package com.twitter.frigate.pushservice.adaptor

import com.twitter.cr\_mixer.thriftscala.FrsTweetRequest

import com.twitter.cr\_mixer.thriftscala.NotificationsContext

import com.twitter.cr\_mixer.thriftscala.Product

import com.twitter.cr\_mixer.thriftscala.ProductContext

import com.twitter.finagle.stats.Counter

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.CandidateSource

import com.twitter.frigate.common.base.CandidateSourceEligible

import com.twitter.frigate.common.base.\_

import com.twitter.frigate.common.predicate.CommonOutNetworkTweetCandidatesSourcePredicates.filterOutReplyTweet

import com.twitter.frigate.pushservice.model.PushTypes.RawCandidate

import com.twitter.frigate.pushservice.model.PushTypes.Target

import com.twitter.frigate.pushservice.params.PushFeatureSwitchParams

import com.twitter.frigate.pushservice.store.CrMixerTweetStore

import com.twitter.frigate.pushservice.store.UttEntityHydrationStore

import com.twitter.frigate.pushservice.util.MediaCRT

import com.twitter.frigate.pushservice.util.PushAdaptorUtil

import com.twitter.frigate.pushservice.util.PushDeviceUtil

import com.twitter.frigate.pushservice.util.TopicsUtil

import com.twitter.frigate.thriftscala.CommonRecommendationType

import com.twitter.hermit.constants.AlgorithmFeedbackTokens

import com.twitter.hermit.model.Algorithm.Algorithm

import com.twitter.hermit.model.Algorithm.CrowdSearchAccounts

import com.twitter.hermit.model.Algorithm.ForwardEmailBook

import com.twitter.hermit.model.Algorithm.ForwardPhoneBook

import com.twitter.hermit.model.Algorithm.ReverseEmailBookIbis

import com.twitter.hermit.model.Algorithm.ReversePhoneBook

import com.twitter.hermit.store.tweetypie.UserTweet

import com.twitter.product\_mixer.core.thriftscala.ClientContext

import com.twitter.stitch.tweetypie.TweetyPie.TweetyPieResult

import com.twitter.storehaus.ReadableStore

import com.twitter.tsp.thriftscala.TopicSocialProofRequest

import com.twitter.tsp.thriftscala.TopicSocialProofResponse

import com.twitter.util.Future

object FRSAlgorithmFeedbackTokenUtil {

private val crtsByAlgoToken = Map(

getAlgorithmToken(ReverseEmailBookIbis) -> CommonRecommendationType.ReverseAddressbookTweet,

getAlgorithmToken(ReversePhoneBook) -> CommonRecommendationType.ReverseAddressbookTweet,

getAlgorithmToken(ForwardEmailBook) -> CommonRecommendationType.ForwardAddressbookTweet,

getAlgorithmToken(ForwardPhoneBook) -> CommonRecommendationType.ForwardAddressbookTweet,

getAlgorithmToken(CrowdSearchAccounts) -> CommonRecommendationType.CrowdSearchTweet

)

def getAlgorithmToken(algorithm: Algorithm): Int = {

AlgorithmFeedbackTokens.AlgorithmToFeedbackTokenMap(algorithm)

}

def getCRTForAlgoToken(algorithmToken: Int): Option[CommonRecommendationType] = {

crtsByAlgoToken.get(algorithmToken)

}

}

case class FRSTweetCandidateAdaptor(

crMixerTweetStore: CrMixerTweetStore,

tweetyPieStore: ReadableStore[Long, TweetyPieResult],

tweetyPieStoreNoVF: ReadableStore[Long, TweetyPieResult],

userTweetTweetyPieStore: ReadableStore[UserTweet, TweetyPieResult],

uttEntityHydrationStore: UttEntityHydrationStore,

topicSocialProofServiceStore: ReadableStore[TopicSocialProofRequest, TopicSocialProofResponse],

globalStats: StatsReceiver)

extends CandidateSource[Target, RawCandidate]

with CandidateSourceEligible[Target, RawCandidate] {

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

private val crtStats = stats.scope("CandidateDistribution")

private val totalRequests = stats.counter("total\_requests")

// Candidate Distribution stats

private val reverseAddressbookCounter = crtStats.counter("reverse\_addressbook")

private val forwardAddressbookCounter = crtStats.counter("forward\_addressbook")

private val frsTweetCounter = crtStats.counter("frs\_tweet")

private val nonReplyTweetsCounter = stats.counter("non\_reply\_tweets")

private val crtToCounterMapping: Map[CommonRecommendationType, Counter] = Map(

CommonRecommendationType.ReverseAddressbookTweet -> reverseAddressbookCounter,

CommonRecommendationType.ForwardAddressbookTweet -> forwardAddressbookCounter,

CommonRecommendationType.FrsTweet -> frsTweetCounter

)

private val emptyTweetyPieResult = stats.stat("empty\_tweetypie\_result")

private[this] val numberReturnedCandidates = stats.stat("returned\_candidates\_from\_earlybird")

private[this] val numberCandidateWithTopic: Counter = stats.counter("num\_can\_with\_topic")

private[this] val numberCandidateWithoutTopic: Counter = stats.counter("num\_can\_without\_topic")

private val userTweetTweetyPieStoreCounter = stats.counter("user\_tweet\_tweetypie\_store")

override val name: String = this.getClass.getSimpleName

private def filterInvalidTweets(

tweetIds: Seq[Long],

target: Target

): Future[Map[Long, TweetyPieResult]] = {

val resMap = {

if (target.params(PushFeatureSwitchParams.EnableF1FromProtectedTweetAuthors)) {

userTweetTweetyPieStoreCounter.incr()

val keys = tweetIds.map { tweetId =>

UserTweet(tweetId, Some(target.targetId))

}

userTweetTweetyPieStore

.multiGet(keys.toSet).map {

case (userTweet, resultFut) =>

userTweet.tweetId -> resultFut

}.toMap

} else {

(if (target.params(PushFeatureSwitchParams.EnableVFInTweetypie)) {

tweetyPieStore

} else {

tweetyPieStoreNoVF

}).multiGet(tweetIds.toSet)

}

}

Future.collect(resMap).map { tweetyPieResultMap =>

// Filter out replies and generate earlybird candidates only for non-empty tweetypie result

val cands = filterOutReplyTweet(tweetyPieResultMap, nonReplyTweetsCounter).collect {

case (id: Long, Some(result)) =>

id -> result

}

emptyTweetyPieResult.add(tweetyPieResultMap.size - cands.size)

cands

}

}

private def buildRawCandidates(

target: Target,

ebCandidates: Seq[FRSTweetCandidate]

): Future[Option[Seq[RawCandidate with TweetCandidate]]] = {

val enableTopic = target.params(PushFeatureSwitchParams.EnableFrsTweetCandidatesTopicAnnotation)

val topicScoreThre =

target.params(PushFeatureSwitchParams.FrsTweetCandidatesTopicScoreThreshold)

val ebTweets = ebCandidates.map { ebCandidate =>

ebCandidate.tweetId -> ebCandidate.tweetyPieResult

}.toMap

val tweetIdLocalizedEntityMapFut = TopicsUtil.getTweetIdLocalizedEntityMap(

target,

ebTweets,

uttEntityHydrationStore,

topicSocialProofServiceStore,

enableTopic,

topicScoreThre

)

Future.join(target.deviceInfo, tweetIdLocalizedEntityMapFut).map {

case (Some(deviceInfo), tweetIdLocalizedEntityMap) =>

val candidates = ebCandidates

.map { ebCandidate =>

val crt = ebCandidate.commonRecType

crtToCounterMapping.get(crt).foreach(\_.incr())

val tweetId = ebCandidate.tweetId

val localizedEntityOpt = {

if (tweetIdLocalizedEntityMap

.contains(tweetId) && tweetIdLocalizedEntityMap.contains(

tweetId) && deviceInfo.isTopicsEligible) {

tweetIdLocalizedEntityMap(tweetId)

} else {

None

}

}

PushAdaptorUtil.generateOutOfNetworkTweetCandidates(

inputTarget = target,

id = ebCandidate.tweetId,

mediaCRT = MediaCRT(

crt,

crt,

crt

),

result = ebCandidate.tweetyPieResult,

localizedEntity = localizedEntityOpt)

}.filter { candidate =>

// If user only has the topic setting enabled, filter out all non-topic cands

deviceInfo.isRecommendationsEligible || (deviceInfo.isTopicsEligible && candidate.semanticCoreEntityId.nonEmpty)

}

candidates.map { candidate =>

if (candidate.semanticCoreEntityId.nonEmpty) {

numberCandidateWithTopic.incr()

} else {

numberCandidateWithoutTopic.incr()

}

}

numberReturnedCandidates.add(candidates.length)

Some(candidates)

case \_ => Some(Seq.empty)

}

}

def getTweetCandidatesFromCrMixer(

inputTarget: Target,

showAllResultsFromFrs: Boolean,

): Future[Option[Seq[RawCandidate with TweetCandidate]]] = {

Future

.join(

inputTarget.seenTweetIds,

inputTarget.pushRecItems,

inputTarget.countryCode,

inputTarget.targetLanguage).flatMap {

case (seenTweetIds, pastRecItems, countryCode, language) =>

val pastUserRecs = pastRecItems.userIds.toSeq

val request = FrsTweetRequest(

clientContext = ClientContext(

userId = Some(inputTarget.targetId),

countryCode = countryCode,

languageCode = language

),

product = Product.Notifications,

productContext = Some(ProductContext.NotificationsContext(NotificationsContext())),

excludedUserIds = Some(pastUserRecs),

excludedTweetIds = Some(seenTweetIds)

)

crMixerTweetStore.getFRSTweetCandidates(request).flatMap {

case Some(response) =>

val tweetIds = response.tweets.map(\_.tweetId)

val validTweets = filterInvalidTweets(tweetIds, inputTarget)

validTweets.flatMap { tweetypieMap =>

val ebCandidates = response.tweets

.map { frsTweet =>

val candidateTweetId = frsTweet.tweetId

val resultFromTweetyPie = tweetypieMap.get(candidateTweetId)

new FRSTweetCandidate {

override val tweetId = candidateTweetId

override val features = None

override val tweetyPieResult = resultFromTweetyPie

override val feedbackToken = frsTweet.frsPrimarySource

override val commonRecType: CommonRecommendationType = feedbackToken

.flatMap(token =>

FRSAlgorithmFeedbackTokenUtil.getCRTForAlgoToken(token)).getOrElse(

CommonRecommendationType.FrsTweet)

}

}.filter { ebCandidate =>

showAllResultsFromFrs || ebCandidate.commonRecType == CommonRecommendationType.ReverseAddressbookTweet

}

numberReturnedCandidates.add(ebCandidates.length)

buildRawCandidates(

inputTarget,

ebCandidates

)

}

case \_ => Future.None

}

}

}

override def get(inputTarget: Target): Future[Option[Seq[RawCandidate with TweetCandidate]]] = {

totalRequests.incr()

val enableResultsFromFrs =

inputTarget.params(PushFeatureSwitchParams.EnableResultFromFrsCandidates)

getTweetCandidatesFromCrMixer(inputTarget, enableResultsFromFrs)

}

override def isCandidateSourceAvailable(target: Target): Future[Boolean] = {

lazy val enableFrsCandidates = target.params(PushFeatureSwitchParams.EnableFrsCandidates)

PushDeviceUtil.isRecommendationsEligible(target).flatMap { isEnabledForRecosSetting =>

PushDeviceUtil.isTopicsEligible(target).map { topicSettingEnabled =>

val isEnabledForTopics =

topicSettingEnabled && target.params(

PushFeatureSwitchParams.EnableFrsTweetCandidatesTopicSetting)

(isEnabledForRecosSetting || isEnabledForTopics) && enableFrsCandidates

}

}

}

}