package com.twitter.product\_mixer.core.quality\_factor

import com.google.common.annotations.VisibleForTesting

import com.twitter.util.Stopwatch

case class QueriesPerSecondBasedQualityFactor(

override val config: QueriesPerSecondBasedQualityFactorConfig)

extends QualityFactor[Int] {

@VisibleForTesting

private[quality\_factor] val queryRateCounter: QueryRateCounter = QueryRateCounter(

config.queriesPerSecondSampleWindow)

private val delayedUntilInMillis = Stopwatch.timeMillis() + config.initialDelay.inMillis

private var state: Double = config.qualityFactorBounds.default

override def currentValue: Double = state

override def update(count: Int = 1): Unit = {

val queryRate = incrementAndGetQueryRateCount(count)

// Only update quality factor until the initial delay past.

// This allows query rate counter get warm up to reflect

// actual traffic load by the time initial delay expires.

if (Stopwatch.timeMillis() >= delayedUntilInMillis) {

if (queryRate > config.maxQueriesPerSecond) {

state = config.qualityFactorBounds.bounds(state - config.delta)

} else {

state = config.qualityFactorBounds.bounds(state + config.delta)

}

}

}

private def incrementAndGetQueryRateCount(count: Int): Double = {

// Int.MaxValue is used as a special signal from [[QueriesPerSecondBasedQualityFactorObserver]]

// to indicate a component failure is observed.

// In this case, we do not update queryRateCounter and instead return Int.MaxValue.

// As the largest Int value, this should result in the threshold qps for quality factor being

// exceeded and directly decrementing quality factor.

if (count == Int.MaxValue) {

Int.MaxValue.toDouble

} else {

queryRateCounter.increment(count)

queryRateCounter.getRate()

}

}

override def buildObserver(): QualityFactorObserver =

QueriesPerSecondBasedQualityFactorObserver(this)

}