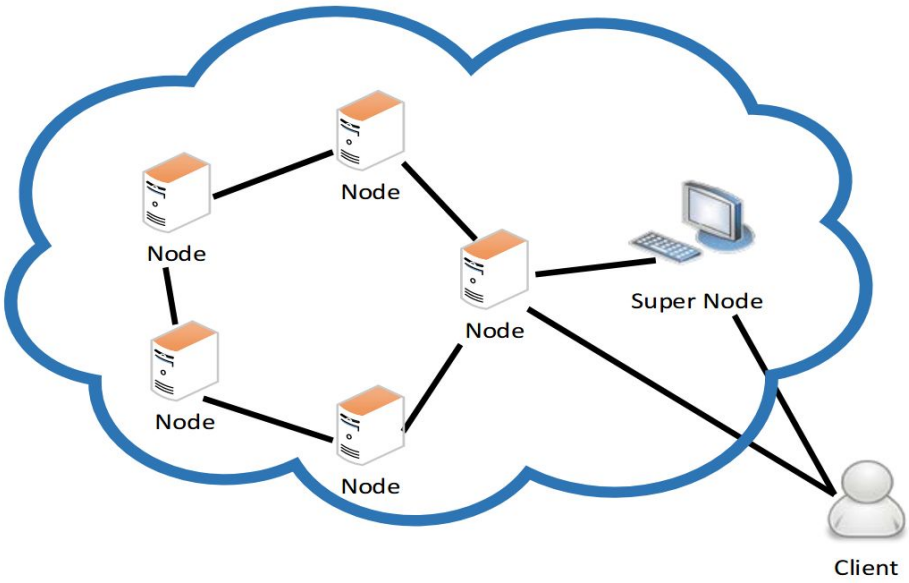
Book Finder System on a DHT
Design Document

Sravya Amudapuram,Saurabh Gupta
(amuda005,gupta555)

We implemented book finder system using Distributed Hash Table(DHT).

- There are 2 main components of Distributed Hash Table in our design: 1) Supernode and 2) DHT Node. We create a circular key space ranging from 0 to 2^m(1<=m<=64).
- We assign a unique random id to a node joining the DHT by hashing the node ip and port combination and divide the key space among the joining nodes.
- We store book title and book genre based on the hash of the book title. The node which is responsible for the key space that contains the hash of the book name will store the book name and genre.
- To fetch the genre of the book, the client contacts the super node and asks ip and port of the DHT Node which is part of DHT and queries it to get the book genre.
- The request for setting and getting book genre is passed using the algorithm defined in the “Chord: A Scalable peer-to-peer Lookup Service for Internet Applications”
- We assume the node count to be limited by 2^m. However, there is no restriction on how many nodes can join the system. System is adaptable to joining of the nodes but doesn’t provide functionality to leave the DHT.



[1] **Supernode** : It acts as the manager of the DHT system. It is the well defined node that client and new node joining the DHT need to connect to get one of the existing nodes in the system. Client uses this node to get and set the book while new node uses this node to get the unique id, and address of a node using which it could join the DHT.

Supernode provides following functions to interact with it:

Interface: SuperNodeService
service{
    string join(1:string ip, 2:int port),
    void postJoin(1:string ip, 2:int port),
    string getNode()
}

// Client Function
getNode() - This function is used by the client to get any random node of the DHT to set and get book.

//DHT Node functions:
postJoin() - Used to release the lock taken by the joining node and also to inform supernode that new node is ready to serve and system is stable

string join(1:string ip, 2:int port) - This function takes a lock when a new node wants to join the node and doesn't let any node to join the DHT if some other node is already joining the DHT.

[2] **DHT Node**:
DHT Node provides following interface for maintaining DHT nodes, distributing the keys, updating predecessor, successor and finger table. It also provides functionality to store and retrieve book title and genre into DHT.

```
namespace java edu.umn
typedef i64 int

struct DHTNode{
        1:string ip,
        2:int port,
        3:int id,
}

struct DHTNodePath{
        1: DHTNode node,
        2: list<DHTNode> visitedNodes,
}

service DHTNodeService{
        //Used by Client
        string getBook(1:string bookTitle, 2:int debugFlag),
        void setBook(1:string bookTitle, 2:string genre, 3:int debugFlag),
        // Used Internally by DHT Nodes
```

```
    void updateFingerTable(1:DHTNode newNode, 2:i32i,3:list<DHTNode>
  visitedNodes),
    DHTNodePath findSuccessor(1:int key,2:list<DHTNode> visitedNodes),
    DHTNodePath findPredecessor(1:int key, 2:list<DHTNode> visitedNodes),
    void setPredecessor(1:DHTNode predecessor),
    map<string,string> transferKeys(1:int key),
}
```

Joining DHT:

DHTNode makes a join call to SuperNode to initiate node joining process. SuperNode returns a new unique identifier, a random node's host, ip and id. All these values are sent to DHTNode in a string with comma separated values.

Update predecessor:

After join call to supernode, DHTNode makes a call to the given random node to fetch its own successor. Once it obtained its successor, it makes RPC call to successor to fetch its predecessor. The new node now places itself between predecessor of successor and successor by making setPredecessor call.

Populate Finger Table:

Then, DHTNode starts initiating its finger table by making findSuccessor RPC calls for each entry.

Update DHT:

Then, it makes updateFingerTable RPC call to nodes whose finger table entries may have new node. updateFingerTable recursively calls predecessor's updateFingerTable.

Transfer keys:

After updateDHT, it makes transferKeys RPC call to successor to fetch its keys. After transferring keys, it calls postJoin of supernode to finish its node joining process.

After all nodes repeat this process, DHT is completely formed.

Setting and getting book titles:

DHTNode provides methods to client - getBook and setBook - to set and get genres for book titles. Client can also set an optional flag-"debugFlag"- to track visited nodes while getting and setting genres of book titles. Once DHTNode

receives a client request to setBook or getBook, it recursively calls findSuccessor to find a DHTNode responsible for the book title. Once it finds the responsible DHTNode, it makes an RPC call to setBook or getBook and returns the response obtained from the corresponding DHTNode.

[3]
Hash Function: We are using **MD5** hash to create the node id and key for the book title. We are using a key space of 2^m and hence we take modulus of resulted hash bytes with 2^m. This result could potentially cause collisions. As the node id should be unique, therefore we are rehashing till we get the node id which is not in the system.

We are using the same hash function for hashing book titles. In case of a collision, we resolved it by chaining the key-value pairs.

Test Cases and Results:
We have tested the system for various positive and negative cases. Detailed instructions to replicate the test cases has been given in the file “Output-DHT.pdf” Some of the test cases are listed as follows:
[1] Setting book and genre using the file which doesn’t exist in the system
[2] Searching for a book genre who name was not added into the system.
[3] Adding book using file (setting in bulk)
[4] Setting one book name and genre in the DHT at a time
[5] Creating DHT by joining node one by one and validating the transfer of keys from successor of new node to successor.
[6] Validating the finger table for existing node when nodes join the DHT consecutively.
[7] Tested tracing of call for get and set book when the trace flag is true.
[8] Tested tracing of call for get and set book when the trace flag is false.