package com.twitter.search.earlybird.common;

import java.util.concurrent.TimeUnit;

import com.google.common.annotations.VisibleForTesting;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

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

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

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

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

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

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

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

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

public final class EarlybirdRequestUtil {

// This logger is setup to log to a separate set of log files (request\_info) and use an

// async logger so as to not block the searcher thread. See search/earlybird/config/log4j.xml

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

@VisibleForTesting

static final SearchMovingAverage REQUESTED\_NUM\_RESULTS\_STAT =

SearchMovingAverage.export("requested\_num\_results");

@VisibleForTesting

static final SearchMovingAverage REQUESTED\_MAX\_HITS\_TO\_PROCESS\_STAT =

SearchMovingAverage.export("requested\_max\_hits\_to\_process");

@VisibleForTesting

static final SearchMovingAverage REQUESTED\_COLLECTOR\_PARAMS\_MAX\_HITS\_TO\_PROCESS\_STAT =

SearchMovingAverage.export("requested\_collector\_params\_max\_hits\_to\_process");

@VisibleForTesting

static final SearchMovingAverage REQUESTED\_RELEVANCE\_OPTIONS\_MAX\_HITS\_TO\_PROCESS\_STAT =

SearchMovingAverage.export("requested\_relevance\_options\_max\_hits\_to\_process");

@VisibleForTesting

static final SearchCounter REQUESTED\_MAX\_HITS\_TO\_PROCESS\_ARE\_DIFFERENT\_STAT =

SearchCounter.export("requested\_max\_hits\_to\_process\_are\_different");

private static final SearchRateCounter REQUEST\_WITH\_MORE\_THAN\_2K\_NUM\_RESULTS\_STAT =

SearchRateCounter.export("request\_with\_more\_than\_2k\_num\_result");

private static final SearchRateCounter REQUEST\_WITH\_MORE\_THAN\_4K\_NUM\_RESULTS\_STAT =

SearchRateCounter.export("request\_with\_more\_than\_4k\_num\_result");

// Stats for tracking clock skew between earlybird and the client-specified request timestamp.

@VisibleForTesting

public static final SearchTimerStats CLIENT\_CLOCK\_DIFF\_ABS =

SearchTimerStats.export("client\_clock\_diff\_abs", TimeUnit.MILLISECONDS, false, true);

@VisibleForTesting

public static final SearchTimerStats CLIENT\_CLOCK\_DIFF\_POS =

SearchTimerStats.export("client\_clock\_diff\_pos", TimeUnit.MILLISECONDS, false, true);

@VisibleForTesting

public static final SearchTimerStats CLIENT\_CLOCK\_DIFF\_NEG =

SearchTimerStats.export("client\_clock\_diff\_neg", TimeUnit.MILLISECONDS, false, true);

@VisibleForTesting

public static final SearchRateCounter CLIENT\_CLOCK\_DIFF\_MISSING =

SearchRateCounter.export("client\_clock\_diff\_missing");

private static final int MAX\_NUM\_RESULTS = 4000;

private static final int OLD\_MAX\_NUM\_RESULTS = 2000;

private EarlybirdRequestUtil() {

}

/\*\*

\* Logs and fixes some potentially excessive values in the given request.

\*/

public static void logAndFixExcessiveValues(EarlybirdRequest request) {

ThriftSearchQuery searchQuery = request.getSearchQuery();

if (searchQuery != null) {

int maxHitsToProcess = 0;

int numResultsToReturn = 0;

if (searchQuery.isSetCollectorParams()) {

numResultsToReturn = searchQuery.getCollectorParams().getNumResultsToReturn();

if (searchQuery.getCollectorParams().isSetTerminationParams()) {

maxHitsToProcess =

searchQuery.getCollectorParams().getTerminationParams().getMaxHitsToProcess();

}

}

if (maxHitsToProcess > 50000) {

LOG.warn("Excessive max hits in " + request.toString());

}

// We used to limit number of results to 2000. These two counters help us track if we receive

// too many requests with large number of results set.

String warningMessageTemplate = "Exceed %d num result in %s";

if (numResultsToReturn > MAX\_NUM\_RESULTS) {

LOG.warn(String.format(warningMessageTemplate, MAX\_NUM\_RESULTS, request.toString()));

REQUEST\_WITH\_MORE\_THAN\_4K\_NUM\_RESULTS\_STAT.increment();

searchQuery.getCollectorParams().setNumResultsToReturn(MAX\_NUM\_RESULTS);

} else if (numResultsToReturn > OLD\_MAX\_NUM\_RESULTS) {

LOG.warn(String.format(warningMessageTemplate, OLD\_MAX\_NUM\_RESULTS, request.toString()));

REQUEST\_WITH\_MORE\_THAN\_2K\_NUM\_RESULTS\_STAT.increment();

}

ThriftSearchRelevanceOptions options = searchQuery.getRelevanceOptions();

if (options != null) {

if (options.getMaxHitsToProcess() > 50000) {

LOG.warn("Excessive max hits in " + request.toString());

}

}

}

}

/\*\*

\* Sets {@code request.searchQuery.collectorParams} if they are not already set.

\*/

public static void checkAndSetCollectorParams(EarlybirdRequest request) {

ThriftSearchQuery searchQuery = request.getSearchQuery();

if (searchQuery == null) {

return;

}

if (!searchQuery.isSetCollectorParams()) {

searchQuery.setCollectorParams(new CollectorParams());

}

if (!searchQuery.getCollectorParams().isSetNumResultsToReturn()) {

searchQuery.getCollectorParams().setNumResultsToReturn(searchQuery.getNumResults());

}

if (!searchQuery.getCollectorParams().isSetTerminationParams()) {

CollectorTerminationParams terminationParams = new CollectorTerminationParams();

if (request.isSetTimeoutMs()) {

terminationParams.setTimeoutMs(request.getTimeoutMs());

}

if (request.isSetMaxQueryCost()) {

terminationParams.setMaxQueryCost(request.getMaxQueryCost());

}

searchQuery.getCollectorParams().setTerminationParams(terminationParams);

}

setMaxHitsToProcess(searchQuery);

}

// Early birds will only look for maxHitsToProcess in CollectorParameters.TerminationParameters.

// Priority to set CollectorParameters.TerminationParameters.maxHitsToProcess is

// 1 Collector parameters

// 2 RelevanceParameters

// 3 ThrfitQuery.maxHitsToProcess

private static void setMaxHitsToProcess(ThriftSearchQuery thriftSearchQuery) {

CollectorTerminationParams terminationParams = thriftSearchQuery

.getCollectorParams().getTerminationParams();

if (!terminationParams.isSetMaxHitsToProcess()) {

if (thriftSearchQuery.isSetRelevanceOptions()

&& thriftSearchQuery.getRelevanceOptions().isSetMaxHitsToProcess()) {

terminationParams.setMaxHitsToProcess(

thriftSearchQuery.getRelevanceOptions().getMaxHitsToProcess());

} else {

terminationParams.setMaxHitsToProcess(thriftSearchQuery.getMaxHitsToProcess());

}

}

}

/\*\*

\* Creates a copy of the given request and unsets the binary fields to make the logged line for

\* this request look nicer.

\*/

public static EarlybirdRequest copyAndClearUnnecessaryValuesForLogging(EarlybirdRequest request) {

EarlybirdRequest copiedRequest = request.deepCopy();

if (copiedRequest.isSetSearchQuery()) {

// These fields are very large and the binary data doesn't play well with formz

copiedRequest.getSearchQuery().unsetTrustedFilter();

copiedRequest.getSearchQuery().unsetDirectFollowFilter();

}

return copiedRequest;

}

/\*\*

\* Updates some hit-related stats based on the parameters in the given request.

\*/

public static void updateHitsCounters(EarlybirdRequest request) {

if ((request == null) || !request.isSetSearchQuery()) {

return;

}

ThriftSearchQuery searchQuery = request.getSearchQuery();

if (searchQuery.isSetNumResults()) {

REQUESTED\_NUM\_RESULTS\_STAT.addSample(searchQuery.getNumResults());

}

if (searchQuery.isSetMaxHitsToProcess()) {

REQUESTED\_MAX\_HITS\_TO\_PROCESS\_STAT.addSample(searchQuery.getMaxHitsToProcess());

}

Integer collectorParamsMaxHitsToProcess = null;

if (searchQuery.isSetCollectorParams()

&& searchQuery.getCollectorParams().isSetTerminationParams()

&& searchQuery.getCollectorParams().getTerminationParams().isSetMaxHitsToProcess()) {

collectorParamsMaxHitsToProcess =

searchQuery.getCollectorParams().getTerminationParams().getMaxHitsToProcess();

REQUESTED\_COLLECTOR\_PARAMS\_MAX\_HITS\_TO\_PROCESS\_STAT

.addSample(collectorParamsMaxHitsToProcess);

}

Integer relevanceOptionsMaxHitsToProcess = null;

if (searchQuery.isSetRelevanceOptions()

&& searchQuery.getRelevanceOptions().isSetMaxHitsToProcess()) {

relevanceOptionsMaxHitsToProcess = searchQuery.getRelevanceOptions().getMaxHitsToProcess();

REQUESTED\_RELEVANCE\_OPTIONS\_MAX\_HITS\_TO\_PROCESS\_STAT

.addSample(relevanceOptionsMaxHitsToProcess);

}

if ((collectorParamsMaxHitsToProcess != null)

&& (relevanceOptionsMaxHitsToProcess != null)

&& (collectorParamsMaxHitsToProcess != relevanceOptionsMaxHitsToProcess)) {

REQUESTED\_MAX\_HITS\_TO\_PROCESS\_ARE\_DIFFERENT\_STAT.increment();

}

}

public static boolean isCachingAllowed(EarlybirdRequest request) {

return !request.isSetCachingParams() || request.getCachingParams().isCache();

}

/\*\*

\* Track the clock difference between this server and its client's specified request time.

\* When there is no clock drift between machines, this will record the inflight time between this

\* server and the client.

\*

\* @param request the incoming earlybird request.

\*/

public static void recordClientClockDiff(EarlybirdRequest request) {

if (request.isSetClientRequestTimeMs()) {

final long timeDiff = System.currentTimeMillis() - request.getClientRequestTimeMs();

final long timeDiffAbs = Math.abs(timeDiff);

if (timeDiff >= 0) {

CLIENT\_CLOCK\_DIFF\_POS.timerIncrement(timeDiffAbs);

} else {

CLIENT\_CLOCK\_DIFF\_NEG.timerIncrement(timeDiffAbs);

}

CLIENT\_CLOCK\_DIFF\_ABS.timerIncrement(timeDiffAbs);

} else {

CLIENT\_CLOCK\_DIFF\_MISSING.increment();

}

}

}