package com.twitter.ann.hnsw

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

import com.twitter.bijection.Injection

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

import java.io.OutputStream

import java.util.concurrent.ConcurrentHashMap

import scala.collection.JavaConverters.\_

private[hnsw] object JMapBasedIdEmbeddingMap {

/\*\*

\* Creates in-memory concurrent hashmap based container that for storing id embedding mapping.

\* @param expectedElements: Expected num of elements for sizing hint, need not be exact.

\*/

def applyInMemory[T](expectedElements: Int): IdEmbeddingMap[T] =

new JMapBasedIdEmbeddingMap[T](

new ConcurrentHashMap[T, EmbeddingVector](expectedElements),

Option.empty

)

/\*\*

\* Creates in-memory concurrent hashmap based container that can be serialized to disk for storing id embedding mapping.

\* @param expectedElements: Expected num of elements for sizing hint, need not be exact.

\* @param injection : Injection for typed Id T to Array[Byte]

\*/

def applyInMemoryWithSerialization[T](

expectedElements: Int,

injection: Injection[T, Array[Byte]]

): IdEmbeddingMap[T] =

new JMapBasedIdEmbeddingMap[T](

new ConcurrentHashMap[T, EmbeddingVector](expectedElements),

Some(injection)

)

/\*\*

\* Loads id embedding mapping in in-memory concurrent hashmap.

\* @param embeddingFile: Local/Hdfs file path for embeddings

\* @param injection : Injection for typed Id T to Array[Byte]

\* @param numElements: Expected num of elements for sizing hint, need not be exact

\*/

def loadInMemory[T](

embeddingFile: AbstractFile,

injection: Injection[T, Array[Byte]],

numElements: Option[Int] = Option.empty

): IdEmbeddingMap[T] = {

val map = numElements match {

case Some(elements) => new ConcurrentHashMap[T, EmbeddingVector](elements)

case None => new ConcurrentHashMap[T, EmbeddingVector]()

}

HnswIOUtil.loadEmbeddings(

embeddingFile,

injection,

new JMapBasedIdEmbeddingMap(map, Some(injection))

)

}

}

private[this] class JMapBasedIdEmbeddingMap[T](

map: java.util.concurrent.ConcurrentHashMap[T, EmbeddingVector],

injection: Option[Injection[T, Array[Byte]]])

extends IdEmbeddingMap[T] {

override def putIfAbsent(id: T, embedding: EmbeddingVector): EmbeddingVector = {

map.putIfAbsent(id, embedding)

}

override def put(id: T, embedding: EmbeddingVector): EmbeddingVector = {

map.put(id, embedding)

}

override def get(id: T): EmbeddingVector = {

map.get(id)

}

override def iter(): Iterator[(T, EmbeddingVector)] =

map

.entrySet()

.iterator()

.asScala

.map(e => (e.getKey, e.getValue))

override def size(): Int = map.size()

override def toDirectory(embeddingFile: OutputStream): Unit = {

HnswIOUtil.saveEmbeddings(embeddingFile, injection.get, iter())

}

}