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

import java.util.List;

import com.google.common.base.Optional;

import com.google.common.collect.Lists;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

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

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

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

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

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

import com.twitter.util.Await;

import com.twitter.util.Function;

import com.twitter.util.Future;

public final class RequestRouterUtil {

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

private RequestRouterUtil() {

}

/\*\*

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

\* than the max ID in the request.

\*

\* @param requestContext The request context that stores the request.

\* @param operator The operator that we're checking against (max\_id or until\_time).

\* @param requestMaxId The maxId specified in the request (in the given operator).

\* @param realtimeResponseFuture The response from the realtime cluster.

\* @param protectedResponseFuture The response from the protected cluster.

\* @param fullArchiveResponseFuture The response from the full archive cluster.

\* @param stat The stat to increment if minSearchedStatusID on the merged response is higher than

\* the max ID in the request.

\* @return A function that checks if the minSearchedStatusID on the merged response is higher than

\* the max ID in the request.

\*/

public static Function<EarlybirdResponse, EarlybirdResponse> checkMinSearchedStatusId(

final EarlybirdRequestContext requestContext,

final String operator,

final Optional<Long> requestMaxId,

final Future<EarlybirdServiceResponse> realtimeResponseFuture,

final Future<EarlybirdServiceResponse> protectedResponseFuture,

final Future<EarlybirdServiceResponse> fullArchiveResponseFuture,

final SearchCounter stat) {

return new Function<EarlybirdResponse, EarlybirdResponse>() {

@Override

public EarlybirdResponse apply(EarlybirdResponse mergedResponse) {

if (requestMaxId.isPresent()

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

&& mergedResponse.isSetSearchResults()

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

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

if (minSearchedStatusId > requestMaxId.get()) {

stat.increment();

// We're logging this only for STRICT RECENCY as it was very spammy for all types of

// request. We don't expect this to happen for STRICT RECENCY but we're tracking

// with the stat when it happens for RELEVANCE and RECENCY

if (requestContext.getEarlybirdRequestType() == EarlybirdRequestType.STRICT\_RECENCY) {

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

+ operator + " ({})."

+ "\nrequest type: {}"

+ "\nrequest: {}"

+ "\nmerged response: {}"

+ "\nrealtime response: {}"

+ "\nprotected response: {}"

+ "\nfull archive response: {}";

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

logMessageParams.add(minSearchedStatusId);

logMessageParams.add(requestMaxId.get());

logMessageParams.add(requestContext.getEarlybirdRequestType());

logMessageParams.add(requestContext.getRequest());

logMessageParams.add(mergedResponse);

// The realtime, protected and full archive response futures are "done" at this point:

// we have to wait for them in order to build the merged response. So it's ok to call

// Await.result() here to get the responses: it's a no-op.

try {

logMessageParams.add(Await.result(realtimeResponseFuture).getResponse());

} catch (Exception e) {

logMessageParams.add(e);

}

try {

logMessageParams.add(Await.result(protectedResponseFuture).getResponse());

} catch (Exception e) {

logMessageParams.add(e);

}

try {

logMessageParams.add(Await.result(fullArchiveResponseFuture).getResponse());

} catch (Exception e) {

logMessageParams.add(e);

}

LOG.warn(logMessage, logMessageParams.toArray());

}

}

}

return mergedResponse;

}

};

}

}