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

import com.twitter.conversions.DurationOps.\_

import com.twitter.heron.util.CommonMetric

import com.twitter.scalding.Args

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

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

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

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

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

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

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

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

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

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

import com.twitter.summingbird.option.\_

import com.twitter.summingbird.storm.option.FlatMapStormMetrics

import com.twitter.summingbird.storm.option.SummerStormMetrics

import com.twitter.summingbird.storm.Storm

import com.twitter.summingbird.storm.StormMetric

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.wtf.summingbird.sources.storm.TimelineEventSource

import java.lang

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

import org.apache.heron.common.basics.ByteAmount

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

import scala.collection.JavaConverters.\_

object TweetJobRunner {

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

GenericRunner(args, TweetStormJob(\_))

}

}

object TweetStormJob {

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 apply(args: Args): StormConfig = {

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

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

// The only SimClusters ENV is Alt. Will clean up soon.

lazy val profile = SimClustersProfile.fetchTweetJobProfile(Environment(env), AltSetting.Alt)

lazy val favoriteEventSource = TimelineEventSource(

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

SubscriberId(profile.timelineEventSourceSubscriberId)

).source

lazy val commonMetric =

StormMetric(new CommonMetric(), CommonMetric.NAME, CommonMetric.POLL\_INTERVAL)

lazy val flatMapMetrics = FlatMapStormMetrics(Iterable(commonMetric))

lazy val summerMetrics = SummerStormMetrics(Iterable(commonMetric))

lazy val entityClusterScoreStore: Storm#Store[

(SimClusterEntity, FullClusterIdBucket),

ClustersWithScores

] = {

Storm.store(

EntityClusterScoreReadableStore

.onlineMergeableStore(profile.entityClusterScorePath, profile.serviceIdentifier(zone)))

}

lazy val tweetTopKStore: Storm#Store[EntityWithVersion, TopKClustersWithScores] = {

Storm.store(

TopKClustersForTweetReadableStore

.onlineMergeableStore(profile.tweetTopKClustersPath, profile.serviceIdentifier(zone)))

}

lazy val clusterTopKTweetsStore: Storm#Store[FullClusterId, TopKTweetsWithScores] = {

Storm.store(

TopKTweetsForClusterReadableStore

.onlineMergeableStore(profile.clusterTopKTweetsPath, profile.serviceIdentifier(zone)))

}

lazy val clusterTopKTweetsLightStore: Option[

Storm#Store[FullClusterId, TopKTweetsWithScores]

] = {

profile.clusterTopKTweetsLightPath.map { lightPath =>

Storm.store(

TopKTweetsForClusterReadableStore

.onlineMergeableStore(lightPath, profile.serviceIdentifier(zone)))

}

}

lazy val userInterestedInService: Storm#Service[Long, ClustersUserIsInterestedIn] = {

Storm.service(

UserInterestedInReadableStore.defaultStoreWithMtls(

ManhattanKVClientMtlsParams(profile.serviceIdentifier(zone)),

modelVersion = profile.modelVersionStr

))

}

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[SimClusterEntity],

SBRunConfig.register[FullClusterIdBucket],

SBRunConfig.register[ClustersWithScores],

SBRunConfig.register[EntityWithVersion],

SBRunConfig.register[FullClusterId],

SBRunConfig.register[EntityWithVersion],

SBRunConfig.register[TopKEntitiesWithScores],

SBRunConfig.register[TopKClustersWithScores],

SBRunConfig.register[TopKTweetsWithScores]

)

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

/\*\*

\* Use vmSettings to configure the VM

\*/

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

private val SourcePerWorker = 1

private val FlatMapPerWorker = 3

private val SummerPerWorker = 3

private val TotalWorker = 150

/\*\*

\* Use transformConfig to set Heron options.

\*/

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

val heronConfig = new HeronConfig()

/\*\*

Component names (subject to change if you add more components, make sure to update this)

Source: Tail-FlatMap-FlatMap-Summer-FlatMap-Source

FlatMap: Tail-FlatMap-FlatMap-Summer-FlatMap, Tail-FlatMap-FlatMap, Tail-FlatMap-FlatMap,

Tail-FlatMap

Summer: Tail-FlatMap-FlatMap-Summer \* 2, Tail, Tail.2

\*/

val sourceName = "Tail-FlatMap-FlatMap-Summer-FlatMap-Source"

val flatMapFlatMapSummerFlatMapName = "Tail-FlatMap-FlatMap-Summer-FlatMap"

// 1 CPU per node, 1 for StreamMgr

// By default, numCpus per component = totalCPUs / total number of components.

// To add more CPUs for a specific component, use heronConfig.setComponentCpu(name, numCPUs)

// add 20% more CPUs to address back pressure issue

val TotalCPU = jLong(

(1.2 \* (SourcePerWorker \* 1 + FlatMapPerWorker \* 4 + SummerPerWorker \* 6 + 1)).ceil.toInt)

heronConfig.setContainerCpuRequested(TotalCPU.toDouble)

// RAM settings

val RamPerSourceGB = 8

val RamPerSummerFlatMap = 8

val RamDefaultPerComponent = 4

// The extra 4GB is not explicitly assigned to the StreamMgr, so it gets 2GB by default, and

// the remaining 2GB is shared among components. Keeping this configuration for now, since

// it seems stable

val TotalRamRB =

RamPerSourceGB \* SourcePerWorker \* 1 +

RamDefaultPerComponent \* FlatMapPerWorker \* 4 +

RamDefaultPerComponent \* SummerPerWorker \* 6 +

4 // reserve 4GB for the StreamMgr

// By default, ramGB per component = totalRAM / total number of components.

// To adjust RAMs for a specific component, use heronConfig.setComponentRam(name, ramGB)

heronConfig.setComponentRam(sourceName, ByteAmount.fromGigabytes(RamPerSourceGB))

heronConfig.setComponentRam(

flatMapFlatMapSummerFlatMapName,

ByteAmount.fromGigabytes(RamPerSummerFlatMap))

heronConfig.setContainerRamRequested(ByteAmount.fromGigabytes(TotalRamRB))

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(

"-XX:MaxMetaspaceSize=256M",

"-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

).mkString(" "),

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

) ++ heronConfig.asScala.toMap

}

/\*\*

\* 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(FlatMapParallelism(TotalWorker \* FlatMapPerWorker))

.set(SourceParallelism(TotalWorker))

.set(SummerBatchMultiplier(1000))

.set(CacheSize(10000))

.set(flatMapMetrics)

.set(summerMetrics),

TweetJob.NodeName.TweetClusterUpdatedScoresFlatMapNodeName -> Options()

.set(FlatMapParallelism(TotalWorker \* FlatMapPerWorker)),

TweetJob.NodeName.TweetClusterScoreSummerNodeName -> Options()

// Most expensive step. Double the capacity.

.set(SummerParallelism(TotalWorker \* SummerPerWorker \* 4))

.set(FlushFrequency(30.seconds)),

TweetJob.NodeName.ClusterTopKTweetsNodeName -> Options()

.set(SummerParallelism(TotalWorker \* SummerPerWorker))

.set(FlushFrequency(30.seconds)),

TweetJob.NodeName.ClusterTopKTweetsLightNodeName -> Options()

.set(SummerParallelism(TotalWorker \* SummerPerWorker))

.set(FlushFrequency(30.seconds)),

TweetJob.NodeName.TweetTopKNodeName -> Options()

.set(SummerParallelism(TotalWorker \* SummerPerWorker))

.set(FlushFrequency(30.seconds))

)

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

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

profile,

favoriteEventSource,

userInterestedInService,

entityClusterScoreStore,

tweetTopKStore,

clusterTopKTweetsStore,

clusterTopKTweetsLightStore

)

}

}

}