package com.twitter.representation\_manager.store

import com.twitter.contentrecommender.store.ApeEntityEmbeddingStore

import com.twitter.contentrecommender.store.InterestsOptOutStore

import com.twitter.contentrecommender.store.SemanticCoreTopicSeedStore

import com.twitter.conversions.DurationOps.\_

import com.twitter.escherbird.util.uttclient.CachedUttClientV2

import com.twitter.finagle.memcached.Client

import com.twitter.finagle.stats.StatsReceiver

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

import com.twitter.frigate.common.util.SeqLongInjection

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

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

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

import com.twitter.interests.thriftscala.InterestsThriftService

import com.twitter.representation\_manager.common.MemCacheConfig

import com.twitter.representation\_manager.common.RepresentationManagerDecider

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

import com.twitter.simclusters\_v2.stores.SimClustersEmbeddingStore

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

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

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

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.TopicId

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

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

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

import com.twitter.storehaus.ReadableStore

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

import com.twitter.tweetypie.util.UserId

import javax.inject.Inject

class TopicSimClustersEmbeddingStore @Inject() (

stratoClient: StratoClient,

cacheClient: Client,

globalStats: StatsReceiver,

mhMtlsParams: ManhattanKVClientMtlsParams,

rmsDecider: RepresentationManagerDecider,

interestService: InterestsThriftService.MethodPerEndpoint,

uttClient: CachedUttClientV2) {

private val stats = globalStats.scope(this.getClass.getSimpleName)

private val interestsOptOutStore = InterestsOptOutStore(interestService)

/\*\*

\* Note this is NOT an embedding store. It is a list of author account ids we use to represent

\* topics

\*/

private val semanticCoreTopicSeedStore: ReadableStore[

SemanticCoreTopicSeedStore.Key,

Seq[UserId]

] = {

/\*

Up to 1000 Long seeds per topic/language = 62.5kb per topic/language (worst case)

Assume ~10k active topic/languages ~= 650MB (worst case)

\*/

val underlying = new SemanticCoreTopicSeedStore(uttClient, interestsOptOutStore)(

stats.scope("semantic\_core\_topic\_seed\_store"))

val memcacheStore = ObservedMemcachedReadableStore.fromCacheClient(

backingStore = underlying,

cacheClient = cacheClient,

ttl = 12.hours)(

valueInjection = SeqLongInjection,

statsReceiver = stats.scope("topic\_producer\_seed\_store\_mem\_cache"),

keyToString = { k => s"tpss:${k.entityId}\_${k.languageCode}" }

)

ObservedCachedReadableStore.from[SemanticCoreTopicSeedStore.Key, Seq[UserId]](

store = memcacheStore,

ttl = 6.hours,

maxKeys = 20e3.toInt,

cacheName = "topic\_producer\_seed\_store\_cache",

windowSize = 5000

)(stats.scope("topic\_producer\_seed\_store\_cache"))

}

private val favBasedTfgTopicEmbedding20m145k2020Store: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

val rawStore =

StratoFetchableStore

.withUnitView[SimClustersEmbeddingId, ThriftSimClustersEmbedding](

stratoClient,

"recommendations/simclusters\_v2/embeddings/favBasedTFGTopic20M145K2020").mapValues(

embedding => SimClustersEmbedding(embedding, truncate = 50).toThrift)

.composeKeyMapping[LocaleEntityId] { localeEntityId =>

SimClustersEmbeddingId(

FavTfgTopic,

Model20m145k2020,

InternalId.LocaleEntityId(localeEntityId))

}

buildLocaleEntityIdMemCacheStore(rawStore, FavTfgTopic, Model20m145k2020)

}

private val logFavBasedApeEntity20M145K2020EmbeddingStore: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

val apeStore = StratoFetchableStore

.withUnitView[SimClustersEmbeddingId, ThriftSimClustersEmbedding](

stratoClient,

"recommendations/simclusters\_v2/embeddings/logFavBasedAPE20M145K2020")

.mapValues(embedding => SimClustersEmbedding(embedding, truncate = 50))

.composeKeyMapping[UserId]({ id =>

SimClustersEmbeddingId(

AggregatableLogFavBasedProducer,

Model20m145k2020,

InternalId.UserId(id))

})

val rawStore = new ApeEntityEmbeddingStore(

semanticCoreSeedStore = semanticCoreTopicSeedStore,

aggregatableProducerEmbeddingStore = apeStore,

statsReceiver = stats.scope("log\_fav\_based\_ape\_entity\_2020\_embedding\_store"))

.mapValues(embedding => SimClustersEmbedding(embedding.toThrift, truncate = 50).toThrift)

.composeKeyMapping[TopicId] { topicId =>

SimClustersEmbeddingId(

LogFavBasedKgoApeTopic,

Model20m145k2020,

InternalId.TopicId(topicId))

}

buildTopicIdMemCacheStore(rawStore, LogFavBasedKgoApeTopic, Model20m145k2020)

}

private def buildTopicIdMemCacheStore(

rawStore: ReadableStore[TopicId, ThriftSimClustersEmbedding],

embeddingType: EmbeddingType,

modelVersion: ModelVersion

): ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding] = {

val observedStore: ObservedReadableStore[TopicId, ThriftSimClustersEmbedding] =

ObservedReadableStore(

store = rawStore

)(stats.scope(embeddingType.name).scope(modelVersion.name))

val storeWithKeyMapping = observedStore.composeKeyMapping[SimClustersEmbeddingId] {

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

topicId

}

MemCacheConfig.buildMemCacheStoreForSimClustersEmbedding(

storeWithKeyMapping,

cacheClient,

embeddingType,

modelVersion,

stats

)

}

private def buildLocaleEntityIdMemCacheStore(

rawStore: ReadableStore[LocaleEntityId, ThriftSimClustersEmbedding],

embeddingType: EmbeddingType,

modelVersion: ModelVersion

): ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding] = {

val observedStore: ObservedReadableStore[LocaleEntityId, ThriftSimClustersEmbedding] =

ObservedReadableStore(

store = rawStore

)(stats.scope(embeddingType.name).scope(modelVersion.name))

val storeWithKeyMapping = observedStore.composeKeyMapping[SimClustersEmbeddingId] {

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

localeEntityId

}

MemCacheConfig.buildMemCacheStoreForSimClustersEmbedding(

storeWithKeyMapping,

cacheClient,

embeddingType,

modelVersion,

stats

)

}

private val underlyingStores: Map[

(EmbeddingType, ModelVersion),

ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding]

] = Map(

// Topic Embeddings

(FavTfgTopic, Model20m145k2020) -> favBasedTfgTopicEmbedding20m145k2020Store,

(LogFavBasedKgoApeTopic, Model20m145k2020) -> logFavBasedApeEntity20M145K2020EmbeddingStore,

)

val topicSimClustersEmbeddingStore: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

SimClustersEmbeddingStore.buildWithDecider(

underlyingStores = underlyingStores,

decider = rmsDecider.decider,

statsReceiver = stats

)

}

}