**Data Structures Project:**

**Distributed Hash Tables**

**Group members:**

* Husnain Rasul (19i-0441)
* Abdullah (19i-0476)
* Abdullah Bin Faiz (19i-0695)

**Section:** CS-A

**Submitted on:** 27th December, 2020

**Contents:**

**Synopsis**

**Design**

**Procedures**

**Synopsis:**

A **Distributed Hash Table** is a convenient database system designed to lookup and store **(key, value) pairs** homogeneous to a hash table, where the mapped key is responsible for providing a certain value, be it a file path, a password, or anything else. The (key, value) pairs are stored inside a machine linked to other machines in a ring, i.e. a peer-to-peer connection. The key in the pair determines which machine is responsible for storing the pairs. The key is passed through a hashing algorithm which is converted to a machine ID, and traversing to said machine ID, the pair is stored inside it. Later, to search up the value, a user can input the key and the machine will retrieve the value from the corresponding machine, should it be present inside the machine. However, inserting or searching in a linear fashion where there can be a million machines connected would cause the program to take a significant amount of time to execute the operation. For this purpose, a **Routing Table** (or a **Finger Table**) is devised for each machine which pinpoints to the machine where said value should be stored.

The way an RT/FT works is that a machine holds, in a successive powers-of-two format, an array of active machine up to **2n-1** Where **n** depicts the identifier bits of the machines (**2n** equals the total amount of machines that can be allocated to the system). The number of machines in the array is limited to a format, the **succeeding** procedure, where the next active machine is calculated using **succ(p + 2i-1)**, **p** being the machine ID from where we lookup or store. When searching, the algorithm seeks the machine ID closest to the decrypted ID from the hashed key and jumps to the machine. As a result, the complexity achieved is **O(log(n))**.

Looking up at the design, out project managed the database described below.

**Design:**

Unidirectional Association:

Machine Node

-owns: Machine Node\*

Machine

Unidirectional Association:

FT Node

-owns: FT Node\*

FT

Unidirectional Association:

AVL Node

-owns: AVL Node\*

AVL Tree

Composition:

FT

-owns: FT

Machine Node

-owner: Machine Node

Aggregation:

AVL Tree

-owns: AVL Tree

Machine Node

-owner: Machine Node

Aggregation:

Machine Node

-owns: Machine Node

FT Node

-owner: FT Node

Aggregation:

Machine Node

-owns: Machine Node

FT

-owner: FT

**Procedures:**

We will tackle the procedures responsible for executing our program step by step, following the operations stated to be included in the project PDF:

1. **“Option to specify the size of identifier space in bits.”**

A simple input from user is taken with no validation. This input is then used as a power-of-two to calculate the maximum number of machines that can be allocated to the database.

1. **“Option to specify the number of machines.”**

A validation was set to enter the number of machines to be incorporated into the Ring DHT. The number of machines to be inserted initially should not exceed the total number of machines that can be allocated to the database (as calculated in **1.**).

1. **“Option to manually assign ID to each machine. If manual IDs are not assigned by the user, the system must automatically assign IDs.”**

Another validation is set when asking for input from user, where the number of machines to be manually assigned must be within the range of 0 and the number of machines entered previously in **2.** exclusive. When manual inputs have been taken (or left for the system to automatically allocate), the keys are passed through a custom ID function where the inputs are passed through a SHA function to get a hash, which is then converted into decimal and passed through a modulus of the maximum number of machines that can be allocated (as in **1.**) to get the Machine ID. This Machine ID is assigned to the Machine Node and respectively inserted into the Machine Circular Linked List.

1. **“Option to insert data in the form of (key, value) pairs from any machine.”**

The (key, value) pair taken from user is passed onto the Ring DHT’s “**insert\_data(template, template, template)**” function of Machine (Ring DHT). The first and second template values will retrieve the Machine ID from where we begin, and the machine ID where we will store in. Both of the type variables will be passed through the custom ID function and return the respective Machine IDs for insertion. The “**insert\_data(int, template, template)**” function of “Machine Node” is called, where first, the Machine Node is searched through the “**succ2(int, template, this)**” function, and then the data is inserted inside the node using the “**Insert\_Data\_Node(Machine Node, template, template)**”. This function simply inserts the (key, value) pair inside the AVL tree (with appropriate rotations) as well as a file located on the system. All of the machine nodes currently existing in the Ring DHT is (re)populated with the succeeding members accordingly, and this function is called for every insertion of machine. The AVL tree of said machine is subsequently printed afterwards.

1. **“Option to remove data by specifying the key from any machine.”**

Similar to **4.**, only the Machine ID and key is asked from user. The Machine ID is hashed and is broken down to retrieve the Node ID and the key is simply hashed to search for in the AVL tree. Once the machine is found using the Routing Table incorporated in the machine, the hashed key is then looked up inside the AVL tree. If the hashed key is found anywhere inside the tree, then that node is removed, and necessary adjustments are made. The AVL tree is then printed afterwards.

1. **“Option to print the routing table of any machine.”**

The Machine ID input from user is passed through the “**displayFT(template)**” function of Machine. The ID is hashed to get the Node ID and that ID is used to access the machine and use its FT object to display its Routing Table.

1. **“Option to print AVL tree maintain on any machine along with the location of files (and line numbers) on which the corresponding values are stored.”**

The Machine ID is taken as an input from user, and it is passed through the SHA function to retrieve the Node ID. That Node ID is simply passed through the Machine array and all of the data inside its AVL tree is printed in Pre-ordered format.

1. **“Option to add new machines on the fly without disrupting the functionality of Ring DHT.”**

Just like in **3.**, a new ID is asked from the user to add into the machine. That ID is passed through a hash function and subsequently a Node ID is retrieved from it. If, on that Node ID, the machine is not empty, it asks the user for another input. Once a unique Node ID is hit where no machine is present, a new Machine Node is inserted inside the array. All of its members are initialized and, should the successor of that node contain data that should belong in the new Node ID, said data is redistributed inside the AVL of the new Machine Node and every machine’s Routing Table is adjusted accordingly.

1. **“Option to remove any machine on the fly without disrupting the functionality of Ring DHT.”**

The concept is similar to **9.**, only that the user is asked for the Machine ID that has to be deleted from the array. Once the machine is found by hashing and getting the Node ID from the Machine ID, all of the data inside the deleted Machine Node is redistributed to the successor of that Node. Necessary adjustments are made and every machine’s Routing Table is adjusted accordingly.