package com.twitter.frigate.pushservice.util

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.TargetUser

import com.twitter.frigate.common.candidate.FrigateHistory

import com.twitter.frigate.common.candidate.ResurrectedUserDetails

import com.twitter.frigate.common.candidate.TargetABDecider

import com.twitter.frigate.common.candidate.UserDetails

import com.twitter.frigate.pushcap.thriftscala.ModelType

import com.twitter.frigate.pushcap.thriftscala.PushcapInfo

import com.twitter.frigate.pushservice.model.PushTypes.Target

import com.twitter.frigate.pushservice.params.PushFeatureSwitchParams

import com.twitter.frigate.scribe.thriftscala.PushCapInfo

import com.twitter.util.Duration

import com.twitter.util.Future

case class PushCapFatigueInfo(

pushcap: Int,

fatigueInterval: Duration) {}

object PushCapUtil {

def getDefaultPushCap(target: Target): Future[Int] = {

Future.value(target.params(PushFeatureSwitchParams.MaxMrPushSends24HoursParam))

}

def getMinimumRestrictedPushcapInfo(

restrictedPushcap: Int,

originalPushcapInfo: PushcapInfo,

statsReceiver: StatsReceiver

): PushcapInfo = {

if (originalPushcapInfo.pushcap < restrictedPushcap) {

statsReceiver

.scope("minModelPushcapRestrictions").counter(

f"num\_users\_adjusted\_from\_${originalPushcapInfo.pushcap}\_to\_${restrictedPushcap}").incr()

PushcapInfo(

pushcap = restrictedPushcap.toShort,

modelType = ModelType.NoModel,

timestamp = 0L,

fatigueMinutes = Some((24L / restrictedPushcap) \* 60L)

)

} else originalPushcapInfo

}

def getPushCapFatigue(

target: Target,

statsReceiver: StatsReceiver

): Future[PushCapFatigueInfo] = {

val pushCapStats = statsReceiver.scope("pushcap\_stats")

target.dynamicPushcap

.map { dynamicPushcapOpt =>

val pushCap: Int = dynamicPushcapOpt match {

case Some(pushcapInfo) => pushcapInfo.pushcap

case \_ => target.params(PushFeatureSwitchParams.MaxMrPushSends24HoursParam)

}

pushCapStats.stat("pushCapValueStats").add(pushCap)

pushCapStats

.scope("pushCapValueCount").counter(f"num\_users\_with\_pushcap\_$pushCap").incr()

target.finalPushcapAndFatigue += "pushPushCap" -> PushCapInfo("pushPushCap", pushCap.toByte)

PushCapFatigueInfo(pushCap, 24.hours)

}

}

def getMinDurationsSincePushWithoutUsingPushCap(

target: TargetUser

with TargetABDecider

with FrigateHistory

with UserDetails

with ResurrectedUserDetails

)(

implicit statsReceiver: StatsReceiver

): Duration = {

val minDurationSincePush =

if (target.params(PushFeatureSwitchParams.EnableGraduallyRampUpNotification)) {

val daysInterval =

target.params(PushFeatureSwitchParams.GraduallyRampUpPhaseDurationDays).inDays.toDouble

val daysSinceActivation =

if (target.isResurrectedUser && target.timeSinceResurrection.isDefined) {

target.timeSinceResurrection.map(\_.inDays.toDouble).get

} else {

target.timeElapsedAfterSignup.inDays.toDouble

}

val phaseInterval =

Math.max(

1,

Math.ceil(daysSinceActivation / daysInterval).toInt

)

val minDuration = 24 / phaseInterval

val finalMinDuration =

Math.max(4, minDuration).hours

statsReceiver

.scope("GraduallyRampUpFinalMinDuration").counter(s"$finalMinDuration.hours").incr()

finalMinDuration

} else {

target.params(PushFeatureSwitchParams.MinDurationSincePushParam)

}

statsReceiver

.scope("minDurationsSincePushWithoutUsingPushCap").counter(

s"$minDurationSincePush.hours").incr()

minDurationSincePush

}

def getMinDurationSincePush(

target: Target,

statsReceiver: StatsReceiver

): Future[Duration] = {

val minDurationStats: StatsReceiver = statsReceiver.scope("pushcapMinDuration\_stats")

val minDurationModifierCalculator =

MinDurationModifierCalculator()

val openedPushByHourAggregatedFut =

if (target.params(PushFeatureSwitchParams.EnableQueryUserOpenedHistory))

target.openedPushByHourAggregated

else Future.None

Future

.join(

target.dynamicPushcap,

target.accountCountryCode,

openedPushByHourAggregatedFut

)

.map {

case (dynamicPushcapOpt, countryCodeOpt, openedPushByHourAggregated) =>

val minDurationSincePush: Duration = {

val isGraduallyRampingUpResurrected = target.isResurrectedUser && target.params(

PushFeatureSwitchParams.EnableGraduallyRampUpNotification)

if (isGraduallyRampingUpResurrected || target.params(

PushFeatureSwitchParams.EnableExplicitPushCap)) {

getMinDurationsSincePushWithoutUsingPushCap(target)(minDurationStats)

} else {

dynamicPushcapOpt match {

case Some(pushcapInfo) =>

pushcapInfo.fatigueMinutes match {

case Some(fatigueMinutes) => (fatigueMinutes / 60).hours

case \_ if pushcapInfo.pushcap > 0 => (24 / pushcapInfo.pushcap).hours

case \_ => getMinDurationsSincePushWithoutUsingPushCap(target)(minDurationStats)

}

case \_ =>

getMinDurationsSincePushWithoutUsingPushCap(target)(minDurationStats)

}

}

}

val modifiedMinDurationSincePush =

if (target.params(PushFeatureSwitchParams.EnableMinDurationModifier)) {

val modifierHourOpt =

minDurationModifierCalculator.getMinDurationModifier(

target,

countryCodeOpt,

statsReceiver.scope("MinDuration"))

modifierHourOpt match {

case Some(modifierHour) => modifierHour.hours

case \_ => minDurationSincePush

}

} else if (target.params(

PushFeatureSwitchParams.EnableMinDurationModifierByUserHistory)) {

val modifierMinuteOpt =

minDurationModifierCalculator.getMinDurationModifierByUserOpenedHistory(

target,

openedPushByHourAggregated,

statsReceiver.scope("MinDuration"))

modifierMinuteOpt match {

case Some(modifierMinute) => modifierMinute.minutes

case \_ => minDurationSincePush

}

} else minDurationSincePush

target.finalPushcapAndFatigue += "pushFatigue" -> PushCapInfo(

"pushFatigue",

modifiedMinDurationSincePush.inHours.toByte)

minDurationStats

.stat("minDurationSincePushValueStats").add(modifiedMinDurationSincePush.inHours)

minDurationStats

.scope("minDurationSincePushValueCount").counter(

s"$modifiedMinDurationSincePush").incr()

modifiedMinDurationSincePush

}

}

}