package com.twitter.cr\_mixer.logging

import com.twitter.cr\_mixer.model.RelatedTweetCandidateGeneratorQuery

import com.twitter.cr\_mixer.model.InitialCandidate

import com.twitter.cr\_mixer.model.ModuleNames

import com.twitter.cr\_mixer.logging.ScribeLoggerUtils.\_

import com.twitter.cr\_mixer.param.decider.CrMixerDecider

import com.twitter.cr\_mixer.param.decider.DeciderConstants

import com.twitter.cr\_mixer.thriftscala.FetchCandidatesResult

import com.twitter.cr\_mixer.thriftscala.GetRelatedTweetsScribe

import com.twitter.cr\_mixer.thriftscala.PerformanceMetrics

import com.twitter.cr\_mixer.thriftscala.PreRankFilterResult

import com.twitter.cr\_mixer.thriftscala.RelatedTweetRequest

import com.twitter.cr\_mixer.thriftscala.RelatedTweetResponse

import com.twitter.cr\_mixer.thriftscala.RelatedTweetResult

import com.twitter.cr\_mixer.thriftscala.RelatedTweetTopLevelApiResult

import com.twitter.cr\_mixer.thriftscala.TweetCandidateWithMetadata

import com.twitter.cr\_mixer.util.CandidateGenerationKeyUtil

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.finagle.tracing.Trace

import com.twitter.logging.Logger

import com.twitter.simclusters\_v2.common.UserId

import com.twitter.util.Future

import com.twitter.util.Stopwatch

import javax.inject.Inject

import javax.inject.Named

import javax.inject.Singleton

@Singleton

case class RelatedTweetScribeLogger @Inject() (

decider: CrMixerDecider,

statsReceiver: StatsReceiver,

@Named(ModuleNames.RelatedTweetsLogger) relatedTweetsScribeLogger: Logger) {

private val scopedStats = statsReceiver.scope("RelatedTweetsScribeLogger")

private val topLevelApiStats = scopedStats.scope("TopLevelApi")

private val topLevelApiNoUserIdStats = scopedStats.scope("TopLevelApiNoUserId")

private val upperFunnelsStats = scopedStats.scope("UpperFunnels")

private val upperFunnelsNoUserIdStats = scopedStats.scope("UpperFunnelsNoUserId")

def scribeInitialCandidates(

query: RelatedTweetCandidateGeneratorQuery,

getResultFn: => Future[Seq[Seq[InitialCandidate]]]

): Future[Seq[Seq[InitialCandidate]]] = {

scribeResultsAndPerformanceMetrics(

RelatedTweetScribeMetadata.from(query),

getResultFn,

convertToResultFn = convertFetchCandidatesResult

)

}

def scribePreRankFilterCandidates(

query: RelatedTweetCandidateGeneratorQuery,

getResultFn: => Future[Seq[Seq[InitialCandidate]]]

): Future[Seq[Seq[InitialCandidate]]] = {

scribeResultsAndPerformanceMetrics(

RelatedTweetScribeMetadata.from(query),

getResultFn,

convertToResultFn = convertPreRankFilterResult

)

}

/\*\*

\* Scribe Top Level API Request / Response and performance metrics

\* for the getRelatedTweets endpoint.

\*/

def scribeGetRelatedTweets(

request: RelatedTweetRequest,

startTime: Long,

relatedTweetScribeMetadata: RelatedTweetScribeMetadata,

getResultFn: => Future[RelatedTweetResponse]

): Future[RelatedTweetResponse] = {

val timer = Stopwatch.start()

getResultFn.onSuccess { response =>

relatedTweetScribeMetadata.clientContext.userId match {

case Some(userId) =>

if (decider.isAvailableForId(userId, DeciderConstants.upperFunnelPerStepScribeRate)) {

topLevelApiStats.counter(relatedTweetScribeMetadata.product.originalName).incr()

val latencyMs = timer().inMilliseconds

val result = convertTopLevelAPIResult(request, response, startTime)

val traceId = Trace.id.traceId.toLong

val scribeMsg =

buildScribeMessage(result, relatedTweetScribeMetadata, latencyMs, traceId)

scribeResult(scribeMsg)

}

case \_ =>

topLevelApiNoUserIdStats.counter(relatedTweetScribeMetadata.product.originalName).incr()

}

}

}

/\*\*

\* Scribe Per-step intermediate results and performance metrics

\* for each step: fetch candidates, filters.

\*/

private def scribeResultsAndPerformanceMetrics[T](

relatedTweetScribeMetadata: RelatedTweetScribeMetadata,

getResultFn: => Future[T],

convertToResultFn: (T, UserId) => RelatedTweetResult

): Future[T] = {

val timer = Stopwatch.start()

getResultFn.onSuccess { input =>

relatedTweetScribeMetadata.clientContext.userId match {

case Some(userId) =>

if (decider.isAvailableForId(userId, DeciderConstants.upperFunnelPerStepScribeRate)) {

upperFunnelsStats.counter(relatedTweetScribeMetadata.product.originalName).incr()

val latencyMs = timer().inMilliseconds

val result = convertToResultFn(input, userId)

val traceId = Trace.id.traceId.toLong

val scribeMsg =

buildScribeMessage(result, relatedTweetScribeMetadata, latencyMs, traceId)

scribeResult(scribeMsg)

}

case \_ =>

upperFunnelsNoUserIdStats.counter(relatedTweetScribeMetadata.product.originalName).incr()

}

}

}

private def convertTopLevelAPIResult(

request: RelatedTweetRequest,

response: RelatedTweetResponse,

startTime: Long

): RelatedTweetResult = {

RelatedTweetResult.RelatedTweetTopLevelApiResult(

RelatedTweetTopLevelApiResult(

timestamp = startTime,

request = request,

response = response

))

}

private def convertFetchCandidatesResult(

candidatesSeq: Seq[Seq[InitialCandidate]],

requestUserId: UserId

): RelatedTweetResult = {

val tweetCandidatesWithMetadata = candidatesSeq.flatMap { candidates =>

candidates.map { candidate =>

TweetCandidateWithMetadata(

tweetId = candidate.tweetId,

candidateGenerationKey = None

) // do not hydrate candidateGenerationKey to save cost

}

}

RelatedTweetResult.FetchCandidatesResult(

FetchCandidatesResult(Some(tweetCandidatesWithMetadata)))

}

private def convertPreRankFilterResult(

candidatesSeq: Seq[Seq[InitialCandidate]],

requestUserId: UserId

): RelatedTweetResult = {

val tweetCandidatesWithMetadata = candidatesSeq.flatMap { candidates =>

candidates.map { candidate =>

val candidateGenerationKey =

CandidateGenerationKeyUtil.toThrift(candidate.candidateGenerationInfo, requestUserId)

TweetCandidateWithMetadata(

tweetId = candidate.tweetId,

candidateGenerationKey = Some(candidateGenerationKey),

authorId = Some(candidate.tweetInfo.authorId),

score = Some(candidate.getSimilarityScore),

numCandidateGenerationKeys = None

)

}

}

RelatedTweetResult.PreRankFilterResult(PreRankFilterResult(Some(tweetCandidatesWithMetadata)))

}

private def buildScribeMessage(

relatedTweetResult: RelatedTweetResult,

relatedTweetScribeMetadata: RelatedTweetScribeMetadata,

latencyMs: Long,

traceId: Long

): GetRelatedTweetsScribe = {

GetRelatedTweetsScribe(

uuid = relatedTweetScribeMetadata.requestUUID,

internalId = relatedTweetScribeMetadata.internalId,

relatedTweetResult = relatedTweetResult,

requesterId = relatedTweetScribeMetadata.clientContext.userId,

guestId = relatedTweetScribeMetadata.clientContext.guestId,

traceId = Some(traceId),

performanceMetrics = Some(PerformanceMetrics(Some(latencyMs))),

impressedBuckets = getImpressedBuckets(scopedStats)

)

}

private def scribeResult(

scribeMsg: GetRelatedTweetsScribe

): Unit = {

publish(logger = relatedTweetsScribeLogger, codec = GetRelatedTweetsScribe, message = scribeMsg)

}

}