# Assignments for Distributed Computing CS G557: DISTRIBUTED HASH TABLE IN KUBERNETES

SANTONU SARKAR (PI)

March 26, 2024

## Instructions

A lab environment will be provided to you at the following URL: `https://naulabportal.netapp.com`. You will be given a username and password to login to your account. The labs will be available from **March 17 through March 29, 2024**.

Your exercise environment contains the following virtual machines:

- One Windows Server system
- A three-node Kubernetes cluster
- An ONTAP 9.9 single-node cluster (Cluster1). The ONTAP cluster can be ignored for this exercise.

When you use the connection information that your instructor assigns to you, you first connect through Remote Desktop Connection to a Windows Server. From this Windows desktop, you connect to the other servers in your exercise environment.

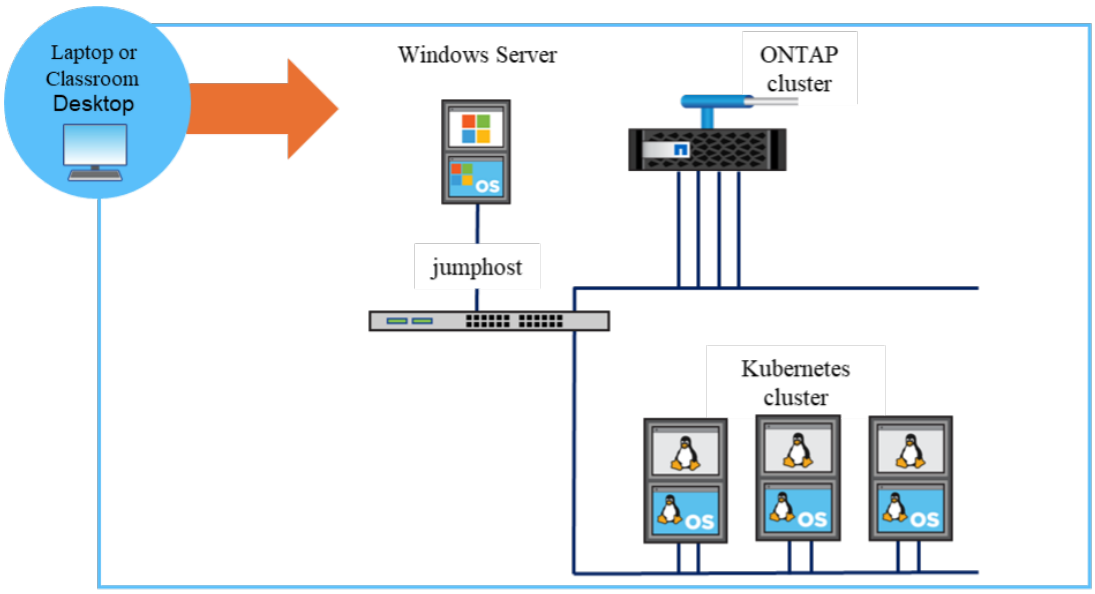


Figure 1: Solution Architecture

Go through the following additional materials for deeper understanding.

- Overview: `https://kubernetes.io/docs/concepts/overview/`
- Kubernetes basics: `https://kubernetes.io/docs/tutorials/kubernetes-basics/`

Table 1: System Description

| System | Host Name | IP Address | Username | Password |
|---|---|---|---|---|
| Windows Server | Jumphost | 192.168.0.5 | DEMO\Administrator | Netapp1! |
| Kubernetes Control Plane | kubmas | 192.168.0.61 | root (case sensitive) | Netapp1! |
| Kubernetes Worker 1 | kubwor1 | 192.168.0.62 | root (case sensitive) | Netapp1! |
| Kubernetes Worker 2 | kubwor2 | 192.168.0.63 | root (case sensitive) | Netapp1! |
| ONTAP cluster mgmt | Cluster1 | 192.168.0.101 | admin (case sensitive) | Netapp1! |
| node1 (Cluster1) | Cluster1-01 | 192.168.0.111 | admin (case sensitive) | Netapp1! |

- Cluster Architecture: `https://kubernetes.io/docs/concepts/architecture/`
- Nodes: `https://kubernetes.io/docs/concepts/architecture/nodes/`
- Communication between nodes and the Control Plane: `https://kubernetes.io/docs/concepts/architecture/control-plane-node-communication/`
- Workloads: `https://kubernetes.io/docs/concepts/workloads/`
- Tools for administrating Kubernetes: `https://kubernetes.io/docs/tasks/tools/`
- API Overview: `https://kubernetes.io/docs/reference/using-api/`
- Creating a cluster with kubeadm `https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/`
- Optional additional training: `https://kubernetes.io/training/`

The machines in this environment are in Table 1 .

# 1 Assignment

You will replicate the DHT described in the class. To make the process simple, we will not consider dynamic addition or deletion of nodes. The nodes with preassigned IDs will stay in the ring right from the beginning. The following digram explains the topology and deployment architecture. Each chord node will have a unique ID, shown in Figure 2. Each chord will have a fixed, hard-coded finger table as shown in Figure 2. The objective is to implment a lookup function.

## 1.1 Design Guideline

Each pod will have one class, `ChordNode` preferrably in Python. This class will have the following:

1. ID: which will be supplied at runtime.
2. pred: ID of the predecessor node of this current node.
3. A finger table FD. Entries will be different for different nodes. This entry needs to be populated at runtime. Should the node require to forward the lookup request to another node say `nn`, the node should be able to figure out the POD address from the ID. For each entry $i = 1..5$, $FT[i] = succ(p + 2^{i-1})$. The successor computation performs a modulo arithmetic. So, for node 18, FT[5]=succ((18+16) mod 32)=4. In this assignment, you do not have to generate the finger table. You can supply the table as a precomputed list.
4. The class in a POD $q$ will run continously, it will sleep when it is not doing anything. When it gets a message from some other POD $p$, it wakes up and executes lookup().
5. The class will implement the lookup() method. Think how many parameters you reuire.

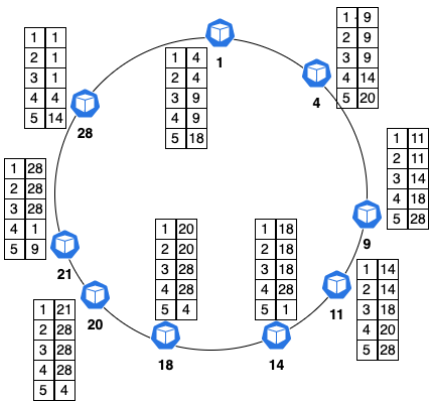


Figure 2: Ring topology, each chord node runs inside a POD

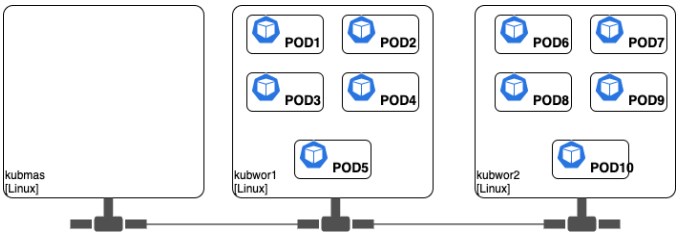


Figure 3: Deployment of PODS in K8S

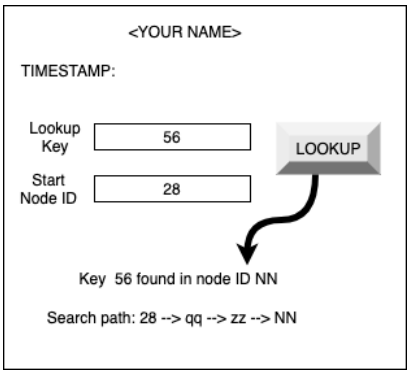


Figure 4: Browser interface

6. If the lookup method of a node $p$ needs to reply, (i.e. the key is in its own jurisdiction), the search should end, and this node should reply.

7. If lookup() of a node $p$ needs to forward the request to another node $q$, it will invoke `lookup()` method remotely to another POD that hosts $q$. Think how a POD can communicate with another POD to invoke this method.

### 1.1.1 Lookup Function

As discussed in the class, the lookup function will perform the following activities

```
def localSuccNode(key): #take module arithmetic during comparison
    if pred < key <= ID return ID       #self has key
    elif ID < key <= FT[1] return FT[1] #successor is responsible
    for i in range(1, m+2):
        if FT[i] <= key < FT[i+1] return FT[i] #FT[i] is responsible
```

1. The comparison operation needs to consider a modulo-$2^m$ arithmetic, where $m$ is this example is 5. According to this modulo arithmetic, $FT[4] = 28 < 30 < FT[5] = 4$ if we consider the finger table of node 18. Keep this in mind.

2. The key you supply in the browser can be any integer. The key needs to be converted using modulo-32 arithmetic, i.e. $key1 = key \mod 2^5$, during lookup.

### 1.2 Deployment Guideline

This code should be containerized and pushed to your docker hub. While deploying a POD, you need to pull this code, supply appropriate parameters and put it inside the POD. Here each POD will have 1 container. Each container should have the exact same code (ChordNode.py). You need to supply finger table values in a file, externally. So there will be 9 finger table files.

You should deploy 9 instances of the ChordNode.py with 9 externally supplied finger tables (shown in Figure 2) in 9 pods distributed across two worker nodes, shown in Figure 3. For demonstration, you should have a webserver hosting page similar to Figure 4. The information present in the page MUST be there. You can add more information if you like.

### 1.3 Tasks

1. Create an appropriate Dockerfile to containerize the ChordNode.py.
2. Push your ChordNode.py in your own docker hub. Pull it to create a container, and a pod.
3. Create YAML file to deploy the pods hosting the chord nodes.
4. The front end container should run ngix:1.24.0-alpine-slim similar to your previous assignment. This should host the webpage shown in Figure 4.
5. Run 9 pods and supply 9 finger table txt files.
6. During demonstration provide a key and a start node, press lookup button. Once you get the result, take a screenshot of the result. This MUST be done in front of the TA.

## 2 Evaluation Scheme

1. Create a zip file named your GROUPID-DHT.zip and include the following: 1. Dockerfile 2. YAML file for deploying pods, 3. commandline screenshot to show that 9 pods+webserver are running (using appropriate command) 4. webpage screenshots (at least 3 for 3 test cases)
2. Upload the zip file
3. Total Marks=10.

Upload the zip file in Google assignment. The Google Assignment will have a due date and time. This will ensure that you finish your assignment ON time. If you can't upload within the time, I will assume that you have not been able to complete the assignment.

We will give partial marks. You get 0 if your solution does not work in any form. Marking scheme will consider the following:

1. Correct Dockerfile
2. Correct Kubenetes YAML file
3. Commandline screenshot to show that 9 pods+webserver are running exactly like Figure 3
4. 3 test case screenshots (ideally they should be screenshots of Figure 4)

### Partial Marks

If you can show that these chordnodes are running as 9 different processes (but not inside container or a pod) and they passed all 3 test cases- then 6 marks. In such a case, you HAVE to upload approapriate screenshots which convince us that your logic is correct.

If you can show that these chordnodes are running as 9 containerized processes but you couldn't run them inside Kubernetes, then you get 6+1=7 marks.

If you can show that your chordnodes and webserver are all running inside ONE worker node, you are unable to distribute them like Figure 3, then you get 8 marks.