package com.twitter.recos.user\_user\_graph

import java.util.Random

import com.google.common.collect.Lists

import com.twitter.concurrent.AsyncQueue

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.graphjet.algorithms.counting.TopSecondDegreeByCountResponse

import com.twitter.graphjet.algorithms.counting.user.TopSecondDegreeByCountForUser

import com.twitter.graphjet.algorithms.counting.user.TopSecondDegreeByCountRequestForUser

import com.twitter.graphjet.algorithms.counting.user.UserRecommendationInfo

import com.twitter.graphjet.algorithms.ConnectingUsersWithMetadata

import com.twitter.graphjet.algorithms.filters.\_

import com.twitter.graphjet.bipartite.NodeMetadataLeftIndexedPowerLawMultiSegmentBipartiteGraph

import com.twitter.logging.Logger

import com.twitter.recos.decider.UserUserGraphDecider

import com.twitter.recos.graph\_common.FinagleStatsReceiverWrapper

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

import com.twitter.recos.recos\_common.thriftscala.UserSocialProofType

import com.twitter.recos.user\_user\_graph.thriftscala.\_

import com.twitter.recos.util.Stats.\_

import com.twitter.servo.request.RequestHandler

import com.twitter.util.Future

import com.twitter.util.Try

import it.unimi.dsi.fastutil.longs.Long2DoubleOpenHashMap

import it.unimi.dsi.fastutil.longs.LongOpenHashSet

import scala.collection.JavaConverters.\_

trait RecommendUsersHandler extends RequestHandler[RecommendUserRequest, RecommendUserResponse]

/\*\*

\* Computes user recommendations based on a RecommendUserRequest by using

\* TopSecondDegree algorithm in GraphJet.

\*/

case class RecommendUsersHandlerImpl(

bipartiteGraph: NodeMetadataLeftIndexedPowerLawMultiSegmentBipartiteGraph,

salsaRunnerConfig: SalsaRunnerConfig,

decider: UserUserGraphDecider,

statsReceiverWrapper: FinagleStatsReceiverWrapper)

extends RecommendUsersHandler {

private val log: Logger = Logger(this.getClass.getSimpleName)

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

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

private val recsStat = stats.stat("recs\_count")

private val emptyCounter = stats.counter("empty")

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 graphJetQueue = new AsyncQueue[TopSecondDegreeByCountForUser]

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

graphJetQueue.offer(

new TopSecondDegreeByCountForUser(

bipartiteGraph,

salsaRunnerConfig.expectedNodesToHitInSalsa,

statsReceiverWrapper.scope(this.getClass.getSimpleName)

)

)

}

/\*\*

\* Given a user\_user\_graph request, make it conform to GraphJet's request format

\*/

private def convertRequestToJava(

request: RecommendUserRequest

): TopSecondDegreeByCountRequestForUser = {

val queryNode = request.requesterId

val leftSeedNodesWithWeight = new Long2DoubleOpenHashMap(

request.seedsWithWeights.keys.toArray,

request.seedsWithWeights.values.toArray

)

val toBeFiltered = new LongOpenHashSet(request.excludedUserIds.getOrElse(Nil).toArray)

val maxNumResults = request.maxNumResults.getOrElse(DefaultRequestParams.MaxNumResults)

val maxNumSocialProofs =

request.maxNumSocialProofs.getOrElse(DefaultRequestParams.MaxNumSocialProofs)

val minUserPerSocialProof = convertMinUserPerSocialProofToJava(request.minUserPerSocialProof)

val socialProofTypes =

UserEdgeTypeMask.getUserUserGraphSocialProofTypes(request.socialProofTypes)

val maxRightNodeAgeInMillis = DefaultRequestParams.MaxRightNodeAgeThreshold

val maxEdgeEngagementAgeInMillis =

request.maxEdgeEngagementAgeInMillis.getOrElse(DefaultRequestParams.MaxEdgeAgeThreshold)

val resultFilterChain = new ResultFilterChain(

Lists.newArrayList(

new SocialProofTypesFilter(statsReceiverWrapper),

new RequestedSetFilter(statsReceiverWrapper)

)

)

new TopSecondDegreeByCountRequestForUser(

queryNode,

leftSeedNodesWithWeight,

toBeFiltered,

maxNumResults,

maxNumSocialProofs,

UserEdgeTypeMask.SIZE.toInt,

minUserPerSocialProof,

socialProofTypes,

maxRightNodeAgeInMillis,

maxEdgeEngagementAgeInMillis,

resultFilterChain

)

}

/\*\*

\* Converts the thrift scala type to the Java equivalent

\*/

private def convertMinUserPerSocialProofToJava(

socialProofInScala: Option[scala.collection.Map[UserSocialProofType, Int]]

): java.util.Map[java.lang.Byte, java.lang.Integer] = {

socialProofInScala

.map {

\_.map {

case (key: UserSocialProofType, value: Int) =>

(new java.lang.Byte(key.getValue.toByte), new java.lang.Integer(value))

}

}

.getOrElse(Map.empty[java.lang.Byte, java.lang.Integer])

.asJava

}

/\*\*

\* Converts a byte-array format of social proofs in Java to its Scala equivalent

\*/

private def convertSocialProofsToScala(

socialProofs: java.util.Map[java.lang.Byte, ConnectingUsersWithMetadata]

): scala.collection.mutable.Map[UserSocialProofType, scala.Seq[Long]] = {

socialProofs.asScala.map {

case (socialProofByte, socialProof) =>

val proofType = UserSocialProofType(socialProofByte.toByte)

val ids = socialProof.getConnectingUsers.asScala.map(\_.toLong)

(proofType, ids)

}

}

/\*\*

\* Converts Java recommendation results to its Scala equivalent

\*/

private def convertResponseToScala(

responseOpt: Option[TopSecondDegreeByCountResponse]

): RecommendUserResponse = {

responseOpt match {

case Some(rawResponse) =>

val userSeq = rawResponse.getRankedRecommendations.asScala.toSeq.flatMap {

case userRecs: UserRecommendationInfo =>

Some(

RecommendedUser(

userRecs.getRecommendation,

userRecs.getWeight,

convertSocialProofsToScala(userRecs.getSocialProof)

)

)

case \_ =>

None

}

recsStat.add(userSeq.size)

if (userSeq.isEmpty) {

emptyCounter.incr()

}

RecommendUserResponse(userSeq)

case None =>

emptyCounter.incr()

RecommendUserResponse(Nil)

}

}

private def getGraphJetResponse(

graphJet: TopSecondDegreeByCountForUser,

request: TopSecondDegreeByCountRequestForUser,

random: Random

)(

implicit statsReceiver: StatsReceiver

): Option[TopSecondDegreeByCountResponse] = {

trackBlockStats(stats) {

// compute recs -- need to catch and print exceptions here otherwise they are swallowed

val recAttempt = Try(graphJet.computeRecommendations(request, random)).onFailure { e =>

failureCounter.incr()

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

}

recAttempt.toOption

}

}

override def apply(request: RecommendUserRequest): Future[RecommendUserResponse] = {

val random = new Random()

val graphJetRequest = convertRequestToJava(request)

pollCounter.incr()

val t0 = System.currentTimeMillis

graphJetQueue.poll().map { graphJetRunner =>

val pollTime = System.currentTimeMillis - t0

pollLatencyStat.add(pollTime)

val response = Try {

if (pollTime < salsaRunnerConfig.timeoutSalsaRunner) {

convertResponseToScala(

getGraphJetResponse(

graphJetRunner,

graphJetRequest,

random

)(statsReceiverWrapper.statsReceiver)

)

} else {

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

log.warning("GraphJet Queue polling timeout")

pollTimeoutCounter.incr()

throw new RuntimeException("GraphJet poll timeout")

RecommendUserResponse(Nil)

}

} ensure {

graphJetQueue.offer(graphJetRunner)

offerCounter.incr()

}

response.toOption.getOrElse(RecommendUserResponse(Nil))

}

}

object DefaultRequestParams {

val MaxNumResults = 100

val MaxNumSocialProofs = 100

val MaxRightNodeAgeThreshold: Long = Long.MaxValue

val MaxEdgeAgeThreshold: Long = Long.MaxValue

}

}