package com.twitter.usersignalservice.signals

import com.twitter.finagle.stats.StatsReceiver

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

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

import com.twitter.strato.client.Client

import com.twitter.strato.data.Conv

import com.twitter.strato.thrift.ScroogeConv

import com.twitter.twistly.thriftscala.RecentTweetClickImpressEvents

import com.twitter.twistly.thriftscala.TweetClickImpressEvent

import com.twitter.usersignalservice.base.Query

import com.twitter.usersignalservice.base.StratoSignalFetcher

import com.twitter.usersignalservice.thriftscala.Signal

import com.twitter.usersignalservice.thriftscala.SignalType

import com.twitter.util.Future

import com.twitter.util.Timer

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

case class TweetClickFetcher @Inject() (

stratoClient: Client,

timer: Timer,

stats: StatsReceiver)

extends StratoSignalFetcher[(UserId, Long), Unit, RecentTweetClickImpressEvents] {

import TweetClickFetcher.\_

override type RawSignalType = TweetClickImpressEvent

override val name: String = this.getClass.getCanonicalName

override val statsReceiver: StatsReceiver = stats.scope(name)

override val stratoColumnPath: String = stratoPath

override val stratoView: Unit = None

override protected val keyConv: Conv[(UserId, Long)] = Conv.ofType

override protected val viewConv: Conv[Unit] = Conv.ofType

override protected val valueConv: Conv[RecentTweetClickImpressEvents] =

ScroogeConv.fromStruct[RecentTweetClickImpressEvents]

override protected def toStratoKey(userId: UserId): (UserId, Long) = (userId, defaultVersion)

override protected def toRawSignals(

stratoValue: RecentTweetClickImpressEvents

): Seq[TweetClickImpressEvent] = {

stratoValue.events

}

override def process(

query: Query,

rawSignals: Future[Option[Seq[TweetClickImpressEvent]]]

): Future[Option[Seq[Signal]]] =

rawSignals.map { events =>

events.map { clicks =>

clicks

.filter(dwelltimeFilter(\_, query.signalType))

.map(signalFromTweetClick(\_, query.signalType))

.sortBy(-\_.timestamp)

.take(query.maxResults.getOrElse(Int.MaxValue))

}

}

}

object TweetClickFetcher {

val stratoPath = "recommendations/twistly/userRecentTweetClickImpress"

private val defaultVersion = 0L

private val minDwellTimeMap: Map[SignalType, Long] = Map(

SignalType.GoodTweetClick -> 2 \* 1000L,

SignalType.GoodTweetClick5s -> 5 \* 1000L,

SignalType.GoodTweetClick10s -> 10 \* 1000L,

SignalType.GoodTweetClick30s -> 30 \* 1000L,

)

def signalFromTweetClick(

tweetClickImpressEvent: TweetClickImpressEvent,

signalType: SignalType

): Signal = {

Signal(

signalType,

tweetClickImpressEvent.engagedAt,

Some(InternalId.TweetId(tweetClickImpressEvent.entityId))

)

}

def dwelltimeFilter(

tweetClickImpressEvent: TweetClickImpressEvent,

signalType: SignalType

): Boolean = {

val goodClickDwellTime = minDwellTimeMap(signalType)

tweetClickImpressEvent.clickImpressEventMetadata.totalDwellTime >= goodClickDwellTime

}

}