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

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.candidate\_sources.base.TwoHopExpansionCandidateSource

import com.twitter.follow\_recommendations.common.clients.socialgraph.RecentEdgesQuery

import com.twitter.follow\_recommendations.common.clients.socialgraph.SocialGraphClient

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

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

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

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

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

import com.twitter.hermit.model.Algorithm

import com.twitter.inject.Logging

import com.twitter.product\_mixer.core.model.common.identifier.CandidateSourceIdentifier

import com.twitter.socialgraph.thriftscala.RelationshipType

import com.twitter.stitch.Stitch

import com.twitter.timelines.configapi.HasParams

import javax.inject.Inject

import javax.inject.Singleton

/\*\*

\* This candidate source is a two hop expansion over the follow graph. The candidates returned from this source is the users that get followed by the target user's recent followings. It will call SocialGraph `n` + 1 times where `n` is the number of recent followings of the target user to be considered.

\*/

@Singleton

class RecentFollowingRecentFollowingExpansionSource @Inject() (

socialGraphClient: SocialGraphClient,

statsReceiver: StatsReceiver)

extends TwoHopExpansionCandidateSource[

HasParams with HasRecentFollowedUserIds,

Long,

Long,

CandidateUser

]

with Logging {

override val identifier: CandidateSourceIdentifier =

RecentFollowingRecentFollowingExpansionSource.Identifier

val stats = statsReceiver.scope(identifier.name)

override def firstDegreeNodes(

target: HasParams with HasRecentFollowedUserIds

): Stitch[Seq[Long]] = Stitch.value(

target.recentFollowedUserIds

.getOrElse(Nil).take(

RecentFollowingRecentFollowingExpansionSource.NumFirstDegreeNodesToRetrieve)

)

override def secondaryDegreeNodes(

target: HasParams with HasRecentFollowedUserIds,

node: Long

): Stitch[Seq[Long]] = socialGraphClient

.getRecentEdgesCached(

RecentEdgesQuery(

node,

Seq(RelationshipType.Following),

Some(RecentFollowingRecentFollowingExpansionSource.NumSecondDegreeNodesToRetrieve)),

useCachedStratoColumn =

target.params(RecentFollowingRecentFollowingExpansionSourceParams.CallSgsCachedColumn)

).map(

\_.take(RecentFollowingRecentFollowingExpansionSource.NumSecondDegreeNodesToRetrieve)).rescue {

case exception: Exception =>

logger.warn(

s"${this.getClass} fails to retrieve second degree nodes for first degree node $node",

exception)

stats.counter("second\_degree\_expansion\_error").incr()

Stitch.Nil

}

override def aggregateAndScore(

target: HasParams with HasRecentFollowedUserIds,

firstDegreeToSecondDegreeNodesMap: Map[Long, Seq[Long]]

): Stitch[Seq[CandidateUser]] = {

val zipped = firstDegreeToSecondDegreeNodesMap.toSeq.flatMap {

case (firstDegreeId, secondDegreeIds) =>

secondDegreeIds.map(secondDegreeId => firstDegreeId -> secondDegreeId)

}

val candidateAndConnections = zipped

.groupBy { case (\_, secondDegreeId) => secondDegreeId }

.mapValues { v => v.map { case (firstDegreeId, \_) => firstDegreeId } }

.toSeq

.sortBy { case (\_, connections) => -connections.size }

.map {

case (candidateId, connections) =>

CandidateUser(

id = candidateId,

score = Some(CandidateUser.DefaultCandidateScore),

reason = Some(

Reason(

Some(AccountProof(followProof = Some(FollowProof(connections, connections.size))))))

).withCandidateSource(identifier)

}

Stitch.value(candidateAndConnections)

}

}

object RecentFollowingRecentFollowingExpansionSource {

val Identifier = CandidateSourceIdentifier(Algorithm.NewFollowingNewFollowingExpansion.toString)

val NumFirstDegreeNodesToRetrieve = 5

val NumSecondDegreeNodesToRetrieve = 20

}