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

import com.twitter.bijection.Injection

import com.twitter.bijection.thrift.CompactThriftCodec

import com.twitter.ml.api.AdaptedFeatureSource

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.IRecordOneToManyAdapter

import com.twitter.ml.api.TypedFeatureSource

import com.twitter.scalding.DateRange

import com.twitter.scalding.RichDate

import com.twitter.scalding.TypedPipe

import com.twitter.scalding.commons.source.VersionedKeyValSource

import com.twitter.scalding.commons.tap.VersionedTap.TapMode

import com.twitter.summingbird.batch.BatchID

import com.twitter.summingbird\_internal.bijection.BatchPairImplicits

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

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

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

import org.apache.hadoop.mapred.JobConf

import scala.collection.JavaConverters.\_

import AggregatesV2Adapter.\_

object AggregatesV2AdaptedSource {

val DefaultTrimThreshold = 0

}

trait AggregatesV2AdaptedSource extends AggregatesV2AdaptedSourceBase[DataRecord] {

override def storageFormatCodec: Injection[DataRecord, Array[Byte]] =

CompactThriftCodec[DataRecord]

override def toDataRecord(v: DataRecord): DataRecord = v

}

trait AggregatesV2AdaptedSourceBase[StorageFormat]

extends TypedFeatureSource[AggregatesV2Tuple]

with AdaptedFeatureSource[AggregatesV2Tuple]

with BatchPairImplicits {

/\* Output root path of aggregates v2 job, excluding store name and version \*/

def rootPath: String

/\* Name of store under root path to read \*/

def storeName: String

// max bijection failures

def maxFailures: Int = 0

/\* Aggregate config used to generate above output \*/

def aggregates: Set[TypedAggregateGroup[\_]]

/\* trimThreshold Trim all aggregates below a certain threshold to save memory \*/

def trimThreshold: Double

def toDataRecord(v: StorageFormat): DataRecord

def sourceVersionOpt: Option[Long]

def enableMostRecentBeforeSourceVersion: Boolean = false

implicit private val aggregationKeyInjection: Injection[AggregationKey, Array[Byte]] =

AggregationKeyInjection

implicit def storageFormatCodec: Injection[StorageFormat, Array[Byte]]

private def filteredAggregates = aggregates.filter(\_.outputStore.name == storeName)

def storePath: String = List(rootPath, storeName).mkString("/")

def mostRecentVkvs: VersionedKeyValSource[\_, \_] = {

VersionedKeyValSource[AggregationKey, (BatchID, StorageFormat)](

path = storePath,

sourceVersion = None,

maxFailures = maxFailures

)

}

private def availableVersions: Seq[Long] =

mostRecentVkvs

.getTap(TapMode.SOURCE)

.getStore(new JobConf(true))

.getAllVersions()

.asScala

.map(\_.toLong)

private def mostRecentVersion: Long = {

require(!availableVersions.isEmpty, s"$storeName has no available versions")

availableVersions.max

}

def versionToUse: Long =

if (enableMostRecentBeforeSourceVersion) {

sourceVersionOpt

.map(sourceVersion =>

availableVersions.filter(\_ <= sourceVersion) match {

case Seq() =>

throw new IllegalArgumentException(

"No version older than version: %s, available versions: %s"

.format(sourceVersion, availableVersions)

)

case versionList => versionList.max

})

.getOrElse(mostRecentVersion)

} else {

sourceVersionOpt.getOrElse(mostRecentVersion)

}

override lazy val adapter: IRecordOneToManyAdapter[AggregatesV2Tuple] =

new AggregatesV2Adapter(filteredAggregates, versionToUse, trimThreshold)

override def getData: TypedPipe[AggregatesV2Tuple] = {

val vkvsToUse: VersionedKeyValSource[AggregationKey, (BatchID, StorageFormat)] = {

VersionedKeyValSource[AggregationKey, (BatchID, StorageFormat)](

path = storePath,

sourceVersion = Some(versionToUse),

maxFailures = maxFailures

)

}

TypedPipe.from(vkvsToUse).map {

case (key, (batch, value)) => (key, (batch, toDataRecord(value)))

}

}

}

/\*

\* Adapted data record feature source from aggregates v2 manhattan output

\* Params documented in parent trait.

\*/

case class AggregatesV2FeatureSource(

override val rootPath: String,

override val storeName: String,

override val aggregates: Set[TypedAggregateGroup[\_]],

override val trimThreshold: Double = 0,

override val maxFailures: Int = 0,

)(

implicit val dateRange: DateRange)

extends AggregatesV2AdaptedSource {

// Increment end date by 1 millisec since summingbird output for date D is stored at (D+1)T00

override val sourceVersionOpt: Some[Long] = Some(dateRange.end.timestamp + 1)

}

/\*

\* Reads most recent available AggregatesV2FeatureSource.

\* There is no constraint on recency.

\* Params documented in parent trait.

\*/

case class AggregatesV2MostRecentFeatureSource(

override val rootPath: String,

override val storeName: String,

override val aggregates: Set[TypedAggregateGroup[\_]],

override val trimThreshold: Double = AggregatesV2AdaptedSource.DefaultTrimThreshold,

override val maxFailures: Int = 0)

extends AggregatesV2AdaptedSource {

override val sourceVersionOpt: None.type = None

}

/\*

\* Reads most recent available AggregatesV2FeatureSource

\* on or before the specified beforeDate.

\* Params documented in parent trait.

\*/

case class AggregatesV2MostRecentFeatureSourceBeforeDate(

override val rootPath: String,

override val storeName: String,

override val aggregates: Set[TypedAggregateGroup[\_]],

override val trimThreshold: Double = AggregatesV2AdaptedSource.DefaultTrimThreshold,

beforeDate: RichDate,

override val maxFailures: Int = 0)

extends AggregatesV2AdaptedSource {

override val enableMostRecentBeforeSourceVersion = true

override val sourceVersionOpt: Some[Long] = Some(beforeDate.timestamp + 1)

}