package com.twitter.follow\_recommendations.services

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.base.StatsUtil

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

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

import com.twitter.follow\_recommendations.models.DebugParams

import com.twitter.follow\_recommendations.models.RecommendationRequest

import com.twitter.follow\_recommendations.models.RecommendationResponse

import com.twitter.stitch.Stitch

import com.twitter.timelines.configapi.Params

import javax.inject.Inject

import javax.inject.Singleton

import scala.util.Random

@Singleton

class ProductPipelineSelector @Inject() (

recommendationsService: RecommendationsService,

productMixerRecommendationService: ProductMixerRecommendationService,

productPipelineSelectorConfig: ProductPipelineSelectorConfig,

baseStats: StatsReceiver) {

private val frsStats = baseStats.scope("follow\_recommendations\_service")

private val stats = frsStats.scope("product\_pipeline\_selector\_parity")

private val readFromProductMixerCounter = stats.counter("select\_product\_mixer")

private val readFromOldFRSCounter = stats.counter("select\_old\_frs")

def selectPipeline(

request: RecommendationRequest,

params: Params

): Stitch[RecommendationResponse] = {

productPipelineSelectorConfig

.getDarkReadAndExpParams(request.displayLocation).map { darkReadAndExpParam =>

if (params(darkReadAndExpParam.expParam)) {

readFromProductMixerPipeline(request, params)

} else if (params(darkReadAndExpParam.darkReadParam)) {

darkReadAndReturnResult(request, params)

} else {

readFromOldFrsPipeline(request, params)

}

}.getOrElse(readFromOldFrsPipeline(request, params))

}

private def readFromProductMixerPipeline(

request: RecommendationRequest,

params: Params

): Stitch[RecommendationResponse] = {

readFromProductMixerCounter.incr()

productMixerRecommendationService.get(request, params)

}

private def readFromOldFrsPipeline(

request: RecommendationRequest,

params: Params

): Stitch[RecommendationResponse] = {

readFromOldFRSCounter.incr()

recommendationsService.get(request, params)

}

private def darkReadAndReturnResult(

request: RecommendationRequest,

params: Params

): Stitch[RecommendationResponse] = {

val darkReadStats = stats.scope("select\_dark\_read", request.displayLocation.toFsName)

darkReadStats.counter("count").incr()

// If no seed is set, create a random one that both requests will use to remove differences

// in randomness for the WeightedCandidateSourceRanker

val randomizationSeed = new Random().nextLong()

val oldFRSPiplelineRequest = request.copy(

debugParams = Some(

request.debugParams.getOrElse(

DebugParams(None, Some(DebugOptions(randomizationSeed = Some(randomizationSeed))))))

)

val productMixerPipelineRequest = request.copy(

debugParams = Some(

request.debugParams.getOrElse(

DebugParams(

None,

Some(DebugOptions(doNotLog = true, randomizationSeed = Some(randomizationSeed))))))

)

StatsUtil

.profileStitch(

readFromOldFrsPipeline(oldFRSPiplelineRequest, params),

darkReadStats.scope("frs\_timing")).applyEffect { frsOldPipelineResponse =>

Stitch.async(

StatsUtil

.profileStitch(

readFromProductMixerPipeline(productMixerPipelineRequest, params),

darkReadStats.scope("product\_mixer\_timing")).liftToOption().map {

case Some(frsProductMixerResponse) =>

darkReadStats.counter("product\_mixer\_pipeline\_success").incr()

compare(request, frsOldPipelineResponse, frsProductMixerResponse)

case None =>

darkReadStats.counter("product\_mixer\_pipeline\_failure").incr()

}

)

}

}

def compare(

request: RecommendationRequest,

frsOldPipelineResponse: RecommendationResponse,

frsProductMixerResponse: RecommendationResponse

): Unit = {

val compareStats = stats.scope("pipeline\_comparison", request.displayLocation.toFsName)

compareStats.counter("total-comparisons").incr()

val oldFrsMap = frsOldPipelineResponse.recommendations.map { user => user.id -> user }.toMap

val productMixerMap = frsProductMixerResponse.recommendations.map { user =>

user.id -> user

}.toMap

compareTopNResults(3, frsOldPipelineResponse, frsProductMixerResponse, compareStats)

compareTopNResults(5, frsOldPipelineResponse, frsProductMixerResponse, compareStats)

compareTopNResults(25, frsOldPipelineResponse, frsProductMixerResponse, compareStats)

compareTopNResults(50, frsOldPipelineResponse, frsProductMixerResponse, compareStats)

compareTopNResults(75, frsOldPipelineResponse, frsProductMixerResponse, compareStats)

// Compare individual matching candidates

oldFrsMap.keys.foreach(userId => {

if (productMixerMap.contains(userId)) {

(oldFrsMap(userId), productMixerMap(userId)) match {

case (oldFrsUser: CandidateUser, productMixerUser: CandidateUser) =>

compareStats.counter("matching-user-count").incr()

compareUser(oldFrsUser, productMixerUser, compareStats)

case \_ =>

compareStats.counter("unknown-user-type-count").incr()

}

} else {

compareStats.counter("missing-user-count").incr()

}

})

}

private def compareTopNResults(

n: Int,

frsOldPipelineResponse: RecommendationResponse,

frsProductMixerResponse: RecommendationResponse,

compareStats: StatsReceiver

): Unit = {

if (frsOldPipelineResponse.recommendations.size >= n && frsProductMixerResponse.recommendations.size >= n) {

val oldFrsPipelineFirstN = frsOldPipelineResponse.recommendations.take(n).map(\_.id)

val productMixerPipelineFirstN = frsProductMixerResponse.recommendations.take(n).map(\_.id)

if (oldFrsPipelineFirstN.sorted == productMixerPipelineFirstN.sorted)

compareStats.counter(s"first-$n-sorted-equal-ids").incr()

if (oldFrsPipelineFirstN == productMixerPipelineFirstN)

compareStats.counter(s"first-$n-unsorted-ids-equal").incr()

else

compareStats.counter(s"first-$n-unsorted-ids-unequal").incr()

}

}

private def compareUser(

oldFrsUser: CandidateUser,

productMixerUser: CandidateUser,

stats: StatsReceiver

): Unit = {

val userStats = stats.scope("matching-user")

if (oldFrsUser.score != productMixerUser.score)

userStats.counter("mismatch-score").incr()

if (oldFrsUser.reason != productMixerUser.reason)

userStats.counter("mismatch-reason").incr()

if (oldFrsUser.userCandidateSourceDetails != productMixerUser.userCandidateSourceDetails)

userStats.counter("mismatch-userCandidateSourceDetails").incr()

if (oldFrsUser.adMetadata != productMixerUser.adMetadata)

userStats.counter("mismatch-adMetadata").incr()

if (oldFrsUser.trackingToken != productMixerUser.trackingToken)

userStats.counter("mismatch-trackingToken").incr()

if (oldFrsUser.dataRecord != productMixerUser.dataRecord)

userStats.counter("mismatch-dataRecord").incr()

if (oldFrsUser.scores != productMixerUser.scores)

userStats.counter("mismatch-scores").incr()

if (oldFrsUser.infoPerRankingStage != productMixerUser.infoPerRankingStage)

userStats.counter("mismatch-infoPerRankingStage").incr()

if (oldFrsUser.params != productMixerUser.params)

userStats.counter("mismatch-params").incr()

if (oldFrsUser.engagements != productMixerUser.engagements)

userStats.counter("mismatch-engagements").incr()

if (oldFrsUser.recommendationFlowIdentifier != productMixerUser.recommendationFlowIdentifier)

userStats.counter("mismatch-recommendationFlowIdentifier").incr()

}

}