package com.twitter.simclusters\_v2.summingbird.stores

import com.twitter.finagle.mtls.authentication.ServiceIdentifier

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

import com.twitter.simclusters\_v2.summingbird.common.Implicits.batcher

import com.twitter.simclusters\_v2.summingbird.common.Implicits.topKClustersWithScoresCodec

import com.twitter.simclusters\_v2.summingbird.common.Implicits.topKClustersWithScoresMonoid

import com.twitter.simclusters\_v2.summingbird.common.SimClustersProfile.Environment

import com.twitter.simclusters\_v2.summingbird.common.ClientConfigs

import com.twitter.simclusters\_v2.summingbird.common.Configs

import com.twitter.simclusters\_v2.summingbird.common.Implicits

import com.twitter.simclusters\_v2.summingbird.common.SimClustersProfile

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

import com.twitter.storehaus.ReadableStore

import com.twitter.storehaus.algebra.MergeableStore

import com.twitter.storehaus\_internal.memcache.Memcache

import com.twitter.summingbird.batch.BatchID

import com.twitter.summingbird.store.ClientStore

import com.twitter.summingbird\_internal.bijection.BatchPairImplicits

import com.twitter.util.Duration

import com.twitter.util.Future

object TopKClustersForTweetReadableStore {

private[summingbird] final lazy val onlineMergeableStore: (

String,

ServiceIdentifier

) => MergeableStore[(EntityWithVersion, BatchID), TopKClustersWithScores] = {

(storePath: String, serviceIdentifier: ServiceIdentifier) =>

Memcache.getMemcacheStore[(EntityWithVersion, BatchID), TopKClustersWithScores](

ClientConfigs.tweetTopKClustersMemcacheConfig(storePath, serviceIdentifier)

)(

BatchPairImplicits.keyInjection[EntityWithVersion](Implicits.topKClustersKeyCodec),

topKClustersWithScoresCodec,

topKClustersWithScoresMonoid

)

}

final lazy val defaultStore: (

String,

ServiceIdentifier

) => ReadableStore[EntityWithVersion, TopKClustersWithScores] = {

(storePath: String, serviceIdentifier: ServiceIdentifier) =>

// note that DefaultTopKClustersForEntityReadableStore is reused here because they share the

// same structure

TopKClustersForEntityReadableStore(

ClientStore(this.onlineMergeableStore(storePath, serviceIdentifier), Configs.batchesToKeep))

}

}

case class TweetKey(

tweetId: Long,

modelVersion: String,

embeddingType: EmbeddingType = EmbeddingType.FavBasedTweet,

halfLife: Duration = Configs.HalfLife) {

lazy val modelVersionThrift: ModelVersion = ModelVersions.toModelVersion(modelVersion)

lazy val simClustersEmbeddingId: SimClustersEmbeddingId =

SimClustersEmbeddingId(embeddingType, modelVersionThrift, InternalId.TweetId(tweetId))

}

object TweetKey {

def apply(simClustersEmbeddingId: SimClustersEmbeddingId): TweetKey = {

simClustersEmbeddingId match {

case SimClustersEmbeddingId(embeddingType, modelVersion, InternalId.TweetId(tweetId)) =>

TweetKey(tweetId, ModelVersions.toKnownForModelVersion(modelVersion), embeddingType)

case id =>

throw new IllegalArgumentException(s"Invalid $id for TweetKey")

}

}

}

case class TopKClustersForTweetKeyReadableStore(

proxyMap: Map[(EmbeddingType, String), ReadableStore[EntityWithVersion, TopKClustersWithScores]],

halfLifeDuration: Duration,

topKClustersWithScoresToSeq: TopKClustersWithScores => Seq[(Int, Double)],

maxResult: Option[Int] = None)

extends ReadableStore[TweetKey, Seq[(Int, Double)]] {

private val modifiedProxyMap = proxyMap.map {

case ((embeddingType, modelVersion), proxy) =>

(embeddingType, modelVersion) -> proxy.composeKeyMapping { key: TweetKey =>

EntityWithVersion(

SimClusterEntity.TweetId(key.tweetId),

// Fast fail if the model version is invalid.

ModelVersions.toModelVersion(modelVersion))

}

}

override def multiGet[K1 <: TweetKey](

keys: Set[K1]

): Map[K1, Future[Option[Seq[(Int, Double)]]]] = {

val (validKeys, invalidKeys) = keys.partition { tweetKey =>

proxyMap.contains((tweetKey.embeddingType, tweetKey.modelVersion)) &&

halfLifeDuration.inMilliseconds == Configs.HalfLifeInMs

}

val resultsFuture = validKeys.groupBy(key => (key.embeddingType, key.modelVersion)).flatMap {

case (typeModelTuple, subKeys) =>

modifiedProxyMap(typeModelTuple).multiGet(subKeys)

}

resultsFuture.mapValues { topKClustersWithScoresFut =>

for (topKClustersWithScoresOpt <- topKClustersWithScoresFut) yield {

for {

topKClustersWithScores <- topKClustersWithScoresOpt

} yield {

val results = topKClustersWithScoresToSeq(topKClustersWithScores)

maxResult match {

case Some(max) =>

results.take(max)

case None =>

results

}

}

}

} ++ invalidKeys.map { key => (key, Future.None) }.toMap

}

}

object TopKClustersForTweetKeyReadableStore {

// Use Prod cache by default

def defaultProxyMap(

serviceIdentifier: ServiceIdentifier

): Map[(EmbeddingType, String), ReadableStore[EntityWithVersion, TopKClustersWithScores]] =

SimClustersProfile.tweetJobProfileMap(Environment.Prod).mapValues { profile =>

TopKClustersForTweetReadableStore

.defaultStore(profile.clusterTopKTweetsPath, serviceIdentifier)

}

val defaultHalfLife: Duration = Duration.fromMilliseconds(Configs.HalfLifeInMs)

def defaultStore(

serviceIdentifier: ServiceIdentifier

): ReadableStore[TweetKey, Seq[(Int, Double)]] =

TopKClustersForTweetKeyReadableStore(

defaultProxyMap(serviceIdentifier),

defaultHalfLife,

getTopClustersWithScoresByFavClusterNormalizedScore

)

def overrideLimitDefaultStore(

maxResult: Int,

serviceIdentifier: ServiceIdentifier

): ReadableStore[TweetKey, Seq[(Int, Double)]] = {

TopKClustersForTweetKeyReadableStore(

defaultProxyMap(serviceIdentifier),

defaultHalfLife,

getTopClustersWithScoresByFavClusterNormalizedScore,

Some(maxResult)

)

}

private def getTopClustersWithScoresByFavClusterNormalizedScore(

topKClustersWithScores: TopKClustersWithScores

): Seq[(Int, Double)] = {

{

for {

clusterIdWIthScores <- topKClustersWithScores.topClustersByFavClusterNormalizedScore

} yield {

(

for {

(clusterId, scores) <- clusterIdWIthScores

favClusterNormalized8HrHalfLifeScore <- scores.favClusterNormalized8HrHalfLifeScore

if favClusterNormalized8HrHalfLifeScore.value > 0.0

} yield {

clusterId -> favClusterNormalized8HrHalfLifeScore.value

}

).toSeq.sortBy(-\_.\_2)

}

}.getOrElse(Nil)

}

}