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

import java.util.List;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.cache.CacheBuilder;

import com.google.common.cache.CacheLoader;

import com.google.common.cache.LoadingCache;

import com.twitter.finagle.Service;

import com.twitter.finagle.SimpleFilter;

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

import com.twitter.search.earlybird.common.ClientIdUtil;

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.ThriftSearchResult;

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

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

import com.twitter.util.Future;

import com.twitter.util.FutureEventListener;

/\*\*

\* Filter that is tracking the engagement stats returned from Earlybirds.

\*/

public class MetadataTrackingFilter extends SimpleFilter<EarlybirdRequest, EarlybirdResponse> {

private static final String SCORING\_SIGNAL\_STAT\_PREFIX = "scoring\_signal\_";

private static final String SCORE\_STAT\_PATTERN = "client\_id\_score\_tracker\_for\_%s\_x100";

@VisibleForTesting

static final SearchMovingAverage SCORING\_SIGNAL\_FAV\_COUNT =

SearchMovingAverage.export(SCORING\_SIGNAL\_STAT\_PREFIX + "fav\_count");

@VisibleForTesting

static final SearchMovingAverage SCORING\_SIGNAL\_REPLY\_COUNT =

SearchMovingAverage.export(SCORING\_SIGNAL\_STAT\_PREFIX + "reply\_count");

@VisibleForTesting

static final SearchMovingAverage SCORING\_SIGNAL\_RETWEET\_COUNT =

SearchMovingAverage.export(SCORING\_SIGNAL\_STAT\_PREFIX + "retweet\_count");

@VisibleForTesting

static final LoadingCache<String, SearchMovingAverage> CLIENT\_SCORE\_METRICS\_LOADING\_CACHE =

CacheBuilder.newBuilder().build(new CacheLoader<String, SearchMovingAverage>() {

public SearchMovingAverage load(String clientId) {

return SearchMovingAverage.export(String.format(SCORE\_STAT\_PATTERN, clientId));

}

});

@Override

public Future<EarlybirdResponse> apply(final EarlybirdRequest request,

Service<EarlybirdRequest, EarlybirdResponse> service) {

Future<EarlybirdResponse> response = service.apply(request);

response.addEventListener(new FutureEventListener<EarlybirdResponse>() {

@Override

public void onSuccess(EarlybirdResponse earlybirdResponse) {

EarlybirdRequestType type = EarlybirdRequestType.of(request);

if (earlybirdResponse.responseCode == EarlybirdResponseCode.SUCCESS

&& type == EarlybirdRequestType.RELEVANCE

&& earlybirdResponse.isSetSearchResults()

&& earlybirdResponse.getSearchResults().isSetResults()) {

List<ThriftSearchResult> searchResults = earlybirdResponse.getSearchResults()

.getResults();

long totalFavoriteAmount = 0;

long totalReplyAmount = 0;

long totalRetweetAmount = 0;

double totalScoreX100 = 0;

for (ThriftSearchResult result : searchResults) {

if (!result.isSetMetadata()) {

continue;

}

ThriftSearchResultMetadata metadata = result.getMetadata();

if (metadata.isSetFavCount()) {

totalFavoriteAmount += metadata.getFavCount();

}

if (metadata.isSetReplyCount()) {

totalReplyAmount += metadata.getReplyCount();

}

if (metadata.isSetRetweetCount()) {

totalRetweetAmount += metadata.getRetweetCount();

}

if (metadata.isSetScore()) {

// Scale up the score by 100 so that scores are at least 1 and visible on viz graph

totalScoreX100 += metadata.getScore() \* 100;

}

}

// We only count present engagement counts but report the full size of the search results.

// This means that we consider the missing counts as being 0.

SCORING\_SIGNAL\_FAV\_COUNT.addSamples(totalFavoriteAmount, searchResults.size());

SCORING\_SIGNAL\_REPLY\_COUNT.addSamples(totalReplyAmount, searchResults.size());

SCORING\_SIGNAL\_RETWEET\_COUNT.addSamples(totalRetweetAmount, searchResults.size());

// Export per client id average scores.

String requestClientId = ClientIdUtil.getClientIdFromRequest(request);

String quotaClientId = ClientIdUtil.getQuotaClientId(requestClientId);

CLIENT\_SCORE\_METRICS\_LOADING\_CACHE.getUnchecked(quotaClientId)

.addSamples((long) totalScoreX100, searchResults.size());

}

}

@Override

public void onFailure(Throwable cause) { }

});

return response;

}

}