package com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.scalding.sources

import com.twitter.ml.api.DailySuffixFeatureSource

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.FixedPathFeatureSource

import com.twitter.ml.api.HourlySuffixFeatureSource

import com.twitter.ml.api.util.SRichDataRecord

import com.twitter.scalding.\_

import com.twitter.scalding\_internal.dalv2.DAL

import com.twitter.scalding\_internal.dalv2.remote\_access.AllowCrossClusterSameDC

import com.twitter.statebird.v2.thriftscala.Environment

import com.twitter.summingbird.\_

import com.twitter.summingbird.scalding.Scalding.pipeFactoryExact

import com.twitter.summingbird.scalding.\_

import com.twitter.summingbird\_internal.sources.SourceFactory

import com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.OfflineAggregateSource

import java.lang.{Long => JLong}

/\*

\* Summingbird offline HDFS source that reads from data records on HDFS.

\*

\* @param offlineSource Underlying offline source that contains

\* all the config info to build this platform-specific (scalding) source.

\*/

case class ScaldingAggregateSource(offlineSource: OfflineAggregateSource)

extends SourceFactory[Scalding, DataRecord] {

val hdfsPath: String = offlineSource.scaldingHdfsPath.getOrElse("")

val suffixType: String = offlineSource.scaldingSuffixType.getOrElse("daily")

val withValidation: Boolean = offlineSource.withValidation

def name: String = offlineSource.name

def description: String =

"Summingbird offline source that reads from data records at: " + hdfsPath

implicit val timeExtractor: TimeExtractor[DataRecord] = TimeExtractor((record: DataRecord) =>

SRichDataRecord(record).getFeatureValue[JLong, JLong](offlineSource.timestampFeature))

def getSourceForDateRange(dateRange: DateRange) = {

suffixType match {

case "daily" => DailySuffixFeatureSource(hdfsPath)(dateRange).source

case "hourly" => HourlySuffixFeatureSource(hdfsPath)(dateRange).source

case "fixed\_path" => FixedPathFeatureSource(hdfsPath).source

case "dal" =>

offlineSource.dalDataSet match {

case Some(dataset) =>

DAL

.read(dataset, dateRange)

.withRemoteReadPolicy(AllowCrossClusterSameDC)

.withEnvironment(Environment.Prod)

.toTypedSource

case \_ =>

throw new IllegalArgumentException(

"cannot provide an empty dataset when defining DAL as the suffix type"

)

}

}

}

/\*\*

\* This method is similar to [[Scalding.sourceFromMappable]] except that this uses [[pipeFactoryExact]]

\* instead of [[pipeFactory]]. [[pipeFactoryExact]] also invokes [[FileSource.validateTaps]] on the source.

\* The validation ensures the presence of \_SUCCESS file before processing. For more details, please refer to

\* https://jira.twitter.biz/browse/TQ-10618

\*/

def sourceFromMappableWithValidation[T: TimeExtractor: Manifest](

factory: (DateRange) => Mappable[T]

): Producer[Scalding, T] = {

Producer.source[Scalding, T](pipeFactoryExact(factory))

}

def source: Producer[Scalding, DataRecord] = {

if (withValidation)

sourceFromMappableWithValidation(getSourceForDateRange)

else

Scalding.sourceFromMappable(getSourceForDateRange)

}

}