package com.twitter.ann.service.loadtest

import com.twitter.ann.common.EmbeddingType.EmbeddingVector

import com.twitter.ann.common.{Appendable, Distance, EntityEmbedding, Queryable, RuntimeParams}

import com.twitter.util.logging.Logger

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

class AnnIndexQueryLoadTest(

worker: AnnLoadTestWorker = new AnnLoadTestWorker()) {

lazy val logger = Logger(getClass.getName)

def performQueries[T, P <: RuntimeParams, D <: Distance[D]](

queryable: Queryable[T, P, D],

qps: Int,

duration: Duration,

queries: Seq[Query[T]],

concurrencyLevel: Int,

runtimeConfigurations: Seq[QueryTimeConfiguration[T, P]]

): Future[Unit] = {

logger.info(s"Query set: ${queries.size}")

val res = Future.traverseSequentially(runtimeConfigurations) { config =>

logger.info(s"Run load test with runtime config $config")

worker.runWithQps(

queryable,

queries,

qps,

duration,

config,

concurrencyLevel

)

}

res.onSuccess { \_ =>

logger.info(s"Done loadtest with $qps for ${duration.inMilliseconds / 1000} sec")

}

res.unit

}

}

/\*\*

\* @param embedding Embedding vector

\* @param trueNeighbours List of true neighbour ids. Empty in case true neighbours dataset not available

\* @tparam T Type of neighbour

\*/

case class Query[T](embedding: EmbeddingVector, trueNeighbours: Seq[T] = Seq.empty)

class AnnIndexBuildLoadTest(

buildRecorder: LoadTestBuildRecorder,

embeddingIndexer: EmbeddingIndexer = new EmbeddingIndexer()) {

lazy val logger = Logger(getClass.getName)

def indexEmbeddings[T, P <: RuntimeParams, D <: Distance[D]](

appendable: Appendable[T, P, D],

indexSet: Seq[EntityEmbedding[T]],

concurrencyLevel: Int

): Future[Queryable[T, P, D]] = {

logger.info(s"Index set: ${indexSet.size}")

val queryable = embeddingIndexer

.indexEmbeddings(

appendable,

buildRecorder,

indexSet,

concurrencyLevel

).onSuccess(\_ => logger.info(s"Done indexing.."))

queryable

}

}