# UMBC, Spring 2015, CMSC 621 (Dr. Kalpakis)

# Distributing the Dictionary of triplets on a CHORD ring using GO and JSON-RPC

Design and develop a distributed peer-to-peer system that implements the CHORD protocol with GO. A client can initiate a connection with any ring node and perform any of the operations (lookup, insert, insertOrUpdate, delete, listKeys, listIDs, shutdown) by using JSON-RPC messages. Ring nodes can join and leave (in a scheduled manner) the system at any time while the remaining ring nodes continue to respond to client requests.

Like project 1, ring nodes manage a collection of triplets (key, relationship, value), where key and relation are valid JSON strings and value is a valid JSON object. The (key, relationship) pair identifies the triplet and is used to store the triplet into the appropriate node. The value of a triplet is a JSON object with structure such as:

{“content”: “any valid JSON string”, “size”: “1KB”, “created”: “3/18/2015, 8:50:26”, “modified”: “3/20/2015, 16:40:03”, “accessed”: “3/ 20/2015, 18:09:54”, “permission”: “RW”}

where: “size” is the size of the value’s content, “content” it can be any valid JSON object and represents the triplet’s content, “created”, “modified”, “accessed” represent when the content created, modified, or accessed respectively, and “permission” describes the type of access the content has. It can be read only (R), and read-write (RW). If the permission is read only the triplet’s content cannot be modified or deleted from the dictionary.

A client/ring node should be able to perform a lookup operation even by partially specifying a triplet’s identifier. Therefore, a JSON-RPC message of the form {“method”: “lookup”, “params” : [“keyA”, “”]} or {“method”: “lookup”, “params” : [“”, “relC”]} are allowed and a response should be returned to the client.

A client/ring node should be able to purge (delete from) “RW” dictionary triplets that have not been accessed since some user specified time.

The interaction’s “id” from project 1 should not be defined at any interaction. Therefore, interactions would have the following structure:

Client -> Server :: {“method” : “lookup” , “params” : [“keyA” , “relA”] }, etc.

Server -> Client :: {“result” : [“keyA”, “relA”, {….}], “error”: null}, etc.

Every client should make **asynchronous** calls to the nodes of the form:

Client.Go(“Requested.method”, params, reply, nil)

Do not pass as params the whole interaction message. The Requested.method can be Requested.Lookup, Requested.Insert, etc.

**Server/Client Configuration**

Similar to project 1.

**What to do:**

Design and develop a distributed peer-to-peer system using Chord protocol. The system should be robust when ring nodes join and leave the system. No information should be lost when a ring node leaves the system after receives the shutdown signal. Your system should respond to all properly-structured JSON-RPC messages. Also, design and develop a GO client that reads a JSON-RPC request message from the standard input, makes the appropriate request to the ring node which is connected, and shows the response (if any). Both your client and server code should take as their 1$^{st}$ command line argument the filename that contains the server configuration JSON object.

**What to submit:**

- a .zip/.tgz archive with all your source code, configuration/setup files, scripts to execute your implementation.
- a 5-8 pages written report describing essential elements of your design and implementation.

**Where to submit:**

At least one member of the group should submit the .zip/.tgz archive and the written report (that refers all the members of the group) on blackboard before the deadline.