package com.twitter.ann.service.query\_server.hnsw

import com.twitter.ann.common.Distance

import com.twitter.ann.common.\_

import com.twitter.ann.common.thriftscala.{RuntimeParams => ServiceRuntimeParams}

import com.twitter.ann.hnsw.HnswCommon

import com.twitter.ann.hnsw.HnswParams

import com.twitter.ann.hnsw.TypedHnswIndex

import com.twitter.ann.service.query\_server.common.QueryableProvider

import com.twitter.ann.service.query\_server.common.RefreshableQueryable

import com.twitter.ann.service.query\_server.common.UnsafeQueryIndexServer

import com.twitter.ann.service.query\_server.common.ValidatedIndexPathProvider

import com.twitter.ann.service.query\_server.common.warmup.Warmup

import com.twitter.bijection.Injection

import com.twitter.conversions.DurationOps.richDurationFromInt

import com.twitter.search.common.file.AbstractFile

import com.twitter.search.common.file.FileUtils

import com.twitter.util.Duration

import com.twitter.util.FuturePool

// Creating a separate hnsw query server object, since unit test require non singleton server.

object HnswQueryIndexServer extends HnswQueryableServer

class HnswQueryableServer extends UnsafeQueryIndexServer[HnswParams] {

private val IndexGroupPrefix = "group\_"

// given a directory, how to load it as a queryable index

def queryableProvider[T, D <: Distance[D]]: QueryableProvider[T, HnswParams, D] =

new QueryableProvider[T, HnswParams, D] {

override def provideQueryable(

dir: AbstractFile

): Queryable[T, HnswParams, D] = {

TypedHnswIndex.loadIndex[T, D](

dimension(),

unsafeMetric.asInstanceOf[Metric[D]],

idInjection[T](),

ReadWriteFuturePool(FuturePool.interruptible(executor)),

dir

)

}

}

private def buildQueryable[T, D <: Distance[D]](

dir: AbstractFile,

grouped: Boolean

): Queryable[T, HnswParams, D] = {

val queryable = if (refreshable()) {

logger.info(s"build refreshable queryable")

val updatableQueryable = new RefreshableQueryable(

grouped,

dir,

queryableProvider.asInstanceOf[QueryableProvider[T, HnswParams, D]],

ValidatedIndexPathProvider(

minIndexSizeBytes(),

maxIndexSizeBytes(),

statsReceiver.scope("validated\_index\_provider")

),

statsReceiver.scope("refreshable\_queryable"),

updateInterval = refreshableInterval().minutes

)

// init first load of index and also schedule the following reloads

updatableQueryable.start()

updatableQueryable.asInstanceOf[QueryableGrouped[T, HnswParams, D]]

} else {

logger.info(s"build non-refreshable queryable")

queryableProvider.provideQueryable(dir).asInstanceOf[Queryable[T, HnswParams, D]]

}

logger.info("Hnsw queryable created....")

queryable

}

override def unsafeQueryableMap[T, D <: Distance[D]]: Queryable[T, HnswParams, D] = {

val dir = FileUtils.getFileHandle(indexDirectory())

buildQueryable(dir, grouped())

}

override val runtimeInjection: Injection[HnswParams, ServiceRuntimeParams] =

HnswCommon.RuntimeParamsInjection

protected override def warmup(): Unit =

if (warmup\_enabled()) new HNSWWarmup(unsafeQueryableMap, dimension()).warmup()

}

class HNSWWarmup(hnsw: Queryable[\_, HnswParams, \_], dimension: Int) extends Warmup {

protected def minSuccessfulTries: Int = 100

protected def maxTries: Int = 1000

protected def timeout: Duration = 50.milliseconds

protected def randomQueryDimension: Int = dimension

def warmup(): Unit = {

run(

name = "queryWithDistance",

f = hnsw

.queryWithDistance(randomQuery(), 100, HnswParams(ef = 800))

)

}

}