package com.twitter.cr\_mixer.blender

import com.twitter.cr\_mixer.model.BlendedCandidate

import com.twitter.cr\_mixer.model.CrCandidateGeneratorQuery

import com.twitter.cr\_mixer.model.InitialCandidate

import com.twitter.cr\_mixer.param.BlenderParams

import com.twitter.cr\_mixer.util.CountWeightedInterleaveUtil

import com.twitter.cr\_mixer.util.InterleaveUtil

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.timelines.configapi.Params

import com.twitter.util.Future

import javax.inject.Inject

import javax.inject.Singleton

/\*\*

\* A weighted round robin interleaving algorithm.

\* The weight of each blending group based on the count of candidates in each blending group.

\* The more candidates under a blending group, the more candidates are selected from it during round

\* robin, which in effect prioritizes this group.

\*

\* Weights sum up to 1. For example:

\* total candidates = 8

\* Group Weight

\* [A1, A2, A3, A4] 4/8 = 0.5 // select 50% of results from group A

\* [B1, B2] 2/8 = 0.25 // 25% from group B

\* [C1, C2] 2/8 = 0.25 // 25% from group C

\*

\* Blended results = [A1, A2, B1, C1, A3, A4, B2, C2]

\* See @linht's go/weighted-interleave

\*/

@Singleton

case class CountWeightedInterleaveBlender @Inject() (globalStats: StatsReceiver) {

import CountWeightedInterleaveBlender.\_

private val name: String = this.getClass.getCanonicalName

private val stats: StatsReceiver = globalStats.scope(name)

def blend(

query: CrCandidateGeneratorQuery,

inputCandidates: Seq[Seq[InitialCandidate]]

): Future[Seq[BlendedCandidate]] = {

val weightedBlenderQuery = CountWeightedInterleaveBlender.paramToQuery(query.params)

countWeightedInterleave(weightedBlenderQuery, inputCandidates)

}

private[blender] def countWeightedInterleave(

query: WeightedBlenderQuery,

inputCandidates: Seq[Seq[InitialCandidate]],

): Future[Seq[BlendedCandidate]] = {

val candidatesAndWeightKeyByIndexId: Seq[(Seq[InitialCandidate], Double)] = {

CountWeightedInterleaveUtil.buildInitialCandidatesWithWeightKeyByFeature(

inputCandidates,

query.rankerWeightShrinkage)

}

val interleavedCandidates =

InterleaveUtil.weightedInterleave(candidatesAndWeightKeyByIndexId, query.maxWeightAdjustments)

stats.stat("candidates").add(interleavedCandidates.size)

val blendedCandidates = BlendedCandidatesBuilder.build(inputCandidates, interleavedCandidates)

Future.value(blendedCandidates)

}

}

object CountWeightedInterleaveBlender {

/\*\*

\* We pass two parameters to the weighted interleaver:

\* @param rankerWeightShrinkage shrinkage parameter between [0, 1] that determines how close we

\* stay to uniform sampling. The bigger the shrinkage the

\* closer we are to uniform round robin

\* @param maxWeightAdjustments max number of weighted sampling to do prior to defaulting to

\* uniform. Set so that we avoid infinite loops (e.g. if weights are

\* 0)

\*/

case class WeightedBlenderQuery(

rankerWeightShrinkage: Double,

maxWeightAdjustments: Int)

def paramToQuery(params: Params): WeightedBlenderQuery = {

val rankerWeightShrinkage: Double =

params(BlenderParams.RankingInterleaveWeightShrinkageParam)

val maxWeightAdjustments: Int =

params(BlenderParams.RankingInterleaveMaxWeightAdjustments)

WeightedBlenderQuery(rankerWeightShrinkage, maxWeightAdjustments)

}

}