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

import java.util.List;

import java.util.Map;

import java.util.Set;

import java.util.TreeSet;

import java.util.concurrent.ConcurrentHashMap;

import javax.annotation.concurrent.ThreadSafe;

import com.google.common.base.Preconditions;

import com.google.common.collect.ImmutableList;

import com.google.common.collect.Maps;

import org.apache.commons.lang.mutable.MutableInt;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.search.common.features.thrift.ThriftSearchFeatureSchema;

import com.twitter.search.common.features.thrift.ThriftSearchFeatureSchemaSpecifier;

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

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

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

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

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

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

@ThreadSafe

public class EarlybirdFeatureSchemaMerger {

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

private static final SearchLongGauge NUM\_FEATURE\_SCHEMAS\_MAP = SearchLongGauge.export(

"earlybird\_feature\_schema\_cached\_cnt");

private class Stats {

public final SearchCounter fieldFormatResponses;

public final SearchCounter mapFormatResponses;

public final SearchCounter mapFormatSavedSchemaResponses;

public final SearchCounter mapFormatAllDownstreamMissingSchema;

public final SearchCounter mapFormatOneDownstreamMissingSchema;

public final SearchCounter mapFormatSchemaCachedMismatch;

public final SearchCounter numInvalidRankingModeRequests;

public final SearchCounter numEmptyResponses;

public Stats(String prefix) {

this.fieldFormatResponses =

SearchCounter.export(

"earlybird\_feature\_schema\_" + prefix + "\_field\_format\_feature\_responses");

this.mapFormatResponses =

SearchCounter.export(

"earlybird\_feature\_schema\_" + prefix + "\_map\_format\_feature\_responses");

this.mapFormatSavedSchemaResponses =

SearchCounter.export(

"earlybird\_feature\_schema\_" + prefix + "\_map\_format\_feature\_saved\_schema\_responses");

this.mapFormatAllDownstreamMissingSchema =

SearchCounter.export(

"earlybird\_feature\_schema\_" + prefix

+ "\_map\_format\_feature\_all\_downstream\_missing\_schema\_error");

this.mapFormatOneDownstreamMissingSchema =

SearchCounter.export(

"earlybird\_feature\_schema\_" + prefix

+ "\_map\_format\_feature\_one\_downstream\_missing\_schema\_error");

this.mapFormatSchemaCachedMismatch =

SearchCounter.export(

"earlybird\_feature\_schema\_" + prefix

+ "\_map\_format\_feature\_schema\_cached\_mismatch\_error");

this.numInvalidRankingModeRequests =

SearchCounter.export(

"earlybird\_feature\_schema\_" + prefix + "\_num\_invalid\_ranking\_mode\_requests");

this.numEmptyResponses =

SearchCounter.export(

"earlybird\_feature\_schema\_" + prefix

+ "\_num\_empty\_response\_without\_schema");

}

}

private final ConcurrentHashMap<ThriftSearchFeatureSchemaSpecifier, ThriftSearchFeatureSchema>

featureSchemas = new ConcurrentHashMap<>();

private final ConcurrentHashMap<String, Stats> mergeStats = new ConcurrentHashMap<>();

/\*\*

\* Get all available cache schema list indicated by the schema specifier.

\* @return identifiers for all the cached schema

\*/

public List<ThriftSearchFeatureSchemaSpecifier> getAvailableSchemaList() {

return ImmutableList.copyOf(featureSchemas.keySet());

}

/\*\*

\* Iterate all the responses and collect and cache feature schemas from response.

\* Set the feature schema for the response in searchResults if needed.

\* (This is done inside earlybird roots)

\*

\* @param searchResults the response

\* @param requestContext the request, which should record the client cached feature schemas

\* @param statPrefix the stats prefix string

\* @param successfulResponses all successfull responses from downstream

\*/

public void collectAndSetFeatureSchemaInResponse(

ThriftSearchResults searchResults,

EarlybirdRequestContext requestContext,

String statPrefix,

List<EarlybirdResponse> successfulResponses) {

Stats stats = getOrCreateMergeStat(statPrefix);

EarlybirdRequest request = requestContext.getRequest();

if (!request.isSetSearchQuery()

|| !request.getSearchQuery().isSetResultMetadataOptions()

|| !request.getSearchQuery().getResultMetadataOptions().isReturnSearchResultFeatures()) {

// If the client does not want to get all features in map format, do not do anything.

stats.fieldFormatResponses.increment();

return;

}

// Find the most occurred schema from per-merge responses and return it in the post-merge

// response.

ThriftSearchFeatureSchemaSpecifier schemaMostOccurred = findMostOccurredSchema(

stats, request, successfulResponses);

if (schemaMostOccurred == null) {

return;

}

Set<ThriftSearchFeatureSchemaSpecifier> availableSchemasInClient =

requestContext.getFeatureSchemasAvailableInClient();

if (availableSchemasInClient != null && availableSchemasInClient.contains(schemaMostOccurred)) {

// The client already knows the schema that we used for this response, so we don't need to

// send it the full schema, just the ThriftSearchFeatureSchemaSpecifier.

ThriftSearchFeatureSchema schema = new ThriftSearchFeatureSchema();

schema.setSchemaSpecifier(schemaMostOccurred);

searchResults.setFeatureSchema(schema);

stats.mapFormatResponses.increment();

stats.mapFormatSavedSchemaResponses.increment();

} else {

ThriftSearchFeatureSchema schema = featureSchemas.get(schemaMostOccurred);

if (schema != null) {

Preconditions.checkState(schema.isSetEntries());

Preconditions.checkState(schema.isSetSchemaSpecifier());

searchResults.setFeatureSchema(schema);

stats.mapFormatResponses.increment();

} else {

stats.mapFormatSchemaCachedMismatch.increment();

LOG.error("The feature schema cache misses the schema entry {} it should cache for {}",

schemaMostOccurred, request);

}

}

}

/\*\*

\* Merge the feature schema from each cluster's response and return it to the client.

\* (This is done inside superroot)

\* @param requestContext the search request context

\* @param mergedResponse the merged result inside the superroot

\* @param realtimeResponse the realtime tier resposne

\* @param protectedResponse the protected tier response

\* @param fullArchiveResponse the full archive tier response

\* @param statsPrefix

\*/

public void mergeFeatureSchemaAcrossClusters(

EarlybirdRequestContext requestContext,

EarlybirdResponse mergedResponse,

String statsPrefix,

EarlybirdResponse realtimeResponse,

EarlybirdResponse protectedResponse,

EarlybirdResponse fullArchiveResponse) {

Stats superrootStats = getOrCreateMergeStat(statsPrefix);

// Only try to merge feature schema if there are search results.

ThriftSearchResults mergedResults = Preconditions.checkNotNull(

mergedResponse.getSearchResults());

if (mergedResults.getResults().isEmpty()) {

mergedResults.unsetFeatureSchema();

superrootStats.numEmptyResponses.increment();

return;

}

EarlybirdRequest request = requestContext.getRequest();

if (!request.isSetSearchQuery()

|| !request.getSearchQuery().isSetResultMetadataOptions()

|| !request.getSearchQuery().getResultMetadataOptions().isReturnSearchResultFeatures()) {

mergedResults.unsetFeatureSchema();

// If the client does not want to get all features in map format, do not do anything.

superrootStats.fieldFormatResponses.increment();

return;

}

if (request.getSearchQuery().getRankingMode() != ThriftSearchRankingMode.RELEVANCE

&& request.getSearchQuery().getRankingMode() != ThriftSearchRankingMode.TOPTWEETS

&& request.getSearchQuery().getRankingMode() != ThriftSearchRankingMode.RECENCY) {

mergedResults.unsetFeatureSchema();

// Only RELEVANCE, TOPTWEETS and RECENCY requests might need a feature schema in the response.

superrootStats.numInvalidRankingModeRequests.increment();

LOG.warn("Request asked for feature schema, but has incorrect ranking mode: {}", request);

return;

}

superrootStats.mapFormatResponses.increment();

ThriftSearchFeatureSchema schema = updateReturnSchemaForClusterResponse(

null, realtimeResponse, request, superrootStats);

schema = updateReturnSchemaForClusterResponse(

schema, protectedResponse, request, superrootStats);

schema = updateReturnSchemaForClusterResponse(

schema, fullArchiveResponse, request, superrootStats);

if (schema != null) {

if (requestContext.getFeatureSchemasAvailableInClient() != null

&& requestContext.getFeatureSchemasAvailableInClient().contains(

schema.getSchemaSpecifier())) {

mergedResults.setFeatureSchema(

new ThriftSearchFeatureSchema().setSchemaSpecifier(schema.getSchemaSpecifier()));

} else {

mergedResults.setFeatureSchema(schema);

}

} else {

superrootStats.mapFormatAllDownstreamMissingSchema.increment();

LOG.error("The response for request {} is missing feature schema from all clusters", request);

}

}

/\*\*

\* Add the schema to both the schema map and and the schema list if it is not there yet.

\*

\* @param schema the feature schema for search results

\*/

private void addNewSchema(ThriftSearchFeatureSchema schema) {

if (!schema.isSetEntries()

|| !schema.isSetSchemaSpecifier()

|| featureSchemas.containsKey(schema.getSchemaSpecifier())) {

return;

}

synchronized (this) {

String oldExportedSchemaName = null;

if (!featureSchemas.isEmpty()) {

oldExportedSchemaName = getExportSchemasName();

}

if (featureSchemas.putIfAbsent(schema.getSchemaSpecifier(), schema) == null) {

LOG.info("Add new feature schema {} into the list", schema);

NUM\_FEATURE\_SCHEMAS\_MAP.set(featureSchemas.size());

if (oldExportedSchemaName != null) {

SearchLongGauge.export(oldExportedSchemaName).reset();

}

SearchLongGauge.export(getExportSchemasName()).set(1);

LOG.info("Expanded feature schema: {}", ImmutableList.copyOf(featureSchemas.keySet()));

}

}

}

private String getExportSchemasName() {

StringBuilder builder = new StringBuilder("earlybird\_feature\_schema\_cached");

TreeSet<String> exportedVersions = new TreeSet<>();

// We do not need checksum for exported vars as all cached schemas are from the majority of the

// responses.

featureSchemas.keySet().stream().forEach(key -> exportedVersions.add(key.getVersion()));

exportedVersions.stream().forEach(version -> {

builder.append('\_');

builder.append(version);

});

return builder.toString();

}

// Get the updated the feature schema based on the earlybird response from the search cluster.

// . If the existingSchema is not null, the function would return the existing schema. Under the

// situation, we would still check whether the feature in earlybird response is valid.

// . Otherwise, the function would extract the feature schema from the earlybird response.

private ThriftSearchFeatureSchema updateReturnSchemaForClusterResponse(

ThriftSearchFeatureSchema existingSchema,

EarlybirdResponse clusterResponse,

EarlybirdRequest request,

Stats stats) {

// If there is no response or search result for this cluster, do not update returned schema.

if ((clusterResponse == null) || !clusterResponse.isSetSearchResults()) {

return existingSchema;

}

ThriftSearchResults results = clusterResponse.getSearchResults();

if (results.getResults().isEmpty()) {

return existingSchema;

}

if (!results.isSetFeatureSchema() || !results.getFeatureSchema().isSetSchemaSpecifier()) {

stats.mapFormatOneDownstreamMissingSchema.increment();

LOG.error("The downstream response {} is missing feature schema for request {}",

clusterResponse, request);

return existingSchema;

}

ThriftSearchFeatureSchema schema = results.getFeatureSchema();

// Even if existingSchema is already set, we would still try to cache the returned schema.

// In this way, the next time earlybird roots don't have to send the full schema back again.

if (schema.isSetEntries()) {

addNewSchema(schema);

} else if (featureSchemas.containsKey(schema.getSchemaSpecifier())) {

stats.mapFormatSavedSchemaResponses.increment();

} else {

stats.mapFormatSchemaCachedMismatch.increment();

LOG.error(

"The feature schema cache misses the schema entry {}, it should cache {} in {}",

schema.getSchemaSpecifier(), request, clusterResponse);

}

ThriftSearchFeatureSchema updatedSchema = existingSchema;

if (updatedSchema == null) {

updatedSchema = featureSchemas.get(schema.getSchemaSpecifier());

if (updatedSchema != null) {

Preconditions.checkState(updatedSchema.isSetEntries());

Preconditions.checkState(updatedSchema.isSetSchemaSpecifier());

}

}

return updatedSchema;

}

private ThriftSearchFeatureSchemaSpecifier findMostOccurredSchema(

Stats stats,

EarlybirdRequest request,

List<EarlybirdResponse> successfulResponses) {

boolean hasResults = false;

Map<ThriftSearchFeatureSchemaSpecifier, MutableInt> schemaCount =

Maps.newHashMapWithExpectedSize(successfulResponses.size());

for (EarlybirdResponse response : successfulResponses) {

if (!response.isSetSearchResults()

|| response.getSearchResults().getResultsSize() == 0) {

continue;

}

hasResults = true;

if (response.getSearchResults().isSetFeatureSchema()) {

ThriftSearchFeatureSchema schema = response.getSearchResults().getFeatureSchema();

if (schema.isSetSchemaSpecifier()) {

MutableInt cnt = schemaCount.get(schema.getSchemaSpecifier());

if (cnt != null) {

cnt.increment();

} else {

schemaCount.put(schema.getSchemaSpecifier(), new MutableInt(1));

}

if (schema.isSetEntries()) {

addNewSchema(schema);

}

}

} else {

stats.mapFormatOneDownstreamMissingSchema.increment();

LOG.error("The downstream response {} is missing feature schema for request {}",

response, request);

}

}

int numMostOccurred = 0;

ThriftSearchFeatureSchemaSpecifier schemaMostOccurred = null;

for (Map.Entry<ThriftSearchFeatureSchemaSpecifier, MutableInt> entry : schemaCount.entrySet()) {

if (entry.getValue().toInteger() > numMostOccurred) {

numMostOccurred = entry.getValue().toInteger();

schemaMostOccurred = entry.getKey();

}

}

if (schemaMostOccurred == null && hasResults) {

stats.mapFormatAllDownstreamMissingSchema.increment();

LOG.error("None of the downstream host returned feature schema for {}", request);

}

return schemaMostOccurred;

}

private Stats getOrCreateMergeStat(String statPrefix) {

Stats stats = mergeStats.get(statPrefix);

if (stats == null) {

Stats newStats = new Stats(statPrefix);

stats = mergeStats.putIfAbsent(statPrefix, newStats);

if (stats == null) {

stats = newStats;

}

}

return stats;

}

}