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

import com.google.inject.Inject

import com.google.inject.Singleton

import com.twitter.conversions.DurationOps.\_

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

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.stitch.Stitch

import com.twitter.strato.generated.client.ml.featureStore.TimelinesUserVertexOnUserClientColumn

import com.twitter.strato.generated.client.onboarding.userrecs.RealGraphScoresMhOnUserClientColumn

import com.twitter.util.Duration

import com.twitter.util.Time

import com.twitter.wtf.real\_time\_interaction\_graph.thriftscala.\_

@Singleton

class RealTimeRealGraphClient @Inject() (

timelinesUserVertexOnUserClientColumn: TimelinesUserVertexOnUserClientColumn,

realGraphScoresMhOnUserClientColumn: RealGraphScoresMhOnUserClientColumn) {

def mapUserVertexToEngagementAndFilter(userVertex: UserVertex): Map[Long, Seq[Engagement]] = {

val minTimestamp = (Time.now - RealTimeRealGraphClient.MaxEngagementAge).inMillis

userVertex.outgoingInteractionMap.mapValues { interactions =>

interactions

.flatMap { interaction => RealTimeRealGraphClient.toEngagement(interaction) }.filter(

\_.timestamp >= minTimestamp)

}.toMap

}

def getRecentProfileViewEngagements(userId: Long): Stitch[Map[Long, Seq[Engagement]]] = {

timelinesUserVertexOnUserClientColumn.fetcher

.fetch(userId).map(\_.v).map { input =>

input

.map { userVertex =>

val targetToEngagements = mapUserVertexToEngagementAndFilter(userVertex)

targetToEngagements.mapValues { engagements =>

engagements.filter(engagement =>

engagement.engagementType == EngagementType.ProfileView)

}

}.getOrElse(Map.empty)

}

}

def getUsersRecentlyEngagedWith(

userId: Long,

engagementScoreMap: Map[EngagementType, Double],

includeDirectFollowCandidates: Boolean,

includeNonDirectFollowCandidates: Boolean

): Stitch[Seq[CandidateUser]] = {

val isNewUser =

SnowflakeId.timeFromIdOpt(userId).exists { signupTime =>

(Time.now - signupTime) < RealTimeRealGraphClient.MaxNewUserAge

}

val updatedEngagementScoreMap =

if (isNewUser)

engagementScoreMap + (EngagementType.ProfileView -> RealTimeRealGraphClient.ProfileViewScore)

else engagementScoreMap

Stitch

.join(

timelinesUserVertexOnUserClientColumn.fetcher.fetch(userId).map(\_.v),

realGraphScoresMhOnUserClientColumn.fetcher.fetch(userId).map(\_.v)).map {

case (Some(userVertex), Some(neighbors)) =>

val engagements = mapUserVertexToEngagementAndFilter(userVertex)

val candidatesAndScores: Seq[(Long, Double, Seq[EngagementType])] =

EngagementScorer.apply(engagements, engagementScoreMap = updatedEngagementScoreMap)

val directNeighbors = neighbors.candidates.map(\_.\_1).toSet

val (directFollows, nonDirectFollows) = candidatesAndScores

.partition {

case (id, \_, \_) => directNeighbors.contains(id)

}

val candidates =

(if (includeNonDirectFollowCandidates) nonDirectFollows else Seq.empty) ++

(if (includeDirectFollowCandidates)

directFollows.take(RealTimeRealGraphClient.MaxNumDirectFollow)

else Seq.empty)

candidates.map {

case (id, score, proof) =>

CandidateUser(id, Some(score))

}

case \_ => Nil

}

}

def getRealGraphWeights(userId: Long): Stitch[Map[Long, Double]] =

realGraphScoresMhOnUserClientColumn.fetcher

.fetch(userId)

.map(

\_.v

.map(\_.candidates.map(candidate => (candidate.userId, candidate.score)).toMap)

.getOrElse(Map.empty[Long, Double]))

}

object RealTimeRealGraphClient {

private def toEngagement(interaction: Interaction): Option[Engagement] = {

// We do not include SoftFollow since it's deprecated

interaction match {

case Interaction.Retweet(Retweet(timestamp)) =>

Some(Engagement(EngagementType.Retweet, timestamp))

case Interaction.Favorite(Favorite(timestamp)) =>

Some(Engagement(EngagementType.Like, timestamp))

case Interaction.Click(Click(timestamp)) => Some(Engagement(EngagementType.Click, timestamp))

case Interaction.Mention(Mention(timestamp)) =>

Some(Engagement(EngagementType.Mention, timestamp))

case Interaction.ProfileView(ProfileView(timestamp)) =>

Some(Engagement(EngagementType.ProfileView, timestamp))

case \_ => None

}

}

val MaxNumDirectFollow = 50

val MaxEngagementAge: Duration = 14.days

val MaxNewUserAge: Duration = 30.days

val ProfileViewScore = 0.4

val EngagementScoreMap = Map(

EngagementType.Like -> 1.0,

EngagementType.Retweet -> 1.0,

EngagementType.Mention -> 1.0

)

val StrongEngagementScoreMap = Map(

EngagementType.Like -> 1.0,

EngagementType.Retweet -> 1.0,

)

}