package com.twitter.representation\_manager.store

import com.twitter.contentrecommender.twistly

import com.twitter.finagle.memcached.Client

import com.twitter.finagle.stats.StatsReceiver

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

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

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

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

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

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

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

import com.twitter.simclusters\_v2.summingbird.stores.ProducerClusterEmbeddingReadableStores

import com.twitter.simclusters\_v2.summingbird.stores.UserInterestedInReadableStore

import com.twitter.simclusters\_v2.summingbird.stores.UserInterestedInReadableStore.getStore

import com.twitter.simclusters\_v2.summingbird.stores.UserInterestedInReadableStore.modelVersionToDatasetMap

import com.twitter.simclusters\_v2.summingbird.stores.UserInterestedInReadableStore.knownModelVersions

import com.twitter.simclusters\_v2.summingbird.stores.UserInterestedInReadableStore.toSimClustersEmbedding

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

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.{SimClustersEmbedding => ThriftSimClustersEmbedding}

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

import com.twitter.storehaus.ReadableStore

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

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

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

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

import com.twitter.tweetypie.util.UserId

import com.twitter.util.Future

import javax.inject.Inject

class UserSimClustersEmbeddingStore @Inject() (

stratoClient: StratoClient,

cacheClient: Client,

globalStats: StatsReceiver,

mhMtlsParams: ManhattanKVClientMtlsParams,

rmsDecider: RepresentationManagerDecider) {

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

private val favBasedProducer20M145KUpdatedEmbeddingStore: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

val rawStore = ProducerClusterEmbeddingReadableStores

.getProducerTopKSimClustersEmbeddingsStore(

mhMtlsParams

).mapValues { topSimClustersWithScore =>

ThriftSimClustersEmbedding(topSimClustersWithScore.topClusters)

}.composeKeyMapping[SimClustersEmbeddingId] {

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

userId

}

buildMemCacheStore(rawStore, FavBasedProducer, Model20m145kUpdated)

}

private val favBasedProducer20M145K2020EmbeddingStore: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

val rawStore = ProducerClusterEmbeddingReadableStores

.getProducerTopKSimClusters2020EmbeddingsStore(

mhMtlsParams

).mapValues { topSimClustersWithScore =>

ThriftSimClustersEmbedding(topSimClustersWithScore.topClusters)

}.composeKeyMapping[SimClustersEmbeddingId] {

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

userId

}

buildMemCacheStore(rawStore, FavBasedProducer, Model20m145k2020)

}

private val followBasedProducer20M145K2020EmbeddingStore: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

val rawStore = ProducerClusterEmbeddingReadableStores

.getProducerTopKSimClustersEmbeddingsByFollowStore(

mhMtlsParams

).mapValues { topSimClustersWithScore =>

ThriftSimClustersEmbedding(topSimClustersWithScore.topClusters)

}.composeKeyMapping[SimClustersEmbeddingId] {

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

userId

}

buildMemCacheStore(rawStore, FollowBasedProducer, Model20m145k2020)

}

private val logFavBasedApe20M145K2020EmbeddingStore: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

val rawStore = StratoFetchableStore

.withUnitView[SimClustersEmbeddingId, ThriftSimClustersEmbedding](

stratoClient,

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

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

buildMemCacheStore(rawStore, AggregatableLogFavBasedProducer, Model20m145k2020)

}

private val rawRelaxedLogFavBasedApe20M145K2020EmbeddingStore: ReadableStore[

SimClustersEmbeddingId,

ThriftSimClustersEmbedding

] = {

StratoFetchableStore

.withUnitView[SimClustersEmbeddingId, ThriftSimClustersEmbedding](

stratoClient,

"recommendations/simclusters\_v2/embeddings/logFavBasedAPERelaxedFavEngagementThreshold20M145K2020")

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

}

private val relaxedLogFavBasedApe20M145K2020EmbeddingStore: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

buildMemCacheStore(

rawRelaxedLogFavBasedApe20M145K2020EmbeddingStore,

RelaxedAggregatableLogFavBasedProducer,

Model20m145k2020)

}

private val relaxedLogFavBasedApe20m145kUpdatedEmbeddingStore: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

val rawStore = rawRelaxedLogFavBasedApe20M145K2020EmbeddingStore

.composeKeyMapping[SimClustersEmbeddingId] {

case SimClustersEmbeddingId(

RelaxedAggregatableLogFavBasedProducer,

Model20m145kUpdated,

internalId) =>

SimClustersEmbeddingId(

RelaxedAggregatableLogFavBasedProducer,

Model20m145k2020,

internalId)

}

buildMemCacheStore(rawStore, RelaxedAggregatableLogFavBasedProducer, Model20m145kUpdated)

}

private val logFavBasedInterestedInFromAPE20M145K2020Store: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

buildUserInterestedInStore(

UserInterestedInReadableStore.defaultIIAPESimClustersEmbeddingStoreWithMtls,

LogFavBasedUserInterestedInFromAPE,

Model20m145k2020)

}

private val followBasedInterestedInFromAPE20M145K2020Store: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

buildUserInterestedInStore(

UserInterestedInReadableStore.defaultIIAPESimClustersEmbeddingStoreWithMtls,

FollowBasedUserInterestedInFromAPE,

Model20m145k2020)

}

private val favBasedUserInterestedIn20M145KUpdatedStore: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

buildUserInterestedInStore(

UserInterestedInReadableStore.defaultSimClustersEmbeddingStoreWithMtls,

FavBasedUserInterestedIn,

Model20m145kUpdated)

}

private val favBasedUserInterestedIn20M145K2020Store: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

buildUserInterestedInStore(

UserInterestedInReadableStore.defaultSimClustersEmbeddingStoreWithMtls,

FavBasedUserInterestedIn,

Model20m145k2020)

}

private val followBasedUserInterestedIn20M145K2020Store: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

buildUserInterestedInStore(

UserInterestedInReadableStore.defaultSimClustersEmbeddingStoreWithMtls,

FollowBasedUserInterestedIn,

Model20m145k2020)

}

private val logFavBasedUserInterestedIn20M145K2020Store: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

buildUserInterestedInStore(

UserInterestedInReadableStore.defaultSimClustersEmbeddingStoreWithMtls,

LogFavBasedUserInterestedIn,

Model20m145k2020)

}

private val favBasedUserInterestedInFromPE20M145KUpdatedStore: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

buildUserInterestedInStore(

UserInterestedInReadableStore.defaultIIPESimClustersEmbeddingStoreWithMtls,

FavBasedUserInterestedInFromPE,

Model20m145kUpdated)

}

private val twistlyUserInterestedInStore: ReadableStore[

SimClustersEmbeddingId,

ThriftSimClustersEmbedding

] = {

val interestedIn20M145KUpdatedStore = {

UserInterestedInReadableStore.defaultStoreWithMtls(

mhMtlsParams,

modelVersion = ModelVersions.Model20M145KUpdated

)

}

val interestedIn20M145K2020Store = {

UserInterestedInReadableStore.defaultStoreWithMtls(

mhMtlsParams,

modelVersion = ModelVersions.Model20M145K2020

)

}

val interestedInFromPE20M145KUpdatedStore = {

UserInterestedInReadableStore.defaultIIPEStoreWithMtls(

mhMtlsParams,

modelVersion = ModelVersions.Model20M145KUpdated)

}

val simClustersInterestedInStore: ReadableStore[

(UserId, ModelVersion),

ClustersUserIsInterestedIn

] = {

new ReadableStore[(UserId, ModelVersion), ClustersUserIsInterestedIn] {

override def get(k: (UserId, ModelVersion)): Future[Option[ClustersUserIsInterestedIn]] = {

k match {

case (userId, Model20m145kUpdated) =>

interestedIn20M145KUpdatedStore.get(userId)

case (userId, Model20m145k2020) =>

interestedIn20M145K2020Store.get(userId)

case \_ =>

Future.None

}

}

}

}

val simClustersInterestedInFromProducerEmbeddingsStore: ReadableStore[

(UserId, ModelVersion),

ClustersUserIsInterestedIn

] = {

new ReadableStore[(UserId, ModelVersion), ClustersUserIsInterestedIn] {

override def get(k: (UserId, ModelVersion)): Future[Option[ClustersUserIsInterestedIn]] = {

k match {

case (userId, ModelVersion.Model20m145kUpdated) =>

interestedInFromPE20M145KUpdatedStore.get(userId)

case \_ =>

Future.None

}

}

}

}

new twistly.interestedin.EmbeddingStore(

interestedInStore = simClustersInterestedInStore,

interestedInFromProducerEmbeddingStore = simClustersInterestedInFromProducerEmbeddingsStore,

statsReceiver = stats

).mapValues(\_.toThrift)

}

private val userNextInterestedIn20m145k2020Store: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

buildUserInterestedInStore(

UserInterestedInReadableStore.defaultNextInterestedInStoreWithMtls,

UserNextInterestedIn,

Model20m145k2020)

}

private val filteredUserInterestedIn20m145kUpdatedStore: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

buildMemCacheStore(twistlyUserInterestedInStore, FilteredUserInterestedIn, Model20m145kUpdated)

}

private val filteredUserInterestedIn20m145k2020Store: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

buildMemCacheStore(twistlyUserInterestedInStore, FilteredUserInterestedIn, Model20m145k2020)

}

private val filteredUserInterestedInFromPE20m145kUpdatedStore: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

buildMemCacheStore(

twistlyUserInterestedInStore,

FilteredUserInterestedInFromPE,

Model20m145kUpdated)

}

private val unfilteredUserInterestedIn20m145kUpdatedStore: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

buildMemCacheStore(

twistlyUserInterestedInStore,

UnfilteredUserInterestedIn,

Model20m145kUpdated)

}

private val unfilteredUserInterestedIn20m145k2020Store: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

buildMemCacheStore(twistlyUserInterestedInStore, UnfilteredUserInterestedIn, Model20m145k2020)

}

// [Experimental] User InterestedIn, generated by aggregating IIAPE embedding from AddressBook

private val logFavBasedInterestedMaxpoolingAddressBookFromIIAPE20M145K2020Store: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

val datasetName = "addressbook\_sims\_embedding\_iiape\_maxpooling"

val appId = "wtf\_embedding\_apollo"

buildUserInterestedInStoreGeneric(

simClustersEmbeddingStoreWithMtls,

LogFavBasedUserInterestedMaxpoolingAddressBookFromIIAPE,

Model20m145k2020,

datasetName = datasetName,

appId = appId,

manhattanCluster = Apollo

)

}

private val logFavBasedInterestedAverageAddressBookFromIIAPE20M145K2020Store: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

val datasetName = "addressbook\_sims\_embedding\_iiape\_average"

val appId = "wtf\_embedding\_apollo"

buildUserInterestedInStoreGeneric(

simClustersEmbeddingStoreWithMtls,

LogFavBasedUserInterestedAverageAddressBookFromIIAPE,

Model20m145k2020,

datasetName = datasetName,

appId = appId,

manhattanCluster = Apollo

)

}

private val logFavBasedUserInterestedBooktypeMaxpoolingAddressBookFromIIAPE20M145K2020Store: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

val datasetName = "addressbook\_sims\_embedding\_iiape\_booktype\_maxpooling"

val appId = "wtf\_embedding\_apollo"

buildUserInterestedInStoreGeneric(

simClustersEmbeddingStoreWithMtls,

LogFavBasedUserInterestedBooktypeMaxpoolingAddressBookFromIIAPE,

Model20m145k2020,

datasetName = datasetName,

appId = appId,

manhattanCluster = Apollo

)

}

private val logFavBasedUserInterestedLargestDimMaxpoolingAddressBookFromIIAPE20M145K2020Store: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

val datasetName = "addressbook\_sims\_embedding\_iiape\_largestdim\_maxpooling"

val appId = "wtf\_embedding\_apollo"

buildUserInterestedInStoreGeneric(

simClustersEmbeddingStoreWithMtls,

LogFavBasedUserInterestedLargestDimMaxpoolingAddressBookFromIIAPE,

Model20m145k2020,

datasetName = datasetName,

appId = appId,

manhattanCluster = Apollo

)

}

private val logFavBasedUserInterestedLouvainMaxpoolingAddressBookFromIIAPE20M145K2020Store: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

val datasetName = "addressbook\_sims\_embedding\_iiape\_louvain\_maxpooling"

val appId = "wtf\_embedding\_apollo"

buildUserInterestedInStoreGeneric(

simClustersEmbeddingStoreWithMtls,

LogFavBasedUserInterestedLouvainMaxpoolingAddressBookFromIIAPE,

Model20m145k2020,

datasetName = datasetName,

appId = appId,

manhattanCluster = Apollo

)

}

private val logFavBasedUserInterestedConnectedMaxpoolingAddressBookFromIIAPE20M145K2020Store: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

val datasetName = "addressbook\_sims\_embedding\_iiape\_connected\_maxpooling"

val appId = "wtf\_embedding\_apollo"

buildUserInterestedInStoreGeneric(

simClustersEmbeddingStoreWithMtls,

LogFavBasedUserInterestedConnectedMaxpoolingAddressBookFromIIAPE,

Model20m145k2020,

datasetName = datasetName,

appId = appId,

manhattanCluster = Apollo

)

}

/\*\*

\* Helper func to build a readable store for some UserInterestedIn embeddings with

\* 1. A storeFunc from UserInterestedInReadableStore

\* 2. EmbeddingType

\* 3. ModelVersion

\* 4. MemCacheConfig

\* \*/

private def buildUserInterestedInStore(

storeFunc: (ManhattanKVClientMtlsParams, EmbeddingType, ModelVersion) => ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

],

embeddingType: EmbeddingType,

modelVersion: ModelVersion

): ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

val rawStore = storeFunc(mhMtlsParams, embeddingType, modelVersion)

.mapValues(\_.toThrift)

val observedStore = ObservedReadableStore(

store = rawStore

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

MemCacheConfig.buildMemCacheStoreForSimClustersEmbedding(

observedStore,

cacheClient,

embeddingType,

modelVersion,

stats

)

}

private def buildUserInterestedInStoreGeneric(

storeFunc: (ManhattanKVClientMtlsParams, EmbeddingType, ModelVersion, String, String,

ManhattanCluster) => ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

],

embeddingType: EmbeddingType,

modelVersion: ModelVersion,

datasetName: String,

appId: String,

manhattanCluster: ManhattanCluster

): ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

val rawStore =

storeFunc(mhMtlsParams, embeddingType, modelVersion, datasetName, appId, manhattanCluster)

.mapValues(\_.toThrift)

val observedStore = ObservedReadableStore(

store = rawStore

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

MemCacheConfig.buildMemCacheStoreForSimClustersEmbedding(

observedStore,

cacheClient,

embeddingType,

modelVersion,

stats

)

}

private def simClustersEmbeddingStoreWithMtls(

mhMtlsParams: ManhattanKVClientMtlsParams,

embeddingType: EmbeddingType,

modelVersion: ModelVersion,

datasetName: String,

appId: String,

manhattanCluster: ManhattanCluster

): ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding] = {

if (!modelVersionToDatasetMap.contains(ModelVersions.toKnownForModelVersion(modelVersion))) {

throw new IllegalArgumentException(

"Unknown model version: " + modelVersion + ". Known model versions: " + knownModelVersions)

}

getStore(appId, mhMtlsParams, datasetName, manhattanCluster)

.composeKeyMapping[SimClustersEmbeddingId] {

case SimClustersEmbeddingId(theEmbeddingType, theModelVersion, InternalId.UserId(userId))

if theEmbeddingType == embeddingType && theModelVersion == modelVersion =>

userId

}.mapValues(toSimClustersEmbedding(\_, embeddingType))

}

private def buildMemCacheStore(

rawStore: ReadableStore[SimClustersEmbeddingId, ThriftSimClustersEmbedding],

embeddingType: EmbeddingType,

modelVersion: ModelVersion

): ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding] = {

val observedStore = ObservedReadableStore(

store = rawStore

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

MemCacheConfig.buildMemCacheStoreForSimClustersEmbedding(

observedStore,

cacheClient,

embeddingType,

modelVersion,

stats

)

}

private val underlyingStores: Map[

(EmbeddingType, ModelVersion),

ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding]

] = Map(

// KnownFor Embeddings

(FavBasedProducer, Model20m145kUpdated) -> favBasedProducer20M145KUpdatedEmbeddingStore,

(FavBasedProducer, Model20m145k2020) -> favBasedProducer20M145K2020EmbeddingStore,

(FollowBasedProducer, Model20m145k2020) -> followBasedProducer20M145K2020EmbeddingStore,

(AggregatableLogFavBasedProducer, Model20m145k2020) -> logFavBasedApe20M145K2020EmbeddingStore,

(

RelaxedAggregatableLogFavBasedProducer,

Model20m145kUpdated) -> relaxedLogFavBasedApe20m145kUpdatedEmbeddingStore,

(

RelaxedAggregatableLogFavBasedProducer,

Model20m145k2020) -> relaxedLogFavBasedApe20M145K2020EmbeddingStore,

// InterestedIn Embeddings

(

LogFavBasedUserInterestedInFromAPE,

Model20m145k2020) -> logFavBasedInterestedInFromAPE20M145K2020Store,

(

FollowBasedUserInterestedInFromAPE,

Model20m145k2020) -> followBasedInterestedInFromAPE20M145K2020Store,

(FavBasedUserInterestedIn, Model20m145kUpdated) -> favBasedUserInterestedIn20M145KUpdatedStore,

(FavBasedUserInterestedIn, Model20m145k2020) -> favBasedUserInterestedIn20M145K2020Store,

(FollowBasedUserInterestedIn, Model20m145k2020) -> followBasedUserInterestedIn20M145K2020Store,

(LogFavBasedUserInterestedIn, Model20m145k2020) -> logFavBasedUserInterestedIn20M145K2020Store,

(

FavBasedUserInterestedInFromPE,

Model20m145kUpdated) -> favBasedUserInterestedInFromPE20M145KUpdatedStore,

(FilteredUserInterestedIn, Model20m145kUpdated) -> filteredUserInterestedIn20m145kUpdatedStore,

(FilteredUserInterestedIn, Model20m145k2020) -> filteredUserInterestedIn20m145k2020Store,

(

FilteredUserInterestedInFromPE,

Model20m145kUpdated) -> filteredUserInterestedInFromPE20m145kUpdatedStore,

(

UnfilteredUserInterestedIn,

Model20m145kUpdated) -> unfilteredUserInterestedIn20m145kUpdatedStore,

(UnfilteredUserInterestedIn, Model20m145k2020) -> unfilteredUserInterestedIn20m145k2020Store,

(UserNextInterestedIn, Model20m145k2020) -> userNextInterestedIn20m145k2020Store,

(

LogFavBasedUserInterestedMaxpoolingAddressBookFromIIAPE,

Model20m145k2020) -> logFavBasedInterestedMaxpoolingAddressBookFromIIAPE20M145K2020Store,

(

LogFavBasedUserInterestedAverageAddressBookFromIIAPE,

Model20m145k2020) -> logFavBasedInterestedAverageAddressBookFromIIAPE20M145K2020Store,

(

LogFavBasedUserInterestedBooktypeMaxpoolingAddressBookFromIIAPE,

Model20m145k2020) -> logFavBasedUserInterestedBooktypeMaxpoolingAddressBookFromIIAPE20M145K2020Store,

(

LogFavBasedUserInterestedLargestDimMaxpoolingAddressBookFromIIAPE,

Model20m145k2020) -> logFavBasedUserInterestedLargestDimMaxpoolingAddressBookFromIIAPE20M145K2020Store,

(

LogFavBasedUserInterestedLouvainMaxpoolingAddressBookFromIIAPE,

Model20m145k2020) -> logFavBasedUserInterestedLouvainMaxpoolingAddressBookFromIIAPE20M145K2020Store,

(

LogFavBasedUserInterestedConnectedMaxpoolingAddressBookFromIIAPE,

Model20m145k2020) -> logFavBasedUserInterestedConnectedMaxpoolingAddressBookFromIIAPE20M145K2020Store,

)

val userSimClustersEmbeddingStore: ReadableStore[

SimClustersEmbeddingId,

SimClustersEmbedding

] = {

SimClustersEmbeddingStore.buildWithDecider(

underlyingStores = underlyingStores,

decider = rmsDecider.decider,

statsReceiver = stats

)

}

}