package com.twitter.cr\_mixer.param.decider

import com.twitter.decider.Decider

import com.twitter.decider.RandomRecipient

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.util.Future

import javax.inject.Inject

import scala.util.control.NoStackTrace

/\*

Provides deciders-controlled load shedding for a given Product from a given endpoint.

The format of the decider keys is:

enable\_loadshedding\_<endpoint name>\_<product name>

E.g.:

enable\_loadshedding\_getTweetRecommendations\_Notifications

Deciders are fractional, so a value of 50.00 will drop 50% of responses. If a decider key is not

defined for a particular endpoint/product combination, those requests will always be

served.

We should therefore aim to define keys for the endpoints/product we care most about in decider.yml,

so that we can control them during incidents.

\*/

case class EndpointLoadShedder @Inject() (

decider: Decider,

statsReceiver: StatsReceiver) {

import EndpointLoadShedder.\_

// Fall back to False for any undefined key

private val deciderWithFalseFallback: Decider = decider.orElse(Decider.False)

private val keyPrefix = "enable\_loadshedding"

private val scopedStats = statsReceiver.scope("EndpointLoadShedder")

def apply[T](endpointName: String, product: String)(serve: => Future[T]): Future[T] = {

/\*

Checks if either per-product or top-level load shedding is enabled

If both are enabled at different percentages, load shedding will not be perfectly calculable due

to salting of hash (i.e. 25% load shed for Product x + 25% load shed for overall does not

result in 50% load shed for x)

\*/

val keyTyped = s"${keyPrefix}\_${endpointName}\_$product"

val keyTopLevel = s"${keyPrefix}\_${endpointName}"

if (deciderWithFalseFallback.isAvailable(keyTopLevel, recipient = Some(RandomRecipient))) {

scopedStats.counter(keyTopLevel).incr

Future.exception(LoadSheddingException)

} else if (deciderWithFalseFallback.isAvailable(keyTyped, recipient = Some(RandomRecipient))) {

scopedStats.counter(keyTyped).incr

Future.exception(LoadSheddingException)

} else serve

}

}

object EndpointLoadShedder {

object LoadSheddingException extends Exception with NoStackTrace

}