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

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

import com.google.inject.Inject

import com.twitter.finagle.TimeoutException

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

import com.twitter.finagle.stats.StatsReceiver

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.HasSimilarToContext

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.FeatureContext

import com.twitter.ml.api.IRecordOneToOneAdapter

import com.twitter.ml.featurestore.catalog.datasets.core.UsersourceEntityDataset

import com.twitter.ml.featurestore.catalog.entities.core.{Author => AuthorEntity}

import com.twitter.ml.featurestore.catalog.entities.core.{AuthorTopic => AuthorTopicEntity}

import com.twitter.ml.featurestore.catalog.entities.core.{CandidateUser => CandidateUserEntity}

import com.twitter.ml.featurestore.catalog.entities.core.{User => UserEntity}

import com.twitter.ml.featurestore.lib.EdgeEntityId

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

import com.twitter.ml.featurestore.lib.TopicId

import com.twitter.ml.featurestore.lib.UserId

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.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.stitch.Stitch

import com.twitter.timelines.configapi.HasParams

import java.util.concurrent.TimeUnit

import com.twitter.conversions.DurationOps.\_

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

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

class FeatureStoreGizmoduckSource @Inject() (

serviceIdentifier: ServiceIdentifier,

stats: StatsReceiver)

extends FeatureSource {

import FeatureStoreGizmoduckSource.\_

val backupSourceStats = stats.scope("feature\_store\_hydration\_gizmoduck")

val adapterStats = backupSourceStats.scope("adapters")

override def id: FeatureSourceId = FeatureSourceId.FeatureStoreGizmoduckSourceId

override def featureContext: FeatureContext = getFeatureContext

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(

UsersourceEntityDataset

).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 adapter: IRecordOneToOneAdapter[PredictionRecord] =

PredictionRecordAdapter.oneToOne(

BoundFeatureSet(allFeatures),

OnlineFeatureGenerationStats(backupSourceStats)

)

override def hydrateFeatures(

target: HasClientContext

with HasPreFetchedFeature

with HasParams

with HasSimilarToContext

with HasDisplayLocation,

candidates: Seq[CandidateUser]

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

target.getOptionalUserId

.map { targetUserId =>

val featureRequests = candidates.map { candidate =>

val userEntityId = UserEntity.withId(UserId(targetUserId))

val candidateEntityId = CandidateUserEntity.withId(UserId(candidate.id))

val similarToUserId = target.similarToUserIds.map(id => AuthorEntity.withId(UserId(id)))

val topicProof = candidate.reason.flatMap(\_.accountProof.flatMap(\_.topicProof))

val authorTopicEntity = if (topicProof.isDefined) {

backupSourceStats.counter("candidates\_with\_topic\_proof").incr()

Set(

AuthorTopicEntity.withId(

EdgeEntityId(UserId(candidate.id), TopicId(topicProof.get.topicId))))

} else Nil

val entities =

Seq(userEntityId, candidateEntityId) ++ similarToUserId ++ authorTopicEntity

FeatureStoreRequest(entities)

}

val predictionRecordsFut = dynamicFeatureStoreClient(featureRequests, target)

val candidateFeatureMap = predictionRecordsFut.map { predictionRecords =>

// we can zip predictionRecords with candidates as the order is preserved in the client

candidates

.zip(predictionRecords).map {

case (candidate, predictionRecord) =>

candidate -> adaptAdditionalFeaturesToDataRecord(

adapter.adaptToDataRecord(predictionRecord),

adapterStats,

FeatureStoreSource.featureAdapters)

}.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 FeatureStoreGizmoduckSource {

private val DatasetCacheScope = "feature\_store\_local\_cache\_gizmoduck"

private val DefaultCacheMaxKeys = 20000

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

FeatureStoreFeatures.candidateUserStatusFeatures ++

FeatureStoreFeatures.similarToUserStatusFeatures ++

FeatureStoreFeatures.targetUserStatusFeatures

val getFeatureContext: FeatureContext =

BoundFeatureSet(allFeatures).toFeatureContext

val dynamicHydrationConfig: DynamicHydrationConfig[HasParams] =

DynamicHydrationConfig(

Set(

GatedFeatures(

boundFeatureSet = BoundFeatureSet(FeatureStoreFeatures.targetUserStatusFeatures),

gate = HasParams

.paramGate(FeatureStoreSourceParams.EnableSeparateClientForGizmoduck) &

HasParams.paramGate(FeatureStoreSourceParams.EnableTargetUserFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(FeatureStoreFeatures.candidateUserStatusFeatures),

gate =

HasParams

.paramGate(FeatureStoreSourceParams.EnableSeparateClientForGizmoduck) &

HasParams.paramGate(FeatureStoreSourceParams.EnableCandidateUserFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(FeatureStoreFeatures.similarToUserStatusFeatures),

gate =

HasParams

.paramGate(FeatureStoreSourceParams.EnableSeparateClientForGizmoduck) &

HasParams.paramGate(FeatureStoreSourceParams.EnableSimilarToUserFeatures)

),

))

}