package com.twitter.recos.user\_video\_graph.relatedTweetHandlers

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.graphjet.bipartite.api.BipartiteGraph

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

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

import com.twitter.servo.request.\_

import com.twitter.util.Duration

import com.twitter.util.Future

import scala.concurrent.duration.HOURS

import com.twitter.simclusters\_v2.common.UserId

import com.twitter.storehaus.ReadableStore

import com.twitter.recos.user\_video\_graph.store.UserRecentFollowersStore

import com.twitter.recos.user\_video\_graph.util.FetchRHSTweetsUtil

import com.twitter.recos.user\_video\_graph.util.FilterUtil

import com.twitter.recos.user\_video\_graph.util.GetRelatedTweetCandidatesUtil

/\*\*

\* Implementation of the Thrift-defined service interface for producerBasedRelatedTweets.

\*

\*/

class ProducerBasedRelatedTweetsHandler(

bipartiteGraph: BipartiteGraph,

userRecentFollowersStore: ReadableStore[UserRecentFollowersStore.Query, Seq[UserId]],

statsReceiver: StatsReceiver)

extends RequestHandler[ProducerBasedRelatedTweetRequest, RelatedTweetResponse] {

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

override def apply(request: ProducerBasedRelatedTweetRequest): Future[RelatedTweetResponse] = {

trackFutureBlockStats(stats) {

val maxResults = request.maxResults.getOrElse(200)

val maxNumFollowers = request.maxNumFollowers.getOrElse(500)

val minScore = request.minScore.getOrElse(0.0)

val maxTweetAge = request.maxTweetAgeInHours.getOrElse(48)

val minResultDegree = request.minResultDegree.getOrElse(50)

val minCooccurrence = request.minCooccurrence.getOrElse(4)

val excludeTweetIds = request.excludeTweetIds.getOrElse(Seq.empty).toSet

val followersFut = fetchFollowers(request.producerId, Some(maxNumFollowers))

followersFut.map { followers =>

val rhsTweetIds = FetchRHSTweetsUtil.fetchRHSTweets(

followers,

bipartiteGraph

)

val scorePreFactor = 1000.0 / followers.size

val relatedTweetCandidates = GetRelatedTweetCandidatesUtil.getRelatedTweetCandidates(

rhsTweetIds,

minCooccurrence,

minResultDegree,

scorePreFactor,

bipartiteGraph)

val relatedTweets = relatedTweetCandidates

.filter { relatedTweet =>

FilterUtil.tweetAgeFilter(

relatedTweet.tweetId,

Duration(maxTweetAge, HOURS)) && (relatedTweet.score > minScore) && (!excludeTweetIds

.contains(relatedTweet.tweetId))

}.take(maxResults)

stats.stat("response\_size").add(relatedTweets.size)

RelatedTweetResponse(tweets = relatedTweets)

}

}

}

private def fetchFollowers(

producerId: Long,

maxNumFollower: Option[Int],

): Future[Seq[Long]] = {

val query =

UserRecentFollowersStore.Query(producerId, maxNumFollower, None)

val followersFut = userRecentFollowersStore.get(query)

followersFut.map { followersOpt =>

val followers = followersOpt.getOrElse(Seq.empty)

val followerIds = followers.distinct.filter { userId =>

val userDegree = bipartiteGraph.getLeftNodeDegree(userId)

// constrain to more active users that have >1 engagement to optimize latency, and <100 engagements to avoid spammy behavior

userDegree > 1 && userDegree < 500

}

stats.stat("follower\_size\_after\_filter").add(followerIds.size)

followerIds

}

}

}