package com.twitter.search.earlybird\_root.mergers;

import java.util.Collections;

import java.util.List;

import java.util.Map;

import java.util.Set;

import java.util.TreeMap;

import java.util.concurrent.TimeUnit;

import java.util.stream.Collectors;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Function;

import com.google.common.base.Preconditions;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.search.common.constants.thriftjava.ThriftLanguage;

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

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

import com.twitter.search.common.util.earlybird.EarlybirdResponseUtil;

import com.twitter.search.common.util.earlybird.ResultsUtil;

import com.twitter.search.earlybird.thrift.EarlybirdRequest;

import com.twitter.search.earlybird.thrift.EarlybirdResponse;

import com.twitter.search.earlybird.thrift.ThriftSearchQuery;

import com.twitter.search.earlybird.thrift.ThriftSearchRankingMode;

import com.twitter.search.earlybird.thrift.ThriftSearchResult;

import com.twitter.search.earlybird.thrift.ThriftSearchResults;

import com.twitter.search.earlybird\_root.collectors.RelevanceMergeCollector;

import com.twitter.search.earlybird\_root.common.EarlybirdFeatureSchemaMerger;

import com.twitter.search.earlybird\_root.common.EarlybirdRequestContext;

import com.twitter.util.Future;

/\*\*

\* Merger class to merge relevance search EarlybirdResponse objects

\*/

public class RelevanceResponseMerger extends EarlybirdResponseMerger {

private static final Logger LOG = LoggerFactory.getLogger(RelevanceResponseMerger.class);

private static final SearchTimerStats TIMER =

SearchTimerStats.export("merge\_relevance", TimeUnit.NANOSECONDS, false, true);

private static final SearchCounter RELVEANCE\_TIER\_MERGE\_EARLY\_TERMINATED\_WITH\_NOT\_ENOUGH\_RESULTS =

SearchCounter.export("merger\_relevance\_tier\_merge\_early\_terminated\_with\_not\_enough\_results");

private static final String PARTITION\_NUM\_RESULTS\_COUNTER\_SKIP\_STATS =

"merger\_relevance\_post\_trimmed\_results\_skip\_stat\_tier\_%s\_partition\_%d";

@VisibleForTesting

public static final String PARTITION\_NUM\_RESULTS\_COUNTER\_NAME\_FORMAT =

"merger\_relevance\_post\_trimmed\_results\_from\_tier\_%s\_partition\_%d";

protected static final Function<EarlybirdResponse, Map<ThriftLanguage, Integer>> LANG\_MAP\_GETTER =

response -> response.getSearchResults() == null

? null

: response.getSearchResults().getLanguageHistogram();

private static final double SUCCESSFUL\_RESPONSE\_THRESHOLD = 0.8;

private final EarlybirdFeatureSchemaMerger featureSchemaMerger;

// The number of partitions are not meaningful when it is invoked through multi-tier merging.

private final int numPartitions;

public RelevanceResponseMerger(EarlybirdRequestContext requestContext,

List<Future<EarlybirdResponse>> responses,

ResponseAccumulator mode,

EarlybirdFeatureSchemaMerger featureSchemaMerger,

int numPartitions) {

super(requestContext, responses, mode);

this.featureSchemaMerger = Preconditions.checkNotNull(featureSchemaMerger);

this.numPartitions = numPartitions;

}

@Override

protected double getDefaultSuccessResponseThreshold() {

return SUCCESSFUL\_RESPONSE\_THRESHOLD;

}

@Override

protected SearchTimerStats getMergedResponseTimer() {

return TIMER;

}

@Override

protected EarlybirdResponse internalMerge(EarlybirdResponse mergedResponse) {

final ThriftSearchQuery searchQuery = requestContext.getRequest().getSearchQuery();

long maxId = findMaxFullySearchedStatusID();

long minId = findMinFullySearchedStatusID();

Preconditions.checkNotNull(searchQuery);

Preconditions.checkState(searchQuery.isSetRankingMode());

Preconditions.checkState(searchQuery.getRankingMode() == ThriftSearchRankingMode.RELEVANCE);

// First get the results in score order (the default comparator for this merge collector).

RelevanceMergeCollector collector = new RelevanceMergeCollector(responses.size());

int totalResultSize = addResponsesToCollector(collector);

ThriftSearchResults searchResults = collector.getAllSearchResults();

TrimStats trimStats = trimResults(searchResults);

featureSchemaMerger.collectAndSetFeatureSchemaInResponse(

searchResults,

requestContext,

"merger\_relevance\_tier",

accumulatedResponses.getSuccessResponses());

mergedResponse.setSearchResults(searchResults);

searchResults = mergedResponse.getSearchResults();

searchResults

.setHitCounts(aggregateHitCountMap())

.setLanguageHistogram(aggregateLanguageHistograms());

if (!accumulatedResponses.getMaxIds().isEmpty()) {

searchResults.setMaxSearchedStatusID(maxId);

}

if (!accumulatedResponses.getMinIds().isEmpty()) {

searchResults.setMinSearchedStatusID(minId);

}

LOG.debug("Hits: {} Removed duplicates: {}", totalResultSize, trimStats.getRemovedDupsCount());

LOG.debug("Hash Partition'ed Earlybird call completed successfully: {}", mergedResponse);

publishNumResultsFromPartitionStatistics(mergedResponse);

return mergedResponse;

}

/\*\*

\* If any of the partitions has an early termination, the tier merge must also early terminate.

\*

\* If a partition early terminated (we haven't fully searched that partition), and we instead

\* moved onto the next tier, there will be a gap of unsearched results.

\*

\* If our early termination condition was only if we had enough results, we could get bad quality

\* results by only looking at 20 hits when asking for 20 results.

\*/

@Override

public boolean shouldEarlyTerminateTierMerge(int totalResultsFromSuccessfulShards,

boolean foundEarlyTermination) {

// Don't use computeNumResultsToKeep because if returnAllResults is true, it will be

// Integer.MAX\_VALUE and we will always log a stat that we didn't get enough results

int resultsRequested;

EarlybirdRequest request = requestContext.getRequest();

if (request.isSetNumResultsToReturnAtRoot()) {

resultsRequested = request.getNumResultsToReturnAtRoot();

} else {

resultsRequested = request.getSearchQuery().getCollectorParams().getNumResultsToReturn();

}

if (foundEarlyTermination && totalResultsFromSuccessfulShards < resultsRequested) {

RELVEANCE\_TIER\_MERGE\_EARLY\_TERMINATED\_WITH\_NOT\_ENOUGH\_RESULTS.increment();

}

return foundEarlyTermination;

}

/\*\*

\* Merge language histograms from all queries.

\*

\* @return Merge per-language count map.

\*/

private Map<ThriftLanguage, Integer> aggregateLanguageHistograms() {

Map<ThriftLanguage, Integer> totalLangCounts = new TreeMap<>(

ResultsUtil.aggregateCountMap(

accumulatedResponses.getSuccessResponses(), LANG\_MAP\_GETTER));

if (totalLangCounts.size() > 0) {

if (responseMessageBuilder.isDebugMode()) {

responseMessageBuilder.append("Language Distrbution:\n");

int count = 0;

for (Map.Entry<ThriftLanguage, Integer> entry : totalLangCounts.entrySet()) {

responseMessageBuilder.append(

String.format(" %10s:%6d", entry.getKey(), entry.getValue()));

if (++count % 5 == 0) {

responseMessageBuilder.append("\n");

}

}

responseMessageBuilder.append("\n");

}

}

return totalLangCounts;

}

/\*\*

\* Find the min status id that has been searched. Since no results are trimmed for Relevance mode,

\* it should be the smallest among the min IDs.

\*/

private long findMinFullySearchedStatusID() {

// The min ID should be the smallest among the min IDs

return accumulatedResponses.getMinIds().isEmpty() ? 0

: Collections.min(accumulatedResponses.getMinIds());

}

/\*\*

\* Find the max status id that has been searched. Since no results are trimmed for Relevance mode,

\* it should be the largest among the max IDs.

\*/

private long findMaxFullySearchedStatusID() {

// The max ID should be the largest among the max IDs

return accumulatedResponses.getMaxIds().isEmpty() ? 0

: Collections.max(accumulatedResponses.getMaxIds());

}

/\*\*

\* Return all the searchResults except duplicates.

\*

\* @param searchResults ThriftSearchResults that hold the to be trimmed List<ThriftSearchResult>

\* @return TrimStats containing statistics about how many results being removed

\*/

private TrimStats trimResults(ThriftSearchResults searchResults) {

if (!searchResults.isSetResults() || searchResults.getResultsSize() == 0) {

// no results, no trimming needed

return TrimStats.EMPTY\_STATS;

}

if (requestContext.getRequest().getSearchQuery().isSetSearchStatusIds()) {

// Not a normal search, no trimming needed

return TrimStats.EMPTY\_STATS;

}

TrimStats trimStats = new TrimStats();

trimExactDups(searchResults, trimStats);

truncateResults(searchResults, trimStats);

return trimStats;

}

private void publishNumResultsFromPartitionStatistics(EarlybirdResponse mergedResponse) {

// Keep track of all of the results that were kept after merging

Set<Long> mergedResults =

EarlybirdResponseUtil.getResults(mergedResponse).getResults()

.stream()

.map(result -> result.getId())

.collect(Collectors.toSet());

// For each successful response (pre merge), count how many of its results were kept post merge.

// Increment the appropriate stat.

for (EarlybirdResponse response : accumulatedResponses.getSuccessResponses()) {

if (!response.isSetEarlybirdServerStats()) {

continue;

}

int numResultsKept = 0;

for (ThriftSearchResult result

: EarlybirdResponseUtil.getResults(response).getResults()) {

if (mergedResults.contains(result.getId())) {

++numResultsKept;

}

}

// We only update partition stats when the partition ID looks sane.

String tierName = response.getEarlybirdServerStats().getTierName();

int partition = response.getEarlybirdServerStats().getPartition();

if (partition >= 0 && partition < numPartitions) {

SearchCounter.export(String.format(PARTITION\_NUM\_RESULTS\_COUNTER\_NAME\_FORMAT,

tierName,

partition))

.add(numResultsKept);

} else {

SearchCounter.export(String.format(PARTITION\_NUM\_RESULTS\_COUNTER\_SKIP\_STATS,

tierName,

partition)).increment();

}

}

}

}