package com.twitter.follow\_recommendations.common.feature\_hydration.sources

import com.github.benmanes.caffeine.cache.Caffeine

import com.google.inject.Inject

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.TimeoutException

import com.twitter.finagle.mtls.authentication.ServiceIdentifier

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.constants.CandidateAlgorithmTypeConstants

import com.twitter.follow\_recommendations.common.feature\_hydration.adapters.CandidateAlgorithmAdapter.remapCandidateSource

import com.twitter.follow\_recommendations.common.feature\_hydration.adapters.PostNuxAlgorithmIdAdapter

import com.twitter.follow\_recommendations.common.feature\_hydration.adapters.PostNuxAlgorithmTypeAdapter

import com.twitter.follow\_recommendations.common.feature\_hydration.common.FeatureSource

import com.twitter.follow\_recommendations.common.feature\_hydration.common.FeatureSourceId

import com.twitter.follow\_recommendations.common.feature\_hydration.common.HasPreFetchedFeature

import com.twitter.follow\_recommendations.common.feature\_hydration.sources.Utils.adaptAdditionalFeaturesToDataRecord

import com.twitter.follow\_recommendations.common.feature\_hydration.sources.Utils.randomizedTTL

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

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

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

import com.twitter.hermit.constants.AlgorithmFeedbackTokens.AlgorithmToFeedbackTokenMap

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.DataRecordMerger

import com.twitter.ml.api.FeatureContext

import com.twitter.ml.api.IRecordOneToOneAdapter

import com.twitter.ml.featurestore.catalog.datasets.customer\_journey.PostNuxAlgorithmIdAggregateDataset

import com.twitter.ml.featurestore.catalog.datasets.customer\_journey.PostNuxAlgorithmTypeAggregateDataset

import com.twitter.ml.featurestore.catalog.entities.onboarding.{WtfAlgorithm => OnboardingWtfAlgoId}

import com.twitter.ml.featurestore.catalog.entities.onboarding.{

WtfAlgorithmType => OnboardingWtfAlgoType

}

import com.twitter.ml.featurestore.catalog.features.customer\_journey.CombineAllFeaturesPolicy

import com.twitter.ml.featurestore.lib.EntityId

import com.twitter.ml.featurestore.lib.WtfAlgorithmId

import com.twitter.ml.featurestore.lib.WtfAlgorithmType

import com.twitter.ml.featurestore.lib.data.PredictionRecord

import com.twitter.ml.featurestore.lib.data.PredictionRecordAdapter

import com.twitter.ml.featurestore.lib.dataset.DatasetId

import com.twitter.ml.featurestore.lib.dataset.online.Hydrator.HydrationResponse

import com.twitter.ml.featurestore.lib.dataset.online.OnlineAccessDataset

import com.twitter.ml.featurestore.lib.dynamic.ClientConfig

import com.twitter.ml.featurestore.lib.dynamic.DynamicFeatureStoreClient

import com.twitter.ml.featurestore.lib.dynamic.DynamicHydrationConfig

import com.twitter.ml.featurestore.lib.dynamic.FeatureStoreParamsConfig

import com.twitter.ml.featurestore.lib.dynamic.GatedFeatures

import com.twitter.ml.featurestore.lib.entity.EntityWithId

import com.twitter.ml.featurestore.lib.feature.BoundFeature

import com.twitter.ml.featurestore.lib.feature.BoundFeatureSet

import com.twitter.ml.featurestore.lib.online.DatasetValuesCache

import com.twitter.ml.featurestore.lib.online.FeatureStoreRequest

import com.twitter.ml.featurestore.lib.online.OnlineFeatureGenerationStats

import com.twitter.product\_mixer.core.model.marshalling.request.HasClientContext

import com.twitter.stitch.Stitch

import com.twitter.timelines.configapi.HasParams

import java.util.concurrent.TimeUnit

import scala.collection.JavaConverters.\_

class FeatureStorePostNuxAlgorithmSource @Inject() (

serviceIdentifier: ServiceIdentifier,

stats: StatsReceiver)

extends FeatureSource {

import FeatureStorePostNuxAlgorithmSource.\_

val backupSourceStats = stats.scope("feature\_store\_hydration\_post\_nux\_algorithm")

val adapterStats = backupSourceStats.scope("adapters")

override def id: FeatureSourceId = FeatureSourceId.FeatureStorePostNuxAlgorithmSourceId

override def featureContext: FeatureContext = getFeatureContext

private val dataRecordMerger = new DataRecordMerger

val clientConfig: ClientConfig[HasParams] = ClientConfig(

dynamicHydrationConfig = dynamicHydrationConfig,

featureStoreParamsConfig =

FeatureStoreParamsConfig(FeatureStoreParameters.featureStoreParams, Map.empty),

/\*\*

\* The smaller one between `timeoutProvider` and `FeatureStoreSourceParams.GlobalFetchTimeout`

\* used below takes effect.

\*/

timeoutProvider = Function.const(800.millis),

serviceIdentifier = serviceIdentifier

)

private val datasetsToCache = Set(

PostNuxAlgorithmIdAggregateDataset,

PostNuxAlgorithmTypeAggregateDataset,

).asInstanceOf[Set[OnlineAccessDataset[\_ <: EntityId, \_]]]

private val datasetValuesCache: DatasetValuesCache =

DatasetValuesCache(

Caffeine

.newBuilder()

.expireAfterWrite(randomizedTTL(12.hours.inSeconds), TimeUnit.SECONDS)

.maximumSize(DefaultCacheMaxKeys)

.build[(\_ <: EntityId, DatasetId), Stitch[HydrationResponse[\_]]]

.asMap,

datasetsToCache,

DatasetCacheScope

)

private val dynamicFeatureStoreClient = DynamicFeatureStoreClient(

clientConfig,

backupSourceStats,

Set(datasetValuesCache)

)

private val adapterToDataRecord: IRecordOneToOneAdapter[PredictionRecord] =

PredictionRecordAdapter.oneToOne(

BoundFeatureSet(allFeatures),

OnlineFeatureGenerationStats(backupSourceStats)

)

// These two calculate the rate for each feature by dividing it by the number of impressions, then

// apply a log transformation.

private val transformAdapters = Seq(PostNuxAlgorithmIdAdapter, PostNuxAlgorithmTypeAdapter)

override def hydrateFeatures(

target: HasClientContext

with HasPreFetchedFeature

with HasParams

with HasSimilarToContext

with HasDisplayLocation,

candidates: Seq[CandidateUser]

): Stitch[Map[CandidateUser, DataRecord]] = {

target.getOptionalUserId

.map { \_: Long =>

val candidateAlgoIdEntities = candidates.map { candidate =>

candidate.id -> candidate.getAllAlgorithms

.flatMap { algo =>

AlgorithmToFeedbackTokenMap.get(remapCandidateSource(algo))

}.map(algoId => OnboardingWtfAlgoId.withId(WtfAlgorithmId(algoId)))

}.toMap

val candidateAlgoTypeEntities = candidateAlgoIdEntities.map {

case (candidateId, algoIdEntities) =>

candidateId -> algoIdEntities

.map(\_.id.algoId)

.flatMap(algoId => CandidateAlgorithmTypeConstants.getAlgorithmTypes(algoId.toString))

.distinct

.map(algoType => OnboardingWtfAlgoType.withId(WtfAlgorithmType(algoType)))

}

val entities = {

candidateAlgoIdEntities.values.flatten ++ candidateAlgoTypeEntities.values.flatten

}.toSeq.distinct

val requests = entities.map(entity => FeatureStoreRequest(Seq(entity)))

val predictionRecordsFut = dynamicFeatureStoreClient(requests, target)

val candidateFeatureMap = predictionRecordsFut.map {

predictionRecords: Seq[PredictionRecord] =>

val entityFeatureMap: Map[EntityWithId[\_], DataRecord] = entities

.zip(predictionRecords).map {

case (entity, predictionRecord) =>

entity -> adaptAdditionalFeaturesToDataRecord(

adapterToDataRecord.adaptToDataRecord(predictionRecord),

adapterStats,

transformAdapters)

}.toMap

// In case we have more than one algorithm ID, or type, for a candidate, we merge the

// resulting DataRecords using the two merging policies below.

val algoIdMergeFn =

CombineAllFeaturesPolicy(PostNuxAlgorithmIdAdapter.getFeatures).getMergeFn

val algoTypeMergeFn =

CombineAllFeaturesPolicy(PostNuxAlgorithmTypeAdapter.getFeatures).getMergeFn

val candidateAlgoIdFeaturesMap = candidateAlgoIdEntities.mapValues { entities =>

val features = entities.flatMap(e => Option(entityFeatureMap.getOrElse(e, null)))

algoIdMergeFn(features)

}

val candidateAlgoTypeFeaturesMap = candidateAlgoTypeEntities.mapValues { entities =>

val features = entities.flatMap(e => Option(entityFeatureMap.getOrElse(e, null)))

algoTypeMergeFn(features)

}

candidates.map { candidate =>

val idDrOpt = candidateAlgoIdFeaturesMap.getOrElse(candidate.id, None)

val typeDrOpt = candidateAlgoTypeFeaturesMap.getOrElse(candidate.id, None)

val featureDr = (idDrOpt, typeDrOpt) match {

case (None, Some(typeDataRecord)) => typeDataRecord

case (Some(idDataRecord), None) => idDataRecord

case (None, None) => new DataRecord()

case (Some(idDataRecord), Some(typeDataRecord)) =>

dataRecordMerger.merge(idDataRecord, typeDataRecord)

idDataRecord

}

candidate -> featureDr

}.toMap

}

Stitch

.callFuture(candidateFeatureMap)

.within(target.params(FeatureStoreSourceParams.GlobalFetchTimeout))(

com.twitter.finagle.util.DefaultTimer)

.rescue {

case \_: TimeoutException =>

Stitch.value(Map.empty[CandidateUser, DataRecord])

}

}.getOrElse(Stitch.value(Map.empty[CandidateUser, DataRecord]))

}

}

object FeatureStorePostNuxAlgorithmSource {

private val DatasetCacheScope = "feature\_store\_local\_cache\_post\_nux\_algorithm"

private val DefaultCacheMaxKeys = 1000 // Both of these datasets have <50 keys total.

val allFeatures: Set[BoundFeature[\_ <: EntityId, \_]] =

FeatureStoreFeatures.postNuxAlgorithmIdAggregateFeatures ++

FeatureStoreFeatures.postNuxAlgorithmTypeAggregateFeatures

val algoIdFinalFeatures = CombineAllFeaturesPolicy(

PostNuxAlgorithmIdAdapter.getFeatures).outputFeaturesPostMerge.toSeq

val algoTypeFinalFeatures = CombineAllFeaturesPolicy(

PostNuxAlgorithmTypeAdapter.getFeatures).outputFeaturesPostMerge.toSeq

val getFeatureContext: FeatureContext =

new FeatureContext().addFeatures((algoIdFinalFeatures ++ algoTypeFinalFeatures).asJava)

val dynamicHydrationConfig: DynamicHydrationConfig[HasParams] =

DynamicHydrationConfig(

Set(

GatedFeatures(

boundFeatureSet =

BoundFeatureSet(FeatureStoreFeatures.postNuxAlgorithmIdAggregateFeatures),

gate = HasParams.paramGate(FeatureStoreSourceParams.EnableAlgorithmAggregateFeatures)

),

GatedFeatures(

boundFeatureSet =

BoundFeatureSet(FeatureStoreFeatures.postNuxAlgorithmTypeAggregateFeatures),

gate = HasParams.paramGate(FeatureStoreSourceParams.EnableAlgorithmAggregateFeatures)

),

))

}