package com.twitter.frigate.pushservice.util

import com.twitter.finagle.stats.Counter

import com.twitter.finagle.stats.Stat

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.Invalid

import com.twitter.frigate.common.base.OK

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

import com.twitter.frigate.pushservice.refresh\_handler.ResultWithDebugInfo

import com.twitter.frigate.pushservice.predicate.BigFilteringEpsilonGreedyExplorationPredicate

import com.twitter.frigate.pushservice.predicate.MlModelsHoldbackExperimentPredicate

import com.twitter.frigate.pushservice.take.candidate\_validator.RFPHCandidateValidator

import com.twitter.frigate.pushservice.thriftscala.PushStatus

import com.twitter.hermit.predicate.NamedPredicate

import com.twitter.util.Future

class RFPHTakeStepUtil()(globalStats: StatsReceiver) {

implicit val statsReceiver: StatsReceiver =

globalStats.scope("RefreshForPushHandler")

private val takeStats: StatsReceiver = statsReceiver.scope("take")

private val notifierStats = takeStats.scope("notifier")

private val validatorStats = takeStats.scope("validator")

private val validatorLatency: Stat = validatorStats.stat("latency")

private val executedPredicatesInTandem: Counter =

takeStats.counter("predicates\_executed\_in\_tandem")

private val bigFilteringEpsGreedyPredicate: NamedPredicate[PushCandidate] =

BigFilteringEpsilonGreedyExplorationPredicate()(takeStats)

private val bigFilteringEpsGreedyStats: StatsReceiver =

takeStats.scope("big\_filtering\_eps\_greedy\_predicate")

private val modelPredicate: NamedPredicate[PushCandidate] =

MlModelsHoldbackExperimentPredicate()(takeStats)

private val mlPredicateStats: StatsReceiver = takeStats.scope("ml\_predicate")

private def updateFilteredStatusExptStats(candidate: PushCandidate, predName: String): Unit = {

val recTypeStat = globalStats.scope(

candidate.commonRecType.toString

)

recTypeStat.counter(PushStatus.Filtered.toString).incr()

recTypeStat

.scope(PushStatus.Filtered.toString)

.counter(predName)

.incr()

}

def isCandidateValid(

candidate: PushCandidate,

candidateValidator: RFPHCandidateValidator

): Future[ResultWithDebugInfo] = {

val predResultFuture = Stat.timeFuture(validatorLatency) {

Future

.join(

bigFilteringEpsGreedyPredicate.apply(Seq(candidate)),

modelPredicate.apply(Seq(candidate))

).flatMap {

case (Seq(true), Seq(true)) =>

executedPredicatesInTandem.incr()

bigFilteringEpsGreedyStats

.scope(candidate.commonRecType.toString)

.counter("passed")

.incr()

mlPredicateStats

.scope(candidate.commonRecType.toString)

.counter("passed")

.incr()

candidateValidator.validateCandidate(candidate).map((\_, Nil))

case (Seq(false), \_) =>

bigFilteringEpsGreedyStats

.scope(candidate.commonRecType.toString)

.counter("filtered")

.incr()

Future.value((Some(bigFilteringEpsGreedyPredicate), Nil))

case (\_, \_) =>

mlPredicateStats

.scope(candidate.commonRecType.toString)

.counter("filtered")

.incr()

Future.value((Some(modelPredicate), Nil))

}

}

predResultFuture.map {

case (Some(pred: NamedPredicate[\_]), candPredicateResults) =>

takeStats.counter("filtered\_by\_named\_general\_predicate").incr()

updateFilteredStatusExptStats(candidate, pred.name)

ResultWithDebugInfo(

Invalid(Some(pred.name)),

candPredicateResults

)

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

takeStats.counter("filtered\_by\_unnamed\_general\_predicate").incr()

updateFilteredStatusExptStats(candidate, predName = "unk")

ResultWithDebugInfo(

Invalid(Some("unnamed\_candidate\_predicate")),

candPredicateResults

)

case (None, candPredicateResults) =>

takeStats.counter("accepted\_push\_ok").incr()

ResultWithDebugInfo(

OK,

candPredicateResults

)

}

}

}