package com.twitter.search.common.util.earlybird;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.concurrent.ExecutionException;

import com.google.common.base.Preconditions;

import com.google.common.cache.LoadingCache;

import com.google.common.collect.ImmutableMap;

import com.google.common.collect.Lists;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.collections.Pair;

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

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

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

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

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.ThriftTweetSource;

/\*\*

\* Utility methods to merge EarlybirdResponses.

\*/

public final class EarlybirdResponseMergeUtil {

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

private static final String INVALID\_RESPONSE\_STATS\_PREFIX = "invalid\_response\_stats\_";

// Stats for invalid earlybird response

private static final ImmutableMap<EarlybirdResponseCode, SearchCounter> ERROR\_EXCEPTIONS;

public static final SearchCounter NULL\_RESPONSE\_COUNTER =

SearchCounter.export(INVALID\_RESPONSE\_STATS\_PREFIX + "null\_response");

public static final SearchCounter SEARCH\_RESULTS\_NOT\_SET\_COUNTER =

SearchCounter.export(INVALID\_RESPONSE\_STATS\_PREFIX + "search\_results\_not\_set");

public static final SearchCounter SEARCH\_RESULTS\_WITH\_RESULTS\_NOT\_SET\_COUNTER =

SearchCounter.export(INVALID\_RESPONSE\_STATS\_PREFIX + "search\_results\_with\_results\_not\_set");

public static final SearchCounter MAX\_SEARCHED\_STATUS\_ID\_NOT\_SET\_COUNTER =

SearchCounter.export(INVALID\_RESPONSE\_STATS\_PREFIX + "max\_searched\_status\_id\_not\_set");

public static final SearchCounter MIN\_SEARCHED\_STATUS\_ID\_NOT\_SET\_COUNTER =

SearchCounter.export(INVALID\_RESPONSE\_STATS\_PREFIX + "min\_searched\_status\_id\_not\_set");

static {

ImmutableMap.Builder<EarlybirdResponseCode, SearchCounter> builder = ImmutableMap.builder();

for (EarlybirdResponseCode responseCode : EarlybirdResponseCode.values()) {

if (responseCode != EarlybirdResponseCode.SUCCESS) {

builder.put(responseCode, SearchCounter.export(

INVALID\_RESPONSE\_STATS\_PREFIX + responseCode.name().toLowerCase()));

}

}

ERROR\_EXCEPTIONS = builder.build();

}

private EarlybirdResponseMergeUtil() {

}

/\*\*

\* Tags the results in the given EarlybirdResponse with the given ThriftTweetSource and adds them

\* to the given list of results.

\*

\* @param results The list of results to which the new results will be added.

\* @param earlybirdResponse The EarlybirdResponse whose results will be added to {@code results}.

\* @param tweetSource The ThriftTweetSource that will be used to mark all results in

\* {@code earlybirdResponse}.

\* @return {@code false} if {@code earlybirdResponse} is {@code null} or doesn't have any results;

\* {@code true}, otherwise.

\*/

public static boolean addResultsToList(List<ThriftSearchResult> results,

EarlybirdResponse earlybirdResponse,

ThriftTweetSource tweetSource) {

return EarlybirdResponseUtil.hasResults(earlybirdResponse)

&& addResultsToList(results,

earlybirdResponse.getSearchResults().getResults(),

tweetSource);

}

/\*\*

\* Tags the results in the given list with the given ThriftTweetSource and adds them to the given

\* list of results.

\*

\* @param results The list of results to which the new results will be added.

\* @param resultsToAdd The list of results to add.

\* @param tweetSource The ThriftTweetSource that will be used to mark all results in

\* {@code resultsToAdd}.

\* @return {@code false} if {@code results} is {@code null} or if {@code resultsToAdd} is

\* {@code null} or doesn't have any results; {@code true}, otherwise.

\*/

public static boolean addResultsToList(List<ThriftSearchResult> results,

List<ThriftSearchResult> resultsToAdd,

ThriftTweetSource tweetSource) {

Preconditions.checkNotNull(results);

if ((resultsToAdd == null) || resultsToAdd.isEmpty()) {

return false;

}

markWithTweetSource(resultsToAdd, tweetSource);

results.addAll(resultsToAdd);

return true;

}

/\*\*

\* Distinct the input ThriftSearchResult by its status id. If there are duplicates, the first

\* instance of the duplicates is returned in the distinct result. If the distinct result is the

\* same as the input result, the initial input result is returned; otherwise, the distinct result

\* is returned.

\*

\* @param results the input result

\* @param dupsStats stats counter track duplicates source

\* @return the input result if there is no duplicate; otherwise, return the distinct result

\*/

public static List<ThriftSearchResult> distinctByStatusId(

List<ThriftSearchResult> results,

LoadingCache<Pair<ThriftTweetSource, ThriftTweetSource>, SearchCounter> dupsStats) {

Map<Long, ThriftTweetSource> seenStatusIdToSourceMap = new HashMap<>();

List<ThriftSearchResult> distinctResults = Lists.newArrayListWithCapacity(results.size());

for (ThriftSearchResult result : results) {

if (seenStatusIdToSourceMap.containsKey(result.getId())) {

ThriftTweetSource source1 = seenStatusIdToSourceMap.get(result.getId());

ThriftTweetSource source2 = result.getTweetSource();

if (source1 != null && source2 != null) {

try {

dupsStats.get(Pair.of(source1, source2)).increment();

} catch (ExecutionException e) {

LOG.warn("Could not increment stat for duplicate results from clusters " + source1

+ " and " + source2, e);

}

}

} else {

distinctResults.add(result);

seenStatusIdToSourceMap.put(result.getId(), result.getTweetSource());

}

}

return results.size() == distinctResults.size() ? results : distinctResults;

}

/\*\*

\* Tags the given results with the given ThriftTweetSource.

\*

\* @param results The results to be tagged.

\* @param tweetSource The ThriftTweetSource to be used to tag the given results.

\*/

public static void markWithTweetSource(List<ThriftSearchResult> results,

ThriftTweetSource tweetSource) {

if (results != null) {

for (ThriftSearchResult result : results) {

result.setTweetSource(tweetSource);

}

}

}

/\*\*

\* Check if an Earlybird response is valid

\*/

public static boolean isValidResponse(final EarlybirdResponse response) {

if (response == null) {

NULL\_RESPONSE\_COUNTER.increment();

return false;

}

if (!EarlybirdResponseUtil.isSuccessfulResponse(response)) {

return false;

}

if (!response.isSetSearchResults()) {

SEARCH\_RESULTS\_NOT\_SET\_COUNTER.increment();

return true;

}

if (!response.getSearchResults().isSetResults()) {

SEARCH\_RESULTS\_WITH\_RESULTS\_NOT\_SET\_COUNTER.increment();

}

// In earlybird, when earlybird terminated, e.g., time out, complex queries - we don't set the

// min/max searched status id.

boolean isEarlyTerminated = response.isSetEarlyTerminationInfo()

&& response.getEarlyTerminationInfo().isEarlyTerminated();

if (!isEarlyTerminated && !response.getSearchResults().isSetMinSearchedStatusID()) {

MIN\_SEARCHED\_STATUS\_ID\_NOT\_SET\_COUNTER.increment();

}

if (!isEarlyTerminated && !response.getSearchResults().isSetMaxSearchedStatusID()) {

MAX\_SEARCHED\_STATUS\_ID\_NOT\_SET\_COUNTER.increment();

}

return true;

}

/\*\*

\* For invalid successful Earlybird Response, return a failed response with debug msg.

\*/

public static EarlybirdResponse transformInvalidResponse(final EarlybirdResponse response,

final String debugMsg) {

if (response == null) {

return failedEarlybirdResponse(EarlybirdResponseCode.PERSISTENT\_ERROR,

debugMsg + ", msg: null response from downstream");

}

Preconditions.checkState(response.getResponseCode() != EarlybirdResponseCode.SUCCESS);

EarlybirdResponseCode newResponseCode;

EarlybirdResponseCode responseCode = response.getResponseCode();

switch (responseCode) {

case TIER\_SKIPPED:

ERROR\_EXCEPTIONS.get(responseCode).increment();

return response;

case REQUEST\_BLOCKED\_ERROR:

case CLIENT\_ERROR:

case SERVER\_TIMEOUT\_ERROR:

case QUOTA\_EXCEEDED\_ERROR:

case CLIENT\_CANCEL\_ERROR:

case TOO\_MANY\_PARTITIONS\_FAILED\_ERROR:

ERROR\_EXCEPTIONS.get(responseCode).increment();

newResponseCode = responseCode;

break;

default:

ERROR\_EXCEPTIONS.get(responseCode).increment();

newResponseCode = EarlybirdResponseCode.PERSISTENT\_ERROR;

}

String newDebugMsg = debugMsg + ", downstream response code: " + responseCode

+ (response.isSetDebugString() ? ", downstream msg: " + response.getDebugString() : "");

return failedEarlybirdResponse(newResponseCode, newDebugMsg);

}

/\*\*

\* Create a new EarlybirdResponse with debug msg

\*/

public static EarlybirdResponse failedEarlybirdResponse(final EarlybirdResponseCode responseCode,

final String debugMsg) {

EarlybirdResponse failedResponse = new EarlybirdResponse();

failedResponse.setResponseCode(responseCode);

failedResponse.setDebugString(debugMsg);

return failedResponse;

}

/\*\*

\* Returns the number of results to keep as part of merge-collection. Recency mode should ignore

\* relevance options. In particular, the flag returnAllResults inside relevance options.

\*/

public static int computeNumResultsToKeep(EarlybirdRequest request) {

ThriftSearchQuery searchQuery = request.getSearchQuery();

if (searchQuery.getRankingMode() != ThriftSearchRankingMode.RECENCY

&& searchQuery.isSetRelevanceOptions()

&& searchQuery.getRelevanceOptions().isReturnAllResults()) {

return Integer.MAX\_VALUE;

}

if (request.isSetNumResultsToReturnAtRoot()) {

return request.getNumResultsToReturnAtRoot();

}

if (searchQuery.isSetCollectorParams()) {

return searchQuery.getCollectorParams().getNumResultsToReturn();

}

return searchQuery.getNumResults();

}

}