package com.twitter.follow\_recommendations.common.candidate\_sources.stp

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.candidate\_sources.addressbook.ForwardEmailBookSource

import com.twitter.follow\_recommendations.common.candidate\_sources.addressbook.ForwardPhoneBookSource

import com.twitter.follow\_recommendations.common.candidate\_sources.addressbook.ReverseEmailBookSource

import com.twitter.follow\_recommendations.common.candidate\_sources.addressbook.ReversePhoneBookSource

import com.twitter.follow\_recommendations.common.clients.real\_time\_real\_graph.RealTimeRealGraphClient

import com.twitter.follow\_recommendations.common.models.HasRecentFollowedUserIds

import com.twitter.follow\_recommendations.common.models.PotentialFirstDegreeEdge

import com.twitter.follow\_recommendations.common.stores.LowTweepCredFollowStore

import com.twitter.hermit.model.Algorithm

import com.twitter.hermit.model.Algorithm.Algorithm

import com.twitter.inject.Logging

import com.twitter.product\_mixer.core.model.marshalling.request.HasClientContext

import com.twitter.stitch.Stitch

import com.twitter.timelines.configapi.HasParams

import com.twitter.util.Duration

import com.twitter.util.Timer

import com.twitter.wtf.scalding.jobs.strong\_tie\_prediction.FirstDegreeEdge

import com.twitter.wtf.scalding.jobs.strong\_tie\_prediction.FirstDegreeEdgeInfo

import com.twitter.wtf.scalding.jobs.strong\_tie\_prediction.FirstDegreeEdgeInfoMonoid

import javax.inject.Inject

import javax.inject.Singleton

// Grabs FirstDegreeNodes from Candidate Sources

@Singleton

class STPFirstDegreeFetcher @Inject() (

realTimeGraphClient: RealTimeRealGraphClient,

reversePhoneBookSource: ReversePhoneBookSource,

reverseEmailBookSource: ReverseEmailBookSource,

forwardEmailBookSource: ForwardEmailBookSource,

forwardPhoneBookSource: ForwardPhoneBookSource,

mutualFollowStrongTiePredictionSource: MutualFollowStrongTiePredictionSource,

lowTweepCredFollowStore: LowTweepCredFollowStore,

timer: Timer,

statsReceiver: StatsReceiver)

extends Logging {

private val stats = statsReceiver.scope("STPFirstDegreeFetcher")

private val stitchRequests = stats.scope("stitchRequests")

private val allStitchRequests = stitchRequests.counter("all")

private val timeoutStitchRequests = stitchRequests.counter("timeout")

private val successStitchRequests = stitchRequests.counter("success")

private implicit val firstDegreeEdgeInfoMonoid: FirstDegreeEdgeInfoMonoid =

new FirstDegreeEdgeInfoMonoid

/\*\*

\* Used to map from algorithm to the correct fetcher and firstDegreeEdgeInfo.

\* Afterward, uses fetcher to get candidates and construct the correct FirstDegreeEdgeInfo.

\* \*/

private def getPotentialFirstEdgesFromFetcher(

userId: Long,

target: HasClientContext with HasParams with HasRecentFollowedUserIds,

algorithm: Algorithm,

weight: Double

): Stitch[Seq[PotentialFirstDegreeEdge]] = {

val (candidates, edgeInfo) = algorithm match {

case Algorithm.MutualFollowSTP =>

(

mutualFollowStrongTiePredictionSource(target),

Some(FirstDegreeEdgeInfo(mutualFollow = true)))

case Algorithm.ReverseEmailBookIbis =>

(reverseEmailBookSource(target), Some(FirstDegreeEdgeInfo(reverseEmail = true)))

case Algorithm.ReversePhoneBook =>

(reversePhoneBookSource(target), Some(FirstDegreeEdgeInfo(reversePhone = true)))

case Algorithm.ForwardEmailBook =>

(forwardEmailBookSource(target), Some(FirstDegreeEdgeInfo(forwardEmail = true)))

case Algorithm.ForwardPhoneBook =>

(forwardPhoneBookSource(target), Some(FirstDegreeEdgeInfo(forwardPhone = true)))

case Algorithm.LowTweepcredFollow =>

(

lowTweepCredFollowStore.getLowTweepCredUsers(target),

Some(FirstDegreeEdgeInfo(lowTweepcredFollow = true)))

case \_ =>

(Stitch.Nil, None)

}

candidates.map(\_.flatMap { candidate =>

edgeInfo.map(PotentialFirstDegreeEdge(userId, candidate.id, algorithm, weight, \_))

})

}

/\*\*

\* Using the DefaultMap (AlgorithmToScore) we iterate through algorithm/weights to get

\* candidates with a set weight. Then, given repeating candidates (by candidate id).

\* Given those candidates we group by the candidateId and sum all below weights and combine

\* the edgeInfos of into one. Then we choose the candidates with most weight. Finally,

\* we attach the realGraphWeight score to those candidates.

\* \*/

def getFirstDegreeEdges(

target: HasClientContext with HasParams with HasRecentFollowedUserIds

): Stitch[Seq[FirstDegreeEdge]] = {

target.getOptionalUserId

.map { userId =>

allStitchRequests.incr()

val firstEdgesQueryStitch = Stitch

.collect(STPFirstDegreeFetcher.DefaultGraphBuilderAlgorithmToScore.map {

case (algorithm, candidateWeight) =>

getPotentialFirstEdgesFromFetcher(userId, target, algorithm, candidateWeight)

}.toSeq)

.map(\_.flatten)

val destinationIdsToEdges = firstEdgesQueryStitch

.map(\_.groupBy(\_.connectingId).map {

case (destinationId: Long, edges: Seq[PotentialFirstDegreeEdge]) =>

val combinedDestScore = edges.map(\_.score).sum

val combinedEdgeInfo: FirstDegreeEdgeInfo =

edges.map(\_.edgeInfo).fold(firstDegreeEdgeInfoMonoid.zero) {

(aggregatedInfo, currentInfo) =>

firstDegreeEdgeInfoMonoid.plus(aggregatedInfo, currentInfo)

}

(destinationId, combinedEdgeInfo, combinedDestScore)

}).map(\_.toSeq)

val topDestinationEdges = destinationIdsToEdges.map(\_.sortBy {

case (\_, \_, combinedDestScore) => -combinedDestScore

}.take(STPFirstDegreeFetcher.MaxNumFirstDegreeEdges))

Stitch

.join(realTimeGraphClient.getRealGraphWeights(userId), topDestinationEdges).map {

case (realGraphWeights, topDestinationEdges) =>

successStitchRequests.incr()

topDestinationEdges.map {

case (destinationId, combinedEdgeInfo, \_) =>

val updatedEdgeInfo = combinedEdgeInfo.copy(

realGraphWeight = realGraphWeights.getOrElse(destinationId, 0.0),

lowTweepcredFollow =

!combinedEdgeInfo.mutualFollow && combinedEdgeInfo.lowTweepcredFollow

)

FirstDegreeEdge(userId, destinationId, updatedEdgeInfo)

}

}.within(STPFirstDegreeFetcher.LongTimeoutFetcher)(timer).rescue {

case ex =>

timeoutStitchRequests.incr()

logger.error("Exception while loading direct edges in OnlineSTP: ", ex)

Stitch.Nil

}

}.getOrElse(Stitch.Nil)

}

}

object STPFirstDegreeFetcher {

val MaxNumFirstDegreeEdges = 200

val DefaultGraphBuilderAlgorithmToScore = Map(

Algorithm.MutualFollowSTP -> 10.0,

Algorithm.LowTweepcredFollow -> 6.0,

Algorithm.ForwardEmailBook -> 7.0,

Algorithm.ForwardPhoneBook -> 9.0,

Algorithm.ReverseEmailBookIbis -> 5.0,

Algorithm.ReversePhoneBook -> 8.0

)

val LongTimeoutFetcher: Duration = 300.millis

}