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

import java.util.ArrayList;

import java.util.EnumMap;

import java.util.List;

import java.util.Map;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import com.google.common.collect.Maps;

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

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

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.ThriftSearchResults;

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

/\*\*

\* Accumulates EarlybirdResponse's and determines when to early terminate.

\*/

public abstract class ResponseAccumulator {

@VisibleForTesting

static class MinMaxSearchedIdStats {

/\*\* How many results did we actually check \*/

private final SearchCounter checkedMaxMinSearchedStatusId;

private final SearchCounter unsetMaxSearchedStatusId;

private final SearchCounter unsetMinSearchedStatusId;

private final SearchCounter unsetMaxAndMinSearchedStatusId;

private final SearchCounter sameMinMaxSearchedIdWithoutResults;

private final SearchCounter sameMinMaxSearchedIdWithOneResult;

private final SearchCounter sameMinMaxSearchedIdWithResults;

private final SearchCounter flippedMinMaxSearchedId;

MinMaxSearchedIdStats(EarlybirdRequestType requestType) {

String statPrefix = "merge\_helper\_" + requestType.getNormalizedName();

checkedMaxMinSearchedStatusId = SearchCounter.export(statPrefix

+ "\_max\_min\_searched\_id\_checks");

unsetMaxSearchedStatusId = SearchCounter.export(statPrefix

+ "\_unset\_max\_searched\_status\_id");

unsetMinSearchedStatusId = SearchCounter.export(statPrefix

+ "\_unset\_min\_searched\_status\_id");

unsetMaxAndMinSearchedStatusId = SearchCounter.export(statPrefix

+ "\_unset\_max\_and\_min\_searched\_status\_id");

sameMinMaxSearchedIdWithoutResults = SearchCounter.export(statPrefix

+ "\_same\_min\_max\_searched\_id\_without\_results");

sameMinMaxSearchedIdWithOneResult = SearchCounter.export(statPrefix

+ "\_same\_min\_max\_searched\_id\_with\_one\_results");

sameMinMaxSearchedIdWithResults = SearchCounter.export(statPrefix

+ "\_same\_min\_max\_searched\_id\_with\_results");

flippedMinMaxSearchedId = SearchCounter.export(statPrefix

+ "\_flipped\_min\_max\_searched\_id");

}

@VisibleForTesting

SearchCounter getCheckedMaxMinSearchedStatusId() {

return checkedMaxMinSearchedStatusId;

}

@VisibleForTesting

SearchCounter getFlippedMinMaxSearchedId() {

return flippedMinMaxSearchedId;

}

@VisibleForTesting

SearchCounter getUnsetMaxSearchedStatusId() {

return unsetMaxSearchedStatusId;

}

@VisibleForTesting

SearchCounter getUnsetMinSearchedStatusId() {

return unsetMinSearchedStatusId;

}

@VisibleForTesting

SearchCounter getUnsetMaxAndMinSearchedStatusId() {

return unsetMaxAndMinSearchedStatusId;

}

@VisibleForTesting

SearchCounter getSameMinMaxSearchedIdWithoutResults() {

return sameMinMaxSearchedIdWithoutResults;

}

@VisibleForTesting

SearchCounter getSameMinMaxSearchedIdWithOneResult() {

return sameMinMaxSearchedIdWithOneResult;

}

@VisibleForTesting

SearchCounter getSameMinMaxSearchedIdWithResults() {

return sameMinMaxSearchedIdWithResults;

}

}

@VisibleForTesting

static final Map<EarlybirdRequestType, MinMaxSearchedIdStats> MIN\_MAX\_SEARCHED\_ID\_STATS\_MAP;

static {

EnumMap<EarlybirdRequestType, MinMaxSearchedIdStats> statsMap

= Maps.newEnumMap(EarlybirdRequestType.class);

for (EarlybirdRequestType earlybirdRequestType : EarlybirdRequestType.values()) {

statsMap.put(earlybirdRequestType, new MinMaxSearchedIdStats(earlybirdRequestType));

}

MIN\_MAX\_SEARCHED\_ID\_STATS\_MAP = Maps.immutableEnumMap(statsMap);

}

// Merge has encountered at least one early terminated response.

private boolean foundEarlyTermination = false;

// Empty but successful response counter (E.g. when a tier or partition is skipped)

private int successfulEmptyResponseCount = 0;

// The list of the successful responses from all earlybird futures. This does not include empty

// responses resulted from null requests.

private final List<EarlybirdResponse> successResponses = new ArrayList<>();

// The list of the error responses from all earlybird futures.

private final List<EarlybirdResponse> errorResponses = new ArrayList<>();

// the list of max statusIds seen in each earlybird.

private final List<Long> maxIds = new ArrayList<>();

// the list of min statusIds seen in each earlybird.

private final List<Long> minIds = new ArrayList<>();

private int numResponses = 0;

private int numResultsAccumulated = 0;

private int numSearchedSegments = 0;

/\*\*

\* Returns a string that can be used for logging to identify a single response out of all the

\* responses that are being merged.

\*

\* @param responseIndex the index of a response's partition or tier, depending on the type of

\* responses being accumulated.

\* @param numTotalResponses the total number of partitions or tiers that are being merged.

\*/

public abstract String getNameForLogging(int responseIndex, int numTotalResponses);

/\*\*

\* Returns a string that is used to export per-EarlybirdResponseCode stats for partitions and tiers.

\*

\* @param responseIndex the index of of a response's partition or tier.

\* @param numTotalResponses the total number of partitions or tiers that are being merged.

\* @return a string that is used to export per-EarlybirdResponseCode stats for partitions and tiers.

\*/

public abstract String getNameForEarlybirdResponseCodeStats(

int responseIndex, int numTotalResponses);

abstract boolean shouldEarlyTerminateMerge(EarlyTerminateTierMergePredicate merger);

/\*\*

\* Add a EarlybirdResponse

\*/

public void addResponse(EarlybirdResponseDebugMessageBuilder responseMessageBuilder,

EarlybirdRequest request,

EarlybirdResponse response) {

numResponses++;

numSearchedSegments += response.getNumSearchedSegments();

if (isSkippedResponse(response)) {

// This is an empty response, no processing is required, just need to update statistics.

successfulEmptyResponseCount++;

handleSkippedResponse(response.getResponseCode());

} else if (isErrorResponse(response)) {

errorResponses.add(response);

handleErrorResponse(response);

} else {

handleSuccessfulResponse(responseMessageBuilder, request, response);

}

}

private boolean isErrorResponse(EarlybirdResponse response) {

return !response.isSetResponseCode()

|| response.getResponseCode() != EarlybirdResponseCode.SUCCESS;

}

private boolean isSkippedResponse(EarlybirdResponse response) {

return response.isSetResponseCode()

&& (response.getResponseCode() == EarlybirdResponseCode.PARTITION\_SKIPPED

|| response.getResponseCode() == EarlybirdResponseCode.TIER\_SKIPPED);

}

/\*\*

\* Record a response corresponding to a skipped partition or skipped tier.

\*/

protected abstract void handleSkippedResponse(EarlybirdResponseCode responseCode);

/\*\*

\* Handle an error response

\*/

protected abstract void handleErrorResponse(EarlybirdResponse response);

/\*\*

\* Subclasses can override this to perform more successful response handling.

\*/

protected void extraSuccessfulResponseHandler(EarlybirdResponse response) { }

/\*\*

\* Whether the helper is for merging results from partitions within a single tier.

\*/

protected final boolean isMergingPartitionsWithinATier() {

return !isMergingAcrossTiers();

}

/\*\*

\* Whether the helper is for merging results across different tiers.

\*/

protected abstract boolean isMergingAcrossTiers();

/\*\*

\* Record a successful response.

\*/

public final void handleSuccessfulResponse(

EarlybirdResponseDebugMessageBuilder responseMessageBuilder,

EarlybirdRequest request,

EarlybirdResponse response) {

successResponses.add(response);

if (response.isSetSearchResults()) {

ThriftSearchResults searchResults = response.getSearchResults();

numResultsAccumulated += searchResults.getResultsSize();

recordMinMaxSearchedIdsAndUpdateStats(responseMessageBuilder, request, response,

searchResults);

}

if (response.isSetEarlyTerminationInfo()

&& response.getEarlyTerminationInfo().isEarlyTerminated()) {

foundEarlyTermination = true;

}

extraSuccessfulResponseHandler(response);

}

private void recordMinMaxSearchedIdsAndUpdateStats(

EarlybirdResponseDebugMessageBuilder responseMessageBuidler,

EarlybirdRequest request,

EarlybirdResponse response,

ThriftSearchResults searchResults) {

boolean isMaxIdSet = searchResults.isSetMaxSearchedStatusID();

boolean isMinIdSet = searchResults.isSetMinSearchedStatusID();

if (isMaxIdSet) {

maxIds.add(searchResults.getMaxSearchedStatusID());

}

if (isMinIdSet) {

minIds.add(searchResults.getMinSearchedStatusID());

}

updateMinMaxIdStats(responseMessageBuidler, request, response, searchResults, isMaxIdSet,

isMinIdSet);

}

private void updateMinMaxIdStats(

EarlybirdResponseDebugMessageBuilder responseMessageBuilder,

EarlybirdRequest request,

EarlybirdResponse response,

ThriftSearchResults searchResults,

boolean isMaxIdSet,

boolean isMinIdSet) {

// Now just track the stats.

EarlybirdRequestType requestType = EarlybirdRequestType.of(request);

MinMaxSearchedIdStats minMaxSearchedIdStats = MIN\_MAX\_SEARCHED\_ID\_STATS\_MAP.get(requestType);

minMaxSearchedIdStats.checkedMaxMinSearchedStatusId.increment();

if (isMaxIdSet && isMinIdSet) {

if (searchResults.getMinSearchedStatusID() > searchResults.getMaxSearchedStatusID()) {

// We do not expect this case to happen in production.

minMaxSearchedIdStats.flippedMinMaxSearchedId.increment();

} else if (searchResults.getResultsSize() == 0

&& searchResults.getMaxSearchedStatusID() == searchResults.getMinSearchedStatusID()) {

minMaxSearchedIdStats.sameMinMaxSearchedIdWithoutResults.increment();

responseMessageBuilder.debugVerbose(

"Got no results, and same min/max searched ids. Request: %s, Response: %s",

request, response);

} else if (searchResults.getResultsSize() == 1

&& searchResults.getMaxSearchedStatusID() == searchResults.getMinSearchedStatusID()) {

minMaxSearchedIdStats.sameMinMaxSearchedIdWithOneResult.increment();

responseMessageBuilder.debugVerbose(

"Got one results, and same min/max searched ids. Request: %s, Response: %s",

request, response);

} else if (searchResults.getMaxSearchedStatusID()

== searchResults.getMinSearchedStatusID()) {

minMaxSearchedIdStats.sameMinMaxSearchedIdWithResults.increment();

responseMessageBuilder.debugVerbose(

"Got multiple results, and same min/max searched ids. Request: %s, Response: %s",

request, response);

}

} else if (!isMaxIdSet && isMinIdSet) {

// We do not expect this case to happen in production.

minMaxSearchedIdStats.unsetMaxSearchedStatusId.increment();

responseMessageBuilder.debugVerbose(

"Got unset maxSearchedStatusID. Request: %s, Response: %s", request, response);

} else if (isMaxIdSet && !isMinIdSet) {

// We do not expect this case to happen in production.

minMaxSearchedIdStats.unsetMinSearchedStatusId.increment();

responseMessageBuilder.debugVerbose(

"Got unset minSearchedStatusID. Request: %s, Response: %s", request, response);

} else {

Preconditions.checkState(!isMaxIdSet && !isMinIdSet);

minMaxSearchedIdStats.unsetMaxAndMinSearchedStatusId.increment();

responseMessageBuilder.debugVerbose(

"Got unset maxSearchedStatusID and minSearchedStatusID. Request: %s, Response: %s",

request, response);

}

}

/\*\*

\* Return partition counts with number of partitions, number of successful responses, and list of

\* responses per tier.

\*/

public abstract AccumulatedResponses.PartitionCounts getPartitionCounts();

public final AccumulatedResponses getAccumulatedResults() {

return new AccumulatedResponses(successResponses,

errorResponses,

maxIds,

minIds,

ResponseMergerUtils.mergeEarlyTerminationInfo(successResponses),

isMergingAcrossTiers(),

getPartitionCounts(),

getNumSearchedSegments());

}

// Getters are only intended to be used by subclasses. Other users should get data from

// AccumulatedResponses

int getNumResponses() {

return numResponses;

}

int getNumSearchedSegments() {

return numSearchedSegments;

}

List<EarlybirdResponse> getSuccessResponses() {

return successResponses;

}

int getNumResultsAccumulated() {

return numResultsAccumulated;

}

int getSuccessfulEmptyResponseCount() {

return successfulEmptyResponseCount;

}

boolean foundError() {

return !errorResponses.isEmpty();

}

boolean foundEarlyTermination() {

return foundEarlyTermination;

}

}