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

import java.util.Collections;

import java.util.Map;

import java.util.Optional;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.collect.Maps;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.finagle.Service;

import com.twitter.finagle.SimpleFilter;

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

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

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

import com.twitter.search.earlybird.config.ServingRange;

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.EarlybirdRequestContext;

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

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

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

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

import com.twitter.util.Future;

/\*\*

\* A Finagle filter used to filter requests to tiers.

\* Parses serialized query on Earlybird request, and extracts since / until / since\_id / max\_id

\* operators. This filter then tests whether the request overlaps with the given tier. If there

\* is no overlap, an empty response is returned without actually forwarding the requests to the

\* underlying service.

\*/

public class EarlybirdTimeRangeFilter extends

SimpleFilter<EarlybirdRequestContext, EarlybirdResponse> {

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

private static final EarlybirdResponse ERROR\_RESPONSE =

new EarlybirdResponse(EarlybirdResponseCode.PERSISTENT\_ERROR, 0)

.setSearchResults(new ThriftSearchResults());

private final ServingRangeProvider servingRangeProvider;

private final Optional<EarlybirdTimeFilterQueryRewriter> queryRewriter;

private static final Map<EarlybirdRequestType, SearchCounter> FAILED\_REQUESTS;

static {

final Map<EarlybirdRequestType, SearchCounter> tempMap =

Maps.newEnumMap(EarlybirdRequestType.class);

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

tempMap.put(requestType, SearchCounter.export(

"time\_range\_filter\_" + requestType.getNormalizedName() + "\_failed\_requests"));

}

FAILED\_REQUESTS = Collections.unmodifiableMap(tempMap);

}

public static EarlybirdTimeRangeFilter newTimeRangeFilterWithQueryRewriter(

ServingRangeProvider servingRangeProvider,

SearchDecider decider) {

return new EarlybirdTimeRangeFilter(servingRangeProvider,

Optional.of(new EarlybirdTimeFilterQueryRewriter(servingRangeProvider, decider)));

}

public static EarlybirdTimeRangeFilter newTimeRangeFilterWithoutQueryRewriter(

ServingRangeProvider servingRangeProvider) {

return new EarlybirdTimeRangeFilter(servingRangeProvider, Optional.empty());

}

/\*\*

\* Construct a filter that avoids forwarding requests to unrelated tiers

\* based on requests' since / until / since\_id / max\_id.

\* @param provider Holds the boundary information.

\*/

EarlybirdTimeRangeFilter(

ServingRangeProvider provider,

Optional<EarlybirdTimeFilterQueryRewriter> rewriter) {

this.servingRangeProvider = provider;

this.queryRewriter = rewriter;

}

public ServingRangeProvider getServingRangeProvider() {

return servingRangeProvider;

}

@Override

public Future<EarlybirdResponse> apply(

EarlybirdRequestContext requestContext,

Service<EarlybirdRequestContext, EarlybirdResponse> service) {

Query parsedQuery = requestContext.getParsedQuery();

if (parsedQuery != null) {

// Only perform filtering if serialized query is set.

try {

IdTimeRanges queryRanges = IdTimeRanges.fromQuery(parsedQuery);

if (queryRanges == null) {

// No time ranges in query.

return issueServiceRequest(service, requestContext);

}

ServingRange servingRange =

servingRangeProvider.getServingRange(

requestContext, requestContext.useOverrideTierConfig());

if (queryDoesNotOverlapWithServingRange(queryRanges, servingRange)) {

return Future.value(tierSkippedResponse(requestContext.getEarlybirdRequestType(),

servingRange));

} else {

return issueServiceRequest(service, requestContext);

}

} catch (QueryParserException e) {

LOG.warn("Unable to get IdTimeRanges from query: " + parsedQuery.serialize());

// The failure here is not due to a miss-formed query from the client, since we already

// were able to successfully get a parsed Query from the request.

// If we can't determine the time ranges, pass the query along to the tier, and just

// restrict it to the timeranges of the tier.

return issueServiceRequest(service, requestContext);

}

} else {

// There's no serialized query. Just pass through like an identity filter.

return issueServiceRequest(service, requestContext);

}

}

private boolean queryDoesNotOverlapWithServingRange(IdTimeRanges queryRanges,

ServingRange servingRange) {

// As long as a query overlaps with the tier serving range on either side,

// the request is not filtered. I.e. we want to be conservative when doing this filtering,

// because it is just an optimization. We ignore the inclusiveness / exclusiveness of the

// boundaries. If the tier boundary and the query boundry happen to be the same, we do not

// filter the request.

return queryRanges.getSinceIDExclusive().or(0L)

> servingRange.getServingRangeMaxId()

|| queryRanges.getMaxIDInclusive().or(Long.MAX\_VALUE)

< servingRange.getServingRangeSinceId()

|| queryRanges.getSinceTimeInclusive().or(0)

> servingRange.getServingRangeUntilTimeSecondsFromEpoch()

|| queryRanges.getUntilTimeExclusive().or(Integer.MAX\_VALUE)

< servingRange.getServingRangeSinceTimeSecondsFromEpoch();

}

private Future<EarlybirdResponse> issueServiceRequest(

Service<EarlybirdRequestContext, EarlybirdResponse> service,

EarlybirdRequestContext requestContext) {

try {

EarlybirdRequestContext request = requestContext;

if (queryRewriter.isPresent()) {

request = queryRewriter.get().rewriteRequest(requestContext);

}

return service.apply(request);

} catch (QueryParserException e) {

FAILED\_REQUESTS.get(requestContext.getEarlybirdRequestType()).increment();

String msg = "Failed to add time filter operators";

LOG.error(msg, e);

// Note that in this case it is not clear whether the error is the client's fault or our

// fault, so we don't necessarily return a CLIENT\_ERROR here.

// Currently this actually returns a PERSISTENT\_ERROR.

if (requestContext.getRequest().getDebugMode() > 0) {

return Future.value(

ERROR\_RESPONSE.deepCopy().setDebugString(msg + ": " + e.getMessage()));

} else {

return Future.value(ERROR\_RESPONSE);

}

}

}

/\*\*

\* Creates a tier skipped response, based on the given request type.

\*

\* For recency, relevance, facets and top tweets requests, this method returns a SUCCESS response

\* with no search results and the minSearchedStatusID and maxSearchedStatusID appropriately set.

\* For term stats response, it returns a TIER\_SKIPPED response, but we need to revisit this.

\*

\* @param requestType The type of the request.

\* @param servingRange The serving range of the tier that we're skipping.

\*/

@VisibleForTesting

public static EarlybirdResponse tierSkippedResponse(

EarlybirdRequestType requestType,

ServingRange servingRange) {

String debugMessage =

"Tier skipped because it does not intersect with query time boundaries.";

if (requestType == EarlybirdRequestType.TERM\_STATS) {

// If it's a term stats request, return a TIER\_SKIPPED response for now.

// But we need to figure out the right thing to do here.

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

.setDebugString(debugMessage);

} else {

// minIds in ServingRange instances are set to tierLowerBoundary - 1, because the

// since\_id operator is exclusive. The max\_id operator on the other hand is inclusive,

// so maxIds in ServingRange instances are also set to tierUpperBoundary - 1.

// Here we want both of them to be inclusive, so we need to increment the minId by 1.

return EarlybirdResponseUtil.tierSkippedRootResponse(

servingRange.getServingRangeSinceId() + 1,

servingRange.getServingRangeMaxId(),

debugMessage);

}

}

}