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

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import com.google.common.base.Preconditions;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.util.Clock;

import com.twitter.finagle.Service;

import com.twitter.search.common.decider.SearchDecider;

import com.twitter.search.common.futures.Futures;

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

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

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

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.ThriftSearchResults;

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

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

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

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

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

import com.twitter.search.earlybird\_root.filters.EarlybirdTimeRangeFilter;

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

import com.twitter.search.queryparser.util.QueryUtil;

import com.twitter.util.Function;

import com.twitter.util.Function0;

import com.twitter.util.Future;

/\*\*

\* For Recency traffic SuperRoot hits realtime and/or protected realtime first and then archive

\*/

public abstract class AbstractRecencyAndRelevanceRequestRouter extends RequestRouter {

public static final String FULL\_ARCHIVE\_AVAILABLE\_FOR\_GET\_PROTECTED\_TWEETS\_ONLY\_DECIDER\_KEY =

"superroot\_full\_archive\_cluster\_available\_for\_get\_protected\_tweets\_only\_requests";

public static final String FULL\_ARCHIVE\_AVAILABLE\_FOR\_NOT\_ENOUGH\_PROTECTED\_RESULTS\_DECIDER\_KEY =

"superroot\_full\_archive\_cluster\_available\_for\_requests\_without\_enough\_protected\_results";

private static final Logger LOG =

LoggerFactory.getLogger(AbstractRecencyAndRelevanceRequestRouter.class);

private final String skipProtectedClusterDeciderKey;

private final String skipFullArchiveClusterDeciderKey;

private final SearchCounter realtimeResponseInvalidCounter;

private final SearchCounter realtimeResponseSearchResultsNotSetCounter;

private final SearchCounter minSearchedStatusIdLargerThanRequestMaxIdCounter;

private final SearchCounter minSearchedStatusIdLargerThanRequestUntilTimeCounter;

private final Service<EarlybirdRequestContext, EarlybirdResponse> realtime;

private final Service<EarlybirdRequestContext, EarlybirdResponse> protectedRealtime;

private final Service<EarlybirdRequestContext, EarlybirdResponse> fullArchive;

private final SuperRootResponseMerger responseMerger;

private final SearchDecider decider;

AbstractRecencyAndRelevanceRequestRouter(

Service<EarlybirdRequestContext, EarlybirdResponse> realtime,

Service<EarlybirdRequestContext, EarlybirdResponse> protectedRealtime,

Service<EarlybirdRequestContext, EarlybirdResponse> fullArchive,

EarlybirdTimeRangeFilter realtimeTimeRangeFilter,

EarlybirdTimeRangeFilter protectedTimeRangeFilter,

EarlybirdTimeRangeFilter fullArchiveTimeRangeFilter,

ThriftSearchRankingMode rankingMode,

Clock clock,

SearchDecider decider,

EarlybirdFeatureSchemaMerger featureSchemaMerger) {

LOG.info("Instantiating AbstractRecencyAndRelevanceRequestRouter");

this.realtime = realtimeTimeRangeFilter.andThen(realtime);

this.protectedRealtime = protectedTimeRangeFilter.andThen(protectedRealtime);

this.fullArchive = fullArchiveTimeRangeFilter.andThen(fullArchive);

this.responseMerger = new SuperRootResponseMerger(rankingMode, featureSchemaMerger, clock);

this.decider = decider;

String rankingModeForStats = rankingMode.name().toLowerCase();

skipProtectedClusterDeciderKey =

String.format("superroot\_skip\_protected\_cluster\_for\_%s\_requests", rankingModeForStats);

skipFullArchiveClusterDeciderKey =

String.format("superroot\_skip\_full\_archive\_cluster\_for\_%s\_requests", rankingModeForStats);

realtimeResponseInvalidCounter =

SearchCounter.export(rankingModeForStats + "\_realtime\_response\_invalid");

realtimeResponseSearchResultsNotSetCounter =

SearchCounter.export(rankingModeForStats + "\_realtime\_response\_search\_results\_not\_set");

minSearchedStatusIdLargerThanRequestMaxIdCounter = SearchCounter.export(

rankingModeForStats + "\_min\_searched\_status\_id\_larger\_than\_request\_max\_id");

minSearchedStatusIdLargerThanRequestUntilTimeCounter = SearchCounter.export(

rankingModeForStats + "\_min\_searched\_status\_id\_larger\_than\_request\_until\_time");

}

private void checkRequestPreconditions(EarlybirdRequest request) {

// CollectorParams should be set in EarlybirdRequestUtil.checkAndSetCollectorParams().

Preconditions.checkNotNull(request.getSearchQuery().getCollectorParams());

// return a Client error if the num results are less than 0

if (request.getSearchQuery().getNumResults() < 0) {

throw new ClientErrorException("The request.searchQuery.numResults field can't be negative");

}

if (request.getSearchQuery().getCollectorParams().getNumResultsToReturn() < 0) {

throw new ClientErrorException("The request.searchQuery.collectorParams.numResultsToReturn "

+ "field can't be negative");

}

}

/\*\*

\* Hit realtime and/or protected realtime first, if not enough results, then hit archive,

\* merge the results.

\*/

@Override

public Future<EarlybirdResponse> route(final EarlybirdRequestContext requestContext) {

EarlybirdRequest request = requestContext.getRequest();

this.checkRequestPreconditions(request);

ArrayList<RequestResponse> savedRequestResponses = new ArrayList<>();

// If clients do not define numResults to return or the numResults requested are 0

// return an empty EarlyBirdResponse without hitting any service.

if (request.getSearchQuery().getNumResults() == 0

|| request.getSearchQuery().getCollectorParams().getNumResultsToReturn() == 0) {

return Future.value(successNoResultsResponse());

}

// Realtime earlybird response is already required. Even if the service is not called

// the result passed to the mergers should be a valid one.

EarlybirdServiceResponse.ServiceState realtimeServiceState =

getRealtimeServiceState(requestContext);

final Future<EarlybirdServiceResponse> realtimeResponseFuture =

realtimeServiceState.serviceWasCalled()

? getRealtimeResponse(savedRequestResponses, requestContext)

: Future.value(EarlybirdServiceResponse.serviceNotCalled(realtimeServiceState));

// If no flock response (followedUserIds) is set, request wont be sent to protected.

EarlybirdServiceResponse.ServiceState protectedServiceState =

getProtectedServiceState(requestContext);

final Future<EarlybirdServiceResponse> protectedResponseFuture =

protectedServiceState.serviceWasCalled()

? getProtectedResponse(savedRequestResponses, requestContext)

: Future.value(EarlybirdServiceResponse.serviceNotCalled(protectedServiceState));

final Future<EarlybirdServiceResponse> archiveResponseFuture =

Futures.flatMap(realtimeResponseFuture, protectedResponseFuture,

new Function0<Future<EarlybirdServiceResponse>>() {

@Override

public Future<EarlybirdServiceResponse> apply() {

EarlybirdServiceResponse realtimeResponse = Futures.get(realtimeResponseFuture);

EarlybirdServiceResponse protectedResponse = Futures.get(protectedResponseFuture);

EarlybirdServiceResponse.ServiceState fullArchiveServiceState =

getFullArchiveServiceState(requestContext, realtimeResponse, protectedResponse);

return fullArchiveServiceState.serviceWasCalled()

? getFullArchiveResponse(savedRequestResponses, requestContext,

realtimeResponse.getResponse(), protectedResponse.getResponse())

: Future.value(

EarlybirdServiceResponse.serviceNotCalled(fullArchiveServiceState));

}

}

);

Future<EarlybirdResponse> mergedResponse = responseMerger.mergeResponseFutures(

requestContext, realtimeResponseFuture, protectedResponseFuture, archiveResponseFuture);

mergedResponse = mergedResponse

.map(RequestRouterUtil.checkMinSearchedStatusId(

requestContext,

"max\_id",

EarlybirdRequestUtil.getRequestMaxId(requestContext.getParsedQuery()),

realtimeResponseFuture,

protectedResponseFuture,

archiveResponseFuture,

minSearchedStatusIdLargerThanRequestMaxIdCounter))

.map(RequestRouterUtil.checkMinSearchedStatusId(

requestContext,

"until\_time",

EarlybirdRequestUtil.getRequestMaxIdFromUntilTime(requestContext.getParsedQuery()),

realtimeResponseFuture,

protectedResponseFuture,

archiveResponseFuture,

minSearchedStatusIdLargerThanRequestUntilTimeCounter));

return this.maybeAttachSentRequestsToDebugInfo(

savedRequestResponses,

requestContext,

mergedResponse

);

}

private EarlybirdResponse successNoResultsResponse() {

return new EarlybirdResponse(EarlybirdResponseCode.SUCCESS, 0)

.setSearchResults(new ThriftSearchResults().setResults(Collections.emptyList()));

}

protected abstract boolean shouldSendRequestToFullArchiveCluster(

EarlybirdRequest request, EarlybirdResponse realtimeResponse);

/\*\* Determines if the protected service is available and if a request should be sent to it. \*/

private EarlybirdServiceResponse.ServiceState getProtectedServiceState(

EarlybirdRequestContext requestContext) {

if (!requestContext.getRequest().isSetFollowedUserIds()

|| requestContext.getRequest().getFollowedUserIds().isEmpty()) {

return EarlybirdServiceResponse.ServiceState.SERVICE\_NOT\_REQUESTED;

}

if (decider.isAvailable(skipProtectedClusterDeciderKey)) {

return EarlybirdServiceResponse.ServiceState.SERVICE\_NOT\_AVAILABLE;

}

return EarlybirdServiceResponse.ServiceState.SERVICE\_CALLED;

}

/\*\* Determines if the realtime service is available and if a request should be sent to it. \*/

private EarlybirdServiceResponse.ServiceState getRealtimeServiceState(

EarlybirdRequestContext requestContext) {

EarlybirdRequest request = requestContext.getRequest();

// SERVICE\_NOT\_REQUESTED should always be returned before other states as

// SuperRootResponseMerger has special logic for this case.

if (request.isSetGetProtectedTweetsOnly() && request.isGetProtectedTweetsOnly()) {

return EarlybirdServiceResponse.ServiceState.SERVICE\_NOT\_REQUESTED;

}

return EarlybirdServiceResponse.ServiceState.SERVICE\_CALLED;

}

/\*\* Determines if the full archive service is available and if a request should be sent to it. \*/

private EarlybirdServiceResponse.ServiceState getFullArchiveServiceState(

EarlybirdRequestContext requestContext,

EarlybirdServiceResponse publicServiceResponse,

EarlybirdServiceResponse protectedServiceResponse) {

// SERVICE\_NOT\_REQUESTED should be always be returned before other states as

// SuperRootResponseMerger has special logic for this case.

if (!requestContext.getRequest().isSetGetOlderResults()

|| !requestContext.getRequest().isGetOlderResults()) {

return EarlybirdServiceResponse.ServiceState.SERVICE\_NOT\_REQUESTED;

}

// allow requesting full archive service when decider is enabled

if (!decider.isAvailable(FULL\_ARCHIVE\_AVAILABLE\_FOR\_GET\_PROTECTED\_TWEETS\_ONLY\_DECIDER\_KEY)

&& requestContext.getRequest().isSetGetProtectedTweetsOnly()

&& requestContext.getRequest().isGetProtectedTweetsOnly()) {

return EarlybirdServiceResponse.ServiceState.SERVICE\_NOT\_REQUESTED;

}

if (decider.isAvailable(skipFullArchiveClusterDeciderKey)) {

return EarlybirdServiceResponse.ServiceState.SERVICE\_NOT\_AVAILABLE;

}

boolean serviceWasCalledForPublic =

getFullArchiveServiceState(requestContext, publicServiceResponse).serviceWasCalled();

boolean serviceWasCalledForProtected =

decider.isAvailable(FULL\_ARCHIVE\_AVAILABLE\_FOR\_NOT\_ENOUGH\_PROTECTED\_RESULTS\_DECIDER\_KEY)

&& getFullArchiveServiceState(requestContext, protectedServiceResponse).serviceWasCalled();

if (!serviceWasCalledForPublic && !serviceWasCalledForProtected) {

return EarlybirdServiceResponse.ServiceState.SERVICE\_NOT\_CALLED;

}

return EarlybirdServiceResponse.ServiceState.SERVICE\_CALLED;

}

private EarlybirdServiceResponse.ServiceState getFullArchiveServiceState(

EarlybirdRequestContext requestContext,

EarlybirdServiceResponse realtimeServiceResponse) {

EarlybirdResponse realtimeResponse = realtimeServiceResponse.getResponse();

if (!EarlybirdResponseMergeUtil.isValidResponse(realtimeResponse)) {

realtimeResponseInvalidCounter.increment();

return EarlybirdServiceResponse.ServiceState.SERVICE\_NOT\_CALLED;

}

if (!realtimeResponse.isSetSearchResults()) {

realtimeResponseSearchResultsNotSetCounter.increment();

return EarlybirdServiceResponse.ServiceState.SERVICE\_NOT\_CALLED;

}

if (!shouldSendRequestToFullArchiveCluster(requestContext.getRequest(), realtimeResponse)) {

return EarlybirdServiceResponse.ServiceState.SERVICE\_NOT\_CALLED;

}

return EarlybirdServiceResponse.ServiceState.SERVICE\_CALLED;

}

/\*\*

\* Modify the original request context based on the followedUserId field and then send the

\* request to the protected cluster.

\*/

private Future<EarlybirdServiceResponse> getProtectedResponse(

ArrayList<RequestResponse> savedRequestResponses,

final EarlybirdRequestContext requestContext) {

EarlybirdRequestContext protectedRequestContext =

EarlybirdRequestContext.newContextWithRestrictFromUserIdFilter64(requestContext);

Preconditions.checkArgument(

protectedRequestContext.getRequest().getSearchQuery().isSetFromUserIDFilter64());

// SERVICE\_NOT\_REQUESTED should be always be returned before other states as

// SuperRootResponseMerger has special logic for this case.

if (protectedRequestContext.getRequest().getSearchQuery().getFromUserIDFilter64().isEmpty()) {

return Future.value(EarlybirdServiceResponse.serviceNotCalled(

EarlybirdServiceResponse.ServiceState.SERVICE\_NOT\_REQUESTED));

}

if (requestContext.getRequest().isSetAdjustedProtectedRequestParams()) {

adjustRequestParams(protectedRequestContext.getRequest(),

requestContext.getRequest().getAdjustedProtectedRequestParams());

}

LOG.debug("Request sent to the protected cluster: {}", protectedRequestContext.getRequest());

return toEarlybirdServiceResponseFuture(

savedRequestResponses,

protectedRequestContext,

"protected",

this.protectedRealtime

);

}

private Future<EarlybirdServiceResponse> getRealtimeResponse(

ArrayList<RequestResponse> savedRequestResponses,

EarlybirdRequestContext requestContext) {

return toEarlybirdServiceResponseFuture(

savedRequestResponses,

requestContext,

"realtime",

this.realtime);

}

/\*\*

\* Modifying the existing max id filter of the request or appending a new

\* max id filter and then send the request to the full archive cluster.

\*/

private Future<EarlybirdServiceResponse> getFullArchiveResponse(

ArrayList<RequestResponse> savedRequestResponses,

EarlybirdRequestContext requestContext,

EarlybirdResponse realtimeResponse,

EarlybirdResponse protectedResponse) {

long realtimeMinId = getMinSearchedId(realtimeResponse);

long protectedMinId = getMinSearchedId(protectedResponse);

// if both realtime and protected min searched ids are available, the larger(newer) one is used

// to make sure no tweets are left out. However, this means it might introduce duplicates for

// the other response. The response merger will dedup the response. This logic is enabled

// when full archive cluster is available for requests without enough protected results.

long minId =

decider.isAvailable(FULL\_ARCHIVE\_AVAILABLE\_FOR\_NOT\_ENOUGH\_PROTECTED\_RESULTS\_DECIDER\_KEY)

? Math.max(realtimeMinId, protectedMinId) : realtimeMinId;

if (minId <= 0) {

// If the realtime response doesn't have a minSearchedStatusID set, get all results from

// the full archive cluster.

minId = Long.MAX\_VALUE;

}

// The [max\_id] operator is inclusive in earlybirds. This means that a query with [max\_id X]

// will return tweet X, if X matches the rest of the query. So we should add a [max\_id (X - 1)]

// operator to the full archive query (instead of [max\_id X]). Otherwise, we could end up with

// duplicates. For example:

//

// realtime response: results = [ 100, 90, 80 ], minSearchedStatusID = 80

// full archive request: [max\_id 80]

// full archive response: results = [ 80, 70, 60 ]

//

// In this case, tweet 80 would be returned from both the realtime and full archive clusters.

EarlybirdRequestContext archiveRequestContext =

EarlybirdRequestContext.copyRequestContext(

requestContext,

QueryUtil.addOrReplaceMaxIdFilter(

requestContext.getParsedQuery(),

minId - 1));

if (requestContext.getRequest().isSetAdjustedFullArchiveRequestParams()) {

adjustRequestParams(archiveRequestContext.getRequest(),

requestContext.getRequest().getAdjustedFullArchiveRequestParams());

}

LOG.debug("Request sent to the full archive cluster: {},", archiveRequestContext.getRequest());

return toEarlybirdServiceResponseFuture(

savedRequestResponses,

archiveRequestContext,

"archive",

this.fullArchive

);

}

private long getMinSearchedId(EarlybirdResponse response) {

return response != null && response.isSetSearchResults()

? response.getSearchResults().getMinSearchedStatusID() : 0;

}

private void adjustRequestParams(EarlybirdRequest request,

AdjustedRequestParams adjustedRequestParams) {

ThriftSearchQuery searchQuery = request.getSearchQuery();

if (adjustedRequestParams.isSetNumResults()) {

searchQuery.setNumResults(adjustedRequestParams.getNumResults());

if (searchQuery.isSetCollectorParams()) {

searchQuery.getCollectorParams().setNumResultsToReturn(

adjustedRequestParams.getNumResults());

}

}

if (adjustedRequestParams.isSetMaxHitsToProcess()) {

searchQuery.setMaxHitsToProcess(adjustedRequestParams.getMaxHitsToProcess());

if (searchQuery.isSetRelevanceOptions()) {

searchQuery.getRelevanceOptions().setMaxHitsToProcess(

adjustedRequestParams.getMaxHitsToProcess());

}

if (searchQuery.isSetCollectorParams()

&& searchQuery.getCollectorParams().isSetTerminationParams()) {

searchQuery.getCollectorParams().getTerminationParams().setMaxHitsToProcess(

adjustedRequestParams.getMaxHitsToProcess());

}

}

if (adjustedRequestParams.isSetReturnAllResults()) {

if (searchQuery.isSetRelevanceOptions()) {

searchQuery.getRelevanceOptions().setReturnAllResults(

adjustedRequestParams.isReturnAllResults());

}

}

}

private Future<EarlybirdServiceResponse> toEarlybirdServiceResponseFuture(

List<RequestResponse> savedRequestResponses,

EarlybirdRequestContext requestContext,

String sentTo,

Service<EarlybirdRequestContext, EarlybirdResponse> service) {

Future<EarlybirdResponse> responseFuture = service.apply(requestContext);

this.saveRequestResponse(

savedRequestResponses, sentTo, requestContext, responseFuture

);

return responseFuture.map(new Function<EarlybirdResponse, EarlybirdServiceResponse>() {

@Override

public EarlybirdServiceResponse apply(EarlybirdResponse response) {

return EarlybirdServiceResponse.serviceCalled(response);

}

});

}

}