CS2510-COMPUTER OPERATING SYSTEMS

mini Google

PROJECT 2 – FINAL REPORT

NATHAN ONG and ANURADHA KULKARNI

12/13/2015

Mini Google is a simple search engine to retrieve documents relevant to simple search queries submitted by users.

1. INTRODUCTION

This goal of this project is to design a simple search engine, referred to as **tiny-Google**. The main objective is to implement basic data-intensive application to index and search large documents. Data-intensive Computing and Cloud Computing are two emerging computing paradigms, which are poised to play an increasingly important role in the way Internet services are deployed and provided. Increasingly, large-scale Internet services are being deployed atop multiple geographically distributed data centers. These services must scale across a large number of machines, tolerate failures, and support a large volume of concurrent requests. This project gives an insight about how both data and work are divided across a cluster of computing machines.

2. TASKS OF tiny-Google

This section describes the three basic operations that are provided to the User through a simple User Interface:

2.1 INDEXING THE DOCUMENT

A client user submits an indexing request to the user interface, referred to as uiShell. The submitted request contains a directory path name, where the document is stored. In response, the uiShell establishes a connection to the tiny-Google server and uses the connection to send to the server the user's request to index the document specified by the path name. The uiShell then awaits the outcome. Upon receiving the request, the tinyGoogle server creates a process, referred as the indexing-master to carry out the document indexing task; the server then returns to wait for the next request from other clients. Upon receiving the indexing query, the master selects a set of indexing helpers, each residing in a different machine, and divides the task of indexing the document among these helpers. Each helper receives a segment of the document and creates a word-count for each word in its assigned segment; it then uses the word-count outcome to update the master inverted index. Upon updating the master index, each helper informs the master of the failure or success of its assigned task. Upon hearing from all helpers, the master informs the uiShell of the final outcome (success or failure), which in turn informs the user.

2.2 SIMPLE SEARCH

The user can also issue a search query to retrieve information related to already indexed documents. The search query contains a number of items, in the form a key words, which are relevant to the search. The uiShell establishes a connection to the tiny-Google server; it then sends the search query request, along with the associated items, to the tiny-Google server and waits for the response. Upon receiving the search query, the tiny-Google server creates inserts the query, along the Internet address and port number of the uiShell, into the "Work Queue", and if wakes up a sleeping search-query master, if one exits, to

handle the query. Then, the tiny-Google server returns to listen to new requests from other clients. The search-query master selects a set of search-query helpers, residing in different machines, and tasks each one of them to (i) search a segment of the master inverted index and (ii) retrieve the name of the documents which contain all the words of the query. Upon completing task (i) and (ii), the helpers "shuffle-exchange" aggregate the partial results for each document. Upon receiving the query outcome from all the helpers, the master sorts the outcome into a final response and sends it to the uiShell client. The uiShell client lists the outcome for the user.

2.3 RETRIEVE RELEVANT DOCUMENT

The user can also issue a search query to retrieve information related to already indexed documents. The search query contains a number of items, in the form a key words, which are relevant to the search. The uiShell establishes a connection to the tiny-Google server; it then sends the search query request, along with the associated items, to the tiny-Google server and waits for the response. Upon receiving the search query, the tiny-Google server creates inserts the query, along the Internet address and port number of the uiShell, into the "Work Queue", and if wakes up a sleeping search-query master, if one exits, to handle the query. Then, the tiny-Google server returns to listen to new requests from other clients. The search-query master selects a set of search-query helpers, residing in different machines, and tasks each one of them to (i) search a segment of the master inverted index and (ii) retrieve the name of the documents which contain all the words of the query. Upon completing task (i) and (ii), the helpers "shuffle-exchange" aggregate the partial results for each document. Upon receiving the query outcome from all the helpers, the master sorts the outcome into a final response and sends it to the uiShell client. The uiShell client lists the outcome for the user.

3. SYSTEM OVERVIEW :

This section identifies the components of the proposed system and defines them by their functionality without getting into details of their individual architecture or the interactions between them. It also briefly talks about the layering of mini Google upon which components act.

3.1 SYSTEM COMPONENTS:

Helper:

This application component runs on a set of end systems of the distributed network and these nodes will be providing services to the Master server to index the document and store the document provided by the client at its end.

Master:

This is the piece of application that runs on a set of end systems of the distributed application which are going to be known as Master Server. This is responsible for providing services to the Clients(s) to issue commands to index the document, search by keyword and retrieve a document. It also provides services to helper to register it with the master and index the document provided by the client. These documents are stored at the helper end.

Client:

This component is the overall consumer of the distributed system. They consume the service provided by the Master and Helper. This part of application will be exposed to end users to issue several requests using known commands to index the document, search by word, search by documents name etc.

3.2 PROTOCOL LAYERING:

The components of the system participate in the protocol layering and integrate as a whole system. The high-level layers of the protocol can be conceptualized as below.

User Interface:

This is the top most or outer most layes which provides a platform to the users to interact with the system. The UI defines a set of commands which are known as form of knowledge document or help file to the user. This layer passes the user to the next layer to interpret. The UI is implemented in the Client component. We have implemented the command-line interface (CLI) for user to issue commands for performing the three tasks.

Control and Document Transfer:

The layer is implemented in the clients and server systems (i.e. master and helper). The high level functionality can be visualized in two dimensions- firstly interpret the commands issued by users at UI and translate them to the below layer by invoking certain primitives and secondly handling requests for directory navigation. This is also responsible for handling exceptions encountered by the below layer and pass as user friendly message to the upper layer to ensure smooth execution of the system.

Network Channel:

This component constructs the fundamental or base of the whole protocol and is found in all systems or nodes participating in distributed architecture. It is responsible for connecting with the system components and provides transferring control and data over network in fundamental.

4. SYSTEM ARCHITECTURE

This section explains functionality and architecture of the components of the system and layered architecture of mini Google in detail.

4.1 DETAILS OF SYSTEM COMPONENTS:

Master:

This is the top most component of the application that is initialized in the beginning or in other words the system bootstraps by initializing the Master first. The client is aware of the logical name of Master and queries Master. Before any client can start communicating with Master, the Helper registers itself with Master. The Master maintains a linked list in its primary memory. The linked list stores the IP address and Port Number of the helper. The helpers are assigned the task based on the round robin algorithm. Round robin Scheduling is used to allocated the fixed amount of time to the each process. Round robin ensures that all of the helpers perform the task same number of times ensuring equal balance of the load. The master is accountable to assign indexing task to the helper. It is also responsible for servicing requests from Client to search the term or word, get the document or index the document.

Helper:

Helper is the second component in hierarchy which gets initialized after Master. Helper is accountable for servicing indexing requests from Master for the documents provided by the client. The software component that runs on Helper bootstraps the system. Helper registers itself to Master so that they are to be known and accessible by master to assign the task.

Client:

Client is the active component of the system which runs on several machines. These are the systems actually exposed to the end users of the application. Clients are aware of the logical naming of master and request to Master for resolve to index the document or retrieve search query result based on term entered or document name. Clients receive the response from the Master.

.

4.2 COMPONENT LIFE CYCLE

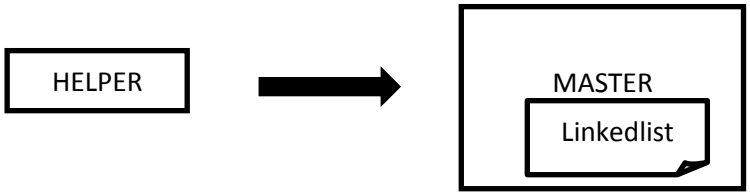
4.2.1 Bootstrapping Master

Step 1: The software component for Master is run to initialize Master.

Step 2: As we are not hard-coding IP address and Port Number of any component to improve portability, Master makes a system call to get its IP and port number. It then writes the same into the file 'master.txt', so the file can be accessed by the Helpers and Clients.

Step 3: Wait for the client or the server to connect. Master is passive after the initialization.

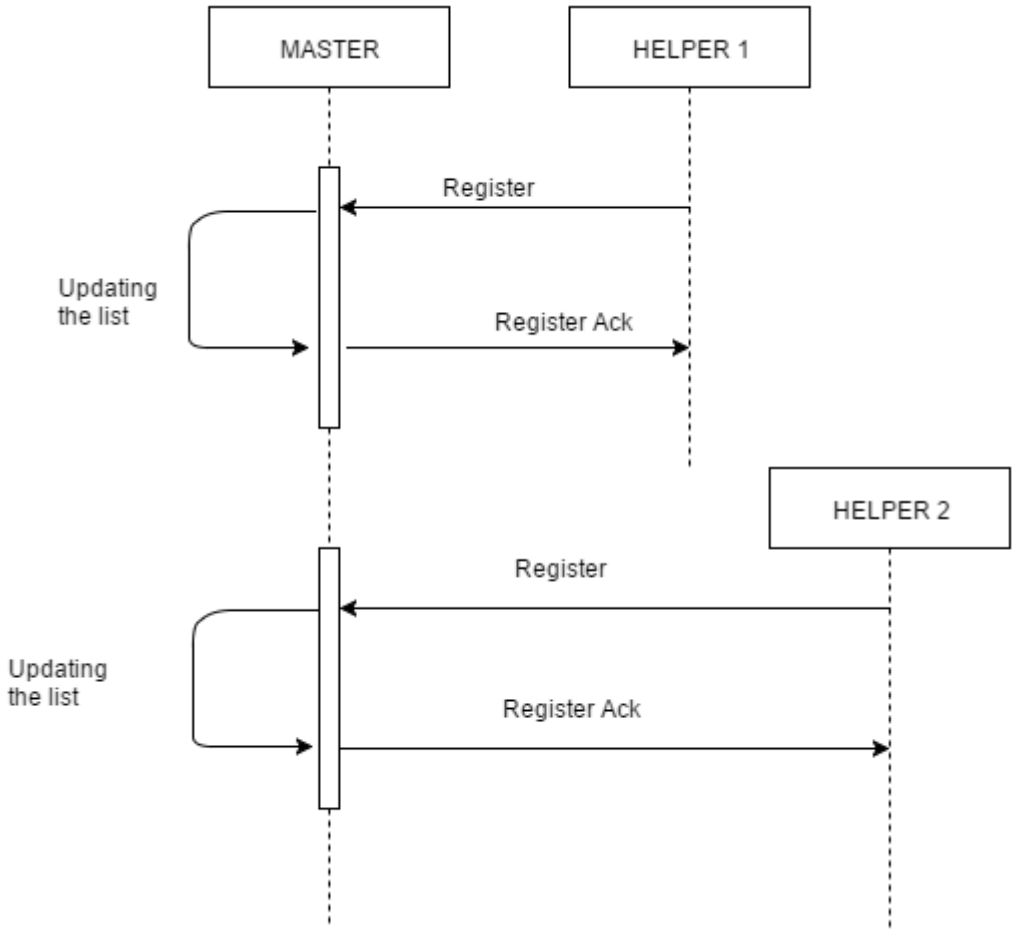
4.2.2 Bootstrapping Helper and Registering at Master



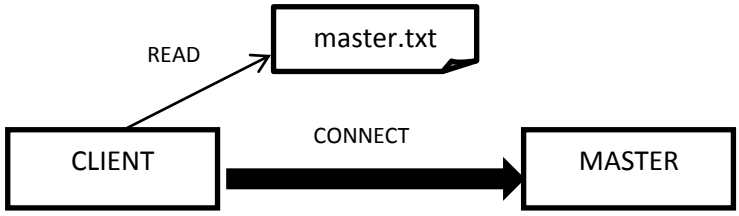
Step 1: The software component for Helper is run to initialize Master.

Step 2: As we are not hard-coding IP address and Port Number of any component to improve portability, Helper makes a system call to get its IP and port number. It then sends the details to master and Master stores the same into the linked list.

Step 3: Wait for the master to send any request.



4.2.3 Interaction between Client and Master



Step 1: The Client connects to the client by reading the port and IP address from the master.txt file.

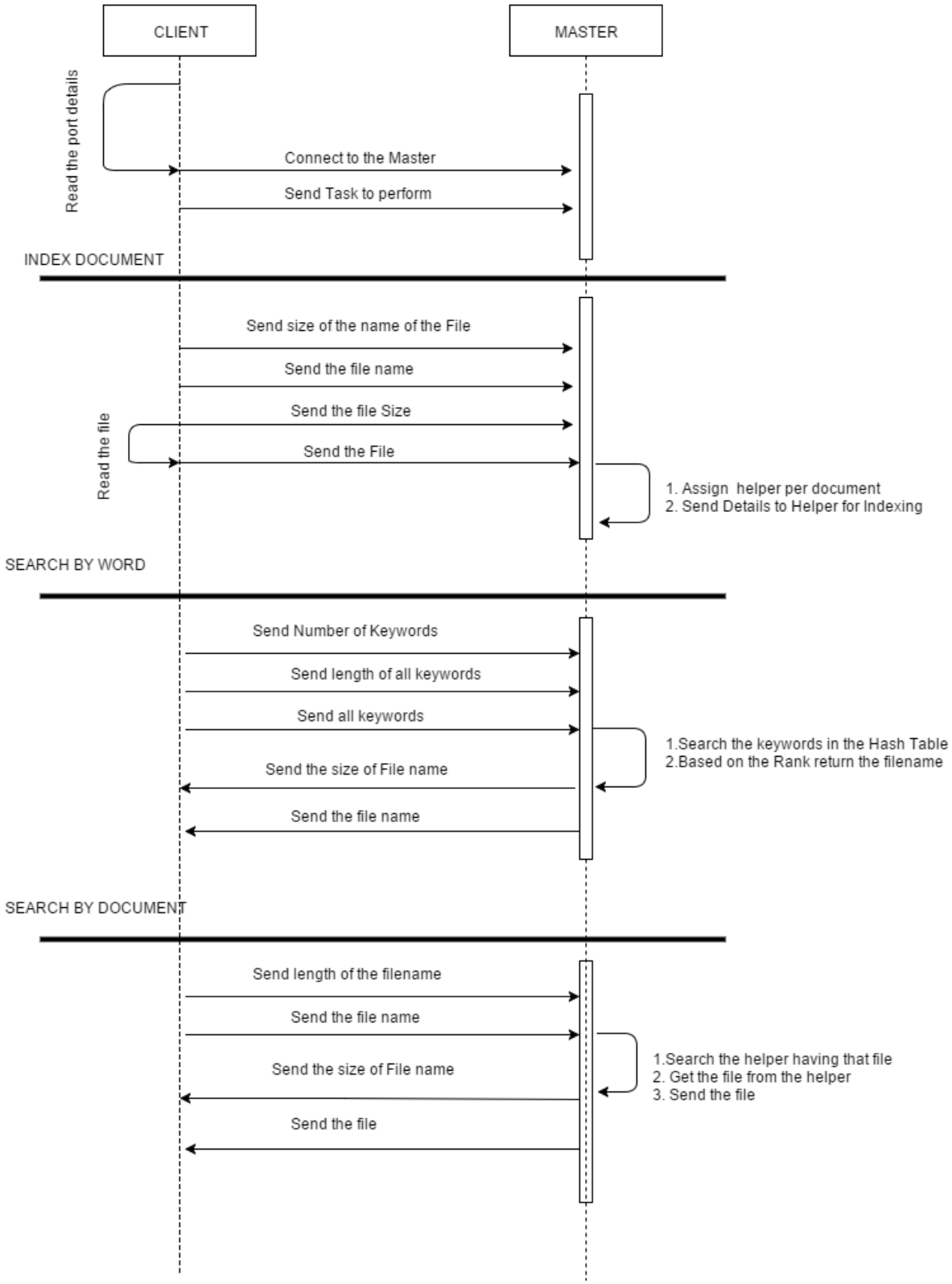
Step 2: The Client identifies the task to be performed and sends the request to the Master.

Step 3: The client choose one task out of these:

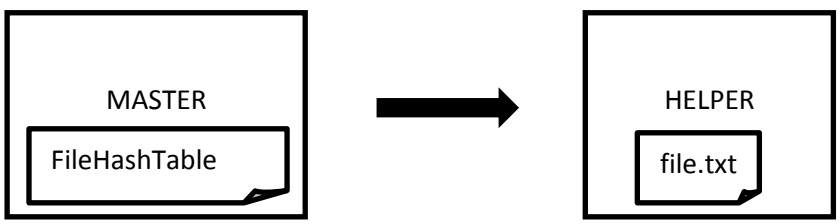
1. Indexing the document:
   Client sends the document to the master in the form of txt file. First it sends size of the filename and size of the file to the master. Then sends the file size to the master. After sending the file a size and filename detail, the file is opened and every term is sent to the master. The master accepts all these details and stores it in another temporary file. This temporary file is then sent to the helper.

2. Search the Query based on the term:
   Client sends the term or no. of terms need to be searched. The details it sends to the master are the number of words, length of the words and the word itself. The master searches the Hash Table to find any match. If some match is found then the filenames that have this keyword is listed out. In order to do that Master first sends the size of the filename and then file name.

3. Search the query based on the document name:
   Client requests the Master to send a particular Document by providing the document name. The Master searches for the Hash Table to find the document. The size of the file name and the file is sent over to the client. The client stores a copy of the file at its end.



4.2.3 Interaction between Master and Helper



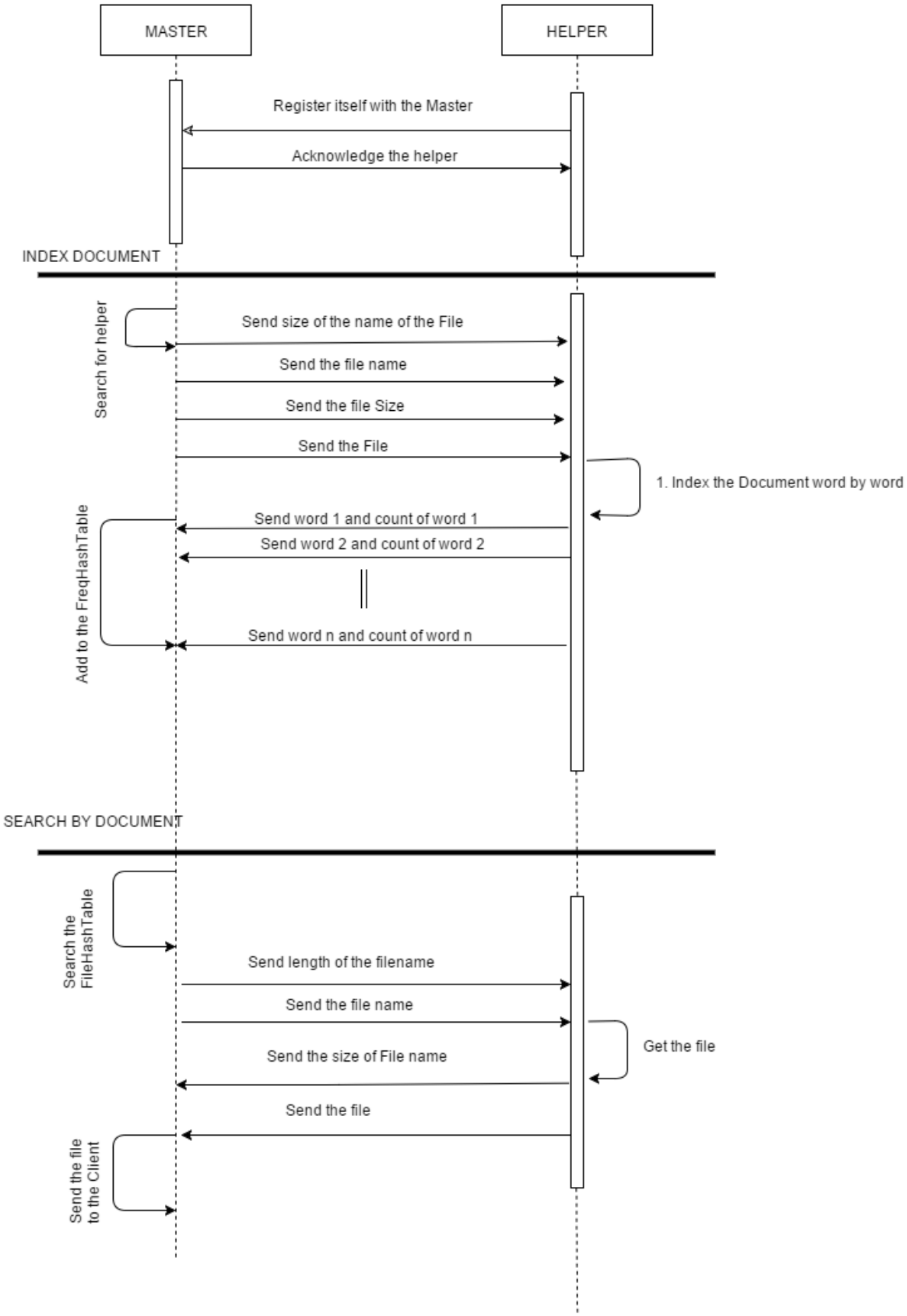
Step 1: The Helper's connect to the client by reading the port and IP address from the master.txt file.

Step 2: Once the Master receives the request from the Client, it contacts the helper accordingly

Step 3: Based on the task the helper is contacted as follows:

1. Indexing the document:
Master first finds the helper that can do the task based on the round robin algorithm. The Master then connects to that helper and sends the document that was sent by the client. First it sends the size of the filename and size of the file to the helper. Then it sends the file size to the master. The file is sent to the Helper. The helper makes a copy of the file and then Index it. The helper runs through the document word by word. It finds the count of every word in the document. It sends the word and count details to the master word by word. After finding the count of every word the master gets the details of the word and it frequency. The master stores the details in the hash Table.

2. Search the query based on the document name:
The Master gets file name length and the file name from the server. The master searches for the table in the File hash table that has the details of the file name and the helper details. The master then sends the request to helper containing the document. It asks the helper to send the document. The helper sends the document size and document to the master. The master then sends it to the client.



5. IMPLEMENTATION

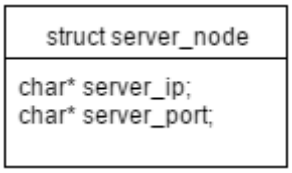
The explains how the tasks that are provided to the client are implemented

5.1 Connection

The client and helpers are aware of the connection details of the Master. Once the master is booted up, the helper's register itself with the master. All the IP Address, Port Number details are stored in the linked list. So when the request is sent from the client, the master connects to the helper that is available at that moment based on round robin algorithm and sends the request to the helper.

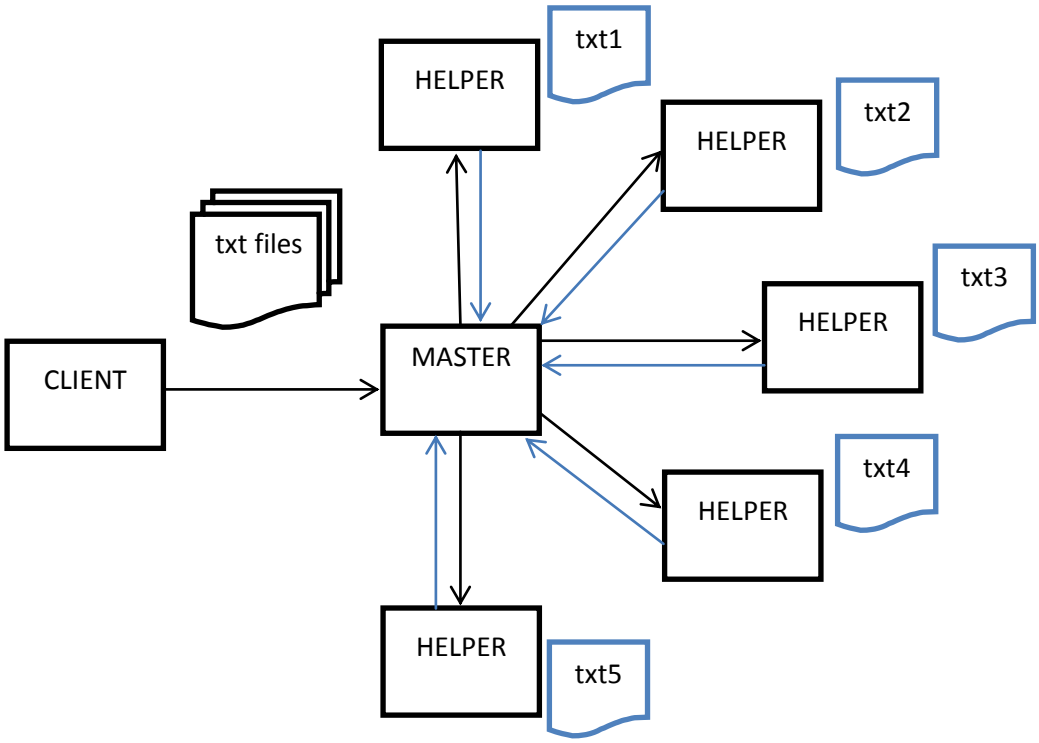
Data Structures and Algorithm:

LinkedList: Linked list is used to store the helper IP Address and Port Number.



5.2 Indexing the Document

The client sends the request to the master to index the document or set of documents. The client sends the document to the master. The master gets the documents that are to be indexed. The master then assigns one document to one helper based on the round robin algorithm. The helper then index the document by finding the word count for every word present in the document. The word and consequent word count details are sent back to the master one word at a time. The master stores these details in a hash table (i.e. FrequencyHashTable). The master also stores the sock address details and filename in another hash table (ie. FileHashTable).



Data Structures and Algorithm

Hash Table : *FrequencyHashtable*

The Hash Table stores the words present in the document as a key and stores the document name and the frequency count as the Value Node. The Master maintains a hash table in its primary memory. The hash is keyed by word (Ex: The). The value of the hash entry is a data structure containing the frequency and file name.

Hash Table : *FileHashtable*

The Hash Table stores the filename as a key and stores the IP Address and Port Number as the Value Node. The Master maintains a hash table in its primary memory. The hash is keyed by filename (Ex: Time.txt). The value of the hash entry is a data structure as shown in following figure containing IP Address, Port Number of the FS and number of clients it is currently servicing.

Round Robin Algorithm:

Round robin Scheduling is used to allocated the fixed amount of time to the each helper. Round robin tournaments ensure that all of the helpers are assigned task. The time slices are assigned to each helper in equal portions and in circular order, handling all helpers without priority.

5.3 Simple Search

The client sends to the master the words to be searched. The word is searched through the Hash Table (i.e. frequencyHashTable). The file name and frequency details are fetched. Based on the number of terms found and the total frequency of those terms, each file is ranked. The filename with highest number of terms first, followed by the frequency count, is sent back to the Client.

Data Structures and Algorithm

Hash Table:

Same as used in the indexing.

Insertion Sort:

It uses a Linked List with nodes that contain the filename, the unique words found in the specified file, and the number of total instances of all terms found in the file.

5.4 Retrieve Relevant Document

The client sends to the master the document name. The document is searched through the Hash Table (i.e. fileHashTable). The master requests the helper holding the document. The helper sends the document to the master and the master sends it back to the client.

Data Structures and Algorithm

Hash Table:

Same as used in the indexing.

6 CONCLUSION

This project gave us an exposure to new programming models of computing and processing large scale data. It gave us an understanding of how the data and the work load is assigned across a cluster of computing machines. It even provided us knowledge base on the algorithms that could be used as part of data-intensive computing.

7 BIBLOGRAPHY

REFERENCES:

1. *mini Google Project Description.* URL: http://people.cs.pitt.edu/~jacklange/teaching/cs2510-f15/project2-2015.pdf
2. *Simple Hash Table.* URL : https://gist.github.com/tonious/1377667