package com.twitter.frigate.pushservice.model.ibis

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.history.History

import com.twitter.frigate.common.rec\_types.RecTypes

import com.twitter.frigate.thriftscala.CommonRecommendationType

import com.twitter.frigate.thriftscala.FrigateNotification

import com.twitter.frigate.thriftscala.OverrideInfo

import com.twitter.util.Duration

import com.twitter.util.Time

object PushOverrideInfo {

private val name: String = this.getClass.getSimpleName

/\*\*

\* Gets all eligible time + override push notification pairs from a target's History

\*

\* @param history: history of push notifications

\* @param lookbackDuration: duration to look back up in history for overriding notifications

\* @return: list of notifications with send timestamps which are eligible for overriding

\*/

def getOverrideEligibleHistory(

history: History,

lookbackDuration: Duration,

): Seq[(Time, FrigateNotification)] = {

history.sortedHistory

.takeWhile { case (notifTimestamp, \_) => lookbackDuration.ago < notifTimestamp }

.filter {

case (\_, notification) => notification.overrideInfo.isDefined

}

}

/\*\*

\* Gets all eligible override push notifications from a target's History

\*

\* @param history Target's History

\* @param lookbackDuration Duration in which we would like to obtain the eligible push notifications

\* @param stats StatsReceiver to track stats for this function

\* @return Returns a list of FrigateNotification

\*/

def getOverrideEligiblePushNotifications(

history: History,

lookbackDuration: Duration,

stats: StatsReceiver,

): Seq[FrigateNotification] = {

val eligibleNotificationsDistribution =

stats.scope(name).stat("eligible\_notifications\_size\_distribution")

val eligibleNotificationsSeq =

getOverrideEligibleHistory(history, lookbackDuration)

.collect {

case (\_, notification) => notification

}

eligibleNotificationsDistribution.add(eligibleNotificationsSeq.size)

eligibleNotificationsSeq

}

/\*\*

\* Gets the OverrideInfo for the last eligible Override Notification FrigateNotification, if it exists

\* @param history Target's History

\* @param lookbackDuration Duration in which we would like to obtain the last override notification

\* @param stats StatsReceiver to track stats for this function

\* @return Returns OverrideInfo of the last MR push, else None

\*/

def getOverrideInfoOfLastEligiblePushNotif(

history: History,

lookbackDuration: Duration,

stats: StatsReceiver

): Option[OverrideInfo] = {

val overrideInfoEmptyOfLastPush = stats.scope(name).counter("override\_info\_empty\_of\_last\_push")

val overrideInfoExistsForLastPush =

stats.scope(name).counter("override\_info\_exists\_for\_last\_push")

val overrideHistory =

getOverrideEligiblePushNotifications(history, lookbackDuration, stats)

if (overrideHistory.isEmpty) {

overrideInfoEmptyOfLastPush.incr()

None

} else {

overrideInfoExistsForLastPush.incr()

overrideHistory.head.overrideInfo

}

}

/\*\*

\* Gets all the MR Push Notifications in the specified override chain

\* @param history Target's History

\* @param overrideChainId Override Chain Identifier

\* @param stats StatsReceiver to track stats for this function

\* @return Returns a sequence of FrigateNotification that exist in the override chain

\*/

def getMrPushNotificationsInOverrideChain(

history: History,

overrideChainId: String,

stats: StatsReceiver

): Seq[FrigateNotification] = {

val notificationInOverrideChain = stats.scope(name).counter("notification\_in\_override\_chain")

val notificationNotInOverrideChain =

stats.scope(name).counter("notification\_not\_in\_override\_chain")

history.sortedHistory.flatMap {

case (\_, notification)

if isNotificationInOverrideChain(notification, overrideChainId, stats) =>

notificationInOverrideChain.incr()

Some(notification)

case \_ =>

notificationNotInOverrideChain.incr()

None

}

}

/\*\*

\* Gets the timestamp (in milliseconds) for the specified FrigateNotification

\* @param notification The FrigateNotification that we would like the timestamp for

\* @param history Target's History

\* @param stats StatsReceiver to track stats for this function

\* @return Returns the timestamp in milliseconds for the specified notification

\* if it exists History, else None

\*/

def getTimestampInMillisForFrigateNotification(

notification: FrigateNotification,

history: History,

stats: StatsReceiver

): Option[Long] = {

val foundTimestampOfNotificationInHistory =

stats.scope(name).counter("found\_timestamp\_of\_notification\_in\_history")

history.sortedHistory

.find(\_.\_2.equals(notification)).map {

case (time, \_) =>

foundTimestampOfNotificationInHistory.incr()

time.inMilliseconds

}

}

/\*\*

\* Gets the oldest frigate notification based on the user's NTab last read position

\* @param overrideCandidatesMap All the NTab Notifications in the override chain

\* @return Returns the oldest frigate notification in the chain

\*/

def getOldestFrigateNotification(

overrideCandidatesMap: Map[Long, FrigateNotification],

): FrigateNotification = {

overrideCandidatesMap.minBy(\_.\_1).\_2

}

/\*\*

\* Gets the impression ids of previous eligible push notification.

\* @param history Target's History

\* @param lookbackDuration Duration in which we would like to obtain previous impression ids

\* @param stats StatsReceiver to track stats for this function

\* @return Returns the impression identifier for the last eligible push notif.

\* if it exists in the target's History, else None.

\*/

def getImpressionIdsOfPrevEligiblePushNotif(

history: History,

lookbackDuration: Duration,

stats: StatsReceiver

): Seq[String] = {

val foundImpressionIdOfLastEligiblePushNotif =

stats.scope(name).counter("found\_impression\_id\_of\_last\_eligible\_push\_notif")

val overrideHistoryEmptyWhenFetchingImpressionId =

stats.scope(name).counter("override\_history\_empty\_when\_fetching\_impression\_id")

val overrideHistory = getOverrideEligiblePushNotifications(history, lookbackDuration, stats)

.filter(frigateNotification =>

// Exclude notifications of nonGenericOverrideTypes from being overridden

!RecTypes.nonGenericOverrideTypes.contains(frigateNotification.commonRecommendationType))

if (overrideHistory.isEmpty) {

overrideHistoryEmptyWhenFetchingImpressionId.incr()

Seq.empty

} else {

foundImpressionIdOfLastEligiblePushNotif.incr()

overrideHistory.flatMap(\_.impressionId)

}

}

/\*\*

\* Gets the impressions ids by eventId, for MagicFanoutEvent candidates.

\*

\* @param history Target's History

\* @param lookbackDuration Duration in which we would like to obtain previous impression ids

\* @param stats StatsReceiver to track stats for this function

\* @param overridableType Specific MagicFanoutEvent CRT

\* @param eventId Event identifier for MagicFanoutEventCandidate.

\* @return Returns the impression identifiers for the last eligible, eventId-matching

\* MagicFanoutEvent push notifications if they exist in the target's history, else None.

\*/

def getImpressionIdsForPrevEligibleMagicFanoutEventCandidates(

history: History,

lookbackDuration: Duration,

stats: StatsReceiver,

overridableType: CommonRecommendationType,

eventId: Long

): Seq[String] = {

val foundImpressionIdOfMagicFanoutEventNotif =

stats.scope(name).counter("found\_impression\_id\_of\_magic\_fanout\_event\_notif")

val overrideHistoryEmptyWhenFetchingImpressionId =

stats

.scope(name).counter(

"override\_history\_empty\_when\_fetching\_impression\_id\_for\_magic\_fanout\_event\_notif")

val overrideHistory =

getOverrideEligiblePushNotifications(history, lookbackDuration, stats)

.filter(frigateNotification =>

// Only override notifications with same CRT and eventId

frigateNotification.commonRecommendationType == overridableType &&

frigateNotification.magicFanoutEventNotification.exists(\_.eventId == eventId))

if (overrideHistory.isEmpty) {

overrideHistoryEmptyWhenFetchingImpressionId.incr()

Seq.empty

} else {

foundImpressionIdOfMagicFanoutEventNotif.incr()

overrideHistory.flatMap(\_.impressionId)

}

}

/\*\*

\* Determines if the provided notification is part of the specified override chain

\* @param notification FrigateNotification that we're trying to identify as within the override chain

\* @param overrideChainId Override Chain Identifier

\* @param stats StatsReceiver to track stats for this function

\* @return Returns true if the provided FrigateNotification is within the override chain, else false

\*/

private def isNotificationInOverrideChain(

notification: FrigateNotification,

overrideChainId: String,

stats: StatsReceiver

): Boolean = {

val notifIsInOverrideChain = stats.scope(name).counter("notif\_is\_in\_override\_chain")

val notifNotInOverrideChain = stats.scope(name).counter("notif\_not\_in\_override\_chain")

notification.overrideInfo match {

case Some(overrideInfo) =>

val isNotifInOverrideChain = overrideInfo.collapseInfo.overrideChainId == overrideChainId

if (isNotifInOverrideChain) {

notifIsInOverrideChain.incr()

true

} else {

notifNotInOverrideChain.incr()

false

}

case \_ =>

notifNotInOverrideChain.incr()

false

}

}

}