package com.twitter.product\_mixer.core.util

import com.twitter.finagle.offload.OffloadFuturePool

import com.twitter.util.Future

object OffloadFuturePools {

def parallelize[In, Out](

inputSeq: Seq[In],

transformer: In => Out,

parallelism: Int

): Future[Seq[Out]] = {

parallelize(inputSeq, transformer.andThen(Some(\_)), parallelism, None).map(\_.flatten)

}

def parallelize[In, Out](

inputSeq: Seq[In],

transformer: In => Out,

parallelism: Int,

default: Out

): Future[Seq[Out]] = {

val threadProcessFutures = (0 until parallelism).map { i =>

OffloadFuturePool.getPool(partitionAndProcessInput(inputSeq, transformer, i, parallelism))

}

val resultMap = Future.collect(threadProcessFutures).map(\_.flatten.toMap)

Future.collect {

inputSeq.indices.map { idx =>

resultMap.map(\_.getOrElse(idx, default))

}

}

}

private def partitionAndProcessInput[In, Out](

inputSeq: Seq[In],

transformer: In => Out,

threadId: Int,

parallelism: Int

): Seq[(Int, Out)] = {

partitionInputForThread(inputSeq, threadId, parallelism)

.map {

case (inputRecord, idx) =>

(idx, transformer(inputRecord))

}

}

private def partitionInputForThread[In](

inputSeq: Seq[In],

threadId: Int,

parallelism: Int

): Seq[(In, Int)] = {

inputSeq.zipWithIndex

.filter {

case (\_, idx) => idx % parallelism == threadId

case \_ => false

}

}

}