package com.twitter.timelines.prediction.common.aggregates.real\_time

import com.twitter.clientapp.thriftscala.LogEvent

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.stats.Counter

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.constant.SharedFeatures

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.summingbird.\_

import com.twitter.summingbird.storm.Storm

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

import com.twitter.summingbird\_internal.sources.storm.remote.ClientEventSourceScrooge2

import com.twitter.timelines.data\_processing.ad\_hoc.suggests.common.AllScribeProcessor

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

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

import com.twitter.timelines.prediction.adapters.client\_log\_event.ClientLogEventAdapter

import com.twitter.timelines.prediction.adapters.client\_log\_event.ProfileClientLogEventAdapter

import com.twitter.timelines.prediction.adapters.client\_log\_event.SearchClientLogEventAdapter

import com.twitter.timelines.prediction.adapters.client\_log\_event.UuaEventAdapter

import com.twitter.unified\_user\_actions.client.config.KafkaConfigs

import com.twitter.unified\_user\_actions.client.summingbird.UnifiedUserActionsSourceScrooge

import com.twitter.unified\_user\_actions.thriftscala.UnifiedUserAction

import scala.collection.JavaConverters.\_

/\*\*

\* Storm Producer for client events generated on Home, Profile, and Search

\*/

class TimelinesStormAggregateSource extends StormAggregateSource {

override val name = "timelines\_rta"

override val timestampFeature = SharedFeatures.TIMESTAMP

private lazy val TimelinesClientEventSourceName = "TL\_EVENTS\_SOURCE"

private lazy val ProfileClientEventSourceName = "PROFILE\_EVENTS\_SOURCE"

private lazy val SearchClientEventSourceName = "SEARCH\_EVENTS\_SOURCE"

private lazy val UuaEventSourceName = "UUA\_EVENTS\_SOURCE"

private lazy val CombinedProducerName = "COMBINED\_PRODUCER"

private lazy val FeatureStoreProducerName = "FEATURE\_STORE\_PRODUCER"

private def isNewUserEvent(event: LogEvent): Boolean = {

event.logBase.flatMap(\_.userId).flatMap(SnowflakeId.timeFromIdOpt).exists(\_.untilNow < 30.days)

}

private def mkDataRecords(event: LogEvent, dataRecordCounter: Counter): Seq[DataRecord] = {

val dataRecords: Seq[DataRecord] =

if (AllScribeProcessor.isValidSuggestTweetEvent(event)) {

ClientLogEventAdapter.adaptToDataRecords(event).asScala

} else {

Seq.empty[DataRecord]

}

dataRecordCounter.incr(dataRecords.size)

dataRecords

}

private def mkProfileDataRecords(

event: LogEvent,

dataRecordCounter: Counter

): Seq[DataRecord] = {

val dataRecords: Seq[DataRecord] =

ProfileClientLogEventAdapter.adaptToDataRecords(event).asScala

dataRecordCounter.incr(dataRecords.size)

dataRecords

}

private def mkSearchDataRecords(

event: LogEvent,

dataRecordCounter: Counter

): Seq[DataRecord] = {

val dataRecords: Seq[DataRecord] =

SearchClientLogEventAdapter.adaptToDataRecords(event).asScala

dataRecordCounter.incr(dataRecords.size)

dataRecords

}

private def mkUuaDataRecords(

event: UnifiedUserAction,

dataRecordCounter: Counter

): Seq[DataRecord] = {

val dataRecords: Seq[DataRecord] =

UuaEventAdapter.adaptToDataRecords(event).asScala

dataRecordCounter.incr(dataRecords.size)

dataRecords

}

override def build(

statsReceiver: StatsReceiver,

jobConfig: RealTimeAggregatesJobConfig

): Producer[Storm, DataRecord] = {

lazy val scopedStatsReceiver = statsReceiver.scope(getClass.getSimpleName)

lazy val dataRecordCounter = scopedStatsReceiver.counter("dataRecord")

// Home Timeline Engagements

// Step 1: => LogEvent

lazy val clientEventProducer: Producer[Storm, HomeEvent[LogEvent]] =

ClientEventSourceScrooge2(

appId = AppId(jobConfig.appId),

topic = "julep\_client\_event\_suggests",

resumeAtLastReadOffset = false,

enableTls = true

).source.map(HomeEvent[LogEvent]).name(TimelinesClientEventSourceName)

// Profile Engagements

// Step 1: => LogEvent

lazy val profileClientEventProducer: Producer[Storm, ProfileEvent[LogEvent]] =

ClientEventSourceScrooge2(

appId = AppId(jobConfig.appId),

topic = "julep\_client\_event\_profile\_real\_time\_engagement\_metrics",

resumeAtLastReadOffset = false,

enableTls = true

).source

.map(ProfileEvent[LogEvent])

.name(ProfileClientEventSourceName)

// Search Engagements

// Step 1: => LogEvent

// Only process events for all users to save resource

lazy val searchClientEventProducer: Producer[Storm, SearchEvent[LogEvent]] =

ClientEventSourceScrooge2(

appId = AppId(jobConfig.appId),

topic = "julep\_client\_event\_search\_real\_time\_engagement\_metrics",

resumeAtLastReadOffset = false,

enableTls = true

).source

.map(SearchEvent[LogEvent])

.name(SearchClientEventSourceName)

// Unified User Actions (includes Home and other product surfaces)

lazy val uuaEventProducer: Producer[Storm, UuaEvent[UnifiedUserAction]] =

UnifiedUserActionsSourceScrooge(

appId = AppId(jobConfig.appId),

parallelism = 10,

kafkaConfig = KafkaConfigs.ProdUnifiedUserActionsEngagementOnly

).source

.filter(StormAggregateSourceUtils.isUuaBCEEventsFromHome(\_))

.map(UuaEvent[UnifiedUserAction])

.name(UuaEventSourceName)

// Combined

// Step 2:

// (a) Combine

// (b) Transform LogEvent => Seq[DataRecord]

// (c) Apply sampler

lazy val combinedClientEventDataRecordProducer: Producer[Storm, Event[DataRecord]] =

profileClientEventProducer // This becomes the bottom branch

.merge(clientEventProducer) // This becomes the middle branch

.merge(searchClientEventProducer)

.merge(uuaEventProducer) // This becomes the top

.flatMap { // LogEvent => Seq[DataRecord]

case e: HomeEvent[LogEvent] =>

mkDataRecords(e.event, dataRecordCounter).map(HomeEvent[DataRecord])

case e: ProfileEvent[LogEvent] =>

mkProfileDataRecords(e.event, dataRecordCounter).map(ProfileEvent[DataRecord])

case e: SearchEvent[LogEvent] =>

mkSearchDataRecords(e.event, dataRecordCounter).map(SearchEvent[DataRecord])

case e: UuaEvent[UnifiedUserAction] =>

mkUuaDataRecords(

e.event,

dataRecordCounter

).map(UuaEvent[DataRecord])

}

.flatMap { // Apply sampler

case e: HomeEvent[DataRecord] =>

jobConfig.sequentiallyTransform(e.event).map(HomeEvent[DataRecord])

case e: ProfileEvent[DataRecord] =>

jobConfig.sequentiallyTransform(e.event).map(ProfileEvent[DataRecord])

case e: SearchEvent[DataRecord] =>

jobConfig.sequentiallyTransform(e.event).map(SearchEvent[DataRecord])

case e: UuaEvent[DataRecord] =>

jobConfig.sequentiallyTransform(e.event).map(UuaEvent[DataRecord])

}

.name(CombinedProducerName)

// Step 3: Join with Feature Store features

lazy val featureStoreDataRecordProducer: Producer[Storm, DataRecord] =

StormAggregateSourceUtils

.wrapByFeatureStoreClient(

underlyingProducer = combinedClientEventDataRecordProducer,

jobConfig = jobConfig,

scopedStatsReceiver = scopedStatsReceiver

).map(\_.event).name(FeatureStoreProducerName)

featureStoreDataRecordProducer

}

}