms != null) {

for (IndexTerm indexTerm : indexTerms) {

termIds[indexTerm.indexId] = indexTerm.termId;

}

}

return termIds;

}

@Override

public ImmutableList<? extends InvertedIndex> getSegmentIndexes() {

return indexes;

}

@Override

public int getNumTerms() {

return this.numTerms;

}

@Override

public int getNumTermEntries() {

return this.numTermEntries;

}

}