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

import java.util.Collections;

import java.util.HashSet;

import java.util.List;

import java.util.Map;

import scala.runtime.BoxedUnit;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Optional;

import com.google.common.base.Preconditions;

import com.google.common.collect.ImmutableList;

import com.google.common.collect.Lists;

import com.google.common.collect.Maps;

import com.google.common.collect.Sets;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

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

import com.twitter.search.common.schema.earlybird.EarlybirdCluster;

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

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

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

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

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.ThriftSearchResult;

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

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

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

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

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

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

import com.twitter.util.Function;

import com.twitter.util.Future;

/\*\*

\* Base EarlybirdResponseMerger containing basic logic to merge EarlybirdResponse objects

\*/

public abstract class EarlybirdResponseMerger implements EarlyTerminateTierMergePredicate {

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

private static final Logger MIN\_SEARCHED\_STATUS\_ID\_LOGGER =

LoggerFactory.getLogger("MinSearchedStatusIdLogger");

private static final SearchCounter NO\_SEARCH\_RESULT\_COUNTER =

SearchCounter.export("no\_search\_result\_count");

private static final SearchCounter NO\_RESPONSES\_TO\_MERGE =

SearchCounter.export("no\_responses\_to\_merge");

private static final SearchCounter EARLYBIRD\_RESPONSE\_NO\_MORE\_RESULTS =

SearchCounter.export("merger\_earlybird\_response\_no\_more\_results");

private static final String PARTITION\_OR\_TIER\_COUNTER\_NAME\_FORMAT =

"merger\_waited\_for\_response\_from\_%s\_counter";

private static final String PARTITION\_OR\_TIER\_ERROR\_COUNTER\_NAME\_FORMAT =

"merger\_num\_error\_responses\_from\_%s";

private static final String PARTITION\_OR\_TIER\_RESPONSE\_CODE\_COUNTER\_NAME\_FORMAT =

"merger\_earlybird\_response\_code\_from\_%s\_%s";

protected final EarlybirdResponseDebugMessageBuilder responseMessageBuilder;

protected final EarlybirdRequestContext requestContext;

protected final ImmutableList<Future<EarlybirdResponse>> responses;

protected AccumulatedResponses accumulatedResponses;

@VisibleForTesting

static final Map<EarlybirdRequestType, SearchCounter> MERGER\_CREATED\_STATS =

perRequestTypeCounterImmutableMap("earlybird\_response\_merger\_%s\_created\_count");

@VisibleForTesting

static final Map<EarlybirdRequestType, SearchCounter>

MIN\_SEARCHED\_STATUS\_ID\_LARGER\_THAN\_REQUEST\_MAX\_ID = perRequestTypeCounterImmutableMap(

"merger\_%s\_min\_searched\_status\_id\_larger\_than\_request\_max\_id");

@VisibleForTesting

static final Map<EarlybirdRequestType, SearchCounter>

MIN\_SEARCHED\_STATUS\_ID\_LARGER\_THAN\_REQUEST\_UNTIL\_TIME = perRequestTypeCounterImmutableMap(

"merger\_%s\_min\_searched\_status\_id\_larger\_than\_request\_until\_time");

private static Map<EarlybirdRequestType, SearchCounter> perRequestTypeCounterImmutableMap(

String statPattern) {

Map<EarlybirdRequestType, SearchCounter> statsMap = Maps.newEnumMap(EarlybirdRequestType.class);

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

String statName = String.format(statPattern, earlybirdRequestType.getNormalizedName());

statsMap.put(earlybirdRequestType, SearchCounter.export(statName));

}

return Maps.immutableEnumMap(statsMap);

}

public static final com.google.common.base.Function<EarlybirdResponse, Map<Long, Integer>>

HIT\_COUNT\_GETTER =

response -> response.getSearchResults() == null

? null

: response.getSearchResults().getHitCounts();

private final ChainMerger chainMerger;

private class ChainMerger {

private final EarlybirdRequestContext requestContext;

private final ResponseAccumulator responseAccumulator;

private final List<Future<EarlybirdResponse>> responses;

private final EarlybirdResponseDebugMessageBuilder responseMessageBuilder;

private int currentFutureIndex = -1;

public ChainMerger(EarlybirdRequestContext requestContext,

ResponseAccumulator responseAccumulator,

List<Future<EarlybirdResponse>> responses,

EarlybirdResponseDebugMessageBuilder responseMessageBuilder) {

this.requestContext = requestContext;

this.responseAccumulator = responseAccumulator;

this.responses = responses;

this.responseMessageBuilder = responseMessageBuilder;

}

public Future<EarlybirdResponse> merge() {

// 'responseFutures' should always be sorted.

// When returned by EarlybirdScatterGather service, the responses are sorted by partition ID.

// When returned by EarlybirdChainedScatterGatherService,

// responses are sorted descending by tier start date. See:

// com.twitter.search.earlybird\_root.EarlybirdChainedScatterGatherService.TIER\_COMPARATOR.

//

// When merging responses from partitions, we want to wait for responses from all partitions,

// so the order in which we wait for those results does not matter. When merging responses

// from tiers, we want to wait for the response from the latest. If we don't need any more

// responses to compute the final response, then we don't need to wait for the responses from

// other tiers. If we cannot terminate early, then we want to wait for the responses from the

// second tier, and so on.

//

// We do not need to have any explicit synchronization, because:

// 1. The callbacks for future\_i are set by the flatMap() callback on future\_{i-1} (when

// recursively calling merge() inside the flatMap()).

// 2. Before setting the callbacks on future\_i, future\_{i-1}.flatMap() adds the response

// results to mergeHelper.

// 3. When the callbacks on future\_i are set, the memory barrier between

// thread\_running\_future\_{i-1} and thread\_running\_future\_i is crossed. This guarantees

// that thread\_running\_future\_i will see the updates to mergeHelper before it sees the

// callbacks. (Or thread\_running\_future\_{i-1} == thread\_running\_future\_i, in which case

// synchronization is not an issue, and correctness is guarateed by the order in which

// things will run.)

// 4. The same reasoning applies to currentFutureIndex.

++currentFutureIndex;

if (currentFutureIndex >= responses.size()) {

return Future.value(getTimedMergedResponse(responseAccumulator.getAccumulatedResults()));

}

final String partitionTierName =

responseAccumulator.getNameForLogging(currentFutureIndex, responses.size());

final String nameForEarlybirdResponseCodeStats =

responseAccumulator.getNameForEarlybirdResponseCodeStats(

currentFutureIndex, responses.size());

// If a tier in the chain throws an exception, convert it to a null response, and let the

// mergeHelper handle it appropriately.

return responses.get(currentFutureIndex)

.handle(Function.func(t -> {

if (FinagleUtil.isCancelException(t)) {

return new EarlybirdResponse()

.setResponseCode(EarlybirdResponseCode.CLIENT\_CANCEL\_ERROR);

} else if (FinagleUtil.isTimeoutException(t)) {

return new EarlybirdResponse()

.setResponseCode(EarlybirdResponseCode.SERVER\_TIMEOUT\_ERROR);

} else {

SearchCounter.export(

String.format(PARTITION\_OR\_TIER\_ERROR\_COUNTER\_NAME\_FORMAT, partitionTierName))

.increment();

if (responseMessageBuilder.isDebugMode()) {

responseMessageBuilder.debugAndLogWarning(

String.format("[%s] failed, exception [%s]",

partitionTierName, t.toString()));

}

LOG.warn("exception response from: " + partitionTierName, t);

return new EarlybirdResponse()

.setResponseCode(EarlybirdResponseCode.TRANSIENT\_ERROR);

}

}))

.flatMap(Function.func(response -> {

Preconditions.checkNotNull(response);

SearchCounter.export(

String.format(PARTITION\_OR\_TIER\_RESPONSE\_CODE\_COUNTER\_NAME\_FORMAT,

nameForEarlybirdResponseCodeStats,

response.getResponseCode().name().toLowerCase()))

.increment();

if ((response.getResponseCode() != EarlybirdResponseCode.PARTITION\_SKIPPED)

&& (response.getResponseCode() != EarlybirdResponseCode.TIER\_SKIPPED)) {

SearchCounter.export(

String.format(PARTITION\_OR\_TIER\_COUNTER\_NAME\_FORMAT, partitionTierName))

.increment();

}

if (response.getResponseCode() == EarlybirdResponseCode.CLIENT\_CANCEL\_ERROR) {

// the request has been cancelled, no need to proceed

return Future.value(response);

}

rewriteResponseCodeIfSearchResultsMissing(requestContext, partitionTierName, response);

responseMessageBuilder.logResponseDebugInfo(

requestContext.getRequest(),

partitionTierName,

response);

responseAccumulator.addResponse(

responseMessageBuilder,

requestContext.getRequest(),

response);

if (responseAccumulator.shouldEarlyTerminateMerge(EarlybirdResponseMerger.this)) {

return Future.value(getTimedMergedResponse(

responseAccumulator.getAccumulatedResults()));

}

return merge();

}));

}

}

private void rewriteResponseCodeIfSearchResultsMissing(

EarlybirdRequestContext earlybirdRequestContext,

String partitionTierName,

EarlybirdResponse response) {

// We always require searchResults to be set, even for term stats and facet requests.

// This is because searchResults contains important info such as pagination cursors

// like minSearchStatusId and minSearchedTimeSinceEpoch.

// We expect all successful responses to have searchResults set.

if (response.isSetResponseCode()

&& response.getResponseCode() == EarlybirdResponseCode.SUCCESS

&& response.getSearchResults() == null) {

NO\_SEARCH\_RESULT\_COUNTER.increment();

LOG.warn("Received Earlybird response with null searchResults from [{}]"

+ " EarlybirdRequest [{}] EarlybirdResponse [{}] ",

partitionTierName, earlybirdRequestContext.getRequest(), response);

response.setResponseCode(EarlybirdResponseCode.TRANSIENT\_ERROR);

}

}

/\*\*

\* Construct a EarlybirdResponseMerger to merge responses from multiple partitions or tiers

\* based on mode.

\*/

EarlybirdResponseMerger(EarlybirdRequestContext requestContext,

List<Future<EarlybirdResponse>> responses,

ResponseAccumulator responseAccumulator) {

this.requestContext = requestContext;

this.responses = ImmutableList.copyOf(responses);

this.responseMessageBuilder =

new EarlybirdResponseDebugMessageBuilder(requestContext.getRequest());

this.chainMerger = new ChainMerger(requestContext, responseAccumulator, responses,

responseMessageBuilder);

}

/\*\*

\* Get a response merger to merge the given responses.

\*/

public static EarlybirdResponseMerger getResponseMerger(

EarlybirdRequestContext requestContext,

List<Future<EarlybirdResponse>> responses,

ResponseAccumulator helper,

EarlybirdCluster cluster,

EarlybirdFeatureSchemaMerger featureSchemaMerger,

int numPartitions) {

EarlybirdRequestType type = requestContext.getEarlybirdRequestType();

MERGER\_CREATED\_STATS.get(type).increment();

switch (type) {

case FACETS:

return new FacetResponseMerger(requestContext, responses, helper);

case TERM\_STATS:

return new TermStatisticsResponseMerger(requestContext, responses, helper);

case RECENCY:

return new RecencyResponseMerger(requestContext, responses, helper, featureSchemaMerger);

case STRICT\_RECENCY:

return new StrictRecencyResponseMerger(

requestContext, responses, helper, featureSchemaMerger, cluster);

case RELEVANCE:

return new RelevanceResponseMerger(

requestContext, responses, helper, featureSchemaMerger, numPartitions);

case TOP\_TWEETS:

return new TopTweetsResponseMerger(requestContext, responses, helper);

default:

throw new RuntimeException("EarlybirdRequestType " + type + "is not supported by merge");

}

}

/\*\*

\* This method can perform two types of merges:

\* 1. merge responses within a tier from different partitions.

\* 2. merge responses from multiple tiers.

\*/

public final Future<EarlybirdResponse> merge() {

return chainMerger.merge()

.onSuccess(checkMinSearchedStatusIdFunction(

"max\_id",

EarlybirdRequestUtil.getRequestMaxId(requestContext.getParsedQuery()),

MIN\_SEARCHED\_STATUS\_ID\_LARGER\_THAN\_REQUEST\_MAX\_ID.get(

requestContext.getEarlybirdRequestType())))

.onSuccess(checkMinSearchedStatusIdFunction(

"until\_time",

EarlybirdRequestUtil.getRequestMaxIdFromUntilTime(requestContext.getParsedQuery()),

MIN\_SEARCHED\_STATUS\_ID\_LARGER\_THAN\_REQUEST\_UNTIL\_TIME.get(

requestContext.getEarlybirdRequestType())));

}

/\*\*

\* Returns the function that checks if the minSearchedStatusID on the merged response is higher

\* than the max ID in the request.

\*/

private Function<EarlybirdResponse, BoxedUnit> checkMinSearchedStatusIdFunction(

final String operator, final Optional<Long> requestMaxId, final SearchCounter stat) {

return Function.cons(mergedResponse -> {

if (requestMaxId.isPresent()

&& requestMaxId.get() != Long.MAX\_VALUE

&& (mergedResponse.getResponseCode() == EarlybirdResponseCode.SUCCESS)

&& mergedResponse.isSetSearchResults()

&& mergedResponse.getSearchResults().isSetMinSearchedStatusID()) {

long minSearchedStatusId = mergedResponse.getSearchResults().getMinSearchedStatusID();

// We sometimes set minSearchedStatusId = max\_id + 1 when a request times out even

// before any search happens.

// Check SEARCH-10134 for more details.

if (minSearchedStatusId > requestMaxId.get() + 1) {

stat.increment();

String logMessage = "Response has a minSearchedStatusID ({}) larger than request "

+ operator + " ({})."

+ "\nrequest type: {}"

+ "\nrequest: {}"

+ "\nmerged response: {}"

+ "\nSuccessful accumulated responses:";

List<Object> logMessageParams = Lists.newArrayList();

logMessageParams.add(minSearchedStatusId);

logMessageParams.add(requestMaxId.get());

logMessageParams.add(requestContext.getEarlybirdRequestType());

logMessageParams.add(requestContext.getRequest());

logMessageParams.add(mergedResponse);

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

logMessage += "\naccumulated response: {}";

logMessageParams.add(response);

}

MIN\_SEARCHED\_STATUS\_ID\_LOGGER.warn(logMessage, logMessageParams.toArray());

}

}

});

}

private EarlybirdResponse getTimedMergedResponse(AccumulatedResponses accResponses) {

long start = System.nanoTime();

try {

return getMergedResponse(accResponses);

} finally {

long totalTime = System.nanoTime() - start;

getMergedResponseTimer().timerIncrement(totalTime);

}

}

private EarlybirdResponse initializeMergedSuccessResponseFromAccumulatedResponses() {

EarlybirdResponse mergedResponse = new EarlybirdResponse();

AccumulatedResponses.PartitionCounts partitionCounts =

accumulatedResponses.getPartitionCounts();

mergedResponse.setNumPartitions(partitionCounts.getNumPartitions())

.setNumSuccessfulPartitions(partitionCounts.getNumSuccessfulPartitions())

.setPerTierResponse(partitionCounts.getPerTierResponse())

.setNumSearchedSegments(accumulatedResponses.getNumSearchedSegments());

mergedResponse.setEarlyTerminationInfo(accumulatedResponses.getMergedEarlyTerminationInfo());

mergedResponse.setResponseCode(EarlybirdResponseCode.SUCCESS);

return mergedResponse;

}

private EarlybirdResponse getMergedResponse(AccumulatedResponses accResponses) {

accumulatedResponses = accResponses;

EarlybirdResponse mergedResponse;

if (accumulatedResponses.getSuccessResponses().isEmpty()

&& !accumulatedResponses.foundError()) {

// No successful or error responses. This means that all tiers / partitions are intentionally

// skipped. Return a blank successful response.

NO\_RESPONSES\_TO\_MERGE.increment();

mergedResponse = new EarlybirdResponse()

.setResponseCode(EarlybirdResponseCode.SUCCESS)

.setSearchResults(new ThriftSearchResults())

.setDebugString("No responses to merge, probably because all tiers/partitions "

+ "were skipped.");

} else if (accumulatedResponses.isMergingAcrossTiers()) {

mergedResponse = getMergedResponseAcrossTiers();

} else {

mergedResponse = getMergedResponseAcrossPartitions();

}

saveMergedDebugString(mergedResponse);

return mergedResponse;

}

private EarlybirdResponse getMergedResponseAcrossTiers() {

Preconditions.checkState(

!accumulatedResponses.getSuccessResponses().isEmpty()

|| accumulatedResponses.foundError());

// When merging across tiers, if we have one failed tier, we should fail the whole

// response. Note that due to early termination, if a tier that is old fails

// but the newer tiers return enough results, the failed tier won't show up

// here in accumulatedResponses -- the only tiers that show up here

// will be successful.

if (accumulatedResponses.foundError()) {

// The TierResponseAccumulator early terminates on the first error, so we should

// never get more than one error. This means that the getMergedErrorResponse will

// return an error response with the error code of that one error, and will never

// have to decide which error response to return if the error responses are all

// different.

// Perhaps we should just return accumulatedResponses.getErrorResponses().get(0);

Preconditions.checkState(accumulatedResponses.getErrorResponses().size() == 1);

return accumulatedResponses.getMergedErrorResponse();

} else {

EarlybirdResponse mergedResponse = initializeMergedSuccessResponseFromAccumulatedResponses();

return internalMerge(mergedResponse);

}

}

private EarlybirdResponse getMergedResponseAcrossPartitions() {

Preconditions.checkState(

!accumulatedResponses.getSuccessResponses().isEmpty()

|| accumulatedResponses.foundError());

EarlybirdResponse mergedResponse;

// Unlike tier merging, one failed response doesn't mean the merged response should

// fail. If we have successful responses we can check the success ratio and if its

// good we can still return a successful merge.

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

// We have at least one successful response, but still need to check the success ratio.

// mergedResponse is a SUCCESS response after this call, but we will

// set it to failure below if necessary.

mergedResponse = initializeMergedSuccessResponseFromAccumulatedResponses();

int numSuccessResponses = mergedResponse.getNumSuccessfulPartitions();

int numPartitions = mergedResponse.getNumPartitions();

double successThreshold = getSuccessResponseThreshold();

if (checkSuccessPartitionRatio(numSuccessResponses, numPartitions, successThreshold)) {

// Success! Proceed with merging.

mergedResponse.setResponseCode(EarlybirdResponseCode.SUCCESS);

mergedResponse = internalMerge(mergedResponse);

} else {

responseMessageBuilder.logBelowSuccessThreshold(

requestContext.getRequest().getSearchQuery(), numSuccessResponses, numPartitions,

successThreshold);

mergedResponse.setResponseCode(EarlybirdResponseCode.TOO\_MANY\_PARTITIONS\_FAILED\_ERROR);

}

} else {

mergedResponse = accumulatedResponses.getMergedErrorResponse();

}

return mergedResponse;

}

/\*\*

\* Derive class should implement the logic to merge the specific type of results (recency,

\* relevance, Top Tweets, etc..)

\*/

protected abstract EarlybirdResponse internalMerge(EarlybirdResponse response);

protected abstract SearchTimerStats getMergedResponseTimer();

/\*\*

\* Do we have enough results so far that we can early terminate and not continue onto next tier?

\*/

public boolean shouldEarlyTerminateTierMerge(int totalResultsFromSuccessfulShards,

boolean foundEarlyTermination) {

// We are taking the most conservative tier response merging.

// This is the most conservative merge logic --- as long as we have some results, we should

// not return anything from the next tier. This may cause not ideal experience where a

// page is not full, but the use can still scroll further.

return foundEarlyTermination || totalResultsFromSuccessfulShards >= 1;

}

private void saveMergedDebugString(EarlybirdResponse mergedResponse) {

if (responseMessageBuilder.isDebugMode()) {

String message = responseMessageBuilder.debugString();

mergedResponse.setDebugString(message);

if (!accumulatedResponses.getSuccessResponses().isEmpty()

&& accumulatedResponses.getSuccessResponses().get(0).isSetDebugInfo()) {

EarlybirdDebugInfo debugInfo =

accumulatedResponses.getSuccessResponses().get(0).getDebugInfo();

mergedResponse.setDebugInfo(debugInfo);

}

}

}

private double getSuccessResponseThreshold() {

EarlybirdRequest request = requestContext.getRequest();

if (request.isSetSuccessfulResponseThreshold()) {

double successfulResponseThreshold = request.getSuccessfulResponseThreshold();

Preconditions.checkArgument(successfulResponseThreshold > 0,

"Invalid successfulResponseThreshold %s", successfulResponseThreshold);

Preconditions.checkArgument(successfulResponseThreshold <= 1.0,

"Invalid successfulResponseThreshold %s", successfulResponseThreshold);

return successfulResponseThreshold;

} else {

return getDefaultSuccessResponseThreshold();

}

}

protected abstract double getDefaultSuccessResponseThreshold();

private static boolean checkSuccessPartitionRatio(

int numSuccessResponses,

int numPartitions,

double goodResponseThreshold) {

Preconditions.checkArgument(goodResponseThreshold > 0.0,

"Invalid goodResponseThreshold %s", goodResponseThreshold);

return numSuccessResponses >= (numPartitions \* goodResponseThreshold);

}

/\*\*

\* Merge hit counts from all results.

\*/

protected Map<Long, Integer> aggregateHitCountMap() {

Map<Long, Integer> hitCounts = ResultsUtil

.aggregateCountMap(accumulatedResponses.getSuccessResponses(), HIT\_COUNT\_GETTER);

if (hitCounts.size() > 0) {

if (responseMessageBuilder.isDebugMode()) {

responseMessageBuilder.append("Hit counts:\n");

for (Map.Entry<Long, Integer> entry : hitCounts.entrySet()) {

responseMessageBuilder.append(String.format(" %10s seconds: %d hits\n",

entry.getKey() / 1000, entry.getValue()));

}

}

return hitCounts;

}

return null;

}

/\*\*

\* Returns the number of results to keep as part of merge-collection.

\*/

protected final int computeNumResultsToKeep() {

return EarlybirdResponseMergeUtil.computeNumResultsToKeep(requestContext.getRequest());

}

/\*\*

\* Remove exact duplicates (same id) from the result set.

\*/

protected static void trimExactDups(ThriftSearchResults searchResults, TrimStats trimStats) {

int numResults = searchResults.getResultsSize();

List<ThriftSearchResult> oldResults = searchResults.getResults();

List<ThriftSearchResult> newResults = Lists.newArrayListWithCapacity(numResults);

HashSet<Long> resultSet = Sets.newHashSetWithExpectedSize(numResults);

for (ThriftSearchResult result : oldResults) {

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

trimStats.increaseRemovedDupsCount();

continue;

}

newResults.add(result);

resultSet.add(result.getId());

}

searchResults.setResults(newResults);

}

protected final int addResponsesToCollector(MultiwayMergeCollector collector) {

int totalResultSize = 0;

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

if (response.isSetSearchResults()) {

totalResultSize += response.getSearchResults().getResultsSize();

}

collector.addResponse(response);

}

return totalResultSize;

}

/\*\*

\* Given a sorted searchResults (for recency, sorted by ID; for relevance, sorted by score),

\* returns the first 'computeNumResultsToKeep()' number of results.

\*

\* @param searchResults the searchResults to be truncated.

\*/

protected final void truncateResults(ThriftSearchResults searchResults, TrimStats trimStats) {

int numResultsRequested = computeNumResultsToKeep();

int to = numResultsRequested == Integer.MAX\_VALUE ? searchResults.getResultsSize()

: Math.min(numResultsRequested, searchResults.getResultsSize());

if (searchResults.getResultsSize() > to) {

trimStats.setResultsTruncatedFromTailCount(searchResults.getResultsSize() - to);

if (to > 0) {

searchResults.setResults(searchResults.getResults().subList(0, to));

} else {

// No more results for the next page

EARLYBIRD\_RESPONSE\_NO\_MORE\_RESULTS.increment();

searchResults.setResults(Collections.<ThriftSearchResult>emptyList());

}

}

}

EarlybirdRequest getEarlybirdRequest() {

return requestContext.getRequest();

}

}