package com.twitter.search.earlybird.stats;

import java.util.EnumMap;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import java.util.concurrent.TimeUnit;

import com.google.common.base.Preconditions;

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

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

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

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

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

import com.twitter.search.common.ranking.thriftjava.ThriftRankingParams;

import com.twitter.search.common.ranking.thriftjava.ThriftScoringFunctionType;

import com.twitter.search.earlybird.EarlybirdSearcher;

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

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

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

/\*\*

\* Manages counter and timer stats for EarlybirdSearcher.

\*/

public class EarlybirdSearcherStats {

private static final TimeUnit TIME\_UNIT = TimeUnit.MICROSECONDS;

private final SearchStatsReceiver earlybirdServerStatsReceiver;

public final SearchCounter thriftQueryWithSerializedQuery;

public final SearchCounter thriftQueryWithLuceneQuery;

public final SearchCounter thriftQueryWithoutTextQuery;

public final SearchCounter addedFilterBadUserRep;

public final SearchCounter addedFilterFromUserIds;

public final SearchCounter addedFilterTweetIds;

public final SearchCounter unsetFiltersForSocialFilterTypeQuery;

public final SearchCounter querySpecificSignalMapTotalSize;

public final SearchCounter querySpecificSignalQueriesUsed;

public final SearchCounter querySpecificSignalQueriesErased;

public final SearchCounter authorSpecificSignalMapTotalSize;

public final SearchCounter authorSpecificSignalQueriesUsed;

public final SearchCounter authorSpecificSignalQueriesErased;

public final SearchCounter nullcastTweetsForceExcluded;

public final SearchCounter nullcastUnexpectedResults;

public final SearchCounter nullcastUnexpectedQueries;

public final SearchCounter relevanceAntiGamingFilterUsed;

public final SearchCounter relevanceAntiGamingFilterNotRequested;

public final SearchCounter relevanceAntiGamingFilterSpecifiedTweetsAndFromUserIds;

public final SearchCounter relevanceAntiGamingFilterSpecifiedTweets;

public final SearchCounter relevanceAntiGamingFilterSpecifiedFromUserIds;

public final SearchCounter numCollectorAdjustedMinSearchedStatusID;

public final Map<EarlybirdSearcher.QueryMode, SearchCounter> numRequestsWithBlankQuery;

private final Map<ThriftScoringFunctionType, SearchTimerStats> latencyByScoringFunctionType;

private final Map<ThriftScoringFunctionType,

Map<String, SearchTimerStats>> latencyByScoringFunctionTypeAndClient;

private final Map<String, SearchTimerStats> latencyByTensorflowModel;

public EarlybirdSearcherStats(SearchStatsReceiver earlybirdServerStatsReceiver) {

this.earlybirdServerStatsReceiver = earlybirdServerStatsReceiver;

this.thriftQueryWithLuceneQuery =

earlybirdServerStatsReceiver.getCounter("thrift\_query\_with\_lucene\_query");

this.thriftQueryWithSerializedQuery =

earlybirdServerStatsReceiver.getCounter("thrift\_query\_with\_serialized\_query");

this.thriftQueryWithoutTextQuery =

earlybirdServerStatsReceiver.getCounter("thrift\_query\_without\_text\_query");

this.addedFilterBadUserRep =

earlybirdServerStatsReceiver.getCounter("added\_filter\_bad\_user\_rep");

this.addedFilterFromUserIds =

earlybirdServerStatsReceiver.getCounter("added\_filter\_from\_user\_ids");

this.addedFilterTweetIds =

earlybirdServerStatsReceiver.getCounter("added\_filter\_tweet\_ids");

this.unsetFiltersForSocialFilterTypeQuery =

earlybirdServerStatsReceiver.getCounter("unset\_filters\_for\_social\_filter\_type\_query");

this.querySpecificSignalMapTotalSize =

earlybirdServerStatsReceiver.getCounter("query\_specific\_signal\_map\_total\_size");

this.querySpecificSignalQueriesUsed =

earlybirdServerStatsReceiver.getCounter("query\_specific\_signal\_queries\_used");

this.querySpecificSignalQueriesErased =

earlybirdServerStatsReceiver.getCounter("query\_specific\_signal\_queries\_erased");

this.authorSpecificSignalMapTotalSize =

earlybirdServerStatsReceiver.getCounter("author\_specific\_signal\_map\_total\_size");

this.authorSpecificSignalQueriesUsed =

earlybirdServerStatsReceiver.getCounter("author\_specific\_signal\_queries\_used");

this.authorSpecificSignalQueriesErased =

earlybirdServerStatsReceiver.getCounter("author\_specific\_signal\_queries\_erased");

this.nullcastTweetsForceExcluded =

earlybirdServerStatsReceiver.getCounter("force\_excluded\_nullcast\_result\_count");

this.nullcastUnexpectedResults =

earlybirdServerStatsReceiver.getCounter("unexpected\_nullcast\_result\_count");

this.nullcastUnexpectedQueries =

earlybirdServerStatsReceiver.getCounter("queries\_with\_unexpected\_nullcast\_results");

this.numCollectorAdjustedMinSearchedStatusID =

earlybirdServerStatsReceiver.getCounter("collector\_adjusted\_min\_searched\_status\_id");

this.relevanceAntiGamingFilterUsed = earlybirdServerStatsReceiver

.getCounter("relevance\_anti\_gaming\_filter\_used");

this.relevanceAntiGamingFilterNotRequested = earlybirdServerStatsReceiver

.getCounter("relevance\_anti\_gaming\_filter\_not\_requested");

this.relevanceAntiGamingFilterSpecifiedTweetsAndFromUserIds = earlybirdServerStatsReceiver

.getCounter("relevance\_anti\_gaming\_filter\_specified\_tweets\_and\_from\_user\_ids");

this.relevanceAntiGamingFilterSpecifiedTweets = earlybirdServerStatsReceiver

.getCounter("relevance\_anti\_gaming\_filter\_specified\_tweets");

this.relevanceAntiGamingFilterSpecifiedFromUserIds = earlybirdServerStatsReceiver

.getCounter("relevance\_anti\_gaming\_filter\_specified\_from\_user\_ids");

this.latencyByScoringFunctionType = new EnumMap<>(ThriftScoringFunctionType.class);

this.latencyByScoringFunctionTypeAndClient = new EnumMap<>(ThriftScoringFunctionType.class);

this.latencyByTensorflowModel = new ConcurrentHashMap<>();

for (ThriftScoringFunctionType type : ThriftScoringFunctionType.values()) {

this.latencyByScoringFunctionType.put(type, getTimerStatsByName(getStatsNameByType(type)));

this.latencyByScoringFunctionTypeAndClient.put(type, new ConcurrentHashMap<>());

}

this.numRequestsWithBlankQuery = new EnumMap<>(EarlybirdSearcher.QueryMode.class);

for (EarlybirdSearcher.QueryMode queryMode : EarlybirdSearcher.QueryMode.values()) {

String counterName =

String.format("num\_requests\_with\_blank\_query\_%s", queryMode.name().toLowerCase());

this.numRequestsWithBlankQuery.put(

queryMode, earlybirdServerStatsReceiver.getCounter(counterName));

}

}

/\*\*

\* Records the latency for a request for the applicable stats.

\* @param timer A stopped timer that timed the request.

\* @param request The request that was timed.

\*/

public void recordRelevanceStats(SearchTimer timer, EarlybirdRequest request) {

Preconditions.checkNotNull(timer);

Preconditions.checkNotNull(request);

Preconditions.checkArgument(!timer.isRunning());

ThriftSearchRelevanceOptions relevanceOptions = request.getSearchQuery().getRelevanceOptions();

// Only record ranking searches with a set type.

if (!relevanceOptions.isSetRankingParams()

|| !relevanceOptions.getRankingParams().isSetType()) {

return;

}

ThriftRankingParams rankingParams = relevanceOptions.getRankingParams();

ThriftScoringFunctionType scoringFunctionType = rankingParams.getType();

latencyByScoringFunctionType.get(scoringFunctionType).stoppedTimerIncrement(timer);

if (request.getClientId() != null) {

getTimerStatsByClient(scoringFunctionType, request.getClientId())

.stoppedTimerIncrement(timer);

}

if (scoringFunctionType != ThriftScoringFunctionType.TENSORFLOW\_BASED) {

return;

}

String modelName = rankingParams.getSelectedTensorflowModel();

if (modelName != null) {

getTimerStatsByTensorflowModel(modelName).stoppedTimerIncrement(timer);

}

}

/\*\*

\* Creates a search timer with options specified by TweetsEarlybirdSearcherStats.

\* @return A new SearchTimer.

\*/

public SearchTimer createTimer() {

return new SearchTimer(new SearchMetricTimerOptions.Builder()

.withTimeUnit(TIME\_UNIT)

.build());

}

private SearchTimerStats getTimerStatsByClient(

ThriftScoringFunctionType type,

String clientId) {

Map<String, SearchTimerStats> latencyByClient = latencyByScoringFunctionTypeAndClient.get(type);

return latencyByClient.computeIfAbsent(clientId,

cid -> getTimerStatsByName(getStatsNameByClientAndType(type, cid)));

}

private SearchTimerStats getTimerStatsByTensorflowModel(String modelName) {

return latencyByTensorflowModel.computeIfAbsent(modelName,

mn -> getTimerStatsByName(getStatsNameByTensorflowModel(mn)));

}

private SearchTimerStats getTimerStatsByName(String name) {

return earlybirdServerStatsReceiver.getTimerStats(

name, TIME\_UNIT, false, true, false);

}

public static String getStatsNameByType(ThriftScoringFunctionType type) {

return String.format(

"search\_relevance\_scoring\_function\_%s\_requests", type.name().toLowerCase());

}

public static String getStatsNameByClientAndType(

ThriftScoringFunctionType type,

String clientId) {

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

}

public static String getStatsNameByTensorflowModel(String modelName) {

return String.format(

"model\_%s\_%s", modelName, getStatsNameByType(ThriftScoringFunctionType.TENSORFLOW\_BASED));

}

}