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

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.stats.DefaultStatsReceiver

import com.twitter.summingbird.Options

import com.twitter.summingbird.online.option.FlatMapParallelism

import com.twitter.summingbird.online.option.SourceParallelism

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

import com.twitter.timelines.data\_processing.ml\_util.transforms.DownsampleTransform

import com.twitter.timelines.data\_processing.ml\_util.transforms.RichITransform

import com.twitter.timelines.data\_processing.ml\_util.transforms.UserDownsampleTransform

import com.twitter.timelines.prediction.common.aggregates.BCELabelTransformFromUUADataRecord

/\*\*

\* Sets up relevant topology parameters. Our primary goal is to handle the

\* LogEvent stream and aggregate (sum) on the parsed DataRecords without falling

\* behind. Our constraint is the resulting write (and read) QPS to the backing

\* memcache store.

\*

\* If the job is falling behind, add more flatMappers and/or Summers after

\* inspecting the viz panels for the respective job (go/heron-ui). An increase in

\* Summers (and/or aggregation keys and features in the config) results in an

\* increase in memcache QPS (go/cb and search for our cache). Adjust with CacheSize

\* settings until QPS is well-controlled.

\*

\*/

object TimelinesRealTimeAggregatesJobConfigs extends RealTimeAggregatesJobConfigs {

import TimelinesOnlineAggregationUtils.\_

/\*\*

\* We remove input records that do not contain a label/engagement as defined in AllTweetLabels, which includes

\* explicit user engagements including public, private and impression events. By avoiding ingesting records without

\* engagemnts, we guarantee that no distribution shifts occur in computed aggregate features when we add a new spout

\* to input aggregate sources. Counterfactual signal is still available since we aggregate on explicit dwell

\* engagements.

\*/

val NegativeDownsampleTransform =

DownsampleTransform(

negativeSamplingRate = 0.0,

keepLabels = AllTweetLabels,

positiveSamplingRate = 1.0)

/\*\*

\* We downsample positive engagements for devel topology to reduce traffic, aiming for equivalent of 10% of prod traffic.

\* First apply consistent downsampling to 10% of users, and then apply downsampling to remove records without

\* explicit labels. We apply user-consistent sampling to more closely approximate prod query patterns.

\*/

val StagingUserBasedDownsampleTransform =

UserDownsampleTransform(

availability = 1000,

featureName = "rta\_devel"

)

override val Prod = RealTimeAggregatesJobConfig(

appId = "summingbird\_timelines\_rta",

topologyWorkers = 1450,

sourceCount = 120,

flatMapCount = 1800,

summerCount = 3850,

cacheSize = 200,

containerRamGigaBytes = 54,

name = "timelines\_real\_time\_aggregates",

teamName = "timelines",

teamEmail = "",

// If one component is hitting GC limit at prod, tune componentToMetaSpaceSizeMap.

// Except for Source bolts. Tune componentToRamGigaBytesMap for Source bolts instead.

componentToMetaSpaceSizeMap = Map(

"Tail-FlatMap" -> "-XX:MaxMetaspaceSize=1024M -XX:MetaspaceSize=1024M",

"Tail" -> "-XX:MaxMetaspaceSize=2560M -XX:MetaspaceSize=2560M"

),

// If either component is hitting memory limit at prod

// its memory need to increase: either increase total memory of container (containerRamGigaBytes),

// or allocate more memory for one component while keeping total memory unchanged.

componentToRamGigaBytesMap = Map(

"Tail-FlatMap-Source" -> 3, // Home source

"Tail-FlatMap-Source.2" -> 3, // Profile source

"Tail-FlatMap-Source.3" -> 3, // Search source

"Tail-FlatMap-Source.4" -> 3, // UUA source

"Tail-FlatMap" -> 8

// Tail will use the leftover memory in the container.

// Make sure to tune topologyWorkers and containerRamGigaBytes such that this is greater than 10 GB.

),

topologyNamedOptions = Map(

"TL\_EVENTS\_SOURCE" -> Options()

.set(SourceParallelism(120)),

"PROFILE\_EVENTS\_SOURCE" -> Options()

.set(SourceParallelism(30)),

"SEARCH\_EVENTS\_SOURCE" -> Options()

.set(SourceParallelism(10)),

"UUA\_EVENTS\_SOURCE" -> Options()

.set(SourceParallelism(10)),

"COMBINED\_PRODUCER" -> Options()

.set(FlatMapParallelism(1800))

),

// The UUA datarecord for BCE events inputted will not have binary labels populated.

// BCELabelTransform will set the datarecord with binary BCE dwell labels features based on the corresponding dwell\_time\_ms.

// It's important to have the BCELabelTransformFromUUADataRecord before ProdNegativeDownsampleTransform

// because ProdNegativeDownsampleTransform will remove datarecord that contains no features from AllTweetLabels.

onlinePreTransforms =

Seq(RichITransform(BCELabelTransformFromUUADataRecord), NegativeDownsampleTransform)

)

/\*\*

\* we downsample 10% computation of devel RTA based on [[StagingNegativeDownsampleTransform]].

\* To better test scalability of topology, we reduce computing resource of components "Tail-FlatMap"

\* and "Tail" to be 10% of prod but keep computing resource of component "Tail-FlatMap-Source" unchanged.

\* hence flatMapCount=110, summerCount=105 and sourceCount=100. Hence topologyWorkers =(110+105+100)/5 = 63.

\*/

override val Devel = RealTimeAggregatesJobConfig(

appId = "summingbird\_timelines\_rta\_devel",

topologyWorkers = 120,

sourceCount = 120,

flatMapCount = 150,

summerCount = 300,

cacheSize = 200,

containerRamGigaBytes = 54,

name = "timelines\_real\_time\_aggregates\_devel",

teamName = "timelines",

teamEmail = "",

// If one component is hitting GC limit at prod, tune componentToMetaSpaceSizeMap

// Except for Source bolts. Tune componentToRamGigaBytesMap for Source bolts instead.

componentToMetaSpaceSizeMap = Map(

"Tail-FlatMap" -> "-XX:MaxMetaspaceSize=1024M -XX:MetaspaceSize=1024M",

"Tail" -> "-XX:MaxMetaspaceSize=2560M -XX:MetaspaceSize=2560M"

),

// If either component is hitting memory limit at prod

// its memory need to increase: either increase total memory of container (containerRamGigaBytes),

// or allocate more memory for one component while keeping total memory unchanged.

componentToRamGigaBytesMap = Map(

"Tail-FlatMap-Source" -> 3, // Home source

"Tail-FlatMap-Source.2" -> 3, // Profile source

"Tail-FlatMap-Source.3" -> 3, // Search source

"Tail-FlatMap-Source.4" -> 3, // UUA source

"Tail-FlatMap" -> 8

// Tail will use the leftover memory in the container.

// Make sure to tune topologyWorkers and containerRamGigaBytes such that this is greater than 10 GB.

),

topologyNamedOptions = Map(

"TL\_EVENTS\_SOURCE" -> Options()

.set(SourceParallelism(120)),

"PROFILE\_EVENTS\_SOURCE" -> Options()

.set(SourceParallelism(30)),

"SEARCH\_EVENTS\_SOURCE" -> Options()

.set(SourceParallelism(10)),

"UUA\_EVENTS\_SOURCE" -> Options()

.set(SourceParallelism(10)),

"COMBINED\_PRODUCER" -> Options()

.set(FlatMapParallelism(150))

),

// It's important to have the BCELabelTransformFromUUADataRecord before ProdNegativeDownsampleTransform

onlinePreTransforms = Seq(

StagingUserBasedDownsampleTransform,

RichITransform(BCELabelTransformFromUUADataRecord),

NegativeDownsampleTransform),

enableUserReindexingNighthawkBtreeStore = true,

enableUserReindexingNighthawkHashStore = true,

userReindexingNighthawkBtreeStoreConfig = NighthawkUnderlyingStoreConfig(

serversetPath =

"/twitter/service/cache-user/test/nighthawk\_timelines\_real\_time\_aggregates\_btree\_test\_api",

// NOTE: table names are prefixed to every pkey so keep it short

tableName = "u\_r\_v1", // (u)ser\_(r)eindexing\_v1

// keep ttl <= 1 day because it's keyed on user, and we will have limited hit rates beyond 1 day

cacheTTL = 1.day

),

userReindexingNighthawkHashStoreConfig = NighthawkUnderlyingStoreConfig(

// For prod: "/s/cache-user/nighthawk\_timelines\_real\_time\_aggregates\_hash\_api",

serversetPath =

"/twitter/service/cache-user/test/nighthawk\_timelines\_real\_time\_aggregates\_hash\_test\_api",

// NOTE: table names are prefixed to every pkey so keep it short

tableName = "u\_r\_v1", // (u)ser\_(r)eindexing\_v1

// keep ttl <= 1 day because it's keyed on user, and we will have limited hit rates beyond 1 day

cacheTTL = 1.day

)

)

}

object TimelinesRealTimeAggregatesJob extends RealTimeAggregatesJobBase {

override lazy val statsReceiver = DefaultStatsReceiver.scope("timelines\_real\_time\_aggregates")

override lazy val jobConfigs = TimelinesRealTimeAggregatesJobConfigs

override lazy val aggregatesToCompute = TimelinesOnlineAggregationConfig.AggregatesToCompute

}