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

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

import com.google.inject.Inject

import com.google.inject.Singleton

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.feature\_hydration.adapters.CandidateAlgorithmAdapter.remapCandidateSource

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.FeatureContext

import com.twitter.ml.api.IRecordOneToOneAdapter

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

import com.twitter.ml.featurestore.catalog.datasets.magicrecs.NotificationSummariesEntityDataset

import com.twitter.ml.featurestore.catalog.datasets.onboarding.MetricCenterUserCountingFeaturesDataset

import com.twitter.ml.featurestore.catalog.datasets.timelines.AuthorFeaturesEntityDataset

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.{Topic => TopicEntity}

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

import com.twitter.ml.featurestore.catalog.entities.core.{UserCandidate => UserCandidateEntity}

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

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

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

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.dataset.DatasetId

import com.twitter.ml.featurestore.lib.dynamic.\_

import com.twitter.ml.featurestore.lib.feature.\_

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.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.WtfAlgorithmId

import com.twitter.onboarding.relevance.adapters.features.featurestore.CandidateAuthorTopicAggregatesAdapter

import com.twitter.onboarding.relevance.adapters.features.featurestore.CandidateTopicEngagementRealTimeAggregatesAdapter

import com.twitter.onboarding.relevance.adapters.features.featurestore.CandidateTopicEngagementUserStateRealTimeAggregatesAdapter

import com.twitter.onboarding.relevance.adapters.features.featurestore.CandidateTopicNegativeEngagementUserStateRealTimeAggregatesAdapter

import com.twitter.onboarding.relevance.adapters.features.featurestore.FeatureStoreAdapter

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

@Singleton

class FeatureStoreSource @Inject() (

serviceIdentifier: ServiceIdentifier,

stats: StatsReceiver)

extends FeatureSource {

import FeatureStoreSource.\_

override val id: FeatureSourceId = FeatureSourceId.FeatureStoreSourceId

override val featureContext: FeatureContext = FeatureStoreSource.getFeatureContext

val hydrateFeaturesStats = stats.scope("hydrate\_features")

val adapterStats = stats.scope("adapters")

val featureSet: BoundFeatureSet = BoundFeatureSet(FeatureStoreSource.allFeatures)

val clientConfig: ClientConfig[HasParams] = ClientConfig(

dynamicHydrationConfig = FeatureStoreSource.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(

MetricCenterUserCountingFeaturesDataset,

UsersourceEntityDataset,

AuthorFeaturesEntityDataset,

NotificationSummariesEntityDataset

).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,

stats,

Set(datasetValuesCache)

)

private val adapter: IRecordOneToOneAdapter[PredictionRecord] =

PredictionRecordAdapter.oneToOne(

BoundFeatureSet(allFeatures),

OnlineFeatureGenerationStats(stats)

)

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 userId = UserId(targetUserId)

val userEntityId = UserEntity.withId(userId)

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

val userCandidateEdgeEntityId =

UserCandidateEntity.withId(EdgeEntityId(userId, UserId(candidate.id)))

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

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

val topicEntities = if (topicProof.isDefined) {

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

val topicId = topicProof.get.topicId

val topicEntityId = TopicEntity.withId(TopicId(topicId))

val authorTopicEntityId =

AuthorTopicEntity.withId(EdgeEntityId(UserId(candidate.id), TopicId(topicId)))

Seq(topicEntityId, authorTopicEntityId)

} else Nil

val candidateAlgorithmsWithScores = candidate.getAllAlgorithms

val userWtfAlgEdgeEntities =

candidateAlgorithmsWithScores.flatMap(algo => {

val algoId = AlgorithmToFeedbackTokenMap.get(remapCandidateSource(algo))

algoId.map(id =>

UserWtfAlgorithmEntity.withId(EdgeEntityId(userId, WtfAlgorithmId(id))))

})

val entities = Seq(

userEntityId,

candidateEntityId,

userCandidateEdgeEntityId) ++ similarToUserId ++ topicEntities ++ userWtfAlgEdgeEntities

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]))

}

}

// list of features that we will be fetching, even if we are only scribing but not scoring with them

object FeatureStoreSource {

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

private val DefaultCacheMaxKeys = 70000

import FeatureStoreFeatures.\_

///////////////////// ALL hydrated features /////////////////////

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

//target user

targetUserFeatures ++

targetUserUserAuthorUserStateRealTimeAggregatesFeature ++

targetUserResurrectionFeatures ++

targetUserWtfImpressionFeatures ++

targetUserStatusFeatures ++

targetUserMetricCountFeatures ++

//candidate user

candidateUserFeatures ++

candidateUserResurrectionFeatures ++

candidateUserAuthorRealTimeAggregateFeatures ++

candidateUserStatusFeatures ++

candidateUserMetricCountFeatures ++

candidateUserTimelinesAuthorAggregateFeatures ++

candidateUserClientFeatures ++

//similar to user

similarToUserFeatures ++

similarToUserStatusFeatures ++

similarToUserMetricCountFeatures ++

similarToUserTimelinesAuthorAggregateFeatures ++

//other

userCandidateEdgeFeatures ++

userCandidateWtfImpressionCandidateFeatures ++

topicFeatures ++

userWtfAlgorithmEdgeFeatures ++

targetUserClientFeatures

val dynamicHydrationConfig: DynamicHydrationConfig[HasParams] =

DynamicHydrationConfig(

Set(

GatedFeatures(

boundFeatureSet = BoundFeatureSet(topicAggregateFeatures),

gate = HasParams.paramGate(FeatureStoreSourceParams.EnableTopicAggregateFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(authorTopicFeatures),

gate =

HasParams

.paramGate(FeatureStoreSourceParams.EnableSeparateClientForTimelinesAuthors).unary\_! &

HasParams.paramGate(FeatureStoreSourceParams.EnableAuthorTopicAggregateFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(userTopicFeatures),

gate = HasParams.paramGate(FeatureStoreSourceParams.EnableUserTopicFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(targetUserFeatures),

gate = HasParams.paramGate(FeatureStoreSourceParams.EnableTargetUserFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(targetUserUserAuthorUserStateRealTimeAggregatesFeature),

gate = HasParams.paramGate(

FeatureStoreSourceParams.EnableTargetUserUserAuthorUserStateRealTimeAggregatesFeature)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(targetUserResurrectionFeatures),

gate = HasParams.paramGate(FeatureStoreSourceParams.EnableTargetUserResurrectionFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(targetUserWtfImpressionFeatures),

gate = HasParams.paramGate(FeatureStoreSourceParams.EnableTargetUserWtfImpressionFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(targetUserStatusFeatures),

gate =

HasParams.paramGate(FeatureStoreSourceParams.EnableSeparateClientForGizmoduck).unary\_! &

HasParams.paramGate(FeatureStoreSourceParams.EnableTargetUserFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(targetUserMetricCountFeatures),

gate = HasParams

.paramGate(

FeatureStoreSourceParams.EnableSeparateClientForMetricCenterUserCounting).unary\_! &

HasParams.paramGate(FeatureStoreSourceParams.EnableTargetUserFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(candidateUserFeatures),

gate = HasParams.paramGate(FeatureStoreSourceParams.EnableCandidateUserFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(candidateUserAuthorRealTimeAggregateFeatures),

gate = HasParams.paramGate(

FeatureStoreSourceParams.EnableCandidateUserAuthorRealTimeAggregateFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(candidateUserResurrectionFeatures),

gate =

HasParams.paramGate(FeatureStoreSourceParams.EnableCandidateUserResurrectionFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(candidateUserStatusFeatures),

gate =

HasParams.paramGate(FeatureStoreSourceParams.EnableSeparateClientForGizmoduck).unary\_! &

HasParams.paramGate(FeatureStoreSourceParams.EnableCandidateUserFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(candidateUserTimelinesAuthorAggregateFeatures),

gate =

HasParams

.paramGate(FeatureStoreSourceParams.EnableSeparateClientForTimelinesAuthors).unary\_! &

HasParams.paramGate(

FeatureStoreSourceParams.EnableCandidateUserTimelinesAuthorAggregateFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(candidateUserMetricCountFeatures),

gate =

HasParams

.paramGate(

FeatureStoreSourceParams.EnableSeparateClientForMetricCenterUserCounting).unary\_! &

HasParams.paramGate(FeatureStoreSourceParams.EnableCandidateUserFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(userCandidateEdgeFeatures),

gate = HasParams.paramGate(FeatureStoreSourceParams.EnableUserCandidateEdgeFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(userCandidateWtfImpressionCandidateFeatures),

gate = HasParams.paramGate(

FeatureStoreSourceParams.EnableUserCandidateWtfImpressionCandidateFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(userWtfAlgorithmEdgeFeatures),

gate = HasParams.paramGate(FeatureStoreSourceParams.EnableUserWtfAlgEdgeFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(similarToUserFeatures),

gate = HasParams.paramGate(FeatureStoreSourceParams.EnableSimilarToUserFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(similarToUserStatusFeatures),

gate =

HasParams.paramGate(FeatureStoreSourceParams.EnableSeparateClientForGizmoduck).unary\_! &

HasParams.paramGate(FeatureStoreSourceParams.EnableSimilarToUserFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(similarToUserTimelinesAuthorAggregateFeatures),

gate =

HasParams

.paramGate(FeatureStoreSourceParams.EnableSeparateClientForTimelinesAuthors).unary\_! &

HasParams.paramGate(FeatureStoreSourceParams.EnableSimilarToUserFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(similarToUserMetricCountFeatures),

gate =

HasParams

.paramGate(

FeatureStoreSourceParams.EnableSeparateClientForMetricCenterUserCounting).unary\_! &

HasParams.paramGate(FeatureStoreSourceParams.EnableSimilarToUserFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(candidateUserClientFeatures),

gate = HasParams.paramGate(FeatureStoreSourceParams.EnableCandidateClientFeatures)

),

GatedFeatures(

boundFeatureSet = BoundFeatureSet(targetUserClientFeatures),

gate = HasParams.paramGate(FeatureStoreSourceParams.EnableUserClientFeatures)

),

)

)

// for calibrating features, e.g. add log transformed topic features

val featureAdapters: Seq[FeatureStoreAdapter] = Seq(

CandidateTopicEngagementRealTimeAggregatesAdapter,

CandidateTopicNegativeEngagementUserStateRealTimeAggregatesAdapter,

CandidateTopicEngagementUserStateRealTimeAggregatesAdapter,

CandidateAuthorTopicAggregatesAdapter

)

val additionalFeatureContext: FeatureContext = FeatureContext.merge(

featureAdapters

.foldRight(new FeatureContext())((adapter, context) =>

context

.addFeatures(adapter.getFeatureContext))

)

val getFeatureContext: FeatureContext =

BoundFeatureSet(allFeatures).toFeatureContext

.addFeatures(additionalFeatureContext)

// The below are aggregated features that are aggregated for a second time over multiple keys.

.addFeatures(maxSumAvgAggregatedFeatureContext)

}