CS 6968/5968 – Data Str Alg Scalable Comp Spring 2023

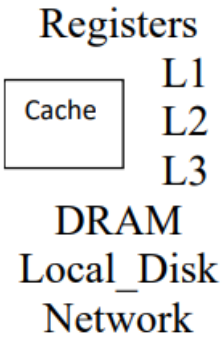
Lecture 18 — 10 April, 2023

Prof. Prashant Pandey *Scribe: Pranjal Patil*

1 Overview

In this lecture we study about how files are distributed across various devices and different solutions to solve the challenges that comes with it. The solutions discussed are Centralized Index System, Gnutella and Distributed Hash table.

2 Memory Hierarchy



The cost of running the Distributed Algorithm is proportional to cost of network traffic. As we go down the hierarchy the speed to access data decreases.

3 Main Goal

The main goal is to handle large dataset that cannot be stores in one machine and hence improving the throughput and scalability challenges to access the dataset.

Let us understand 2 basic terms:

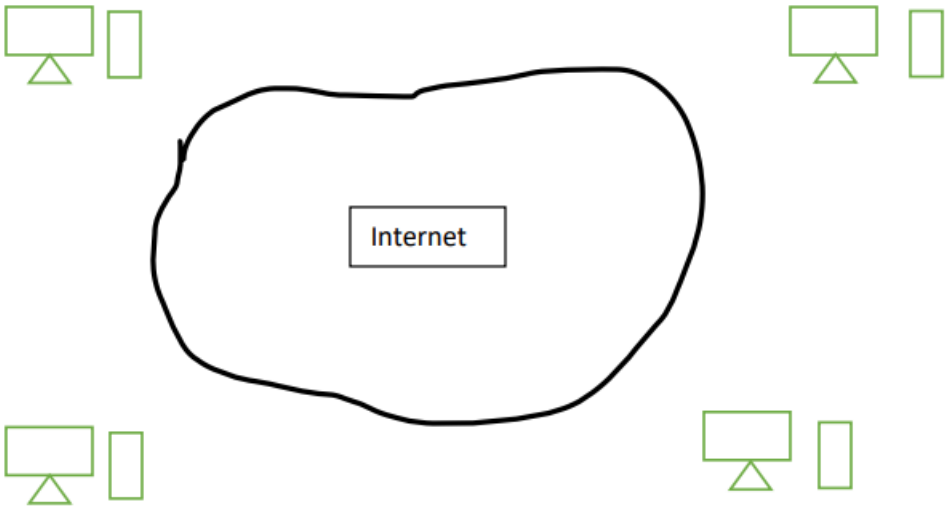
- **Scaling in/down** - Here we try to compress the data to improve the performance. The example can be like suffix tree, VEB trees etc.

- **Scaling up** - Here we try to improve the throughput bu increasing the number of resources for example increasing RAM, number of cores etc.

3.1 USE CASE (NAPSTER)

The key Idea - share the content, storage, and bandwidth of individual users.
Let us understand this through a diagram:

Figure 1: Distributed Systems



Model:
Music is distributed across different computer. Each user is a computer node and they store a subset of files. Every user has access to files from all users in the system.

The major challenges are:

- searching a particular file.
- Scaling up to thousands/millions of machines
- Dynamicity - Machines can go and come anytime hence updating the whole network can be challenging

The next sections provides solutions and their advantages and disadvantages.

3.2 Centralized Index System

In this we design a design a centralized index system that maps the item to machines that are alive. To query an item/file we just have to query the index system and return the machine that stores the index file.

Advantages:

- Simplicity - Easy to implement.

Disadvantages:

- The system is not scalable in terms of size throughput.
- Robustness - Eg: in case single machine goes down create the entire index again.

3.3 Gnutella

In this the main idea is to distribute the File locations. To query we flood the network with requests.

- Send request to node. Then it sends request to all neighbors.
- The neighbors recursively multicast the request.
- Eventually the machine that has file receives the request and send back the answer.

Advantages:

- It is a **decentralized system**
- Highly robust

Disadvantages:

- This is not scalable. In the case there are many requests, bandwidth gets saturated and hence it will take long time
- the entire network can be swamped with requests

To alleviate this there are two methods, first is **caching** in every node and second is each request has **Time to Live**. After a particular number of hops the process stops and restarts again.

Important Points to note:
The network here is Adhoc Topology
Queries are flooded for bounded number of hops.
There are no guarantees on recall.

The above two points can be solved by adding a few machines to decrease diameter of overall network.

3.4 Distributed Hash Table(DHT)

Abstraction: This is a distributed hash table data structure.
API's :

- Insert(id, item)- insert item into table by calculating the hashvalue of id.
- Query(id) - Query the item corresponding to the id

Here the item can be anything like data object, document file, pointer file etc.

There are various proposals for Distributed Hash Tables - CAN, **Chord**, Kademila, Pastry, Tapestry etc.

DHT design goals:-

- Scaling the system to hundreds of millions of models
- Handles rapid arrival departure of nodes in the network
- Make sure that an item identified is always found.

3.5 CHORD

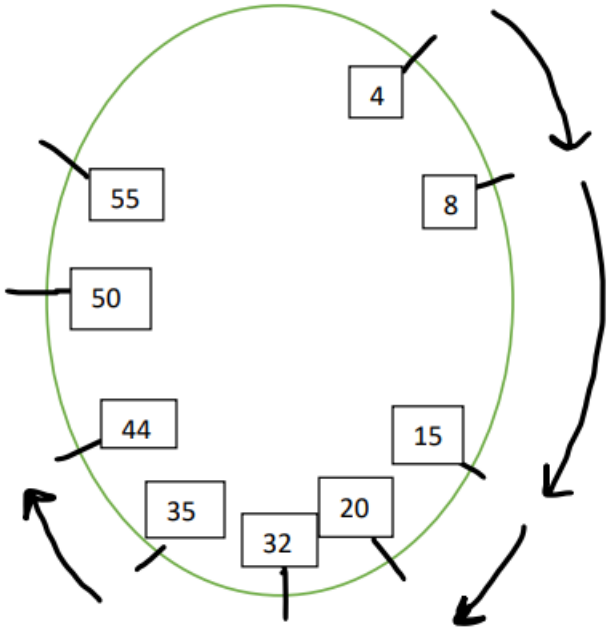
Associate to each node and items a unique id in a uni-dimentional(we have discussed ring topology in this case) space 0..... 2^{m-1}. Here m is number of nodes.

Key Design Decision- Decouple the correctness from efficiency.
Properties:-

- Routing Table size is log(N). here N is total number of nodes.
- It guarantees a file is found in O(logN) steps.

Let us understand this through a diagram.

Figure 2: Ring Topology of network



The numbers 4, 8, 15, 20, 32, 35, 44, 55 are the nodes in the network. The node are hashed to a hash value that is used in this topology. The items/files are also hashed in this system. The hash values of items/files lying between 4(exlusive of 4) and 8(inclusive of 8) are stored in 8. In same way 15 stores item whose hashvalue is between 8(exclusive) and 15(inclusive). Here each node maintains a pointer to its successor/predecessor.

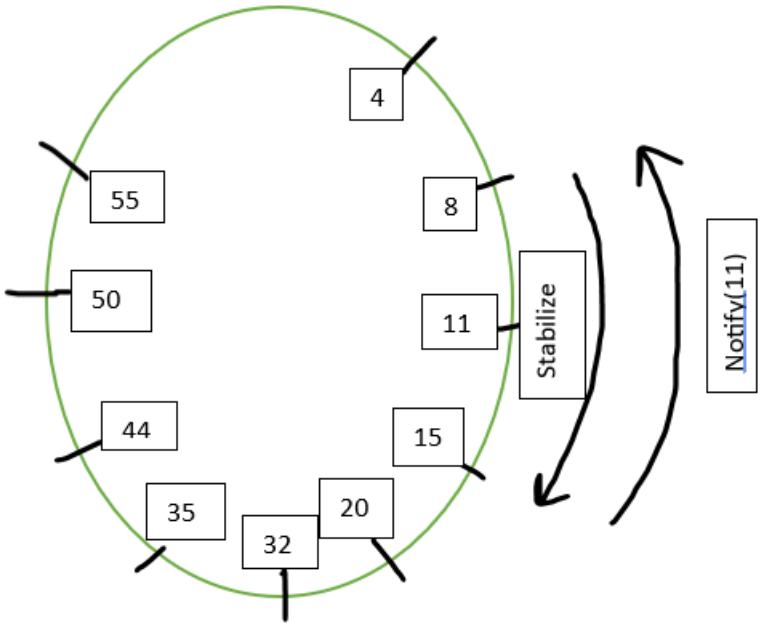
Look up of file [1]:
Each node maintains the successor pointer hence during look up the first idea is to route packet(id, data) to the node responsible for id using successor pointer.
Note that the search is done always in **clock-wise direction**.

Joining Operation:
This is the operation that confirms/updates the successor and predecessor of a node in case of addition of new node.
The process is like:

- Each node A periodically sends **stabilize()** message to its successor.
- Upon receiving a stabilize() message, node B returns its predecessor B' = pred(B) to A by sending a **notify(B')** message.
- Upon receiving notify(B') from B there are two cases possible.

- If B' is between A and B then A updates its successor to B'. else A does'nt do anything.

Figure 3: Ring Topology of Network



Let us understand this using an example. Suppose there is a new node 11 inserted between 8 and 15. When 8 sends the stabilize() message to 15 (currently the successor of 8), 15 send a notify(11) message to 8 that indicates that the predecessor of 15 is 11 and not 8. Hence then 8 changes its successor to 11.

This above look up method has a linear complexity in terms of number of nodes. But we stated in properties that lookup is logarithmic in terms of N(number of nodes). To achieve this goal we add a table in each node known as Finger Table.

FINGER TABLE:

This is a brief information of what is available in the network. It stores the node that are in range of power of 2. For example consider the figure used in Joining Operation. Equation to find finger table at each node is:

$\mathbf{id \geq (n+ 2^{i})mod\ (2^{m})}$
Here id is nodes in network, n is current node, 2^m gives total number of possible nodes.
To understand this equation lets find the finger table of node 8.

i	FT[i]	
0	(8+1)%256	11
1	(8+2)%256	11
2	(8+4)%256	15
3	(8+8)%256	20
4	(8+16)%256	32

Using the finger Table we can traverse to the node that is closest to the query node. This will reduce the search in half in every iteration. Hence the complexity of lookup is O(log N) N is number of nodes.

References

[1] Ion Stoica, Robert Tappan Morris, David R. Karger, M. Frans Kaashoek, Hari Balakrishnan:. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. SIGCOMM 2001: 149-160