package com.twitter.search.feature\_update\_service.stats;

import java.util.concurrent.ConcurrentHashMap;

import java.util.concurrent.ConcurrentMap;

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

import com.twitter.search.feature\_update\_service.thriftjava.FeatureUpdateResponseCode;

/\*\* Stat tracking for the feature update ingester service. \*/

public class FeatureUpdateStats {

public static final String PREFIX = "feature\_update\_service\_";

private final SearchRateCounter requestRate = SearchRateCounter.export(

PREFIX + "requests");

private ConcurrentMap<String, SearchRateCounter> perClientRequestRate =

new ConcurrentHashMap<>();

private ConcurrentMap<String, SearchRateCounter> responseCodeRate =

new ConcurrentHashMap<>();

private ConcurrentMap<String, SearchRateCounter> preClientResponseCodeRate =

new ConcurrentHashMap<>();

/\*\*

\* Record metrics for a single incoming request.

\*/

public void clientRequest(String clientID) {

// 1. Track total request rate. It's better to precompute than compute the per client sum at

// query time.

requestRate.increment();

// 2. Track request rate per client.

incrementPerClientCounter(perClientRequestRate, clientRequestRateKey(clientID));

}

/\*\*

\* Record metrics for a single response.

\*/

public void clientResponse(String clientID, FeatureUpdateResponseCode responseCode) {

String code = responseCode.toString().toLowerCase();

// 1. Track rates per response code.

incrementPerClientCounter(responseCodeRate, responseCodeKey(code));

// 2. Track rates per client per response code.

incrementPerClientCounter(preClientResponseCodeRate, clientResponseCodeKey(clientID, code));

}

/\*\*

\* Returns the total number of requests.

\*/

public long getRequestRateCount() {

return requestRate.getCount();

}

/\*\*

\* Returns the total number of requests for the specified client.

\*/

public long getClientRequestCount(String clientID) {

String key = clientRequestRateKey(clientID);

if (perClientRequestRate.containsKey(key)) {

return perClientRequestRate.get(key).getCount();

}

return 0;

}

/\*\*

\* Returns the total number of responses with the specified code.

\*/

public long getResponseCodeCount(FeatureUpdateResponseCode responseCode) {

String code = responseCode.toString().toLowerCase();

String key = responseCodeKey(code);

if (responseCodeRate.containsKey(key)) {

return responseCodeRate.get(key).getCount();

}

return 0;

}

/\*\*

\* Returns the total number of responses to the specified client with the specified code.

\*/

public long getClientResponseCodeCount(String clientID, FeatureUpdateResponseCode responseCode) {

String code = responseCode.toString().toLowerCase();

String key = clientResponseCodeKey(clientID, code);

if (preClientResponseCodeRate.containsKey(key)) {

return preClientResponseCodeRate.get(key).getCount();

}

return 0;

}

private static String clientRequestRateKey(String clientID) {

return String.format(PREFIX + "requests\_for\_client\_id\_%s", clientID);

}

private static String responseCodeKey(String responseCode) {

return String.format(PREFIX + "response\_code\_%s", responseCode);

}

private static String clientResponseCodeKey(String clientID, String responseCode) {

return String.format(PREFIX + "response\_for\_client\_id\_%s\_code\_%s", clientID, responseCode);

}

private void incrementPerClientCounter(

ConcurrentMap<String, SearchRateCounter> rates,

String key

) {

rates.putIfAbsent(key, SearchRateCounter.export(key));

rates.get(key).increment();

}

}