package com.twitter.home\_mixer.module

import com.google.inject.Provides

import com.twitter.bijection.Injection

import com.twitter.bijection.scrooge.BinaryScalaCodec

import com.twitter.bijection.scrooge.CompactScalaCodec

import com.twitter.bijection.thrift.ThriftCodec

import com.twitter.conversions.DurationOps.\_

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

import com.twitter.home\_mixer.param.HomeMixerInjectionNames.\_

import com.twitter.home\_mixer.util.InjectionTransformerImplicits.\_

import com.twitter.home\_mixer.util.LanguageUtil

import com.twitter.home\_mixer.util.TensorFlowUtil

import com.twitter.inject.TwitterModule

import com.twitter.manhattan.v1.{thriftscala => mh}

import com.twitter.ml.api.{thriftscala => ml}

import com.twitter.ml.featurestore.lib.UserId

import com.twitter.ml.featurestore.{thriftscala => fs}

import com.twitter.onboarding.relevance.features.{thriftjava => rf}

import com.twitter.product\_mixer.shared\_library.manhattan\_client.ManhattanClientBuilder

import com.twitter.scalding\_internal.multiformat.format.keyval.KeyValInjection.ScalaBinaryThrift

import com.twitter.search.common.constants.{thriftscala => scc}

import com.twitter.service.metastore.gen.{thriftscala => smg}

import com.twitter.servo.cache.\_

import com.twitter.servo.manhattan.ManhattanKeyValueRepository

import com.twitter.servo.repository.CachingKeyValueRepository

import com.twitter.servo.repository.ChunkingStrategy

import com.twitter.servo.repository.KeyValueRepository

import com.twitter.servo.repository.Repository

import com.twitter.servo.repository.keysAsQuery

import com.twitter.servo.util.Transformer

import com.twitter.storage.client.manhattan.bijections.Bijections

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

import com.twitter.timelines.author\_features.v1.{thriftjava => af}

import com.twitter.timelines.suggests.common.dense\_data\_record.{thriftscala => ddr}

import com.twitter.user\_session\_store.{thriftscala => uss\_scala}

import com.twitter.user\_session\_store.{thriftjava => uss}

import com.twitter.util.Duration

import com.twitter.util.Try

import java.nio.ByteBuffer

import javax.inject.Named

import javax.inject.Singleton

import org.apache.thrift.protocol.TCompactProtocol

import org.apache.thrift.transport.TMemoryInputTransport

import org.apache.thrift.transport.TTransport

object ManhattanFeatureRepositoryModule extends TwitterModule {

private val DEFAULT\_RPC\_CHUNK\_SIZE = 50

private val ThriftEntityIdInjection = ScalaBinaryThrift(fs.EntityId)

private val FeatureStoreUserIdKeyTransformer = new Transformer[Long, ByteBuffer] {

override def to(userId: Long): Try[ByteBuffer] = {

Try(ByteBuffer.wrap(ThriftEntityIdInjection.apply(UserId(userId).toThrift)))

}

override def from(b: ByteBuffer): Try[Long] = ???

}

private val FloatTensorTransformer = new Transformer[ByteBuffer, ml.FloatTensor] {

override def to(input: ByteBuffer): Try[ml.FloatTensor] = {

val floatTensor = TensorFlowUtil.embeddingByteBufferToFloatTensor(input)

Try(floatTensor)

}

override def from(b: ml.FloatTensor): Try[ByteBuffer] = ???

}

private val LanguageTransformer = new Transformer[ByteBuffer, Seq[scc.ThriftLanguage]] {

override def to(input: ByteBuffer): Try[Seq[scc.ThriftLanguage]] = {

Try.fromScala(

Bijections

.BinaryScalaInjection(smg.UserLanguages)

.andThen(Bijections.byteBuffer2Buf.inverse)

.invert(input).map(LanguageUtil.computeLanguages(\_)))

}

override def from(b: Seq[scc.ThriftLanguage]): Try[ByteBuffer] = ???

}

private val LongKeyTransformer = Injection

.connect[Long, Array[Byte]]

.toByteBufferTransformer()

// manhattan clients

@Provides

@Singleton

@Named(ManhattanApolloClient)

def providesManhattanApolloClient(

serviceIdentifier: ServiceIdentifier

): mh.ManhattanCoordinator.MethodPerEndpoint = {

ManhattanClientBuilder

.buildManhattanV1FinagleClient(

ManhattanClusters.apollo,

serviceIdentifier

)

}

@Provides

@Singleton

@Named(ManhattanAthenaClient)

def providesManhattanAthenaClient(

serviceIdentifier: ServiceIdentifier

): mh.ManhattanCoordinator.MethodPerEndpoint = {

ManhattanClientBuilder

.buildManhattanV1FinagleClient(

ManhattanClusters.athena,

serviceIdentifier

)

}

@Provides

@Singleton

@Named(ManhattanOmegaClient)

def providesManhattanOmegaClient(

serviceIdentifier: ServiceIdentifier

): mh.ManhattanCoordinator.MethodPerEndpoint = {

ManhattanClientBuilder

.buildManhattanV1FinagleClient(

ManhattanClusters.omega,

serviceIdentifier

)

}

@Provides

@Singleton

@Named(ManhattanStarbuckClient)

def providesManhattanStarbuckClient(

serviceIdentifier: ServiceIdentifier

): mh.ManhattanCoordinator.MethodPerEndpoint = {

ManhattanClientBuilder

.buildManhattanV1FinagleClient(

ManhattanClusters.starbuck,

serviceIdentifier

)

}

// non-cached manhattan repositories

@Provides

@Singleton

@Named(MetricCenterUserCountingFeatureRepository)

def providesMetricCenterUserCountingFeatureRepository(

@Named(ManhattanStarbuckClient) client: mh.ManhattanCoordinator.MethodPerEndpoint

): KeyValueRepository[Seq[Long], Long, rf.MCUserCountingFeatures] = {

val valueTransformer = ThriftCodec

.toBinary[rf.MCUserCountingFeatures]

.toByteBufferTransformer()

.flip

batchedManhattanKeyValueRepository[Long, rf.MCUserCountingFeatures](

client = client,

keyTransformer = LongKeyTransformer,

valueTransformer = valueTransformer,

appId = "wtf\_ml",

dataset = "mc\_user\_counting\_features\_v0\_starbuck",

timeoutInMillis = 100

)

}

/\*\*

\* A repository of the offline aggregate feature metadata necessary to decode

\* DenseCompactDataRecords.

\*

\* This repository is expected to virtually always pick up the metadata form the local cache with

\* nearly 0 latency.

\*/

@Provides

@Singleton

@Named(TimelineAggregateMetadataRepository)

def providesTimelineAggregateMetadataRepository(

@Named(ManhattanAthenaClient) client: mh.ManhattanCoordinator.MethodPerEndpoint

): Repository[Int, Option[ddr.DenseFeatureMetadata]] = {

val keyTransformer = Injection

.connect[Int, Array[Byte]]

.toByteBufferTransformer()

val valueTransformer = new Transformer[ByteBuffer, ddr.DenseFeatureMetadata] {

private val compactProtocolFactory = new TCompactProtocol.Factory

def to(buffer: ByteBuffer): Try[ddr.DenseFeatureMetadata] = Try {

val transport = transportFromByteBuffer(buffer)

ddr.DenseFeatureMetadata.decode(compactProtocolFactory.getProtocol(transport))

}

// Encoding intentionally not implemented as it is never used

def from(metadata: ddr.DenseFeatureMetadata): Try[ByteBuffer] = ???

}

val inProcessCache: Cache[Int, Cached[ddr.DenseFeatureMetadata]] = InProcessLruCacheFactory(

ttl = Duration.fromMinutes(20),

lruSize = 30

).apply(serializer = Transformer(\_ => ???, \_ => ???)) // Serialization is not necessary here.

val keyValueRepository = new ManhattanKeyValueRepository(

client = client,

keyTransformer = keyTransformer,

valueTransformer = valueTransformer,

appId = "timelines\_dense\_aggregates\_encoding\_metadata", // Expected QPS is negligible.

dataset = "user\_session\_dense\_feature\_metadata",

timeoutInMillis = 100

)

KeyValueRepository

.singular(

new CachingKeyValueRepository[Seq[Int], Int, ddr.DenseFeatureMetadata](

keyValueRepository,

new NonLockingCache(inProcessCache),

keysAsQuery[Int]

)

)

}

@Provides

@Singleton

@Named(RealGraphFeatureRepository)

def providesRealGraphFeatureRepository(

@Named(ManhattanAthenaClient) client: mh.ManhattanCoordinator.MethodPerEndpoint

): Repository[Long, Option[uss\_scala.UserSession]] = {

val valueTransformer = CompactScalaCodec(uss\_scala.UserSession).toByteBufferTransformer().flip

KeyValueRepository.singular(

new ManhattanKeyValueRepository(

client = client,

keyTransformer = LongKeyTransformer,

valueTransformer = valueTransformer,

appId = "real\_graph",

dataset = "split\_real\_graph\_features",

timeoutInMillis = 100,

)

)

}

// cached manhattan repositories

@Provides

@Singleton

@Named(AuthorFeatureRepository)

def providesAuthorFeatureRepository(

@Named(ManhattanAthenaClient) client: mh.ManhattanCoordinator.MethodPerEndpoint,

@Named(HomeAuthorFeaturesCacheClient) cacheClient: Memcache

): KeyValueRepository[Seq[Long], Long, af.AuthorFeatures] = {

val valueInjection = ThriftCodec

.toCompact[af.AuthorFeatures]

val keyValueRepository = batchedManhattanKeyValueRepository(

client = client,

keyTransformer = LongKeyTransformer,

valueTransformer = valueInjection.toByteBufferTransformer().flip,

appId = "timelines\_author\_feature\_store\_athena",

dataset = "timelines\_author\_features",

timeoutInMillis = 100

)

val remoteCacheRepo = buildMemCachedRepository(

keyValueRepository = keyValueRepository,

cacheClient = cacheClient,

cachePrefix = "AuthorFeatureHydrator",

ttl = 12.hours,

valueInjection = valueInjection)

buildInProcessCachedRepository(

keyValueRepository = remoteCacheRepo,

ttl = 15.minutes,

size = 8000,

valueInjection = valueInjection

)

}

@Provides

@Singleton

@Named(TwhinAuthorFollowFeatureRepository)

def providesTwhinAuthorFollowFeatureRepository(

@Named(ManhattanApolloClient) client: mh.ManhattanCoordinator.MethodPerEndpoint,

@Named(TwhinAuthorFollowFeatureCacheClient) cacheClient: Memcache

): KeyValueRepository[Seq[Long], Long, ml.FloatTensor] = {

val keyValueRepository =

batchedManhattanKeyValueRepository(

client = client,

keyTransformer = FeatureStoreUserIdKeyTransformer,

valueTransformer = FloatTensorTransformer,

appId = "ml\_features\_apollo",

dataset = "twhin\_author\_follow\_embedding\_fsv1\_\_v1\_thrift\_\_embedding",

timeoutInMillis = 100

)

val valueInjection: Injection[ml.FloatTensor, Array[Byte]] =

BinaryScalaCodec(ml.FloatTensor)

buildMemCachedRepository(

keyValueRepository = keyValueRepository,

cacheClient = cacheClient,

cachePrefix = "twhinAuthorFollows",

ttl = 24.hours,

valueInjection = valueInjection

)

}

@Provides

@Singleton

@Named(UserLanguagesRepository)

def providesUserLanguagesFeatureRepository(

@Named(ManhattanStarbuckClient) client: mh.ManhattanCoordinator.MethodPerEndpoint

): KeyValueRepository[Seq[Long], Long, Seq[scc.ThriftLanguage]] = {

batchedManhattanKeyValueRepository(

client = client,

keyTransformer = LongKeyTransformer,

valueTransformer = LanguageTransformer,

appId = "user\_metadata",

dataset = "languages",

timeoutInMillis = 70

)

}

@Provides

@Singleton

@Named(TwhinUserFollowFeatureRepository)

def providesTwhinUserFollowFeatureRepository(

@Named(ManhattanApolloClient) client: mh.ManhattanCoordinator.MethodPerEndpoint

): KeyValueRepository[Seq[Long], Long, ml.FloatTensor] = {

batchedManhattanKeyValueRepository(

client = client,

keyTransformer = FeatureStoreUserIdKeyTransformer,

valueTransformer = FloatTensorTransformer,

appId = "ml\_features\_apollo",

dataset = "twhin\_user\_follow\_embedding\_fsv1\_\_v1\_thrift\_\_embedding",

timeoutInMillis = 100

)

}

@Provides

@Singleton

@Named(TimelineAggregatePartARepository)

def providesTimelineAggregatePartARepository(

@Named(ManhattanApolloClient) client: mh.ManhattanCoordinator.MethodPerEndpoint,

): Repository[Long, Option[uss.UserSession]] =

timelineAggregateRepository(

mhClient = client,

mhDataset = "timelines\_aggregates\_v2\_features\_by\_user\_part\_a\_apollo",

mhAppId = "timelines\_aggregates\_v2\_features\_by\_user\_part\_a\_apollo"

)

@Provides

@Singleton

@Named(TimelineAggregatePartBRepository)

def providesTimelineAggregatePartBRepository(

@Named(ManhattanApolloClient) client: mh.ManhattanCoordinator.MethodPerEndpoint,

): Repository[Long, Option[uss.UserSession]] =

timelineAggregateRepository(

mhClient = client,

mhDataset = "timelines\_aggregates\_v2\_features\_by\_user\_part\_b\_apollo",

mhAppId = "timelines\_aggregates\_v2\_features\_by\_user\_part\_b\_apollo"

)

@Provides

@Singleton

@Named(TwhinUserEngagementFeatureRepository)

def providesTwhinUserEngagementFeatureRepository(

@Named(ManhattanApolloClient) client: mh.ManhattanCoordinator.MethodPerEndpoint

): KeyValueRepository[Seq[Long], Long, ml.FloatTensor] = {

batchedManhattanKeyValueRepository(

client = client,

keyTransformer = FeatureStoreUserIdKeyTransformer,

valueTransformer = FloatTensorTransformer,

appId = "ml\_features\_apollo",

dataset = "twhin\_user\_engagement\_embedding\_fsv1\_\_v1\_thrift\_\_embedding",

timeoutInMillis = 100

)

}

private def buildMemCachedRepository[K, V](

keyValueRepository: KeyValueRepository[Seq[K], K, V],

cacheClient: Memcache,

cachePrefix: String,

ttl: Duration,

valueInjection: Injection[V, Array[Byte]]

): CachingKeyValueRepository[Seq[K], K, V] = {

val cachedSerializer = CachedSerializer.binary(

valueInjection.toByteArrayTransformer()

)

val cache = MemcacheCacheFactory(

cacheClient,

ttl,

PrefixKeyTransformerFactory(cachePrefix)

)[K, Cached[V]](cachedSerializer)

new CachingKeyValueRepository(

keyValueRepository,

new NonLockingCache(cache),

keysAsQuery[K]

)

}

private def buildInProcessCachedRepository[K, V](

keyValueRepository: KeyValueRepository[Seq[K], K, V],

ttl: Duration,

size: Int,

valueInjection: Injection[V, Array[Byte]]

): CachingKeyValueRepository[Seq[K], K, V] = {

val cachedSerializer = CachedSerializer.binary(

valueInjection.toByteArrayTransformer()

)

val cache = InProcessLruCacheFactory(

ttl = ttl,

lruSize = size

)[K, Cached[V]](cachedSerializer)

new CachingKeyValueRepository(

keyValueRepository,

new NonLockingCache(cache),

keysAsQuery[K]

)

}

private def batchedManhattanKeyValueRepository[K, V](

client: mh.ManhattanCoordinator.MethodPerEndpoint,

keyTransformer: Transformer[K, ByteBuffer],

valueTransformer: Transformer[ByteBuffer, V],

appId: String,

dataset: String,

timeoutInMillis: Int,

chunkSize: Int = DEFAULT\_RPC\_CHUNK\_SIZE

): KeyValueRepository[Seq[K], K, V] =

KeyValueRepository.chunked(

new ManhattanKeyValueRepository(

client = client,

keyTransformer = keyTransformer,

valueTransformer = valueTransformer,

appId = appId,

dataset = dataset,

timeoutInMillis = timeoutInMillis

),

chunker = ChunkingStrategy.equalSize(chunkSize)

)

private def transportFromByteBuffer(buffer: ByteBuffer): TTransport =

new TMemoryInputTransport(

buffer.array(),

buffer.arrayOffset() + buffer.position(),

buffer.remaining())

private def timelineAggregateRepository(

mhClient: mh.ManhattanCoordinator.MethodPerEndpoint,

mhDataset: String,

mhAppId: String

): Repository[Long, Option[uss.UserSession]] = {

val valueInjection = ThriftCodec

.toCompact[uss.UserSession]

KeyValueRepository.singular(

new ManhattanKeyValueRepository(

client = mhClient,

keyTransformer = LongKeyTransformer,

valueTransformer = valueInjection.toByteBufferTransformer().flip,

appId = mhAppId,

dataset = mhDataset,

timeoutInMillis = 100

)

)

}

}