PROJECT 2 – CMSC 621

Implementation of CHORD protocol/ Distributed Hash Table
as a
GoLang Distributed Application

Project Group Members:

Nipun Ramagiri (nipunr1@umbc.edu)
Rishabh Sachdeva (rishabs1@umbc.edu)
Arshita Jain (a253@umbc.edu)

Instead of sample datafile(s), and sample (test) output, we have included everything in our main file itself.

We are implementing CHORD protocol/ Distributed Hash Table as a GoLang Distributed Application. We initialized the ring with 6 nodes. The main function calls all the distinct APIs for their respective functions, creating a whole scenario of the usage of these APIs.

Main GoLang Routine: coordinator

```go
func main() {
  //publish coordinator thread for sending commands
  node_addresses = make(map[int]string)
  nodes_in_ring = append(nodes_in_ring, "tcp://127.0.0.1:5501")
  node_addresses[0] = "tcp://127.0.0.1:5500"
  node_addresses[1] = "tcp://127.0.0.1:5501"
  bucket_firstNode := make(map[int]int)
  bucket_firstNode[0] = 1
  bucket_firstNode[1] = 1
  bucket_firstNode[2] = 1
  bucket_firstNode[3] = 1
  bucket_firstNode[4] = 1
  node := &Node{
    Key:     1,
    InRing:  true,
    Address: "tcp://127.0.0.1:5501",
    Bucket:  bucket_firstNode, //make(map[string]string),
    Pre:     1,
    Succ:    1,
  }
  go worker(node)

  for i := 1; i < int(math.Pow(2, num_nodes_order-1)); i++ {
    port := 5501 + i
    node_addresses[i+1] = "tcp://127.0.0.1:" + strconv.Itoa(port)
    node := &Node{
      Key:     i + 1,
      InRing:  false,
      Address: "tcp://127.0.0.1:" + strconv.Itoa(port),
      Bucket:  make(map[int]int),
    }
    go worker(node)
  }
```

List of JSON Requests

{"do": "join-ring", "sponsoring-node": address }

Adds a new node to the existing Chord Ring

The joining node looks for the successor of the sponsoring node for joining. After finalizing this, the node adds up, updating successor and predecessor. The finalization is done by checking that the joining node is in the right position with respect to the other key values. This is done by hopping around, finding the right predecessor and successor for that node.

Code Snippet:

```
join_14 := &Command{
  Do:             "join-ring",
  SponsoringNode: "tcp://127.0.0.1:5501", // node_addresses[1]
}
executeCommand("tcp://127.0.0.1:5514", join_14) // executeCommand(node_addresses[14], join_14)

time.Sleep(1 * time.Second)
```

Output:

```
>Command Node 14 receives: {join-ring tcp://127.0.0.1:5501   <nil> <nil> <nil>
0 0}
join-ring>> Doing join-ring for 14 from tcp://127.0.0.1:5501
join-ring>>Finding successor of tcp://127.0.0.1:5501
>>> Message received after command execution: [ACKNOWLEDGED] - join-ring
>Command Node 1 receives: {find-ring-successor   tcp://127.0.0.1:5514 <nil> <ni
l> <nil> 0 0}
find-ring-successor>> Node 1 Successor - 8
--------------

join-ring>> recvSucc from SponsorNode tcp://127.0.0.1:5501 - 8
join-ring>> Finding Sponsor Node from SponsorNodeSucc 8
>Command Node 8 receives: {find-ring-predecessor   tcp://127.0.0.1:5514 <nil> <
nil> <nil> 0 0}
find-ring-predecessor>> Node 8 Predecessor - 1
--------------

join-ring>> recvPre from SponsorNodeSucc 8 - 1
>Command Node 1 receives: {find-ring-successor   tcp://127.0.0.1:5514 <nil> <ni
l> <nil> 0 0}
find-ring-successor>> Node 1 Successor - 8
--------------

join-ring>> recvSucc - 8
>Command Node 8 receives: {find-ring-successor   tcp://127.0.0.1:5514 <nil> <ni
l> <nil> 0 0}
find-ring-successor>> Node 8 Successor - 1
--------------
```

```
join-ring>> recvSucc - 1
join-ring>> Finalized Succ and Pre for Node  14 : 1 8
>Command Node 8 receives: {update-successor    <nil> <nil> <nil> 0 14}
update-successor>> Updated Node 8 successor 14
>>> Message received after command execution: [ACKNOWLEDGEMENT] - update-succes
sor, Updated Succ
>Command Node 1 receives: {update-predecessor    <nil> <nil> <nil> 14 0}
update-predecessor>> Updated Node 1 predecessor 14
--------------

>>> Message received after command execution: [ACKNOWLEDGEMENT] - update-predec
essor, Updated Pre
join-ring>> join-ring for 14 from tcp://127.0.0.1:5501 Complete
join-ring>> Node 14 details &{14 1 8 [0 0 0 0 0] tcp://127.0.0.1:5514 true map[
]}
--------------
```

{"do": "leave-ring" "mode": "immediate or orderly"}

Removes the specified node from the Chord ring

Subsequently, the node's successor has its predecessor updated while the node's predecessor has its successor updated.

Code Snippet:

```go
//immediate leave-ring node 12
immLeave12 := &Command{
  Do:   "leave-ring",
  Mode: "immediate",
}
executeCommand("tcp://127.0.0.1:5512", immLeave12)

time.Sleep(1 * time.Second)
```

Output:

```
>Command Node 12 receives: {leave-ring  immediate  <nil> <nil> <nil> 0 0}
leave-ring>> Performing immediate leave-ring action on Node address:  tcp://127
.0.0.1:5512
>>> Message received after command execution: [ACKNOWLEDGED] - leave-ring
>Command Node 9 receives: {update-successor    <nil> <nil> <nil> 0 14}
update-successor>> Updated Node 9 successor 14
>>> Message received after command execution: [ACKNOWLEDGEMENT] - update-succes
sor, Updated Succ
>Command Node 14 receives: {update-predecessor    <nil> <nil> <nil> 9 0}
update-predecessor>> Updated Node 14 predecessor 9
```

{"do": "init-ring-fingers" }

It is used to initialize the finger tables of all the nodes present in the chord ring.

Code Snippet:

```go
    } else if unMarshalledCommand.Do == "init-ring-fingers" {
      node.FingerTable = [num_nodes_order - 1]int{}
      workerServer.Send("[[ACKNOWLEDGED]] - init-ring-fingers", 0)
      fmt.Println("init-ring-fingers >> Initialized ring fingers", nodes_in_ring, "\n--------------\n")
```

{"do": "fix-ring-fingers" }

It updates all the finger tables with their relevant key data, keeping in mind the context of all the nodes present in the Chord.

Code Snippet:

```
//joining node 13
join13 := &Command{
  Do:             "join-ring",
  SponsoringNode: "tcp://127.0.0.1:5501",
}
executeCommand("tcp://127.0.0.1:5513", join13)

// upd_cmd := &Command{
//  Do:      "update-bucket",
//  ReplyTo: "tcp://127.0.0.1:5508",
// }
// executeCommand("tcp://127.0.0.1:5501", upd_cmd)

// Command to fix the finger tables
time.Sleep(1 * time.Second)
fix_finger_table_cmd := &Command{
  Do: "fix-ring-fingers",
}
```

Output:

```
update finger table of: tcp://127.0.0.1:5509
>Command Node 9 receives: {fix-ring-fingers    <nil> <nil> <nil> 0 0}
>>> Message received after command execution: [ACKNOWLEDGED] - fix-ring-fingers
>Command Node 13 receives: {find-ring-successor   tcp://127.0.0.1:5509 <nil> <n
il> <nil> 0 0}
find-ring-successor>> Node 13 Successor - 14
--------------

>Command Node 14 receives: {find-ring-successor   tcp://127.0.0.1:5509 <nil> <n
il> <nil> 0 0}
find-ring-successor>> Node 14 Successor - 24
--------------

>Command Node 24 receives: {find-ring-successor   tcp://127.0.0.1:5509 <nil> <n
il> <nil> 0 0}
find-ring-successor>> Node 24 Successor - 1
--------------

fix-ring-fingers>> Fixed ring fingers of [tcp://127.0.0.1:5501 tcp://127.0.0.1:
5508 tcp://127.0.0.1:5514 tcp://127.0.0.1:5509 tcp://127.0.0.1:5524 tcp://127.0
.0.1:5513]
--------------
```

{"do": "ring-notify", "reply-to": address }

{"do": "stabilize-ring" }

Notify updates the predecessors while stabilize updates the successors to their appropriate node values in CHORD.

Code Snippet:

```go
fmt.Println("join-ring>> Finalized Succ and Pre for Node ", node.Key, ":", node.Succ, node.Pre)
node.InRing = true

// Updating the Succ and Pre of node.Pre and node.Succ respectivly
updateSuccCommand := &Command{
    Do:           "stabilize-ring",
    NewSuccessor: node.Key,
}
executeCommand(node_addresses[node.Pre], updateSuccCommand)
updatePreCommand := &Command{
    Do:             "ring-notify",
    NewPredecessor: node.Key,
}
executeCommand(node_addresses[node.Succ], updatePreCommand)

fmt.Println("join-ring>> join-ring for", node.Key, "from", sponsoringNodeAddress, "Complete")
fmt.Println("join-ring>> Node", node.Key, "details", node, "\n--------------\n")

nodes_in_ring = append(nodes_in_ring, my_add)
```

Output:

```
join-ring>> recvSucc - 14
join-ring>> Finalized Succ and Pre for Node  13 : 14 9
>Command Node 9 receives: {stabilize-ring    <nil> <nil> <nil> 0 13}
stabilize-ring>> Updated Node 9 successor 13
--------------
```

```
>>> Message received after command execution: [ACKNOWLEDGEMENT] - stabilize-ring, Updated Succ
>Command Node 14 receives: {ring-notify    <nil> <nil> <nil> 13 0}
ring-notify>> Updated Node 14 predecessor 13
--------------
```

{"do": "get-ring-fingers", "reply-to": address }

Displays the specified node’s finger table.

```
>Command Node 9 receives: {get-ring-fingers   tcp://127.0.0.1:5501 <nil> <nil>
<nil> 0 0}
get-ring-fingers>> Obtianed finger table of Node 9 : [13 13 13 24 1]
--------------
```

{"do": "find-ring-successor", "reply-to": address}

Finds the successor of the specified node

```
>Command Node 9 receives: {find-ring-successor   tcp://127.0.0.1:5501 <nil> <ni
l> <nil> 0 0}
find-ring-successor>> Node 9 Successor - 13
--------------
```

{"do": "find-ring-predecessor", "reply-to": address}

Finds the predecessor of the specified node

```
>Command Node 1 receives: {find-ring-predecessor   tcp://127.0.0.1:5508 <nil> <
nil> <nil> 0 0}
find-ring-predecessor>> Node 1 Predecessor - 1
--------------
```

{"do": "put", "data": { "key" : "a key", "value" : "a value" }, "reply-to": address}

Instructs the recipient node to store the given (key,value) pair in the appropriate ring node.

{"do": "get", "data": { "key" : "a key" }, "reply-to": address}

Instructs the recipient node to retrieve the value associated with the key stored in the ring

{"do": "remove", "data": { "key" : "a key" }, "reply-to": address}

Instructs the recipient node to remove the (key,value) pair from the ring

Code Snippet:

```
// Put data commands
time.Sleep(1 * time.Second)
put_data_cmd := &Command{
    Do: "put",
    Data: &DataStruct{
        Key:   11,
        Value: 100,
    },
    ReplyTo: "tcp://127.0.0.1:5501",
}
executeCommand("tcp://127.0.0.1:5501", put_data_cmd)
```

```
time.Sleep(1 * time.Second)
get_cmd := &Command{
    Do: "get",
    Data: &DataStruct{
        Key: 11,
    },
    ReplyTo: "tcp://127.0.0.1:5509",
}
executeCommand("tcp://127.0.0.1:5509", get_cmd)
```

```
// Changing the nodes_in_ring
for i, n := range nodes_in_ring {
  if my_add == n {
    nodes_in_ring = remove(nodes_in_ring, i)
    break
  }
}
fmt.Println("leave-ring >> Updated nodes_in_ring:", nodes_in_ring, "\n--------------\n")
```

Output:

```
================

>Command Node 1 receives: {put   tcp://127.0.0.1:5501 0xc420230860 <nil> <nil> 0 0}

================

put >> Finger table of Node  1 : [8 8 8 9 24]
put >> Data Key:  11
>>> Message received after command execution: [ACKNOWLEDGED] - put,get or remove
>Command Node 9 receives: {get-ring-fingers   tcp://127.0.0.1:5501 <nil> <nil> <nil> 0 0}
get-ring-fingers>> Obtianed finger table of Node 9 : [13 13 13 24 1]
--------------

[13 13 13 24 1]
put>> Sent put bucket data command at address: tcp://127.0.0.1:5513 tcp://127.0.0.1:5501
put>> Data being sent: &{11 1100}
>Command Node 9 receives: {put-bucket-data   tcp://127.0.0.1:5501 0xc420230980 <nil> <nil> 0 0}
put-bucket-data>> Updating bucket in Node  9 with &{11 1100}
put>> ACK Received : [ACKNOWLEDGED] - put-bucket-data
put>> PUT COMPELTE
================
```

{"do": "list-items", "reply-to": address}

Instructs the recipient node to respond with a list of the key-value pairs stored at its bucket.

Code Snippet:

```go
    } else if unMarshalledCommand.Do == "list-items" {
      bucketJson := BucketStruct{Bucket: node.Bucket}
      marshalledBucketJson, _ := json.Marshal(bucketJson)
      fmt.Println("Bucket:", node.Bucket)
      workerServer.Send(string(marshalledBucketJson), 0)
    }
```

```go
  time.Sleep(1 * time.Second)
  get_list_cmd := &Command{
    Do:       "list-items",
    ReplyTo: "tcp://127.0.0.1:5508",
  }
  executeCommand("tcp://127.0.0.1:5514", get_list_cmd)

```

Output:

```
>Command Node 14 receives: {list-items   tcp://127.0.0.1:5508 <nil> <nil> <nil>
 0 0}
Bucket: map[]
>>> Message received after command execution: {"Bucket":{}}
```

The output of the code from where all the snippets have been provided is present in finalOutput.txt. The command to run this code is: **go run finalChord.go**