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

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.stats.NullStatsReceiver

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

import com.twitter.scalding.Args

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

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

import com.twitter.simclusters\_v2.summingbird.common.Monoids.PersistentSimClustersEmbeddingLongestL2NormMonoid

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

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

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

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

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

import com.twitter.simclusters\_v2.summingbird.stores.PersistentTweetEmbeddingStore.PersistentTweetEmbeddingId

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

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

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

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

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

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

import com.twitter.storehaus.FutureCollector

import com.twitter.summingbird.online.option.\_

import com.twitter.summingbird.option.\_

import com.twitter.summingbird.storm.Storm

import com.twitter.summingbird.Options

import com.twitter.summingbird.TailProducer

import com.twitter.summingbird\_internal.runner.common.JobName

import com.twitter.summingbird\_internal.runner.common.SBRunConfig

import com.twitter.summingbird\_internal.runner.storm.GenericRunner

import com.twitter.summingbird\_internal.runner.storm.StormConfig

import com.twitter.tormenta\_internal.spout.eventbus.SubscriberId

import com.twitter.tweetypie.thriftscala.StatusCounts

import com.twitter.wtf.summingbird.sources.storm.TimelineEventSource

import java.lang

import java.util.{HashMap => JMap}

import org.apache.heron.api.{Config => HeronConfig}

import org.apache.storm.{Config => BTConfig}

object PersistentTweetJobRunner {

def main(args: Array[String]): Unit = {

GenericRunner(args, PersistentTweetStormJob(\_))

}

}

object PersistentTweetStormJob {

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

def jLong(num: Long): lang.Long = java.lang.Long.valueOf(num)

def jInt(num: Int): Integer = java.lang.Integer.valueOf(num)

def jFloat(num: Float): lang.Float = java.lang.Float.valueOf(num)

def apply(args: Args): StormConfig = {

lazy val env: String = args.getOrElse("env", "prod")

lazy val zone: String = args.getOrElse("dc", "atla")

lazy val alt: String = args.getOrElse("alt", default = "normal")

lazy val profile =

SimClustersProfile.fetchPersistentJobProfile(Environment(env), AltSetting(alt))

lazy val stratoClient = ClientConfigs.stratoClient(profile.serviceIdentifier(zone))

lazy val favoriteEventSource = TimelineEventSource(

// Note: do not share the same subsriberId with other jobs. Apply a new one if needed

SubscriberId(profile.timelineEventSourceSubscriberId)

).kafkaSource

lazy val persistentTweetEmbeddingStore =

PersistentTweetEmbeddingStore

.persistentTweetEmbeddingStore(stratoClient, profile.persistentTweetStratoPath)

lazy val persistentTweetEmbeddingStoreWithLatestAggregation: Storm#Store[

PersistentTweetEmbeddingId,

PersistentSimClustersEmbedding

] = {

import com.twitter.storehaus.algebra.StoreAlgebra.\_

lazy val mergeableStore =

persistentTweetEmbeddingStore.toMergeable(

mon = Implicits.persistentSimClustersEmbeddingMonoid,

fc = implicitly[FutureCollector])

Storm.onlineOnlyStore(mergeableStore)

}

lazy val persistentTweetEmbeddingStoreWithLongestL2NormAggregation: Storm#Store[

PersistentTweetEmbeddingId,

PersistentSimClustersEmbedding

] = {

import com.twitter.storehaus.algebra.StoreAlgebra.\_

val longestL2NormMonoid = new PersistentSimClustersEmbeddingLongestL2NormMonoid()

lazy val mergeableStore =

persistentTweetEmbeddingStore.toMergeable(

mon = longestL2NormMonoid,

fc = implicitly[FutureCollector])

Storm.onlineOnlyStore(mergeableStore)

}

lazy val tweetStatusCountsService: Storm#Service[TweetId, StatusCounts] =

Storm.service(

ObservedCachedReadableStore.from[TweetId, StatusCounts](

TweetStatusCountsStore.tweetStatusCountsStore(stratoClient, "tweetypie/core.Tweet"),

ttl = 1.minute,

maxKeys = 10000, // 10K is enough for Heron Job.

cacheName = "tweet\_status\_count",

windowSize = 10000L

)(NullStatsReceiver)

)

lazy val tweetEmbeddingService: Storm#Service[TweetId, ThriftSimClustersEmbedding] =

Storm.service(

TopKClustersForTweetKeyReadableStore

.overrideLimitDefaultStore(50, profile.serviceIdentifier(zone))

.composeKeyMapping { tweetId: TweetId =>

TweetKey(tweetId, profile.modelVersionStr, profile.coreEmbeddingType)

}.mapValues { value => SimClustersEmbedding(value).toThrift })

new StormConfig {

val jobName: JobName = JobName(profile.jobName)

implicit val jobID: JobId = JobId(jobName.toString)

/\*\*

\* Add registrars for chill serialization for user-defined types.

\*/

override def registrars =

List(

SBRunConfig.register[StatusCounts],

SBRunConfig.register[ThriftSimClustersEmbedding],

SBRunConfig.register[PersistentSimClustersEmbedding]

)

/\*\*\*\*\* Job configuration settings \*\*\*\*\*/

/\*\*

\* Use vmSettings to configure the VM

\*/

override def vmSettings: Seq[String] = Seq()

private val SourcePerWorker = 1

private val FlatMapPerWorker = 1

private val SummerPerWorker = 1

private val TotalWorker = 60

/\*\*

\* Use transformConfig to set Heron options.

\*/

override def transformConfig(config: Map[String, AnyRef]): Map[String, AnyRef] = {

val heronJvmOptions = new JMap[String, AnyRef]()

val MetaspaceSize = jLong(256L \* 1024 \* 1024)

val DefaultHeapSize = jLong(2L \* 1024 \* 1024 \* 1024)

val HighHeapSize = jLong(4L \* 1024 \* 1024 \* 1024)

val TotalCPU = jLong(

SourcePerWorker \* 1 + FlatMapPerWorker \* 4 + SummerPerWorker \* 3 + 1

)

// reserve 4GB for the StreamMgr

val TotalRam = jLong(

DefaultHeapSize \* (SourcePerWorker \* 1 + FlatMapPerWorker \* 4)

+ HighHeapSize \* SummerPerWorker \* 3

+ MetaspaceSize \* 8 // Applies to all workers

+ 4L \* 1024 \* 1024 \* 1024)

// These settings help prevent GC issues in the most memory intensive steps of the job by

// dedicating more memory to the new gen heap designated by the -Xmn flag.

Map(

"Tail" -> HighHeapSize

).foreach {

case (stage, heap) =>

HeronConfig.setComponentJvmOptions(

heronJvmOptions,

stage,

s"-Xmx$heap -Xms$heap -Xmn${heap / 2}"

)

}

super.transformConfig(config) ++ List(

BTConfig.TOPOLOGY\_TEAM\_NAME -> "cassowary",

BTConfig.TOPOLOGY\_TEAM\_EMAIL -> "no-reply@twitter.com",

BTConfig.TOPOLOGY\_WORKERS -> jInt(TotalWorker),

BTConfig.TOPOLOGY\_ACKER\_EXECUTORS -> jInt(0),

BTConfig.TOPOLOGY\_MESSAGE\_TIMEOUT\_SECS -> jInt(30),

BTConfig.TOPOLOGY\_WORKER\_CHILDOPTS -> List(

"-Djava.security.auth.login.config=config/jaas.conf",

"-Dsun.security.krb5.debug=true",

"-Dcom.twitter.eventbus.client.EnableKafkaSaslTls=true",

"-Dcom.twitter.eventbus.client.zoneName=" + zone,

s"-XX:MaxMetaspaceSize=$MetaspaceSize"

).mkString(" "),

HeronConfig.TOPOLOGY\_CONTAINER\_CPU\_REQUESTED -> TotalCPU,

HeronConfig.TOPOLOGY\_CONTAINER\_RAM\_REQUESTED -> TotalRam,

"storm.job.uniqueId" -> jobID.get

)

}

/\*\*

\* Use getNamedOptions to set Summingbird runtime options

\* The list of available options: com.twitter.summingbird.online.option

\*/

override def getNamedOptions: Map[String, Options] = Map(

"DEFAULT" -> Options()

.set(SummerParallelism(TotalWorker \* SummerPerWorker))

.set(FlatMapParallelism(TotalWorker \* FlatMapPerWorker))

.set(SourceParallelism(TotalWorker \* SourcePerWorker))

.set(CacheSize(10000))

.set(FlushFrequency(30.seconds))

)

/\*\* Required job generation call for your job, defined in Job.scala \*/

override def graph: TailProducer[Storm, Any] = PersistentTweetJob.generate[Storm](

favoriteEventSource,

tweetStatusCountsService,

tweetEmbeddingService,

persistentTweetEmbeddingStoreWithLatestAggregation,

persistentTweetEmbeddingStoreWithLongestL2NormAggregation

)

}

}

}