package com.twitter.servo.util

import com.twitter.util.{Duration, Time}

/\*\*

\* Calculate a running average of data points

\*/

trait Average {

def value: Option[Double]

def record(dataPoint: Double, count: Double = 1.0): Unit

}

/\*\*

\* Calculates a running average using two windows of data points, a

\* current one and a previous one. When the current window is full,

\* it is rolled into the previous and the current window starts

\* filling up again.

\*/

class WindowedAverage(val windowSize: Long, initialValue: Option[Double] = None) extends Average {

private[this] val average = new ResettableAverage(None)

private[this] var lastAverage: Option[Double] = initialValue

def value: Option[Double] =

synchronized {

lastAverage match {

case Some(lastAvg) =>

// currentCount can temporarily exceed windowSize

val currentWeight = (average.count / windowSize) min 1.0

Some((1.0 - currentWeight) \* lastAvg + currentWeight \* average.value.getOrElse(0.0))

case None => average.value

}

}

def record(dataPoint: Double, count: Double = 1.0): Unit =

synchronized {

if (average.count >= windowSize) {

lastAverage = value

average.reset()

}

average.record(dataPoint, count)

}

}

/\*\*

\* Calculates a recent average using the past windowDuration of data points. Old average is mixed

\* with the new average during windowDuration. If new data points are not recorded the average

\* will revert towards defaultAverage.

\*/

class RecentAverage(

val windowDuration: Duration,

val defaultAverage: Double,

currentTime: Time = Time.now // passing in start time to simplify scalacheck tests

) extends Average {

private[this] val default = Some(defaultAverage)

private[this] val currentAverage = new ResettableAverage(Some(defaultAverage))

private[this] var prevAverage: Option[Double] = None

private[this] var windowStart: Time = currentTime

private[this] def mix(fractOfV2: Double, v1: Double, v2: Double): Double = {

val f = 0.0.max(1.0.min(fractOfV2))

(1.0 - f) \* v1 + f \* v2

}

private[this] def timeFract: Double =

0.0.max(windowStart.untilNow.inNanoseconds.toDouble / windowDuration.inNanoseconds)

def value: Some[Double] =

synchronized {

timeFract match {

case f if f < 1.0 =>

Some(mix(f, prevAverage.getOrElse(defaultAverage), currentAverage.getValue))

case f if f < 2.0 => Some(mix(f - 1.0, currentAverage.getValue, defaultAverage))

case f => default

}

}

def getValue: Double = value.get

def record(dataPoint: Double, count: Double = 1.0): Unit =

synchronized {

// if we're past windowDuration, roll average

val now = Time.now

if (now - windowStart > windowDuration) {

prevAverage = value

windowStart = now

currentAverage.reset()

}

currentAverage.record(dataPoint, count)

}

override def toString =

s"RecentAverage(window=$windowDuration, default=$defaultAverage, " +

s"prevValue=$prevAverage, value=$value, timeFract=$timeFract)"

}

private class ResettableAverage[DoubleOpt <: Option[Double]](defaultAverage: DoubleOpt)

extends Average {

private[this] var currentCount: Double = 0

private[this] var currentValue: Double = 0

def reset(): Unit = {

currentCount = 0

currentValue = 0

}

def record(dataPoint: Double, count: Double): Unit = {

currentCount += count

currentValue += dataPoint

}

def value: Option[Double] =

if (currentCount == 0) defaultAverage

else Some(currentValue / currentCount)

def getValue(implicit ev: DoubleOpt <:< Some[Double]): Double =

value.get

def count: Double = currentCount

}