package com.twitter.representation\_manager.common

import com.twitter.bijection.scrooge.BinaryScalaCodec

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.memcached.Client

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.hashing.KeyHasher

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

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

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

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

import com.twitter.simclusters\_v2.thriftscala.EmbeddingType

import com.twitter.simclusters\_v2.thriftscala.EmbeddingType.\_

import com.twitter.simclusters\_v2.thriftscala.ModelVersion

import com.twitter.simclusters\_v2.thriftscala.ModelVersion.\_

import com.twitter.simclusters\_v2.thriftscala.SimClustersEmbeddingId

import com.twitter.simclusters\_v2.thriftscala.{SimClustersEmbedding => ThriftSimClustersEmbedding}

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Duration

/\*

\* NOTE - ALL the cache configs here are just placeholders, NONE of them is used anyweher in RMS yet

\* \*/

sealed trait MemCacheParams

sealed trait MemCacheConfig

/\*

\* This holds params that is required to set up a memcache cache for a single embedding store

\* \*/

case class EnabledMemCacheParams(ttl: Duration) extends MemCacheParams

object DisabledMemCacheParams extends MemCacheParams

/\*

\* We use this MemcacheConfig as the single source to set up the memcache for all RMS use cases

\* NO OVERRIDE FROM CLIENT

\* \*/

object MemCacheConfig {

val keyHasher: KeyHasher = KeyHasher.FNV1A\_64

val hashKeyPrefix: String = "RMS"

val simclustersEmbeddingCacheKeyBuilder =

SimClustersEmbeddingIdCacheKeyBuilder(keyHasher.hashKey, hashKeyPrefix)

val cacheParamsMap: Map[

(EmbeddingType, ModelVersion),

MemCacheParams

] = Map(

// Tweet Embeddings

(LogFavBasedTweet, Model20m145kUpdated) -> EnabledMemCacheParams(ttl = 10.minutes),

(LogFavBasedTweet, Model20m145k2020) -> EnabledMemCacheParams(ttl = 10.minutes),

(LogFavLongestL2EmbeddingTweet, Model20m145kUpdated) -> EnabledMemCacheParams(ttl = 10.minutes),

(LogFavLongestL2EmbeddingTweet, Model20m145k2020) -> EnabledMemCacheParams(ttl = 10.minutes),

// User - KnownFor Embeddings

(FavBasedProducer, Model20m145kUpdated) -> EnabledMemCacheParams(ttl = 12.hours),

(FavBasedProducer, Model20m145k2020) -> EnabledMemCacheParams(ttl = 12.hours),

(FollowBasedProducer, Model20m145k2020) -> EnabledMemCacheParams(ttl = 12.hours),

(AggregatableLogFavBasedProducer, Model20m145k2020) -> EnabledMemCacheParams(ttl = 12.hours),

(RelaxedAggregatableLogFavBasedProducer, Model20m145kUpdated) -> EnabledMemCacheParams(ttl =

12.hours),

(RelaxedAggregatableLogFavBasedProducer, Model20m145k2020) -> EnabledMemCacheParams(ttl =

12.hours),

// User - InterestedIn Embeddings

(LogFavBasedUserInterestedInFromAPE, Model20m145k2020) -> EnabledMemCacheParams(ttl = 12.hours),

(FollowBasedUserInterestedInFromAPE, Model20m145k2020) -> EnabledMemCacheParams(ttl = 12.hours),

(FavBasedUserInterestedIn, Model20m145kUpdated) -> EnabledMemCacheParams(ttl = 12.hours),

(FavBasedUserInterestedIn, Model20m145k2020) -> EnabledMemCacheParams(ttl = 12.hours),

(FollowBasedUserInterestedIn, Model20m145k2020) -> EnabledMemCacheParams(ttl = 12.hours),

(LogFavBasedUserInterestedIn, Model20m145k2020) -> EnabledMemCacheParams(ttl = 12.hours),

(FavBasedUserInterestedInFromPE, Model20m145kUpdated) -> EnabledMemCacheParams(ttl = 12.hours),

(FilteredUserInterestedIn, Model20m145kUpdated) -> EnabledMemCacheParams(ttl = 12.hours),

(FilteredUserInterestedIn, Model20m145k2020) -> EnabledMemCacheParams(ttl = 12.hours),

(FilteredUserInterestedInFromPE, Model20m145kUpdated) -> EnabledMemCacheParams(ttl = 12.hours),

(UnfilteredUserInterestedIn, Model20m145kUpdated) -> EnabledMemCacheParams(ttl = 12.hours),

(UnfilteredUserInterestedIn, Model20m145k2020) -> EnabledMemCacheParams(ttl = 12.hours),

(UserNextInterestedIn, Model20m145k2020) -> EnabledMemCacheParams(ttl =

30.minutes), //embedding is updated every 2 hours, keeping it lower to avoid staleness

(

LogFavBasedUserInterestedMaxpoolingAddressBookFromIIAPE,

Model20m145k2020) -> EnabledMemCacheParams(ttl = 12.hours),

(

LogFavBasedUserInterestedAverageAddressBookFromIIAPE,

Model20m145k2020) -> EnabledMemCacheParams(ttl = 12.hours),

(

LogFavBasedUserInterestedBooktypeMaxpoolingAddressBookFromIIAPE,

Model20m145k2020) -> EnabledMemCacheParams(ttl = 12.hours),

(

LogFavBasedUserInterestedLargestDimMaxpoolingAddressBookFromIIAPE,

Model20m145k2020) -> EnabledMemCacheParams(ttl = 12.hours),

(

LogFavBasedUserInterestedLouvainMaxpoolingAddressBookFromIIAPE,

Model20m145k2020) -> EnabledMemCacheParams(ttl = 12.hours),

(

LogFavBasedUserInterestedConnectedMaxpoolingAddressBookFromIIAPE,

Model20m145k2020) -> EnabledMemCacheParams(ttl = 12.hours),

// Topic Embeddings

(FavTfgTopic, Model20m145k2020) -> EnabledMemCacheParams(ttl = 12.hours),

(LogFavBasedKgoApeTopic, Model20m145k2020) -> EnabledMemCacheParams(ttl = 12.hours),

)

def getCacheSetup(

embeddingType: EmbeddingType,

modelVersion: ModelVersion

): MemCacheParams = {

// When requested (embeddingType, modelVersion) doesn't exist, we return DisabledMemCacheParams

cacheParamsMap.getOrElse((embeddingType, modelVersion), DisabledMemCacheParams)

}

def getCacheKeyPrefix(embeddingType: EmbeddingType, modelVersion: ModelVersion) =

s"${embeddingType.value}\_${modelVersion.value}\_"

def getStatsName(embeddingType: EmbeddingType, modelVersion: ModelVersion) =

s"${embeddingType.name}\_${modelVersion.name}\_mem\_cache"

/\*\*

\* Build a ReadableStore based on MemCacheConfig.

\*

\* If memcache is disabled, it will return a normal readable store wrapper of the rawStore,

\* with SimClustersEmbedding as value;

\* If memcache is enabled, it will return a ObservedMemcachedReadableStore wrapper of the rawStore,

\* with memcache set up according to the EnabledMemCacheParams

\* \*/

def buildMemCacheStoreForSimClustersEmbedding(

rawStore: ReadableStore[SimClustersEmbeddingId, ThriftSimClustersEmbedding],

cacheClient: Client,

embeddingType: EmbeddingType,

modelVersion: ModelVersion,

stats: StatsReceiver

): ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding] = {

val cacheParams = getCacheSetup(embeddingType, modelVersion)

val store = cacheParams match {

case DisabledMemCacheParams => rawStore

case EnabledMemCacheParams(ttl) =>

val memCacheKeyPrefix = MemCacheConfig.getCacheKeyPrefix(

embeddingType,

modelVersion

)

val statsName = MemCacheConfig.getStatsName(

embeddingType,

modelVersion

)

ObservedMemcachedReadableStore.fromCacheClient(

backingStore = rawStore,

cacheClient = cacheClient,

ttl = ttl

)(

valueInjection = LZ4Injection.compose(BinaryScalaCodec(ThriftSimClustersEmbedding)),

statsReceiver = stats.scope(statsName),

keyToString = { k => memCacheKeyPrefix + k.toString }

)

}

store.mapValues(SimClustersEmbedding(\_))

}

}