CSE 535 - Asynchronous Systems
Professor - Y.Annie Liu
Project Report
Implementing and Evaluating DNS using Chord

Team No - 14
Akanksha Mahajan - 112074564
Mehul Jain - 112072812
Pragesh Jagnani - 112045076

1. Problem and plan

1.1 Introduction:

Domain Name Server (DNS) provides a solution to map human-readable strings to IP Addresses so that Internet users don't have to remember IP addresses to access websites. The earlier shortcoming of traditional DNS systems requires significant expertise to administer these systems and as noted in the book by Cricket Liu and Paul Albitz [1] one of the most common name servers problems are related to configurations. Another study by Danzig et al. [6] found that one-third of the DNS traffic that traversed the NSFNet was directed to one of the seven root nodes. These studies lead to the motivation to serve DNS data using Chord since it eliminates the need to have an expert in running name servers and provides better load balance by eliminating the concept of root server [3].

1.2 Problem Statement:

The main goal of this project is to use the existing implementation[4] of Chord in DistAlgo[8] to solve issues in the traditional implementation of DNS. This project also aims to evaluate the performance and correctness of Chord implementation in DistAlgo and compare it with one of the best implementations in Java[5]. The results will be visually analyzed to better understand the implementations and can be used as a reference for making choice amongst different implementation.

1.3 State of the Art:

One of the implementations of Chord in Java can be found here[5]. This implementation is good because of the following reasons:

1. It includes implementation of all key aspects related to Chord handling stabilization of nodes.
2. It is easy to understand with the help of documentation provided along with the source code.
3. The running instructions for the code are quite clear which makes it easy to run.

Chord.java, provides the interface with all implementations in ChordImpl.java to create nodes, retrieve keys, delete keys, insert data, and leave nodes. The second implementation of Chord in Distalgo can be found here [4]. This implementation considered all the key aspects related to Chord, and also focuses on improving efficiency by applying the following strategies:

1. **Maintaining a reverse finger table:** Each node along with the normal finger table maintains a reverse finger table as well. In this reverse finger table, let us say node A is finger of node B, then node B maintains a separate finger table to maintain information related to node A. This is done to optimize queries which are destined to some node in the key range of A and B. In this case, the request can be directly forwarded to A instead of traversing the entire circle.
2. **Maintaining successor and predecessor list:** This is maintained for easier lookups to find successor and predecessor to handle scenarios whenever a node leaves or joins the network.

There is one file for this implementation named Chord_Distalgo_Optimization.da. The program contains two main processes - Driver, ChordNode. Driver process is responsible to simulate scenarios like key lookups, inserting keys, inserting nodes, and removing nodes. ChordNode implements all the functionalities related to Node in Chord algorithm like updating finger table, updating successor and predecessor list, inserting and finding keys and Stabilization.

Since the first and second implementations include most of the aspects of Chord and the second one also includes some strategies to improve the efficiency, it would be worth to compare the performance and correctness of both the implementations.

1.4 Project Tasks:

This section aims at breaking down this project into smaller tasks, which will help both the student as well Professor in tracking down the progress of the project.

1. Task - 1: **Read and Understand Previous Work**
    a. Read and Understand the working of the existing implementation of Chord in DistAlgo[4].
    Task Performer - Mehul
    b. Read and Understand the working of the existing implementation of Chord in Java [5]. ***Task Performer - Pragesh***
    c. Better understand metrics as calculated in these papers [2], [3].
    Task Performer - Akanksha

2. Task - 2: **Prepare Input Data**
    a. Download a data set having DNS data, primarily a mapping from Domain Name to IP Address.
    Task Performer - Mehul
    b. Convert the data into a CSV file that can be used by our chord Implementation for testing.
    Task Performer - Mehul

3. Task - 3: **Design and Development**
    a. Develop and implement the code in DistAlgo to test performance and correctness of Chord Implementation(DistAlgo) by making minimal changes to the existing code. ***Task Performer - Mehul***
    b. Develop and implement the code in Java to test performance and correctness of Chord Implementation(Java) by making minimal changes to the existing code. ***Task Performer - Pragesh***
    c. Incorporate DNS data (DomainName - Key & IP Address - Value) in both these implementations as an Input file.
    Task Performer - Akanksha
    d. Compare the results obtained from the above two tasks.
    Task Performer - Pragesh, Akanksha
    e. Report results in the form of a visual graph for a better understanding of the reader.
    Task Performer - Mehul, Akanksha

4. Task - 4: **Final Report and Presentation**
    a. Prepare Final Project Report that summarizes each and every phase of the project along with the results obtained.
    Task Performer - Mehul, Pragesh, Akanksha
    b. Prepare and practice Final Project Presentation.
    Task Performer -Mehul, Pragesh, Akanksha

1.5 Metrics Evaluation:

Following parameters have been taken from research papers [2],[3], [6], [9], [10], [11]:

1. **Path Length/Hop count:** Chord's performance depends in part on the number of nodes that must be visited to resolve a query. The path length is the number of nodes traversed during a lookup operation. So during performance analysis, each node in our experiment will pick a random set of keys to query from the system, and we will measure each query's path length.
2. **Latency:** Latency is the time taken for a lookup in the chord. In our evaluation, we will measure the latency for both successful and unsuccessful queries and then compare the lookup latency distribution for both the implementations.
3. **Storage balance:** Irregularities in the random placement can cause some nodes to store more data than others. We will plot a graph that will show a cumulative distribution for the number of records stored on each node.
4. **Load balance:** It is the ability of consistent hashing to allocate keys to nodes evenly. In a network with N nodes and K keys, we would like the distribution of keys to nodes to be tight around N/K. For this, we will vary the total number of keys and then will run the experiments with different random number generator seeds, counting the number of queries assigned to each node in each experiment.
5. **Lookups:** Since Records are distributed randomly, we need to make sure that nodes that are responsible for popular records are not required to serve a disproportionate amount of RPCs. We will evaluate the performance and accuracy of Chord lookups when nodes are continuously joining and leaving.

6. **Simultaneous Node Failures:** We will evaluate the impact of a massive failure on Chord's performance and on its ability to perform correct lookups. Once the network becomes stable, each node will be made to fail with probability p. After the failures occur, we will perform random lookups. For each lookup, we will record the number of timeouts experienced by the lookup, and the number of nodes contacted during the lookup.

1.6 Input and Output:

Following parameters are taken as input and output for this project.

1. **Input:**
    a. Number of bits for node and key identifiers ('m')
    b. Total number of Nodes
    c. Number of Successors
    d. Number of Queries
    e. Stabilize Delay
    f. Fix Finger Delay
    g. Check Predecessor Delay
    h. Probability of Failure
2. **Output**:
    a. Table to visualize the comparison between both the implementations based on different parameters.
    b. Visualization directory that includes graphs that compare results of both the implementations based on different parameters

1.7 Examples & Application:

Following are some of the best industry-wide use-cases of Chord.

1. **Co-operative mirroring:** It is a load balancing mechanism where a product's availability is of importance. This becomes an apt application for Chord which allows many computers to load balance instead of all the work being done by a single centralized server [7].
2. **Time-shared usage:** This is a mechanism in which a computer's data is distributed throughout the network so that its data is available even if it exits from the network abruptly.
3. **P2P file transfer:** This is a mechanism where a file's data which is needed is spread across the network. It reduces the load on a node as the data is distributed across the networks. This also allows offline file access where the node which was sharing the data exited the network.

1.8 Project Plan:

This section aims at dividing the tasks listed in the previous section among four weeks.

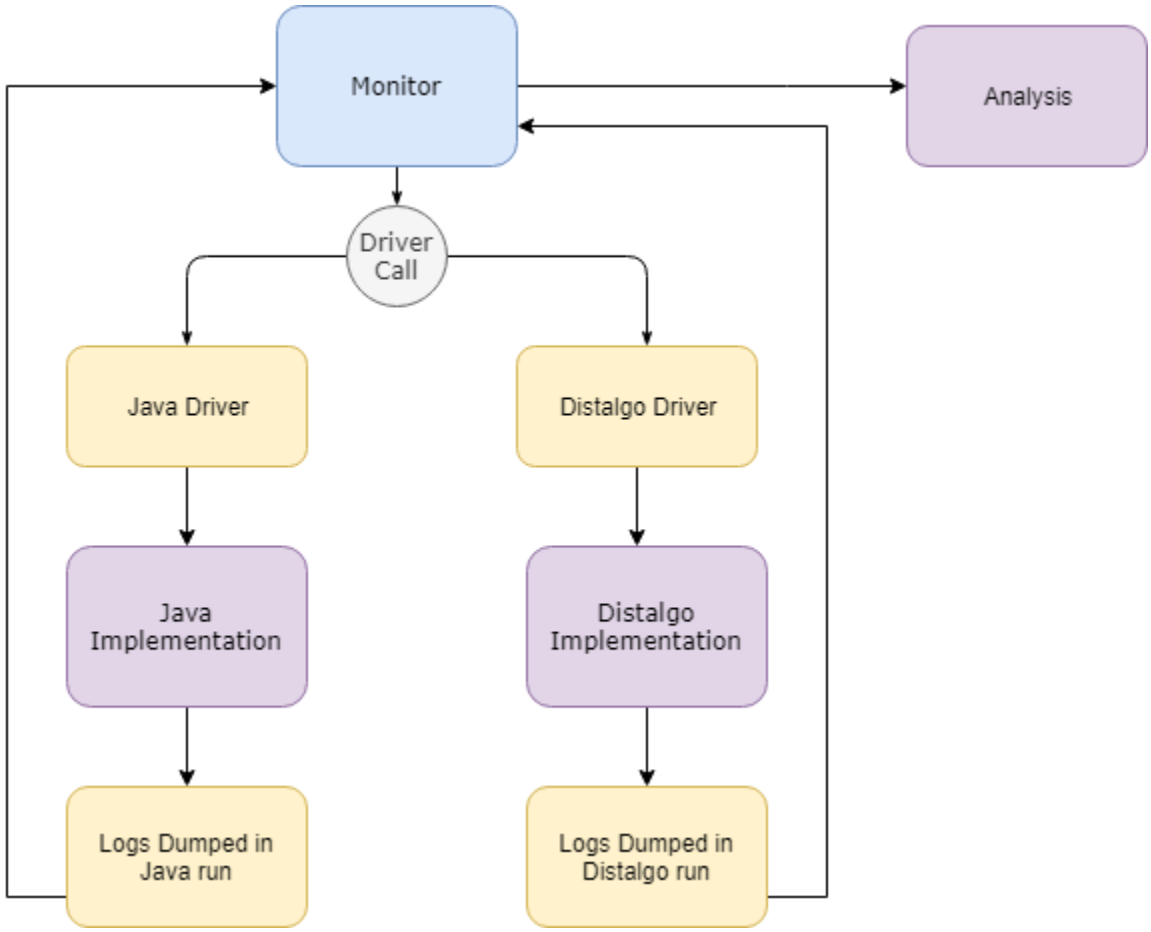
1. **Week - 1**
    a. Finish Reading and Understanding past projects and implementations.
    b. Finalize Input and Output Data.
    c. Download and successfully run the existing code on our local machine.
2. **Week - 2**
    a. Prepare Input Data
    b. Design the testing and performance framework
    c. Start with the Implementation
3. **Week - 3**
    a. Finish Implementation
    b. Perform testing and performance comparison
    c. Begin Final Project Report
4. **Week - 4**
    a. Finish final project report
    b. Prepare final presentation

2. Design

2.1 Driver Program:

The below Figure shows a schematic architecture diagram of our design. We are using separate drivers for each of our implementations. In our figure, we can see the Implementations of Chord that we have taken in Java and DistAlgo respectively, as they are the examples we have taken. The corresponding drivers are used to control these implementations. The drivers are coded in DistAlgo and JAVA and are designed to capture CPU time and memory usage details of each implementation by varying the input parameters like number of runs, number of participating processes, message delay etc.

1. The driver programs would coordinate with the implementation to log output to the files.
2. We plan to use these logs to evaluate performance and generate visualization.



2.2 Monitor Program:

• We are designing a single monitor program that will be responsible for starting all the drivers in the system. Also, this monitor program will read all the logs that the drivers generate as outputs.
• From those captured logs, we plan to use the monitor program to create charts and graphs, to compare the time and space complexities of the different implementations.
• This decoupled architecture will help us to maintain an abstraction between the main monitor program and the end implementations. The modular nature would ensure that we could insert any other algorithms or implementations in our framework. We would just have to write a driver for the program, and our monitor would be able to compare performances their performances.

3. Implementation

This section includes the implementation details of different code which we wrote to do performance and correctness testing. We have cited the research papers used to decide these metrics here and in the code. The code is pushed to the github repository which can be found here.

3.1 Chord Implementation in DistAlgo

The Existing Chord Implementation was written in 2013 and since then DistAlgo has released versions in which some commands became deprecated. Due to this, the code gave a lot of compile-time error when we first tried to run this code. Also, there were a lot of issues, out of which one of them was pointed out on our google-Group for which our team provided a fix.

Following are the list of some changes we made in DistAlgo Chord Implementation:

1. Fixed issues in Join_Correct Method
2. Removed old keywords such as purged_received() with reset(received).
3. Added a Probabilistic Failure and Hop Count feature to test some of the performance metrics as mentioned in paper[9]
4. Instead of performing the operation such as 'perform_periodic_operations()', 'fix_fingers()' and 'check_dead_nodes()' in every run, we have introduced a time-frequency based model (performing these operations at a certain frequency) as mentioned in this [9] paper to do performance analysis.
5. Appropriately handled exiting of Chord Nodes to ensure a smooth testing.
6. The was a bug in the FindKey method of Chord(Improper handling of Received values), which caused correctness to fail. But we fixed it.

Finally, I feel without the above mentioned fix, there is no way anyone can perform correctness or performance evaluation.

3.2 Java Implementation in DistAlgo

We have chosen open-chord implementation written in Java to compare with Distalgo. To compare it with Distalgo version we added following features:

1. Added a Probabilistic Failure and Hop Count feature of nodes in the chord ring as mentioned in paper[9] to test few performance metrics.
2. Added a feature to calculate storage and query load count for each nodes.

3.3 Driver Implementation

Both Distalgo and Java Implementation have below methods in their drivers:

1. Creating a Chord Ring: This function creates the chord ring with the number of nodes specified by the parameter N.

2. Insert Data: This function is used to insert data in the chord ring after creation of the ring. This function reads the DNS data from the csv file and stores sites - IP address value pairs in the chord ring.
3. Run Queries: This function runs the number of queries, Q which is specified by the parameter passed to drivers from Monitor program. To run these queries, driver does following steps:
    a. Probability Failure: This function temporary fails certain number of nodes and brings some of the nodes back to life. The number of nodes is decided by another parameter which is the probability passed from monitor program. In the driver, for each node in the ring we are generating a random number and if this number is greater than the probability then we are keeping or making the node alive. Then for each node in the ring we are again generating a random number and if this number is less than the probability then we are failing this node.
    b. Randomly select DNS data: This function randomly select the dns data from the list which will be used to query to the chord ring.
    c. Randomly select Chord Node: This function randomly select the chord node in the ring. The lookup will be called on this node.
    d. Retrieve Data: This function will call the lookup command on the selected node with the dns key.
    e. Calculate Performance Metrics: This function will calculate the different performance metrics i.e. Average Hop Count, Average Latency, Success Lookup Ratio and Time Out Ratio.
4. Shutdown Nodes: This function sends kill message to all the nodes in the chord ring to perform cleanup. Then the nodes send back their storage count which represents number of keys stored at each nodes and their query load count which represents number of times the query requests handled by the node in this run. Then these values along with above metrics are logged in files which will be read by Monitor program to compare the results, create plots and tables

3.4 Monitor Program

The monitor program performs following tasks :

1. It makes the system call to Distalgo driver and Java driver using python subprocess API.
2. Depending upon the type of run pass, the parameter is varied and 5 runs are done by keeping all parameters fixed and varying that given parameter
3. Run type are: Nodes or Failure or Stabilization_Delay or FixFinger_Delay or CheckPred_Delay or Storage_Load

4. Java and Distalgo implementations dump the Hop Count, Average Latency, Lookup Success ratio and Timeout ratio for all those 5 runs in files DistalgoResults.txt and Java_result.txt respectively.
5. Then Monitor program reads these 2 output files and plot the graphs with x axis as Number of nodes we are varying and y axis as performance parameters for comparison and stores them in results folder. Detailed analysis is mentioned in section 4.
6. For “Storage_Load run type”, Monitor program calls Distalgo driver in 5 runs and Distalgo implementation dumps Load and Storage balance of each node during insertion and deletion of keys and dump this information in DistalgoResult2.txt.
   Distalgo also dumps latency of all nodes over 5 runs in file latencyCDF.txt.
   Monitor program read this file and plot load balance v\s storage balance and Cumulative Fraction of Lookups vs Latency.

3.5 Comparison of both Implementations

Parameters	**Distalgo Implementation**	**Java Implementation**
Ease of Development	Easy to setup and develop	Easy to setup but not trivial to develop
Abstraction	Very Good	Good
Lines of Code	984	86500
Program Clarity	Readable and Understable	Difficult to debug
Correctness	Pass	Pass

Facts about existing Java Implementation:

1. Abstracted the communication layer by providing an interface to develop. This helps in testing the code on local (where it uses thread communication) and same code can be used to run in cloud environment (where it uses socket communication).
2. Simulating different scenarios to test chord correctness and performance, distalgo is much better than Java implementation because it provides APIs to write effective code without going much into the intricacies of low level communication protocols.

4. Testing and evaluation

In this section, we will majorly be comparing the performance of Chord and Java Implementation based on some varying parameter. The major performance metric across these Implementations would be the average Latency, average Hop Count, Success Lookup Ratio and Time Out Ratio in case of probability.

Machine Configurations:
Below testing is done on mac
hine with following configurations:
Processor: 2.3 GHz Intel Core i5
RAM: 8GB
OS Version: macOS Mojave Version 10.14.1

4.1 Correctness Verification:

To verify correctness for DNS implementation using Chord we are evaluating following things:

1. Verify the value for the given key is returned correctly: In a chord ring, once data is inserted and we are performing a lookup for a key the data returned for that key should match the expected data.
2. Verify if there is no data for the key then it should return an empty value.

We are performing correctness verification for each of the runs by reading the CorrectnessData.csv file which contains test data. In this test data, there are few key and value pairs which are present in the chord ring and few which are not. We are verifying both the above conditions using this data. If the correctness is violated, we are reporting this violation.

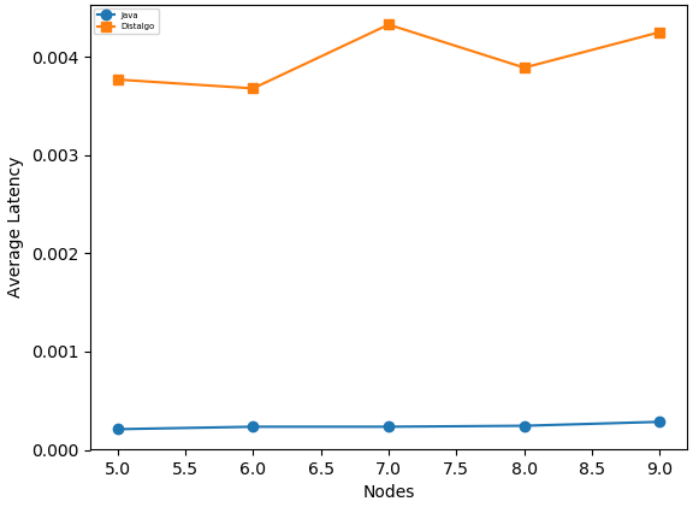
4.2 Performance Analysis:

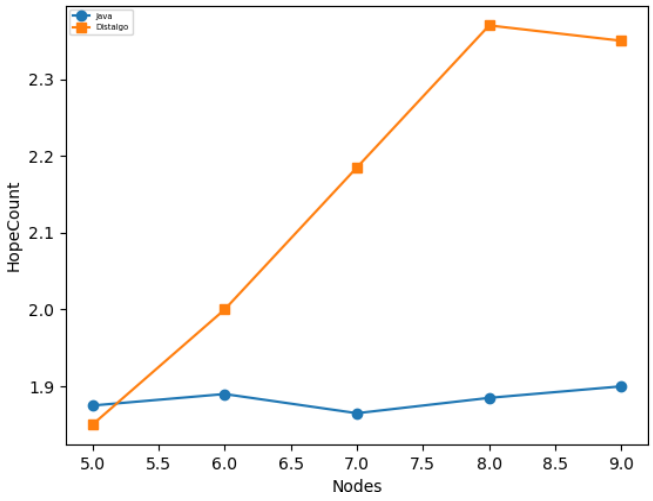
In below graphs: Orange line is Distalgo and Blue line is Java

4.2.1 Varying Number of Chord Nodes

Parameters :
No of bits : 5, *successor list* : 2, *Total number of queries* : 200, *Stabilization delay* : 0.1, *Fix Finger Delay* : 0.1, *Check Predecessor Delay* : 0.1, Probability of failure = 0
Number of Nodes varies from 5 to 9 over 5 runs

The idea to vary this parameter is taken from [9].
Inference : As stated in chord's paper [12] on page 18 at section 3 **" Scalability: The cost of a Chord lookup grows as the log of the number of nodes, so even very large systems are feasible".** As the number of nodes increases from 5 to 9 the hop count in distalgo also increases whereas in Java because of replication and lesser number of nodes, average hop count oscillates between 1 to 2. In case of average latency, its value increases as number of nodes increases in distaglo but in java it remains between 0.00021s to 0.00028s.



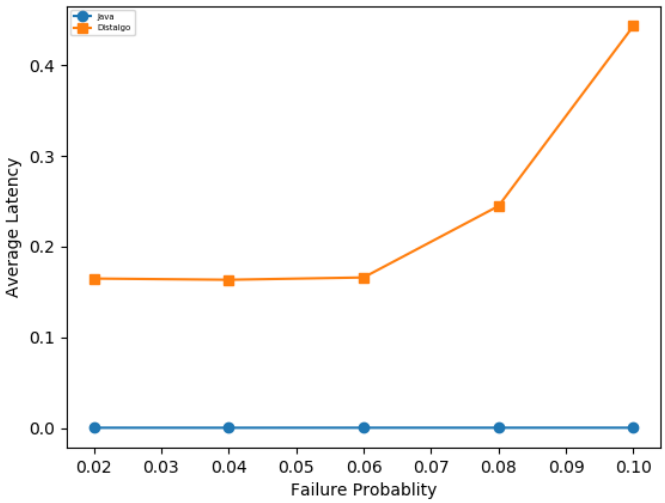


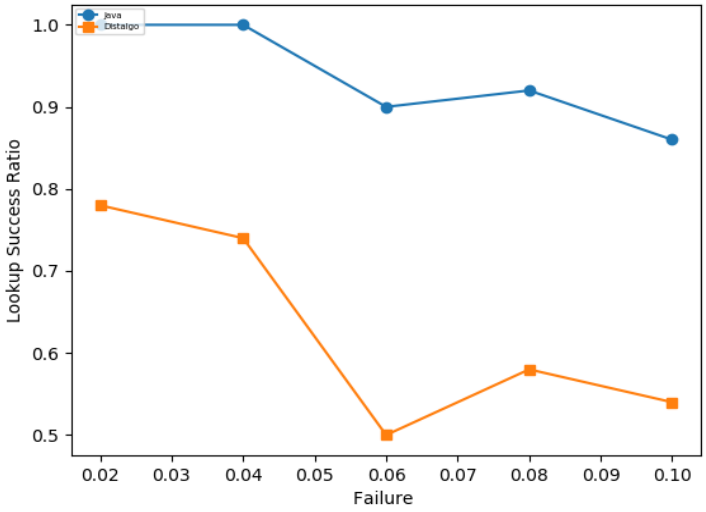
Nodes	Java HopCount	Distalgo HopCount	Java Average Latency	Distalgo Average Latency
5	1.875	1.85	0.00021	0.00377184
6	1.89	2	0.000235	0.00368156
7	1.865	2.185	0.000235	0.00433148
8	1.885	2.37	0.000245	0.00389279
9	1.9	2.35	0.000285	0.00425465

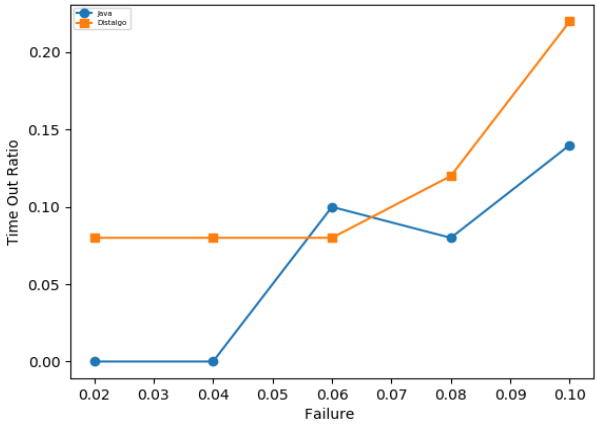
4.2 Varying the Probability of Failure

Parameter:
No of bits: 5, Number of Nodes: 7, *Successor list* : 2, *Total number of queries* : 200, *Stabilization delay* : 0.1, *Fix Finger Delay* : 0.1, *Check Predecessor Delay* : 0.1
Probability of failure varies from 0.02 to 0.1 over 5 runs

The idea to plot these graphs are taken from research paper [2]
part D of section V.







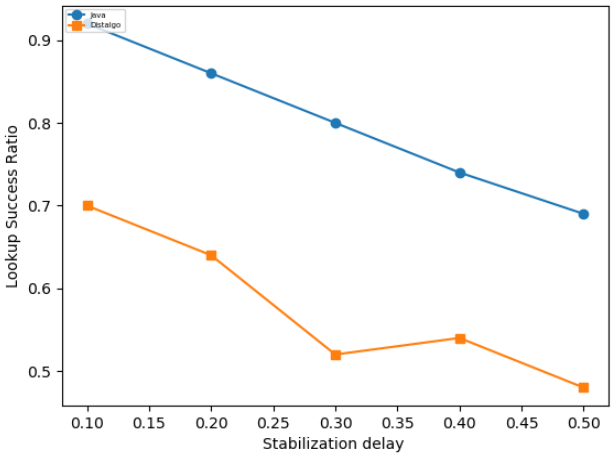
Inference: As stated in Chord's paper [2] on page 28 table 3 and also We got the same inference as stated in Figure 12 in research paper [10] that is as the failure probability is increased, the lookup success ratio decreases. As seen in below graphs and table, both for Java and Distalgo implementation as we are increasing the failure probability lookup success ratio decreases, time out ratio increases, and average latency increases.

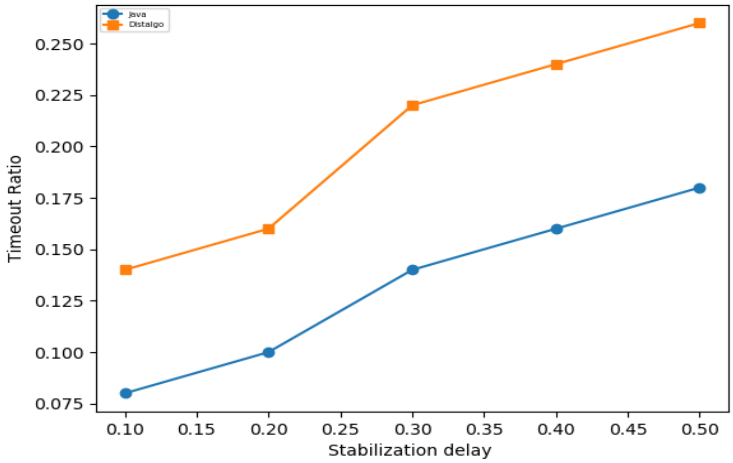
Failure Probability	Java HopCount	Distalgo HopCount	Java Average Latency	Distalgo Average Latency	Java Lookup ratio	Distalgo Lookup ratio	Java Timeout ratio	Distalgo Timeout ratio
0.02	1.86	2.98	0.0003	0.164892	1	0.78	0	0.08
0.04	1.8	1.62	0.00028	0.163537	1	0.74	0	0.08
0.06	1.86	4.2	0.00032	0.166077	0.9	0.5	0.1	0.08
0.08	1.92	2.06	0.00032	0.244868	0.92	0.58	0.08	0.12
0.1	1.88	1.36	0.00032	0.443499	0.86	0.54	0.14	0.22

4.2.2 Varying Stabilization Delay

Parameters :
No of bits : 5, Number of Nodes : 7, *Successor list* : 2, *Total number of queries* : 200, *Fix Finger Delay* : 0.1, *Check Predecessor Delay* : 0.1, *Probability of failure : 0.1*
Stabilization delay varies from 0.1 to 0.5 over 5 runs
The idea to plot these graphs are taken from research paper [9] section 6.

Inference: As stated in chord's paper [12] on page 24 **"In the final case, the nodes in the affected region have incorrect successor pointers, or keys may not yet have migrated to newly joined nodes, and the lookup may fail "** and also in research paper [9] section 6, Figure 1a that as the stabilization delay increases the lookup success ratio should decrease. This is the same inference which we got here for both the implementations, as we are increasing the stabilization delay from 0.10s to 0.50s, we see a decrease in lookup success ratio. Additionally, we plotted timeout ratio vs stabilization delay and as we can see from this graph, timeout ratio increases as we increase stabilization delay.





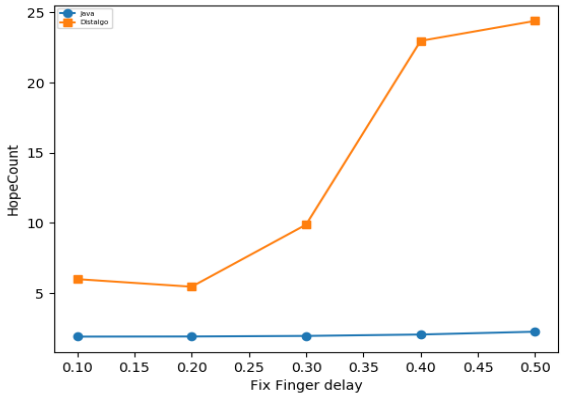
Stabilization delay	Java HopCount	Distalgo HopCount	Java Average Latency	Distalgo Average Latency	Java Lookup ratio	Distalgo Lookup ratio	Java Timeout ratio	Distalgo Timeout ratio
0.1	1.88	3.72	0.00026	0.297364	0.92	0.7	0.08	0.14
0.2	1.88	1.7	0.00026	0.327241	0.86	0.64	0.1	0.16
0.3	1.92	1.46	0.00026	0.489931	0.8	0.52	0.14	0.22
0.4	1.94	1.62	0.00026	0.448709	0.74	0.54	0.16	0.24
0.5	1.9	1.58	0.00028	0.248311	0.69	0.48	0.18	0.26

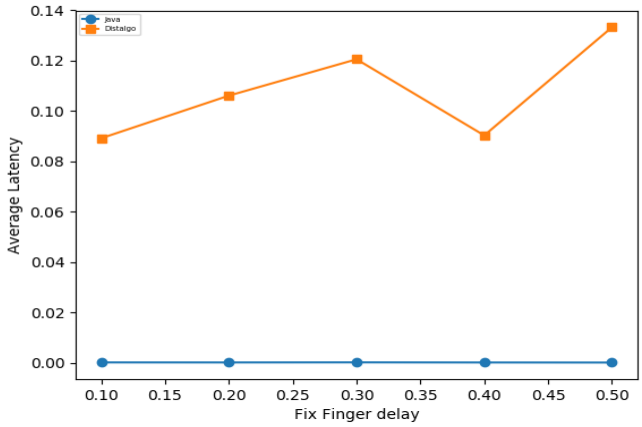
4.2.3 Varying Fix Finger Delay

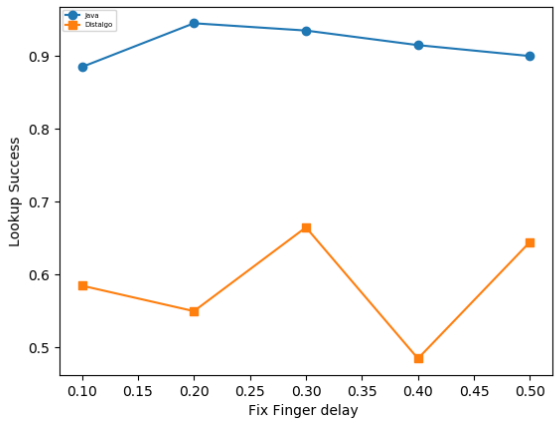
Parameters :
No of bits : 5, Number of Nodes : 7, *Successor list* : 2, *Total number of queries* : 200, *Check Predecessor Delay* : 0.1, *Stabilization Delay* : 0.1, *Probability of failure : 0.1*
Fix Finger Delay delay varies from 0.1 to 0.5 over 5 runs

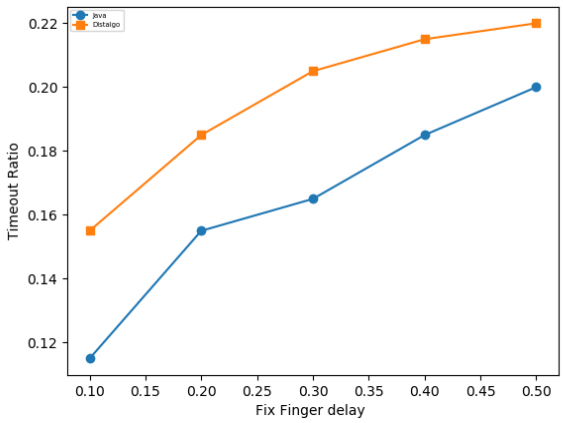
The idea to plot these graphs are taken from research paper [9] section 6.

Inference: We got the related inference as stated in research paper [9] section 6 and also mentioned in chord's paper [2] section on page 24 **"The second case is where successor pointers are correct, but fingers are inaccurate. This yields correct lookups, but they may be slower"** that is as the fix Finger Delay increases, there will be no impact on lookup success ratio as keys will eventually find their correct locations. But the hop count will increase because finger table is not updated and keys will take longer route to find their correct node location and this will also result in increase in average latency.







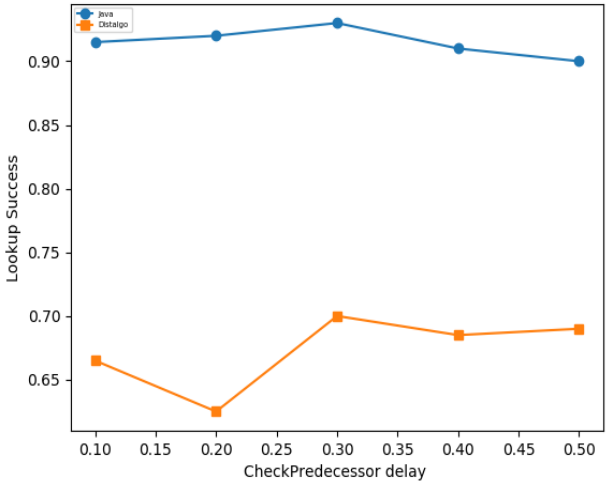


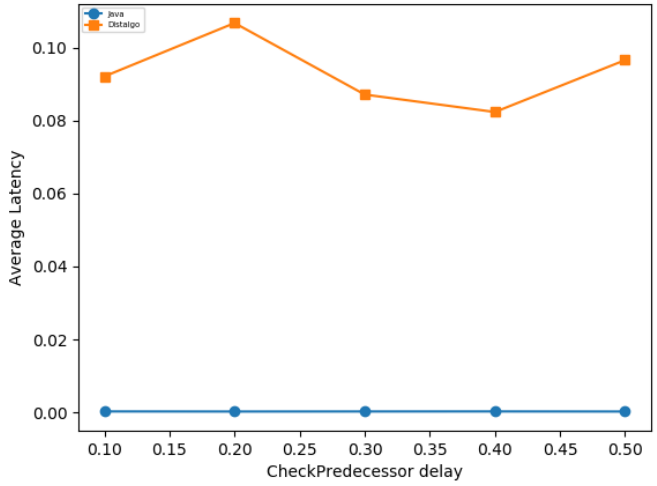
Fix Finger delay	Java HopCount	Distalgo HopCount	Java Average Latency	Distalgo Average Latency	Java Lookup ratio	Distalgo Lookup ratio	Java Timeout ratio	Distalgo Timeout ratio
0.1	1.9	5.995	0.000255	0.0891866	0.885	0.585	0.115	0.155
0.2	1.91	5.455	0.00024	0.106103	0.945	0.55	0.155	0.185
0.3	1.95	9.88	0.000275	0.120531	0.935	0.665	0.165	0.205
0.4	2.05	22.975	0.00023	0.0903252	0.915	0.485	0.185	0.215
0.5	2.25	24.395	0.000205	0.13324	0.9	0.645	0.2	0.22

4.2.4 Varying Check Predecessor Delay

Parameters :
No of bits : 5, Number of Nodes : 7, *Successor list* : 2, *Total number of queries* : 200
Fix Finger *Delay* : 0.1, *Stabilization Delay* : 0.1, *Probability of failure : 0.1*
Check Predecessor Delay : 0.1 to 0.5 over 5 runs

The idea to plot these graphs are taken from research paper [9] section 6.
Inference: We got the related inference as stated in chord's paper [2] section on page 24 **" The main way in which newly joined nodes can influence lookup speed is if the new nodes' IDs are between the target's predecessor and the target. But unless a tremendous number of nodes joins the system, the number of nodes between two old nodes is likely to be very small, so the impact on lookup is negligible."** that is as the check predecessor delay increases, there will be not be significant impact on lookup success ratio and average latency.





4.2.5 Constant Parameter for Multiple Runs

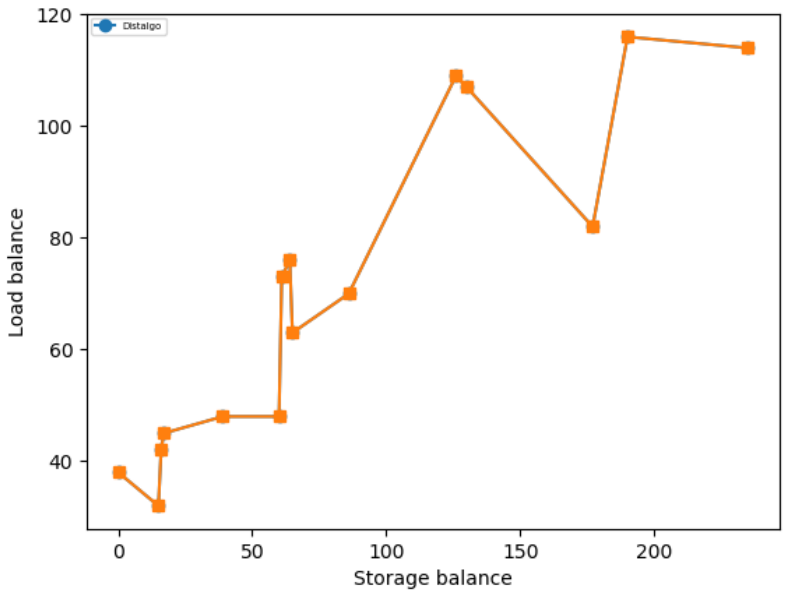
Parameters :
No of bits : 5, Number of Nodes : 7, *Successor list* : 2, *Total number of queries* : 200, *Check Predecessor Delay* : 0.1, *Stabilization Delay* : 0.1, *Probability of failure : 0.1*
Fix Finger Delay : 0.1

In this section, we will analyze some interesting properties of chord protocol. This analysis will be specific to Chord Implementation in DistAlgo.

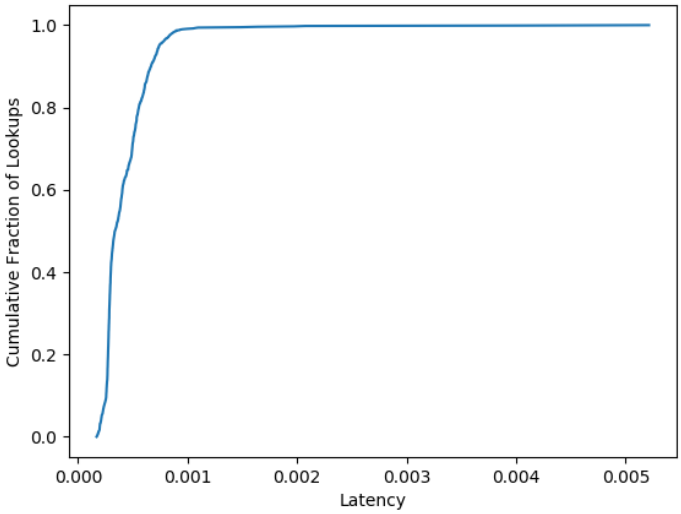
- **Storage Vs Load**

  The idea to plot this graph is taken from research paper [9] section 6.

Inference: We can see that as Storage balance increases, load balance also increases as chord needs to store more keys in the nodes, it needs to increase the load for the same.



- **Cumulative Fraction of Lookups vs Latency**
  This graph is taken from Research Paper [3] figure 6 and our results matches with that of paper results. For each query we dumped it's lookup and latency and then plotted cumulative lookups vs latency.



4.3 Instructions to Run the Code:

Below are the instructions to run this code. We have also mentioned the instructions to run our code in Readme.md file in the github repository.

1. **Install the dependencies:** We are using below libraries in the code to plot the graph, call driver programs, etc. Please run below commands to install below libraries before running this code:

a. pip install numpy
b. pip install matplotlib
c. pip install beautifultable
d. pip install psutil

2. **Compile the Java code:** First navigate to open-chord directory and then run the below command according to your machine to compile open-chord java code

Windows:

```
javac -d bin -sourcepath src -cp
./lib/*;./config/chord.properties src/myapp/driver/Driver.java
```

MacOS/Unix (Using Make File):

```
make
```

3. **Run the Monitor file:** Execute following command to run monitor.py file:
   python monitor.py runType m n r q s_d f_d p_d prob

Parameters to be passed:

runType : where runType is string that you want to vary keeping other parameters fixed. It can be Nodes, Failure, Stabilization_Delay, FixFinger_Delay, CheckPred_Delay, Storage_Load
m = number of bits for finger table size and maximum number of nodes in chord ring. Default value: 5. Constraint: This value should be in between 2 - 16.
n = number of nodes in the chord ring. Default value: 6
r = success list size. Default value: 2
q = number of queries. Default value: 200
s_d = Stabilization delay. Default value: 0.1s. Constraint: This value should be in between 0-1.
f_d = Fix Finger delay. Default value: 0.1s. Constraint: This value should be in between 0-1.
p_d = check predecessor delay. Default value: 0.1s. Constraint: This value should be in between 0 - 1.
prob = Probability failure ratio. Default value: 0.1. Constraint: This value should be in between 0-1.

After running the program, chordTabularResults.txt will contain the tabular results and graphs will be stored in results folder.

Most of these constraints above are placed due to differences in Java and Distalgo implementations, and we want to run both of them together to compare the results.

Conclusion:

After understanding the Chord Protocol and performing comparisons between these two different implementation, we have come to the following conclusion:

1. The results and graph trends obtained from DistAlgo version were more realistic and were comparable to the results obtained in the corresponding papers.
2. DistAlgo is very good language for better understanding distributed Algorithm as it abstracts the developer from internal details such as message handling which helps a developer to focus on making the code correct and efficient rather than focusing on setting up the environment. Understanding how nodes are communicating was tough in Java as compared to Distalgo which abstracts all the internal details.
3. We took the best Java Implementation which had Replication (Fault Tolerance) and tried to compare it with DistAlgo. Therefore some of the parameters could not be tested appropriately in JAVA. But we could have done that if we had included more number of Nodes.
4. Finally, after this project our take home lesson is that it is really difficult to write a correct distributed algorithm, as even a small mistake can cause a system to fail and it is very difficult to replicate the same scenario in future to make corrections.

Future/Additional Scope:

1. Adding Fault Tolerance / Replication in the existing implementation of Chord in DistAlgo.

References:

[1] Paul Albitz and Cricket Liu - *DNS and BIND. O'Reilly & Associates, 1998.*

[2] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan - Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications - IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 11, NO. 1, FEBRUARY 2003

[3] Russ Cox, Athicha Muthitacharoen, Robert T. Morris - Serving DNS using a Peer-to-Peer Lookup Service - International Workshop on Peer-to-Peer Systems 2002 pp 155-165

[4] Chidambaram Ramanathan - https://github.com/ChidambaramR/Asynchronous-Systems

[5] Josh Tan - https://github.com/jtan189/open-chord

[6] Peter Danzig, Katia Obraczka, Anant Kumar - An Analysis of Wide Area Name Server Traffic - Conference proceedings on Communications architectures & protocols, Volume: 22

[7] http://en.wikipedia.org/wiki/Chord_(peer-to-peer)

[8] Annie Liu, Bo Lin - https://github.com/DistAlgo

[9] Farida Chowdhury, Mario Kolberg - Performance Evaluation of Structured Peer-to-Peer Overlays for Use on Mobile Networks - 2013 International Conference on Developments in eSystems Engineering (DeSE)

[10] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan - Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications - In Proc. ACM SIGCOMM'01, San Diego, CA, Aug. 2001

[11] Pamela Zave - Reasoning about Identifier Spaces: How to Make Chord Correct - IEEE Transactions on Software Engineering, 43(12):1144-1156, December 2017