package com.twitter.representation\_manager

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

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.store.strato.StratoFetchableStore

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

import com.twitter.hermit.store.common.ObservedReadableStore

import com.twitter.representation\_manager.config.ClientConfig

import com.twitter.representation\_manager.config.DisabledInMemoryCacheParams

import com.twitter.representation\_manager.config.EnabledInMemoryCacheParams

import com.twitter.representation\_manager.thriftscala.SimClustersEmbeddingView

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

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

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

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

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

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

import com.twitter.storehaus.ReadableStore

import com.twitter.strato.client.{Client => StratoClient}

import com.twitter.strato.thrift.ScroogeConvImplicits.\_

/\*\*

\* This is the class that offers features to build readable stores for a given

\* SimClustersEmbeddingView (i.e. embeddingType and modelVersion). It applies ClientConfig

\* for a particular service and build ReadableStores which implement that config.

\*/

class StoreBuilder(

clientConfig: ClientConfig,

stratoClient: StratoClient,

memCachedClient: MemcachedClient,

globalStats: StatsReceiver,

) {

private val stats =

globalStats.scope("representation\_manager\_client").scope(this.getClass.getSimpleName)

// Column consts

private val ColPathPrefix = "recommendations/representation\_manager/"

private val SimclustersTweetColPath = ColPathPrefix + "simClustersEmbedding.Tweet"

private val SimclustersUserColPath = ColPathPrefix + "simClustersEmbedding.User"

private val SimclustersTopicIdColPath = ColPathPrefix + "simClustersEmbedding.TopicId"

private val SimclustersLocaleEntityIdColPath =

ColPathPrefix + "simClustersEmbedding.LocaleEntityId"

def buildSimclustersTweetEmbeddingStore(

embeddingColumnView: SimClustersEmbeddingView

): ReadableStore[Long, SimClustersEmbedding] = {

val rawStore = StratoFetchableStore

.withView[Long, SimClustersEmbeddingView, ThriftSimClustersEmbedding](

stratoClient,

SimclustersTweetColPath,

embeddingColumnView)

.mapValues(SimClustersEmbedding(\_))

addCacheLayer(rawStore, embeddingColumnView)

}

def buildSimclustersUserEmbeddingStore(

embeddingColumnView: SimClustersEmbeddingView

): ReadableStore[Long, SimClustersEmbedding] = {

val rawStore = StratoFetchableStore

.withView[Long, SimClustersEmbeddingView, ThriftSimClustersEmbedding](

stratoClient,

SimclustersUserColPath,

embeddingColumnView)

.mapValues(SimClustersEmbedding(\_))

addCacheLayer(rawStore, embeddingColumnView)

}

def buildSimclustersTopicIdEmbeddingStore(

embeddingColumnView: SimClustersEmbeddingView

): ReadableStore[TopicId, SimClustersEmbedding] = {

val rawStore = StratoFetchableStore

.withView[TopicId, SimClustersEmbeddingView, ThriftSimClustersEmbedding](

stratoClient,

SimclustersTopicIdColPath,

embeddingColumnView)

.mapValues(SimClustersEmbedding(\_))

addCacheLayer(rawStore, embeddingColumnView)

}

def buildSimclustersLocaleEntityIdEmbeddingStore(

embeddingColumnView: SimClustersEmbeddingView

): ReadableStore[LocaleEntityId, SimClustersEmbedding] = {

val rawStore = StratoFetchableStore

.withView[LocaleEntityId, SimClustersEmbeddingView, ThriftSimClustersEmbedding](

stratoClient,

SimclustersLocaleEntityIdColPath,

embeddingColumnView)

.mapValues(SimClustersEmbedding(\_))

addCacheLayer(rawStore, embeddingColumnView)

}

def buildSimclustersTweetEmbeddingStoreWithEmbeddingIdAsKey(

embeddingColumnView: SimClustersEmbeddingView

): ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding] = {

val rawStore = StratoFetchableStore

.withView[Long, SimClustersEmbeddingView, ThriftSimClustersEmbedding](

stratoClient,

SimclustersTweetColPath,

embeddingColumnView)

.mapValues(SimClustersEmbedding(\_))

val embeddingIdAsKeyStore = rawStore.composeKeyMapping[SimClustersEmbeddingId] {

case SimClustersEmbeddingId(\_, \_, InternalId.TweetId(tweetId)) =>

tweetId

}

addCacheLayer(embeddingIdAsKeyStore, embeddingColumnView)

}

def buildSimclustersUserEmbeddingStoreWithEmbeddingIdAsKey(

embeddingColumnView: SimClustersEmbeddingView

): ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding] = {

val rawStore = StratoFetchableStore

.withView[Long, SimClustersEmbeddingView, ThriftSimClustersEmbedding](

stratoClient,

SimclustersUserColPath,

embeddingColumnView)

.mapValues(SimClustersEmbedding(\_))

val embeddingIdAsKeyStore = rawStore.composeKeyMapping[SimClustersEmbeddingId] {

case SimClustersEmbeddingId(\_, \_, InternalId.UserId(userId)) =>

userId

}

addCacheLayer(embeddingIdAsKeyStore, embeddingColumnView)

}

def buildSimclustersTopicEmbeddingStoreWithEmbeddingIdAsKey(

embeddingColumnView: SimClustersEmbeddingView

): ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding] = {

val rawStore = StratoFetchableStore

.withView[TopicId, SimClustersEmbeddingView, ThriftSimClustersEmbedding](

stratoClient,

SimclustersTopicIdColPath,

embeddingColumnView)

.mapValues(SimClustersEmbedding(\_))

val embeddingIdAsKeyStore = rawStore.composeKeyMapping[SimClustersEmbeddingId] {

case SimClustersEmbeddingId(\_, \_, InternalId.TopicId(topicId)) =>

topicId

}

addCacheLayer(embeddingIdAsKeyStore, embeddingColumnView)

}

def buildSimclustersTopicIdEmbeddingStoreWithEmbeddingIdAsKey(

embeddingColumnView: SimClustersEmbeddingView

): ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding] = {

val rawStore = StratoFetchableStore

.withView[TopicId, SimClustersEmbeddingView, ThriftSimClustersEmbedding](

stratoClient,

SimclustersTopicIdColPath,

embeddingColumnView)

.mapValues(SimClustersEmbedding(\_))

val embeddingIdAsKeyStore = rawStore.composeKeyMapping[SimClustersEmbeddingId] {

case SimClustersEmbeddingId(\_, \_, InternalId.TopicId(topicId)) =>

topicId

}

addCacheLayer(embeddingIdAsKeyStore, embeddingColumnView)

}

def buildSimclustersLocaleEntityIdEmbeddingStoreWithEmbeddingIdAsKey(

embeddingColumnView: SimClustersEmbeddingView

): ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding] = {

val rawStore = StratoFetchableStore

.withView[LocaleEntityId, SimClustersEmbeddingView, ThriftSimClustersEmbedding](

stratoClient,

SimclustersLocaleEntityIdColPath,

embeddingColumnView)

.mapValues(SimClustersEmbedding(\_))

val embeddingIdAsKeyStore = rawStore.composeKeyMapping[SimClustersEmbeddingId] {

case SimClustersEmbeddingId(\_, \_, InternalId.LocaleEntityId(localeEntityId)) =>

localeEntityId

}

addCacheLayer(embeddingIdAsKeyStore, embeddingColumnView)

}

private def addCacheLayer[K](

rawStore: ReadableStore[K, SimClustersEmbedding],

embeddingColumnView: SimClustersEmbeddingView,

): ReadableStore[K, SimClustersEmbedding] = {

// Add in-memory caching based on ClientConfig

val inMemCacheParams = clientConfig.inMemoryCacheConfig

.getCacheSetup(embeddingColumnView.embeddingType, embeddingColumnView.modelVersion)

val statsPerStore = stats

.scope(embeddingColumnView.embeddingType.name).scope(embeddingColumnView.modelVersion.name)

inMemCacheParams match {

case DisabledInMemoryCacheParams =>

ObservedReadableStore(

store = rawStore

)(statsPerStore)

case EnabledInMemoryCacheParams(ttl, maxKeys, cacheName) =>

ObservedCachedReadableStore.from[K, SimClustersEmbedding](

rawStore,

ttl = ttl,

maxKeys = maxKeys,

cacheName = cacheName,

windowSize = 10000L

)(statsPerStore)

}

}

}