package com.twitter.timelineranker.visibility

import com.twitter.finagle.stats.Stat

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.servo.repository.KeyValueRepository

import com.twitter.servo.util.Gate

import com.twitter.timelineranker.core.FollowGraphData

import com.twitter.timelineranker.core.FollowGraphDataFuture

import com.twitter.timelines.clients.socialgraph.SocialGraphClient

import com.twitter.timelines.model.UserId

import com.twitter.timelines.util.FailOpenHandler

import com.twitter.util.Future

import com.twitter.util.Stopwatch

import com.twitter.wtf.candidate.thriftscala.CandidateSeq

object RealGraphFollowGraphDataProvider {

val EmptyRealGraphResponse = CandidateSeq(Nil)

}

/\*\*

\* Wraps an underlying FollowGraphDataProvider (which in practice will usually be a

\* [[SgsFollowGraphDataProvider]]) and supplements the list of followings provided by the

\* underlying provider with additional followings fetched from RealGraph if it looks like the

\* underlying provider did not get the full list of the user's followings.

\*

\* First checks whether the size of the underlying following list is >= the max requested following

\* count, which implies that there were additional followings beyond the max requested count. If so,

\* fetches the full set of followings from RealGraph (go/realgraph), which will be at most 2000.

\*

\* Because the RealGraph dataset is not realtime and thus can potentially include stale followings,

\* the provider confirms that the followings fetched from RealGraph are valid using SGS's

\* getFollowOverlap method, and then merges the valid RealGraph followings with the underlying

\* followings.

\*

\* Note that this supplementing is expected to be very rare as most users do not have more than

\* the max followings we fetch from SGS. Also note that this class is mainly intended for use

\* in the home timeline materialization path, with the goal of preventing a case where users

\* who follow a very large number of accounts may not see Tweets from their earlier follows if we

\* used SGS-based follow fetching alone.

\*/

class RealGraphFollowGraphDataProvider(

underlying: FollowGraphDataProvider,

realGraphClient: KeyValueRepository[Seq[UserId], UserId, CandidateSeq],

socialGraphClient: SocialGraphClient,

supplementFollowsWithRealGraphGate: Gate[UserId],

statsReceiver: StatsReceiver)

extends FollowGraphDataProvider {

import RealGraphFollowGraphDataProvider.\_

private[this] val scopedStatsReceiver = statsReceiver.scope("realGraphFollowGraphDataProvider")

private[this] val requestCounter = scopedStatsReceiver.counter("requests")

private[this] val atMaxCounter = scopedStatsReceiver.counter("followsAtMax")

private[this] val totalLatencyStat = scopedStatsReceiver.stat("totalLatencyWhenSupplementing")

private[this] val supplementLatencyStat = scopedStatsReceiver.stat("supplementFollowsLatency")

private[this] val realGraphResponseSizeStat = scopedStatsReceiver.stat("realGraphFollows")

private[this] val realGraphEmptyCounter = scopedStatsReceiver.counter("realGraphEmpty")

private[this] val nonOverlappingSizeStat = scopedStatsReceiver.stat("nonOverlappingFollows")

private[this] val failOpenHandler = new FailOpenHandler(scopedStatsReceiver)

override def get(userId: UserId, maxFollowingCount: Int): Future[FollowGraphData] = {

getAsync(userId, maxFollowingCount).get()

}

override def getAsync(userId: UserId, maxFollowingCount: Int): FollowGraphDataFuture = {

val startTime = Stopwatch.timeMillis()

val underlyingResult = underlying.getAsync(userId, maxFollowingCount)

if (supplementFollowsWithRealGraphGate(userId)) {

val supplementedFollows = underlyingResult.followedUserIdsFuture.flatMap { sgsFollows =>

supplementFollowsWithRealGraph(userId, maxFollowingCount, sgsFollows, startTime)

}

underlyingResult.copy(followedUserIdsFuture = supplementedFollows)

} else {

underlyingResult

}

}

override def getFollowing(userId: UserId, maxFollowingCount: Int): Future[Seq[UserId]] = {

val startTime = Stopwatch.timeMillis()

val underlyingFollows = underlying.getFollowing(userId, maxFollowingCount)

if (supplementFollowsWithRealGraphGate(userId)) {

underlying.getFollowing(userId, maxFollowingCount).flatMap { sgsFollows =>

supplementFollowsWithRealGraph(userId, maxFollowingCount, sgsFollows, startTime)

}

} else {

underlyingFollows

}

}

private[this] def supplementFollowsWithRealGraph(

userId: UserId,

maxFollowingCount: Int,

sgsFollows: Seq[Long],

startTime: Long

): Future[Seq[UserId]] = {

requestCounter.incr()

if (sgsFollows.size >= maxFollowingCount) {

atMaxCounter.incr()

val supplementedFollowsFuture = realGraphClient(Seq(userId))

.map(\_.getOrElse(userId, EmptyRealGraphResponse))

.map(\_.candidates.map(\_.userId))

.flatMap {

case realGraphFollows if realGraphFollows.nonEmpty =>

realGraphResponseSizeStat.add(realGraphFollows.size)

// Filter out "stale" follows from realgraph by checking them against SGS

val verifiedRealGraphFollows =

socialGraphClient.getFollowOverlap(userId, realGraphFollows)

verifiedRealGraphFollows.map { follows =>

val combinedFollows = (sgsFollows ++ follows).distinct

val additionalFollows = combinedFollows.size - sgsFollows.size

if (additionalFollows > 0) nonOverlappingSizeStat.add(additionalFollows)

combinedFollows

}

case \_ =>

realGraphEmptyCounter.incr()

Future.value(sgsFollows)

}

.onSuccess { \_ => totalLatencyStat.add(Stopwatch.timeMillis() - startTime) }

Stat.timeFuture(supplementLatencyStat) {

failOpenHandler(supplementedFollowsFuture) { \_ => Future.value(sgsFollows) }

}

} else {

Future.value(sgsFollows)

}

}

override def getMutuallyFollowingUserIds(

userId: UserId,

followingIds: Seq[UserId]

): Future[Set[UserId]] = {

underlying.getMutuallyFollowingUserIds(userId, followingIds)

}

}