package com.twitter.search.earlybird\_root;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import javax.inject.Inject;

import com.google.common.base.Preconditions;

import com.google.common.collect.Lists;

import com.google.common.collect.Maps;

import com.google.common.collect.Sets;

import com.twitter.search.common.partitioning.base.PartitionDataType;

import com.twitter.search.common.partitioning.base.PartitionMappingManager;

import com.twitter.search.common.root.ScatterGatherSupport;

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

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

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.EarlybirdFeatureSchemaMerger;

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

import com.twitter.search.earlybird\_root.mergers.EarlybirdResponseMerger;

import com.twitter.search.earlybird\_root.mergers.PartitionResponseAccumulator;

import com.twitter.search.queryparser.query.Query;

import com.twitter.util.Future;

import static com.twitter.search.earlybird\_root.visitors.MultiTermDisjunctionPerPartitionVisitor.NO\_MATCH\_CONJUNCTION;

public class EarlybirdServiceScatterGatherSupport

implements ScatterGatherSupport<EarlybirdRequestContext, EarlybirdResponse> {

private static final EarlybirdResponse EMPTY\_RESPONSE = newEmptyResponse();

private final PartitionMappingManager partitionMappingManager;

private final EarlybirdCluster cluster;

private final EarlybirdFeatureSchemaMerger featureSchemaMerger;

@Inject

protected EarlybirdServiceScatterGatherSupport(PartitionMappingManager partitionMappingManager,

EarlybirdCluster cluster,

EarlybirdFeatureSchemaMerger featureSchemaMerger) {

this.partitionMappingManager = partitionMappingManager;

this.cluster = cluster;

this.featureSchemaMerger = featureSchemaMerger;

}

/\*\*

\* Fans out the original request to all partitions.

\*/

private List<EarlybirdRequestContext> fanoutToAllPartitions(

EarlybirdRequestContext requestContext, int numPartitions) {

// We don't need to create a deep copy of the original requestContext for every partition,

// because requests are not rewritten once they get to this level: our roots have filters

// that rewrite the requests at the top-level, but we do not rewrite requests per-partition.

List<EarlybirdRequestContext> requestContexts = new ArrayList<>(numPartitions);

for (int i = 0; i < numPartitions; ++i) {

requestContexts.add(requestContext);

}

return requestContexts;

}

private Map<Integer, List<Long>> populateIdsForPartition(EarlybirdRequestContext requestContext) {

Map<Integer, List<Long>> perPartitionIds = Maps.newHashMap();

// Based on partition type, populate map for every partition if needed.

if (partitionMappingManager.getPartitionDataType() == PartitionDataType.USER\_ID

&& requestContext.getRequest().getSearchQuery().getFromUserIDFilter64Size() > 0) {

for (long userId : requestContext.getRequest().getSearchQuery().getFromUserIDFilter64()) {

int userPartition = partitionMappingManager.getPartitionIdForUserId(userId);

if (!perPartitionIds.containsKey(userPartition)) {

perPartitionIds.put(userPartition, Lists.newArrayList());

}

perPartitionIds.get(userPartition).add(userId);

}

} else if (partitionMappingManager.getPartitionDataType() == PartitionDataType.TWEET\_ID

&& requestContext.getRequest().getSearchQuery().getSearchStatusIdsSize() > 0) {

for (long id : requestContext.getRequest().getSearchQuery().getSearchStatusIds()) {

int tweetPartition = partitionMappingManager.getPartitionIdForTweetId(id);

if (!perPartitionIds.containsKey(tweetPartition)) {

perPartitionIds.put(tweetPartition, Lists.newArrayList());

}

perPartitionIds.get(tweetPartition).add(id);

}

}

return perPartitionIds;

}

private void setPerPartitionIds(EarlybirdRequest request, List<Long> ids) {

if (partitionMappingManager.getPartitionDataType() == PartitionDataType.USER\_ID) {

request.getSearchQuery().setFromUserIDFilter64(ids);

} else {

request.getSearchQuery().setSearchStatusIds(Sets.newHashSet(ids));

}

}

@Override

public EarlybirdResponse emptyResponse() {

return EMPTY\_RESPONSE;

}

public static final EarlybirdResponse newEmptyResponse() {

return new EarlybirdResponse(EarlybirdResponseCode.PARTITION\_SKIPPED, 0)

.setSearchResults(new ThriftSearchResults());

}

@Override

public List<EarlybirdRequestContext> rewriteRequest(

EarlybirdRequestContext requestContext, int rootNumPartitions) {

int numPartitions = partitionMappingManager.getNumPartitions();

Preconditions.checkState(rootNumPartitions == numPartitions,

"Root's configured numPartitions is different from that configured in database.yml.");

// Rewrite query based on "multi\_term\_disjunction id/from\_user\_id" and partition id if needed.

Map<Integer, Query> perPartitionQueryMap =

requestContext.getRequest().getSearchQuery().getSearchStatusIdsSize() == 0

? EarlybirdRootQueryUtils.rewriteMultiTermDisjunctionPerPartitionFilter(

requestContext.getParsedQuery(),

partitionMappingManager,

numPartitions)

: Maps.newHashMap();

// Key: partition Id; Value: valid ids list for this partition

Map<Integer, List<Long>> perPartitionIds = populateIdsForPartition(requestContext);

if (perPartitionQueryMap.isEmpty() && perPartitionIds.isEmpty()) {

return fanoutToAllPartitions(requestContext, numPartitions);

}

List<EarlybirdRequestContext> requestContexts = new ArrayList<>(numPartitions);

for (int i = 0; i < numPartitions; ++i) {

requestContexts.add(null);

}

// Rewrite per partition queries if exist.

for (int i = 0; i < numPartitions; ++i) {

if (perPartitionIds.containsKey(i)) {

if (!perPartitionQueryMap.containsKey(i)) {

// Query does not need to be rewritten for the partition

// But we still need to create a copy, because we're gonna

// set fromUserIDFilter64/searchStatusIds

requestContexts.set(i, requestContext.deepCopy());

setPerPartitionIds(requestContexts.get(i).getRequest(), perPartitionIds.get(i));

} else if (perPartitionQueryMap.get(i) != NO\_MATCH\_CONJUNCTION) {

requestContexts.set(i, EarlybirdRequestContext.copyRequestContext(

requestContext, perPartitionQueryMap.get(i)));

setPerPartitionIds(requestContexts.get(i).getRequest(), perPartitionIds.get(i));

}

} else if (perPartitionIds.isEmpty()) {

// The fromUserIDFilter64/searchStatusIds field is not set on the original request,

// perPartitionQueryMap should decide if we send a request to this partition or not

if (!perPartitionQueryMap.containsKey(i)) {

// Query does not need to be rewritten for the partition

// Don't need to create a copy, because request context won't be changed afterwards

requestContexts.set(i, requestContext);

} else if (perPartitionQueryMap.get(i) != NO\_MATCH\_CONJUNCTION) {

requestContexts.set(i, EarlybirdRequestContext.copyRequestContext(

requestContext, perPartitionQueryMap.get(i)));

}

}

}

return requestContexts;

}

/\*\*

\* Merges all the sub-results indexed by the partition id. Sub-results with value null

\* indicate an error with that partition such as timeout etc.

\*/

@Override

public Future<EarlybirdResponse> merge(EarlybirdRequestContext requestContext,

List<Future<EarlybirdResponse>> responses) {

EarlybirdResponseMerger merger = EarlybirdResponseMerger.getResponseMerger(

requestContext,

responses,

new PartitionResponseAccumulator(),

cluster,

featureSchemaMerger,

partitionMappingManager.getNumPartitions());

return merger.merge();

}

@Override

public boolean isSuccess(EarlybirdResponse earlybirdResponse) {

return EarlybirdResponseUtil.isSuccessfulResponse(earlybirdResponse);

}

@Override

public boolean isTimeout(EarlybirdResponse earlybirdResponse) {

return earlybirdResponse.getResponseCode() == EarlybirdResponseCode.SERVER\_TIMEOUT\_ERROR;

}

@Override

public boolean isClientCancel(EarlybirdResponse earlybirdResponse) {

return earlybirdResponse.getResponseCode() == EarlybirdResponseCode.CLIENT\_CANCEL\_ERROR;

}

@Override

public EarlybirdResponse errorResponse(String debugString) {

return new EarlybirdResponse()

.setResponseCode(EarlybirdResponseCode.TRANSIENT\_ERROR)

.setDebugString(debugString);

}

}