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

import javax.annotation.Nullable;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Function;

import com.google.common.base.Joiner;

import com.google.common.collect.Iterables;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.search.common.logging.DebugMessageBuilder;

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

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.ThriftSearchQuery;

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

/\*\*

\* Collects debug messages to attach to EarlybirdResponse

\*/

class EarlybirdResponseDebugMessageBuilder {

private static final Logger LOG =

LoggerFactory.getLogger(EarlybirdResponseDebugMessageBuilder.class);

private static final Logger TOO\_MANY\_FAILED\_PARTITIONS\_LOG =

LoggerFactory.getLogger(String.format("%s\_too\_many\_failed\_partitions",

EarlybirdResponseDebugMessageBuilder.class.getName()));

@VisibleForTesting

protected final SearchCounter insufficientValidResponseCounter =

SearchCounter.export("insufficient\_valid\_partition\_responses\_count");

@VisibleForTesting

protected final SearchCounter validPartitionResponseCounter =

SearchCounter.export("valid\_partition\_response\_count");

// the combined debug string for all earlybird responses

private final StringBuilder debugString;

/\*\*

\* A message builder backed by the same {@link #debugString} above.

\*/

private final DebugMessageBuilder debugMessageBuilder;

private static final Joiner JOINER = Joiner.on(", ");

EarlybirdResponseDebugMessageBuilder(EarlybirdRequest request) {

this(getDebugLevel(request));

}

EarlybirdResponseDebugMessageBuilder(DebugMessageBuilder.Level level) {

this.debugString = new StringBuilder();

this.debugMessageBuilder = new DebugMessageBuilder(debugString, level);

}

private static DebugMessageBuilder.Level getDebugLevel(EarlybirdRequest request) {

if (request.isSetDebugMode() && request.getDebugMode() > 0) {

return DebugMessageBuilder.getDebugLevel(request.getDebugMode());

} else if (request.isSetDebugOptions()) {

return DebugMessageBuilder.Level.DEBUG\_BASIC;

} else {

return DebugMessageBuilder.Level.DEBUG\_NONE;

}

}

protected boolean isDebugMode() {

return debugMessageBuilder.getDebugLevel() > 0;

}

void append(String msg) {

debugString.append(msg);

}

void debugAndLogWarning(String msg) {

if (isDebugMode()) {

debugString.append(msg).append('\n');

}

LOG.warn(msg);

}

void debugDetailed(String format, Object... args) {

debugAtLevel(DebugMessageBuilder.Level.DEBUG\_DETAILED, format, args);

}

void debugVerbose(String format, Object... args) {

debugAtLevel(DebugMessageBuilder.Level.DEBUG\_VERBOSE, format, args);

}

void debugVerbose2(String format, Object... args) {

debugAtLevel(DebugMessageBuilder.Level.DEBUG\_VERBOSE\_2, format, args);

}

void debugAtLevel(DebugMessageBuilder.Level level, String format, Object... args) {

boolean levelOK = debugMessageBuilder.isAtLeastLevel(level);

if (levelOK || LOG.isDebugEnabled()) {

// We check both modes here in order to build the formatted message only once.

String message = String.format(format, args);

LOG.debug(message);

if (levelOK) {

debugString.append(message).append('\n');

}

}

}

String debugString() {

return debugString.toString();

}

DebugMessageBuilder getDebugMessageBuilder() {

return debugMessageBuilder;

}

void logBelowSuccessThreshold(ThriftSearchQuery searchQuery, int numSuccessResponses,

int numPartitions, double successThreshold) {

String rawQuery = (searchQuery != null && searchQuery.isSetRawQuery())

? "[" + searchQuery.getRawQuery() + "]" : "null";

String serializedQuery = (searchQuery != null && searchQuery.isSetSerializedQuery())

? "[" + searchQuery.getSerializedQuery() + "]" : "null";

// Not enough successful responses from partitions.

String errorMessage = String.format(

"Only %d valid responses returned out of %d partitions for raw query: %s"

+ " serialized query: %s. Lower than threshold of %s",

numSuccessResponses, numPartitions, rawQuery, serializedQuery, successThreshold);

TOO\_MANY\_FAILED\_PARTITIONS\_LOG.warn(errorMessage);

insufficientValidResponseCounter.increment();

validPartitionResponseCounter.add(numSuccessResponses);

debugString.append(errorMessage);

}

@VisibleForTesting

void logResponseDebugInfo(EarlybirdRequest earlybirdRequest,

String partitionTierName,

EarlybirdResponse response) {

if (response.isSetDebugString() && !response.getDebugString().isEmpty()) {

debugString.append(String.format("Received response from [%s] with debug string [%s]",

partitionTierName, response.getDebugString())).append("\n");

}

if (!response.isSetResponseCode()) {

debugAndLogWarning(String.format(

"Received Earlybird null response code for query [%s] from [%s]",

earlybirdRequest, partitionTierName));

} else if (response.getResponseCode() != EarlybirdResponseCode.SUCCESS

&& response.getResponseCode() != EarlybirdResponseCode.PARTITION\_SKIPPED

&& response.getResponseCode() != EarlybirdResponseCode.PARTITION\_DISABLED

&& response.getResponseCode() != EarlybirdResponseCode.TIER\_SKIPPED) {

debugAndLogWarning(String.format(

"Received Earlybird response error [%s] for query [%s] from [%s]",

response.getResponseCode(), earlybirdRequest, partitionTierName));

}

if (debugMessageBuilder.isVerbose2()) {

debugVerbose2("Earlybird [%s] returned response: %s", partitionTierName, response);

} else if (debugMessageBuilder.isVerbose()) {

if (response.isSetSearchResults() && response.getSearchResults().getResultsSize() > 0) {

String ids = JOINER.join(Iterables.transform(

response.getSearchResults().getResults(),

new Function<ThriftSearchResult, Long>() {

@Nullable

@Override

public Long apply(ThriftSearchResult result) {

return result.getId();

}

}));

debugVerbose("Earlybird [%s] returned TweetIDs: %s", partitionTierName, ids);

}

}

}

}