package com.twitter.timelineranker.common

import com.twitter.finagle.stats.Stat

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.servo.util.FutureArrow

import com.twitter.timelineranker.core.CandidateEnvelope

import com.twitter.timelineranker.model.RecapQuery.DependencyProvider

import com.twitter.timelineranker.parameters.recap.RecapQueryContext

import com.twitter.timelineranker.parameters.in\_network\_tweets.InNetworkTweetParams.RecycledMaxFollowedUsersEnableAntiDilutionParam

import com.twitter.timelineranker.visibility.FollowGraphDataProvider

import com.twitter.timelines.earlybird.common.options.AuthorScoreAdjustments

import com.twitter.util.Future

/\*\*

\* Transform which conditionally augments follow graph data using the real graph,

\* derived from the earlybirdOptions passed in the query

\*

\* @param followGraphDataProvider data provider to be used for fetching updated mutual follow info

\* @param maxFollowedUsersProvider max number of users to return

\* @param enableRealGraphUsersProvider should we augment using real graph data?

\* @param maxRealGraphAndFollowedUsersProvider max combined users to return, overrides maxFollowedUsersProvider above

\* @param statsReceiver scoped stats received

\*/

class FollowAndRealGraphCombiningTransform(

followGraphDataProvider: FollowGraphDataProvider,

maxFollowedUsersProvider: DependencyProvider[Int],

enableRealGraphUsersProvider: DependencyProvider[Boolean],

maxRealGraphAndFollowedUsersProvider: DependencyProvider[Int],

imputeRealGraphAuthorWeightsProvider: DependencyProvider[Boolean],

imputeRealGraphAuthorWeightsPercentileProvider: DependencyProvider[Int],

statsReceiver: StatsReceiver)

extends FutureArrow[CandidateEnvelope, CandidateEnvelope] {

// Number of authors in the seedset after mixing followed users and real graph users

// Only have this stat if user follows >= maxFollowedUsers and enableRealGraphUsers is true and onlyRealGraphUsers is false

val numFollowAndRealGraphUsersStat: Stat = statsReceiver.stat("numFollowAndRealGraphUsers")

val numFollowAndRealGraphUsersFromFollowGraphStat =

statsReceiver.scope("numFollowAndRealGraphUsers").stat("FollowGraphUsers")

val numFollowAndRealGraphUsersFromRealGraphStat =

statsReceiver.scope("numFollowAndRealGraphUsers").stat("RealGraphUsers")

val numFollowAndRealGraphUsersFromRealGraphCounter =

statsReceiver.scope("numFollowAndRealGraphUsers").counter()

// Number of authors in the seedset with only followed users

// Only have this stat if user follows >= maxFollowedUsers and enableRealGraphUsers is false

val numFollowedUsersStat: Stat = statsReceiver.stat("numFollowedUsers")

// Number of authors in the seedset with only followed users

// Only have this stat if user follows < maxFollowedUsers

val numFollowedUsersLessThanMaxStat: Stat = statsReceiver.stat("numFollowedUsersLessThanMax")

val numFollowedUsersLessThanMaxCounter =

statsReceiver.scope("numFollowedUsersLessThanMax").counter()

val numFollowedUsersMoreThanMaxStat: Stat = statsReceiver.stat("numFollowedUsersMoreThanMax")

val numFollowedUsersMoreThanMaxCounter =

statsReceiver.scope("numFollowedUsersMoreThanMax").counter()

val realGraphAuthorWeightsSumProdStat: Stat = statsReceiver.stat("realGraphAuthorWeightsSumProd")

val realGraphAuthorWeightsSumMinExpStat: Stat =

statsReceiver.stat("realGraphAuthorWeightsSumMinExp")

val realGraphAuthorWeightsSumP50ExpStat: Stat =

statsReceiver.stat("realGraphAuthorWeightsSumP50Exp")

val realGraphAuthorWeightsSumP95ExpStat: Stat =

statsReceiver.stat("realGraphAuthorWeightsSumP95Exp")

val numAuthorsWithoutRealgraphScoreStat: Stat =

statsReceiver.stat("numAuthorsWithoutRealgraphScore")

override def apply(envelope: CandidateEnvelope): Future[CandidateEnvelope] = {

val realGraphData = envelope.query.earlybirdOptions

.map(\_.authorScoreAdjustments.authorScoreMap)

.getOrElse(Map.empty)

Future

.join(

envelope.followGraphData.followedUserIdsFuture,

envelope.followGraphData.mutedUserIdsFuture

).map {

case (followedUserIds, mutedUserIds) =>

// Anti-dilution param for DDG-16198

val recycledMaxFollowedUsersEnableAntiDilutionParamProvider =

DependencyProvider.from(RecycledMaxFollowedUsersEnableAntiDilutionParam)

val maxFollowedUsers = {

if (followedUserIds.size > RecapQueryContext.MaxFollowedUsers.default && recycledMaxFollowedUsersEnableAntiDilutionParamProvider(

envelope.query)) {

// trigger experiment

maxFollowedUsersProvider(envelope.query)

} else {

maxFollowedUsersProvider(envelope.query)

}

}

val filteredRealGraphUserIds = realGraphData.keySet

.filterNot(mutedUserIds)

.take(maxFollowedUsers)

.toSeq

val filteredFollowedUserIds = followedUserIds.filterNot(mutedUserIds)

if (followedUserIds.size < maxFollowedUsers) {

numFollowedUsersLessThanMaxStat.add(filteredFollowedUserIds.size)

// stats

numFollowedUsersLessThanMaxCounter.incr()

(filteredFollowedUserIds, false)

} else {

numFollowedUsersMoreThanMaxStat.add(filteredFollowedUserIds.size)

numFollowedUsersMoreThanMaxCounter.incr()

if (enableRealGraphUsersProvider(envelope.query)) {

val maxRealGraphAndFollowedUsersNum =

maxRealGraphAndFollowedUsersProvider(envelope.query)

require(

maxRealGraphAndFollowedUsersNum >= maxFollowedUsers,

"maxRealGraphAndFollowedUsers must be greater than or equal to maxFollowedUsers."

)

val recentFollowedUsersNum = RecapQueryContext.MaxFollowedUsers.bounds

.apply(maxRealGraphAndFollowedUsersNum - filteredRealGraphUserIds.size)

val recentFollowedUsers =

filteredFollowedUserIds

.filterNot(filteredRealGraphUserIds.contains)

.take(recentFollowedUsersNum)

val filteredFollowAndRealGraphUserIds =

recentFollowedUsers ++ filteredRealGraphUserIds

// Track the size of recentFollowedUsers from SGS

numFollowAndRealGraphUsersFromFollowGraphStat.add(recentFollowedUsers.size)

// Track the size of filteredRealGraphUserIds from real graph dataset.

numFollowAndRealGraphUsersFromRealGraphStat.add(filteredRealGraphUserIds.size)

numFollowAndRealGraphUsersFromRealGraphCounter.incr()

numFollowAndRealGraphUsersStat.add(filteredFollowAndRealGraphUserIds.size)

(filteredFollowAndRealGraphUserIds, true)

} else {

numFollowedUsersStat.add(followedUserIds.size)

(filteredFollowedUserIds, false)

}

}

}.map {

case (updatedFollowSeq, shouldUpdateMutualFollows) =>

val updatedMutualFollowing = if (shouldUpdateMutualFollows) {

followGraphDataProvider.getMutuallyFollowingUserIds(

envelope.query.userId,

updatedFollowSeq)

} else {

envelope.followGraphData.mutuallyFollowingUserIdsFuture

}

val followGraphData = envelope.followGraphData.copy(

followedUserIdsFuture = Future.value(updatedFollowSeq),

mutuallyFollowingUserIdsFuture = updatedMutualFollowing

)

val authorIdsWithRealgraphScore = realGraphData.keySet

val authorIdsWithoutRealgraphScores =

updatedFollowSeq.filterNot(authorIdsWithRealgraphScore.contains)

//stat for logging the percentage of users' followings that do not have a realgraph score

if (updatedFollowSeq.nonEmpty)

numAuthorsWithoutRealgraphScoreStat.add(

authorIdsWithoutRealgraphScores.size / updatedFollowSeq.size \* 100)

if (imputeRealGraphAuthorWeightsProvider(envelope.query) && realGraphData.nonEmpty) {

val imputedScorePercentile =

imputeRealGraphAuthorWeightsPercentileProvider(envelope.query) / 100.0

val existingAuthorIdScores = realGraphData.values.toList.sorted

val imputedScoreIndex = Math.min(

existingAuthorIdScores.length - 1,

(existingAuthorIdScores.length \* imputedScorePercentile).toInt)

val imputedScore = existingAuthorIdScores(imputedScoreIndex)

val updatedAuthorScoreMap = realGraphData ++ authorIdsWithoutRealgraphScores

.map(\_ -> imputedScore).toMap

imputedScorePercentile match {

case 0.0 =>

realGraphAuthorWeightsSumMinExpStat.add(updatedAuthorScoreMap.values.sum.toFloat)

case 0.5 =>

realGraphAuthorWeightsSumP50ExpStat.add(updatedAuthorScoreMap.values.sum.toFloat)

case 0.95 =>

realGraphAuthorWeightsSumP95ExpStat.add(updatedAuthorScoreMap.values.sum.toFloat)

case \_ =>

}

val earlybirdOptionsWithUpdatedAuthorScoreMap = envelope.query.earlybirdOptions

.map(\_.copy(authorScoreAdjustments = AuthorScoreAdjustments(updatedAuthorScoreMap)))

val updatedQuery =

envelope.query.copy(earlybirdOptions = earlybirdOptionsWithUpdatedAuthorScoreMap)

envelope.copy(query = updatedQuery, followGraphData = followGraphData)

} else {

envelope.query.earlybirdOptions

.map(\_.authorScoreAdjustments.authorScoreMap.values.sum.toFloat).foreach {

realGraphAuthorWeightsSumProdStat.add(\_)

}

envelope.copy(followGraphData = followGraphData)

}

}

}

}