package com.twitter.usersignalservice.signals

import com.twitter.bijection.Codec

import com.twitter.bijection.scrooge.BinaryScalaCodec

import com.twitter.dds.jobs.repeated\_profile\_visits.thriftscala.ProfileVisitSet

import com.twitter.dds.jobs.repeated\_profile\_visits.thriftscala.ProfileVisitorInfo

import com.twitter.experiments.general\_metrics.thriftscala.IdType

import com.twitter.finagle.stats.StatsReceiver

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

import com.twitter.storage.client.manhattan.kv.ManhattanKVClientMtlsParams

import com.twitter.storehaus\_internal.manhattan.Apollo

import com.twitter.storehaus\_internal.manhattan.ManhattanCluster

import com.twitter.twistly.common.UserId

import com.twitter.usersignalservice.base.ManhattanSignalFetcher

import com.twitter.usersignalservice.base.Query

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

case class ProfileVisitMetadata(

targetId: Option[Long],

totalTargetVisitsInLast14Days: Option[Int],

totalTargetVisitsInLast90Days: Option[Int],

totalTargetVisitsInLast180Days: Option[Int],

latestTargetVisitTimestampInLast90Days: Option[Long])

@Singleton

case class ProfileVisitsFetcher @Inject() (

manhattanKVClientMtlsParams: ManhattanKVClientMtlsParams,

timer: Timer,

stats: StatsReceiver)

extends ManhattanSignalFetcher[ProfileVisitorInfo, ProfileVisitSet] {

import ProfileVisitsFetcher.\_

override type RawSignalType = ProfileVisitMetadata

override val manhattanAppId: String = MHAppId

override val manhattanDatasetName: String = MHDatasetName

override val manhattanClusterId: ManhattanCluster = Apollo

override val manhattanKeyCodec: Codec[ProfileVisitorInfo] = BinaryScalaCodec(ProfileVisitorInfo)

override val manhattanRawSignalCodec: Codec[ProfileVisitSet] = BinaryScalaCodec(ProfileVisitSet)

override protected def toManhattanKey(userId: UserId): ProfileVisitorInfo =

ProfileVisitorInfo(userId, IdType.User)

override protected def toRawSignals(manhattanValue: ProfileVisitSet): Seq[ProfileVisitMetadata] =

manhattanValue.profileVisitSet

.map {

\_.collect {

// only keep the Non-NSFW and not-following profile visits

case profileVisit

if profileVisit.targetId.nonEmpty

// The below check covers 180 days, not only 90 days as the name implies.

// See comment on [[ProfileVisit.latestTargetVisitTimestampInLast90Days]] thrift.

&& profileVisit.latestTargetVisitTimestampInLast90Days.nonEmpty

&& !profileVisit.isTargetNSFW.getOrElse(false)

&& !profileVisit.doesSourceIdFollowTargetId.getOrElse(false) =>

ProfileVisitMetadata(

targetId = profileVisit.targetId,

totalTargetVisitsInLast14Days = profileVisit.totalTargetVisitsInLast14Days,

totalTargetVisitsInLast90Days = profileVisit.totalTargetVisitsInLast90Days,

totalTargetVisitsInLast180Days = profileVisit.totalTargetVisitsInLast180Days,

latestTargetVisitTimestampInLast90Days =

profileVisit.latestTargetVisitTimestampInLast90Days

)

}.toSeq

}.getOrElse(Seq.empty)

override val name: String = this.getClass.getCanonicalName

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

override def process(

query: Query,

rawSignals: Future[Option[Seq[ProfileVisitMetadata]]]

): Future[Option[Seq[Signal]]] = rawSignals.map { profiles =>

profiles

.map {

\_.filter(profileVisitMetadata => visitCountFilter(profileVisitMetadata, query.signalType))

.sortBy(profileVisitMetadata =>

-visitCountMap(query.signalType)(profileVisitMetadata).getOrElse(0))

.map(profileVisitMetadata =>

signalFromProfileVisit(profileVisitMetadata, query.signalType))

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

}

}

}

object ProfileVisitsFetcher {

private val MHAppId = "repeated\_profile\_visits\_aggregated"

private val MHDatasetName = "repeated\_profile\_visits\_aggregated"

private val minVisitCountMap: Map[SignalType, Int] = Map(

SignalType.RepeatedProfileVisit14dMinVisit2V1 -> 2,

SignalType.RepeatedProfileVisit14dMinVisit2V1NoNegative -> 2,

SignalType.RepeatedProfileVisit90dMinVisit6V1 -> 6,

SignalType.RepeatedProfileVisit90dMinVisit6V1NoNegative -> 6,

SignalType.RepeatedProfileVisit180dMinVisit6V1 -> 6,

SignalType.RepeatedProfileVisit180dMinVisit6V1NoNegative -> 6

)

private val visitCountMap: Map[SignalType, ProfileVisitMetadata => Option[Int]] = Map(

SignalType.RepeatedProfileVisit14dMinVisit2V1 ->

((profileVisitMetadata: ProfileVisitMetadata) =>

profileVisitMetadata.totalTargetVisitsInLast14Days),

SignalType.RepeatedProfileVisit14dMinVisit2V1NoNegative ->

((profileVisitMetadata: ProfileVisitMetadata) =>

profileVisitMetadata.totalTargetVisitsInLast14Days),

SignalType.RepeatedProfileVisit90dMinVisit6V1 ->

((profileVisitMetadata: ProfileVisitMetadata) =>

profileVisitMetadata.totalTargetVisitsInLast90Days),

SignalType.RepeatedProfileVisit90dMinVisit6V1NoNegative ->

((profileVisitMetadata: ProfileVisitMetadata) =>

profileVisitMetadata.totalTargetVisitsInLast90Days),

SignalType.RepeatedProfileVisit180dMinVisit6V1 ->

((profileVisitMetadata: ProfileVisitMetadata) =>

profileVisitMetadata.totalTargetVisitsInLast180Days),

SignalType.RepeatedProfileVisit180dMinVisit6V1NoNegative ->

((profileVisitMetadata: ProfileVisitMetadata) =>

profileVisitMetadata.totalTargetVisitsInLast180Days)

)

def signalFromProfileVisit(

profileVisitMetadata: ProfileVisitMetadata,

signalType: SignalType

): Signal = {

Signal(

signalType,

profileVisitMetadata.latestTargetVisitTimestampInLast90Days.get,

profileVisitMetadata.targetId.map(targetId => InternalId.UserId(targetId))

)

}

def visitCountFilter(

profileVisitMetadata: ProfileVisitMetadata,

signalType: SignalType

): Boolean = {

visitCountMap(signalType)(profileVisitMetadata).exists(\_ >= minVisitCountMap(signalType))

}

}