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

import com.twitter.simclusters\_v2.common.TweetId

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

import com.twitter.simclusters\_v2.summingbird.common.Monoids.PersistentSimClustersEmbeddingLongestL2NormMonoid

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

import com.twitter.simclusters\_v2.summingbird.stores.PersistentTweetEmbeddingStore.{

LatestEmbeddingVersion,

LongestL2EmbeddingVersion,

PersistentTweetEmbeddingId

}

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

PersistentSimClustersEmbedding,

SimClustersEmbedding,

SimClustersEmbeddingMetadata

}

import com.twitter.summingbird.option.JobId

import com.twitter.summingbird.{Platform, Producer, TailProducer}

import com.twitter.timelineservice.thriftscala.Event

import com.twitter.tweetypie.thriftscala.StatusCounts

/\*\*

\* The job to save the qualified tweet SimClustersEmbedding into Strato Store(Back by Manhattan).

\*

\* The steps

\* 1. Read from Favorite Stream.

\* 2. Join with Tweet Status Count Service.

\* 3. Filter out the tweets whose favorite count < 8.

\* We consider these tweets' SimClusters embedding is too noisy and untrustable.

\* 4. Update the SimClusters Tweet embedding with timestamp 0L.

\* 0L is reserved for the latest tweet embedding. It's also used to maintain the tweet count.

\* 5. If the SimClusters Tweet embedding's update count is 2 power N & N >= 3.

\* Persistent the embeddings with the timestamp as part of the LK.

\*\*/

private[storm] object PersistentTweetJob {

import StatsUtil.\_

private val MinFavoriteCount = 8

type Timestamp = Long

val longestL2NormMonoid = new PersistentSimClustersEmbeddingLongestL2NormMonoid()

def generate[P <: Platform[P]](

timelineEventSource: Producer[P, Event],

tweetStatusCountService: P#Service[TweetId, StatusCounts],

tweetEmbeddingService: P#Service[TweetId, SimClustersEmbedding],

persistentTweetEmbeddingStoreWithLatestAggregation: P#Store[

PersistentTweetEmbeddingId,

PersistentSimClustersEmbedding

],

persistentTweetEmbeddingStoreWithLongestL2NormAggregation: P#Store[

PersistentTweetEmbeddingId,

PersistentSimClustersEmbedding

]

)(

implicit jobId: JobId

): TailProducer[P, Any] = {

val timelineEvents: Producer[P, (TweetId, Timestamp)] = timelineEventSource

.collect {

case Event.Favorite(favoriteEvent) =>

(favoriteEvent.tweetId, favoriteEvent.eventTimeMs)

}

val filteredEvents = timelineEvents

.leftJoin[StatusCounts](tweetStatusCountService)

.filter {

case (\_, (\_, Some(statusCounts))) =>

// Only consider tweets which has more than 8 favorite

statusCounts.favoriteCount.exists(\_ >= MinFavoriteCount)

case \_ =>

false

}

.leftJoin[SimClustersEmbedding](tweetEmbeddingService)

val latestAndPersistentEmbeddingProducer = filteredEvents

.collect {

case (tweetId, ((eventTimeMs, \_), Some(tweetEmbedding))) =>

(

// This special timestamp is a reserved space for the latest tweet embedding.

PersistentTweetEmbeddingId(tweetId, timestampInMs = LatestEmbeddingVersion),

PersistentSimClustersEmbedding(

tweetEmbedding,

SimClustersEmbeddingMetadata(updatedAtMs = Some(eventTimeMs), updatedCount = Some(1))

))

}

.observe("num\_of\_embedding\_updates")

.sumByKey(persistentTweetEmbeddingStoreWithLatestAggregation)(

Implicits.persistentSimClustersEmbeddingMonoid)

.name("latest\_embedding\_producer")

.flatMap {

case (persistentTweetEmbeddingId, (maybeEmbedding, deltaEmbedding)) =>

lastQualifiedUpdatedCount(

maybeEmbedding.flatMap(\_.metadata.updatedCount),

deltaEmbedding.metadata.updatedCount

).map { newUpdateCount =>

(

persistentTweetEmbeddingId.copy(timestampInMs =

deltaEmbedding.metadata.updatedAtMs.getOrElse(0L)),

deltaEmbedding.copy(metadata =

deltaEmbedding.metadata.copy(updatedCount = Some(newUpdateCount)))

)

}

}

.observe("num\_of\_extra\_embedding")

.sumByKey(persistentTweetEmbeddingStoreWithLatestAggregation)(

Implicits.persistentSimClustersEmbeddingMonoid)

.name("persistent\_embeddings\_producer")

val longestL2NormEmbeddingProducer = filteredEvents

.collect {

case (tweetId, ((eventTimeMs, Some(statusCounts)), Some(tweetEmbedding))) =>

(

// This special timestamp is a reserved space for the latest tweet embedding.

PersistentTweetEmbeddingId(tweetId, timestampInMs = LongestL2EmbeddingVersion),

PersistentSimClustersEmbedding(

tweetEmbedding,

SimClustersEmbeddingMetadata(

updatedAtMs = Some(eventTimeMs),

// We're not aggregating the existing embedding, we're replacing it. The count

// therefore needs to be the absolute fav count for this tweet, not the delta.

updatedCount = statusCounts.favoriteCount.map(\_ + 1)

)

))

}

.observe("num\_longest\_l2\_norm\_updates")

.sumByKey(persistentTweetEmbeddingStoreWithLongestL2NormAggregation)(longestL2NormMonoid)

.name("longest\_l2\_norm\_embedding\_producer")

latestAndPersistentEmbeddingProducer.also(longestL2NormEmbeddingProducer)

}

/\*

If this change in counts crosses one or more powers of 2 (8,16,32...), return the last boundary

that was crossed. In the case where a count delta is large, it may skip a power of 2, and

thus we may not store embeddings for all 2^(i+3) where 0 <= i <= tweetFavCount.

\*/

private def lastQualifiedUpdatedCount(

existingUpdateCount: Option[Long],

deltaUpdateCount: Option[Long]

): Option[Int] = {

val existing = existingUpdateCount.getOrElse(0L)

val sum = existing + deltaUpdateCount.getOrElse(0L)

qualifiedSet.filter { i => (existing < i) && (i <= sum) }.lastOption

}

// Only 2 Power n while n >= 3 is qualified for Persistent. The max = 16,777,216

private lazy val qualifiedSet = 3

.until(25).map { i => Math.pow(2, i).toInt }.toSet

}