package com.twitter.cr\_mixer.module

import com.google.inject.Provides

import com.google.inject.Singleton

import com.twitter.bijection.thrift.CompactThriftCodec

import com.twitter.ads.entities.db.thriftscala.LineItemObjective

import com.twitter.bijection.Injection

import com.twitter.conversions.DurationOps.\_

import com.twitter.cr\_mixer.model.ModuleNames

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

import com.twitter.finagle.memcached.{Client => MemcachedClient}

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.hermit.store.common.ObservedCachedReadableStore

import com.twitter.hermit.store.common.ObservedMemcachedReadableStore

import com.twitter.inject.TwitterModule

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.DataType

import com.twitter.ml.api.Feature

import com.twitter.ml.api.GeneralTensor

import com.twitter.ml.api.RichDataRecord

import com.twitter.relevance\_platform.common.injection.LZ4Injection

import com.twitter.relevance\_platform.common.injection.SeqObjectInjection

import com.twitter.simclusters\_v2.common.TweetId

import com.twitter.storage.client.manhattan.kv.ManhattanKVClientMtlsParams

import com.twitter.storehaus.ReadableStore

import com.twitter.storehaus\_internal.manhattan.ManhattanRO

import com.twitter.storehaus\_internal.manhattan.ManhattanROConfig

import com.twitter.storehaus\_internal.manhattan.Revenue

import com.twitter.storehaus\_internal.util.ApplicationID

import com.twitter.storehaus\_internal.util.DatasetName

import com.twitter.storehaus\_internal.util.HDFSPath

import com.twitter.util.Future

import javax.inject.Named

import scala.collection.JavaConverters.\_

object ActivePromotedTweetStoreModule extends TwitterModule {

case class ActivePromotedTweetStore(

activePromotedTweetMHStore: ReadableStore[String, DataRecord],

statsReceiver: StatsReceiver)

extends ReadableStore[TweetId, Seq[LineItemInfo]] {

override def get(tweetId: TweetId): Future[Option[Seq[LineItemInfo]]] = {

activePromotedTweetMHStore.get(tweetId.toString).map {

\_.map { dataRecord =>

val richDataRecord = new RichDataRecord(dataRecord)

val lineItemIdsFeature: Feature[GeneralTensor] =

new Feature.Tensor("active\_promoted\_tweets.line\_item\_ids", DataType.INT64)

val lineItemObjectivesFeature: Feature[GeneralTensor] =

new Feature.Tensor("active\_promoted\_tweets.line\_item\_objectives", DataType.INT64)

val lineItemIdsTensor: GeneralTensor = richDataRecord.getFeatureValue(lineItemIdsFeature)

val lineItemObjectivesTensor: GeneralTensor =

richDataRecord.getFeatureValue(lineItemObjectivesFeature)

val lineItemIds: Seq[Long] =

if (lineItemIdsTensor.getSetField == GeneralTensor.\_Fields.INT64\_TENSOR && lineItemIdsTensor.getInt64Tensor.isSetLongs) {

lineItemIdsTensor.getInt64Tensor.getLongs.asScala.map(\_.toLong)

} else Seq.empty

val lineItemObjectives: Seq[LineItemObjective] =

if (lineItemObjectivesTensor.getSetField == GeneralTensor.\_Fields.INT64\_TENSOR && lineItemObjectivesTensor.getInt64Tensor.isSetLongs) {

lineItemObjectivesTensor.getInt64Tensor.getLongs.asScala.map(objective =>

LineItemObjective(objective.toInt))

} else Seq.empty

val lineItemInfo =

if (lineItemIds.size == lineItemObjectives.size) {

lineItemIds.zipWithIndex.map {

case (lineItemId, index) =>

LineItemInfo(

lineItemId = lineItemId,

lineItemObjective = lineItemObjectives(index)

)

}

} else Seq.empty

lineItemInfo

}

}

}

}

@Provides

@Singleton

def providesActivePromotedTweetStore(

manhattanKVClientMtlsParams: ManhattanKVClientMtlsParams,

@Named(ModuleNames.UnifiedCache) crMixerUnifiedCacheClient: MemcachedClient,

crMixerStatsReceiver: StatsReceiver

): ReadableStore[TweetId, Seq[LineItemInfo]] = {

val mhConfig = new ManhattanROConfig {

val hdfsPath = HDFSPath("")

val applicationID = ApplicationID("ads\_bigquery\_features")

val datasetName = DatasetName("active\_promoted\_tweets")

val cluster = Revenue

override def statsReceiver: StatsReceiver =

crMixerStatsReceiver.scope("active\_promoted\_tweets\_mh")

}

val mhStore: ReadableStore[String, DataRecord] =

ManhattanRO

.getReadableStoreWithMtls[String, DataRecord](

mhConfig,

manhattanKVClientMtlsParams

)(

implicitly[Injection[String, Array[Byte]]],

CompactThriftCodec[DataRecord]

)

val underlyingStore =

ActivePromotedTweetStore(mhStore, crMixerStatsReceiver.scope("ActivePromotedTweetStore"))

val memcachedStore = ObservedMemcachedReadableStore.fromCacheClient(

backingStore = underlyingStore,

cacheClient = crMixerUnifiedCacheClient,

ttl = 60.minutes,

asyncUpdate = false

)(

valueInjection = LZ4Injection.compose(SeqObjectInjection[LineItemInfo]()),

statsReceiver = crMixerStatsReceiver.scope("memCachedActivePromotedTweetStore"),

keyToString = { k: TweetId => s"apt/$k" }

)

ObservedCachedReadableStore.from(

memcachedStore,

ttl = 30.minutes,

maxKeys = 250000, // size of promoted tweet is around 200,000

windowSize = 10000L,

cacheName = "active\_promoted\_tweet\_cache",

maxMultiGetSize = 20

)(crMixerStatsReceiver.scope("inMemoryCachedActivePromotedTweetStore"))

}

}