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

import java.util.List;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import javax.inject.Inject;

import scala.runtime.BoxedUnit;

import com.twitter.common.util.Clock;

import com.twitter.finagle.Service;

import com.twitter.finagle.SimpleFilter;

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

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

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

import com.twitter.search.common.query.thriftjava.CollectorParams;

import com.twitter.search.common.query.thriftjava.CollectorTerminationParams;

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.ThriftSearchQuery;

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

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

import com.twitter.snowflake.id.SnowflakeId;

import com.twitter.util.Function;

import com.twitter.util.Future;

public class RequestResultStatsFilter

extends SimpleFilter<EarlybirdRequest, EarlybirdResponse> {

private final Clock clock;

private final RequestResultStats stats;

static class RequestResultStats {

private static final String PREFIX = "request\_result\_properties\_";

private final SearchCounter resultsRequestedCount;

private final SearchCounter resultsReturnedCount;

private final SearchCounter maxHitsToProcessCount;

private final SearchCounter hitsProcessedCount;

private final SearchCounter docsProcessedCount;

private final SearchCounter timeoutMsCount;

private Map<String, Percentile<Integer>> requestedNumResultsPercentileByClientId;

private Map<String, Percentile<Integer>> returnedNumResultsPercentileByClientId;

private Map<String, Percentile<Long>> oldestResultPercentileByClientId;

RequestResultStats() {

// Request properties

resultsRequestedCount = SearchCounter.export(PREFIX + "results\_requested\_cnt");

maxHitsToProcessCount = SearchCounter.export(PREFIX + "max\_hits\_to\_process\_cnt");

timeoutMsCount = SearchCounter.export(PREFIX + "timeout\_ms\_cnt");

requestedNumResultsPercentileByClientId = new ConcurrentHashMap<>();

// Result properties

resultsReturnedCount = SearchCounter.export(PREFIX + "results\_returned\_cnt");

hitsProcessedCount = SearchCounter.export(PREFIX + "hits\_processed\_cnt");

docsProcessedCount = SearchCounter.export(PREFIX + "docs\_processed\_cnt");

returnedNumResultsPercentileByClientId = new ConcurrentHashMap<>();

oldestResultPercentileByClientId = new ConcurrentHashMap<>();

}

SearchCounter getResultsRequestedCount() {

return resultsRequestedCount;

}

SearchCounter getResultsReturnedCount() {

return resultsReturnedCount;

}

SearchCounter getMaxHitsToProcessCount() {

return maxHitsToProcessCount;

}

SearchCounter getHitsProcessedCount() {

return hitsProcessedCount;

}

SearchCounter getDocsProcessedCount() {

return docsProcessedCount;

}

SearchCounter getTimeoutMsCount() {

return timeoutMsCount;

}

Percentile<Long> getOldestResultPercentile(String clientId) {

return oldestResultPercentileByClientId.computeIfAbsent(clientId,

key -> PercentileUtil.createPercentile(statName(clientId, "oldest\_result\_age\_seconds")));

}

Percentile<Integer> getRequestedNumResultsPercentile(String clientId) {

return requestedNumResultsPercentileByClientId.computeIfAbsent(clientId,

key -> PercentileUtil.createPercentile(statName(clientId, "requested\_num\_results")));

}

Percentile<Integer> getReturnedNumResultsPercentile(String clientId) {

return returnedNumResultsPercentileByClientId.computeIfAbsent(clientId,

key -> PercentileUtil.createPercentile(statName(clientId, "returned\_num\_results")));

}

private String statName(String clientId, String suffix) {

return String.format("%s%s\_%s", PREFIX, ClientIdUtil.formatClientId(clientId), suffix);

}

}

@Inject

RequestResultStatsFilter(Clock clock, RequestResultStats stats) {

this.clock = clock;

this.stats = stats;

}

private void updateRequestStats(EarlybirdRequest request) {

ThriftSearchQuery searchQuery = request.getSearchQuery();

CollectorParams collectorParams = searchQuery.getCollectorParams();

if (collectorParams != null) {

stats.getResultsRequestedCount().add(collectorParams.numResultsToReturn);

if (request.isSetClientId()) {

stats.getRequestedNumResultsPercentile(request.getClientId())

.record(collectorParams.numResultsToReturn);

}

CollectorTerminationParams terminationParams = collectorParams.getTerminationParams();

if (terminationParams != null) {

if (terminationParams.isSetMaxHitsToProcess()) {

stats.getMaxHitsToProcessCount().add(terminationParams.maxHitsToProcess);

}

if (terminationParams.isSetTimeoutMs()) {

stats.getTimeoutMsCount().add(terminationParams.timeoutMs);

}

}

} else {

if (searchQuery.isSetNumResults()) {

stats.getResultsRequestedCount().add(searchQuery.numResults);

if (request.isSetClientId()) {

stats.getRequestedNumResultsPercentile(request.getClientId())

.record(searchQuery.numResults);

}

}

if (searchQuery.isSetMaxHitsToProcess()) {

stats.getMaxHitsToProcessCount().add(searchQuery.maxHitsToProcess);

}

if (request.isSetTimeoutMs()) {

stats.getTimeoutMsCount().add(request.timeoutMs);

}

}

}

private void updateResultsStats(String clientId, ThriftSearchResults results) {

stats.getResultsReturnedCount().add(results.getResultsSize());

if (results.isSetNumHitsProcessed()) {

stats.getHitsProcessedCount().add(results.numHitsProcessed);

}

if (clientId != null) {

if (results.getResultsSize() > 0) {

List<ThriftSearchResult> resultsList = results.getResults();

long lastId = resultsList.get(resultsList.size() - 1).getId();

long tweetTime = SnowflakeId.timeFromId(lastId).inLongSeconds();

long tweetAge = (clock.nowMillis() / 1000) - tweetTime;

stats.getOldestResultPercentile(clientId).record(tweetAge);

}

stats.getReturnedNumResultsPercentile(clientId).record(results.getResultsSize());

}

}

@Override

public Future<EarlybirdResponse> apply(

EarlybirdRequest request,

Service<EarlybirdRequest, EarlybirdResponse> service) {

updateRequestStats(request);

return service.apply(request).onSuccess(

new Function<EarlybirdResponse, BoxedUnit>() {

@Override

public BoxedUnit apply(EarlybirdResponse response) {

if (response.isSetSearchResults()) {

updateResultsStats(request.getClientId(), response.searchResults);

}

return BoxedUnit.UNIT;

}

});

}

}