package com.twitter.ann.hnsw;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.nio.ByteBuffer;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.Set;

import java.util.stream.Collectors;

import com.google.common.collect.ImmutableList;

import org.apache.thrift.TDeserializer;

import org.apache.thrift.TException;

import org.apache.thrift.TSerializer;

import org.apache.thrift.protocol.TBinaryProtocol;

import org.apache.thrift.protocol.TProtocol;

import org.apache.thrift.transport.TIOStreamTransport;

import org.apache.thrift.transport.TTransportException;

import com.twitter.ann.common.thriftjava.HnswGraphEntry;

import com.twitter.ann.common.thriftjava.HnswInternalIndexMetadata;

import com.twitter.bijection.Injection;

import com.twitter.mediaservices.commons.codec.ArrayByteBufferCodec;

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

public final class HnswIndexIOUtil {

private HnswIndexIOUtil() {

}

/\*\*

\* Save thrift object in file

\*/

public static <T> void saveMetadata(

HnswMeta<T> graphMeta,

int efConstruction,

int maxM,

int numElements,

Injection<T, byte[]> injection,

OutputStream outputStream

) throws IOException, TException {

final int maxLevel = graphMeta.getMaxLevel();

final HnswInternalIndexMetadata metadata = new HnswInternalIndexMetadata(

maxLevel,

efConstruction,

maxM,

numElements

);

if (graphMeta.getEntryPoint().isPresent()) {

metadata.setEntryPoint(injection.apply(graphMeta.getEntryPoint().get()));

}

final TSerializer serializer = new TSerializer(new TBinaryProtocol.Factory());

outputStream.write(serializer.serialize(metadata));

outputStream.close();

}

/\*\*

\* Load Hnsw index metadata

\*/

public static HnswInternalIndexMetadata loadMetadata(AbstractFile file)

throws IOException, TException {

final HnswInternalIndexMetadata obj = new HnswInternalIndexMetadata();

final TDeserializer deserializer = new TDeserializer(new TBinaryProtocol.Factory());

deserializer.deserialize(obj, file.getByteSource().read());

return obj;

}

/\*\*

\* Load Hnsw graph entries from file

\*/

public static <T> Map<HnswNode<T>, ImmutableList<T>> loadHnswGraph(

AbstractFile file,

Injection<T, byte[]> injection,

int numElements

) throws IOException, TException {

final InputStream stream = file.getByteSource().openBufferedStream();

final TProtocol protocol = new TBinaryProtocol(new TIOStreamTransport(stream));

final Map<HnswNode<T>, ImmutableList<T>> graph =

new HashMap<>(numElements);

while (true) {

try {

final HnswGraphEntry entry = new HnswGraphEntry();

entry.read(protocol);

final HnswNode<T> node = HnswNode.from(entry.level,

injection.invert(ArrayByteBufferCodec.decode(entry.key)).get());

final List<T> list = entry.getNeighbours().stream()

.map(bb -> injection.invert(ArrayByteBufferCodec.decode(bb)).get())

.collect(Collectors.toList());

graph.put(node, ImmutableList.copyOf(list.iterator()));

} catch (TException e) {

if (e instanceof TTransportException

&& TTransportException.class.cast(e).getType() == TTransportException.END\_OF\_FILE) {

stream.close();

break;

}

stream.close();

throw e;

}

}

return graph;

}

/\*\*

\* Save hnsw graph in file

\*

\* @return number of keys in the graph

\*/

public static <T> int saveHnswGraphEntries(

Map<HnswNode<T>, ImmutableList<T>> graph,

OutputStream outputStream,

Injection<T, byte[]> injection

) throws IOException, TException {

final TProtocol protocol = new TBinaryProtocol(new TIOStreamTransport(outputStream));

final Set<HnswNode<T>> nodes = graph.keySet();

for (HnswNode<T> node : nodes) {

final HnswGraphEntry entry = new HnswGraphEntry();

entry.setLevel(node.level);

entry.setKey(injection.apply(node.item));

final List<ByteBuffer> nn = graph.getOrDefault(node, ImmutableList.of()).stream()

.map(t -> ByteBuffer.wrap(injection.apply(t)))

.collect(Collectors.toList());

entry.setNeighbours(nn);

entry.write(protocol);

}

outputStream.close();

return nodes.size();

}

}