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

import java.util.EnumSet;

import java.util.Set;

import java.util.concurrent.TimeUnit;

import scala.runtime.BoxedUnit;

import com.google.common.collect.ImmutableMap;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.finagle.Service;

import com.twitter.finagle.SimpleFilter;

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

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

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

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

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

import com.twitter.search.queryparser.query.Query;

import com.twitter.search.queryparser.query.QueryParserException;

import com.twitter.search.queryparser.query.annotation.Annotation;

import com.twitter.search.queryparser.query.search.SearchOperator;

import com.twitter.search.queryparser.query.search.SearchOperatorConstants;

import com.twitter.search.queryparser.visitors.DetectAnnotationVisitor;

import com.twitter.search.queryparser.visitors.DetectVisitor;

import com.twitter.util.Future;

/\*\*

\* For a given query, increments counters if that query has a number of search operators or

\* annotations applied to it. Used to detect unusual traffic patterns.

\*/

public class QueryOperatorStatFilter

extends SimpleFilter<EarlybirdRequestContext, EarlybirdResponse> {

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

private final SearchCounter numQueryOperatorDetectionErrors =

SearchCounter.export("query\_operator\_detection\_errors");

private final SearchCounter numQueryOperatorConsideredRequests =

SearchCounter.export("query\_operator\_requests\_considered");

private final ImmutableMap<String, SearchTimerStats> filterOperatorStats;

// Keeps track of the number of queries with a filter applied, whose type we don't care about.

private final SearchCounter numUnknownFilterOperatorRequests =

SearchCounter.export("query\_operator\_filter\_unknown\_requests");

private final ImmutableMap<String, SearchTimerStats> includeOperatorStats;

// Keeps track of the number of queries with an include operator applied, whose type we don't

// know about.

private final SearchCounter numUnknownIncludeOperatorRequests =

SearchCounter.export("query\_operator\_include\_unknown\_requests");

private final ImmutableMap<SearchOperator.Type, SearchTimerStats> operatorTypeStats;

private final SearchCounter numVariantRequests =

SearchCounter.export("query\_operator\_variant\_requests");

/\*\*

\* Construct this QueryOperatorStatFilter by getting the complete set of possible filters a query

\* might have and associating each with a counter.

\*/

public QueryOperatorStatFilter() {

ImmutableMap.Builder<String, SearchTimerStats> filterBuilder = new ImmutableMap.Builder<>();

for (String operand : SearchOperatorConstants.VALID\_FILTER\_OPERANDS) {

filterBuilder.put(

operand,

SearchTimerStats.export(

"query\_operator\_filter\_" + operand + "\_requests",

TimeUnit.MILLISECONDS,

false,

true));

}

filterOperatorStats = filterBuilder.build();

ImmutableMap.Builder<String, SearchTimerStats> includeBuilder = new ImmutableMap.Builder<>();

for (String operand : SearchOperatorConstants.VALID\_INCLUDE\_OPERANDS) {

includeBuilder.put(

operand,

SearchTimerStats.export(

"query\_operator\_include\_" + operand + "\_requests",

TimeUnit.MILLISECONDS,

false,

true));

}

includeOperatorStats = includeBuilder.build();

ImmutableMap.Builder<SearchOperator.Type, SearchTimerStats> operatorBuilder =

new ImmutableMap.Builder<>();

for (SearchOperator.Type operatorType : SearchOperator.Type.values()) {

operatorBuilder.put(

operatorType,

SearchTimerStats.export(

"query\_operator\_" + operatorType.name().toLowerCase() + "\_requests",

TimeUnit.MILLISECONDS,

false,

true

));

}

operatorTypeStats = operatorBuilder.build();

}

@Override

public Future<EarlybirdResponse> apply(

EarlybirdRequestContext requestContext,

Service<EarlybirdRequestContext, EarlybirdResponse> service) {

numQueryOperatorConsideredRequests.increment();

Query parsedQuery = requestContext.getParsedQuery();

if (parsedQuery == null) {

return service.apply(requestContext);

}

SearchTimer timer = new SearchTimer();

timer.start();

return service.apply(requestContext).ensure(() -> {

timer.stop();

try {

updateTimersForOperatorsAndOperands(parsedQuery, timer);

updateCountersIfVariantAnnotation(parsedQuery);

} catch (QueryParserException e) {

LOG.warn("Unable to test if query has operators defined", e);

numQueryOperatorDetectionErrors.increment();

}

return BoxedUnit.UNIT;

});

}

/\*\*

\* Tracks request stats for operators and operands.

\*

\* @param parsedQuery the query to check.

\*/

private void updateTimersForOperatorsAndOperands(Query parsedQuery, SearchTimer timer)

throws QueryParserException {

final DetectVisitor detectVisitor = new DetectVisitor(false, SearchOperator.Type.values());

parsedQuery.accept(detectVisitor);

Set<SearchOperator.Type> detectedOperatorTypes = EnumSet.noneOf(SearchOperator.Type.class);

for (Query query : detectVisitor.getDetectedQueries()) {

// This detectVisitor only matches on SearchOperators.

SearchOperator operator = (SearchOperator) query;

SearchOperator.Type operatorType = operator.getOperatorType();

detectedOperatorTypes.add(operatorType);

if (operatorType == SearchOperator.Type.INCLUDE) {

updateOperandStats(

operator,

includeOperatorStats,

timer,

numUnknownIncludeOperatorRequests);

}

if (operatorType == SearchOperator.Type.FILTER) {

updateOperandStats(

operator,

filterOperatorStats,

timer,

numUnknownFilterOperatorRequests);

}

}

for (SearchOperator.Type type : detectedOperatorTypes) {

operatorTypeStats.get(type).stoppedTimerIncrement(timer);

}

}

private void updateOperandStats(

SearchOperator operator,

ImmutableMap<String, SearchTimerStats> operandRequestStats,

SearchTimer timer,

SearchCounter unknownOperandStat) {

String operand = operator.getOperand();

SearchTimerStats stats = operandRequestStats.get(operand);

if (stats != null) {

stats.stoppedTimerIncrement(timer);

} else {

unknownOperandStat.increment();

}

}

private void updateCountersIfVariantAnnotation(Query parsedQuery) throws QueryParserException {

DetectAnnotationVisitor visitor = new DetectAnnotationVisitor(Annotation.Type.VARIANT);

if (parsedQuery.accept(visitor)) {

numVariantRequests.increment();

}

}

}