package com.twitter.recos.user\_tweet\_entity\_graph

import java.util.Random

import com.twitter.concurrent.AsyncQueue

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.graphjet.bipartite.NodeMetadataLeftIndexedPowerLawMultiSegmentBipartiteGraph

import com.twitter.graphjet.algorithms.{

RecommendationInfo,

RecommendationType => JavaRecommendationType

}

import com.twitter.graphjet.algorithms.socialproof.{

NodeMetadataSocialProofGenerator,

NodeMetadataSocialProofResult,

NodeMetadataSocialProofRequest => SocialProofJavaRequest,

SocialProofResponse => SocialProofJavaResponse

}

import com.twitter.logging.Logger

import com.twitter.recos.model.SalsaQueryRunner.SalsaRunnerConfig

import com.twitter.recos.user\_tweet\_entity\_graph.thriftscala.{

RecommendationType => ThriftRecommendationType,

RecommendationSocialProofRequest => SocialProofThriftRequest

}

import com.twitter.util.{Future, Try}

import it.unimi.dsi.fastutil.bytes.{Byte2ObjectArrayMap, Byte2ObjectMap}

import it.unimi.dsi.fastutil.ints.{IntOpenHashSet, IntSet}

import it.unimi.dsi.fastutil.longs.{Long2DoubleMap, Long2DoubleOpenHashMap}

import scala.collection.JavaConverters.\_

/\*\*

\* EntitySocialProofRunner creates a queue of reader threads, NodeMetadataProofGenerator,

\* and each one reads from the graph and computes social proofs.

\*/

class EntitySocialProofRunner(

graph: NodeMetadataLeftIndexedPowerLawMultiSegmentBipartiteGraph,

salsaRunnerConfig: SalsaRunnerConfig,

statsReceiver: StatsReceiver) {

private val log: Logger = Logger()

private val stats = statsReceiver.scope(this.getClass.getSimpleName)

private val socialProofSizeStat = stats.stat("socialProofSize")

private val socialProofFailureCounter = stats.counter("failure")

private val pollCounter = stats.counter("poll")

private val pollTimeoutCounter = stats.counter("pollTimeout")

private val offerCounter = stats.counter("offer")

private val pollLatencyStat = stats.stat("pollLatency")

private val socialProofRunnerPool = initSocialProofRunnerPool()

private def initSocialProofRunnerPool(): AsyncQueue[NodeMetadataSocialProofGenerator] = {

val socialProofQueue = new AsyncQueue[NodeMetadataSocialProofGenerator]

(0 until salsaRunnerConfig.numSalsaRunners).foreach { \_ =>

socialProofQueue.offer(new NodeMetadataSocialProofGenerator(graph))

}

socialProofQueue

}

/\*\*

\* Helper method to interpret the output of SocialProofJavaResponse

\*

\* @param socialProofResponse is the response from running NodeMetadataSocialProof

\* @return a sequence of SocialProofResult

\*/

private def transformSocialProofResponse(

socialProofResponse: Option[SocialProofJavaResponse]

): Seq[RecommendationInfo] = {

socialProofResponse match {

case Some(response) =>

val scalaResponse = response.getRankedRecommendations.asScala

scalaResponse.foreach { result =>

socialProofSizeStat.add(

result.asInstanceOf[NodeMetadataSocialProofResult].getSocialProofSize)

}

scalaResponse.toSeq

case \_ => Nil

}

}

/\*\*

\* Helper method to run social proof computation and convert the results to Option

\*

\* @param socialProof is socialProof reader on bipartite graph

\* @param request is the socialProof request

\* @return is an option of SocialProofJavaResponse

\*/

private def getSocialProofResponse(

socialProof: NodeMetadataSocialProofGenerator,

request: SocialProofJavaRequest,

random: Random

)(

implicit statsReceiver: StatsReceiver

): Option[SocialProofJavaResponse] = {

val attempt = Try(socialProof.computeRecommendations(request, random)).onFailure { e =>

socialProofFailureCounter.incr()

log.error(e, "SocialProof computation failed")

}

attempt.toOption

}

/\*\*

\* Attempt to retrieve a NodeMetadataSocialProof thread from the runner pool

\* to execute a socialProofRequest

\*/

private def handleSocialProofRequest(socialProofRequest: SocialProofJavaRequest) = {

pollCounter.incr()

val t0 = System.currentTimeMillis()

socialProofRunnerPool.poll().map { entitySocialProof =>

val pollTime = System.currentTimeMillis - t0

pollLatencyStat.add(pollTime)

val socialProofResponse = Try {

if (pollTime < salsaRunnerConfig.timeoutSalsaRunner) {

val response =

getSocialProofResponse(entitySocialProof, socialProofRequest, new Random())(

statsReceiver

)

transformSocialProofResponse(response)

} else {

// if we did not get a social proof in time, then fail fast here and immediately put it back

log.warning("socialProof polling timeout")

pollTimeoutCounter.incr()

throw new RuntimeException("socialProof poll timeout")

Nil

}

} ensure {

socialProofRunnerPool.offer(entitySocialProof)

offerCounter.incr()

}

socialProofResponse.toOption getOrElse Nil

}

}

/\*\*

\* This apply() supports requests coming from the new social proof endpoint in UTEG that works for

\* tweet social proof generation, as well as hashtag and url social proof generation.

\* Currently this endpoint supports url social proof generation for Guide.

\*/

def apply(request: SocialProofThriftRequest): Future[Seq[RecommendationInfo]] = {

val nodeMetadataTypeToIdsMap: Byte2ObjectMap[IntSet] = new Byte2ObjectArrayMap[IntSet]()

request.recommendationIdsForSocialProof.collect {

case (ThriftRecommendationType.Url, urlIds) =>

// We must convert the Long url ids into type Int since the underlying library expects Int type metadata ids.

val urlIntIds = urlIds.map(\_.toInt)

nodeMetadataTypeToIdsMap.put(

JavaRecommendationType.URL.getValue.toByte,

new IntOpenHashSet(urlIntIds.toArray)

)

case (ThriftRecommendationType.Hashtag, hashtagIds) =>

// We must convert the Long hashtag ids into type Int since the underlying library expects Int type metadata ids.

val hashtagIntIds = hashtagIds.map(\_.toInt)

nodeMetadataTypeToIdsMap.put(

JavaRecommendationType.HASHTAG.getValue.toByte,

new IntOpenHashSet(hashtagIntIds.toArray)

)

}

val leftSeedNodes: Long2DoubleMap = new Long2DoubleOpenHashMap(

request.seedsWithWeights.keys.toArray,

request.seedsWithWeights.values.toArray

)

val socialProofRequest = new SocialProofJavaRequest(

nodeMetadataTypeToIdsMap,

leftSeedNodes,

UserTweetEdgeTypeMask.getUserTweetGraphSocialProofTypes(request.socialProofTypes)

)

handleSocialProofRequest(socialProofRequest)

}

}