Fall '15

CS 2510: Computer Operating Systems, Project 2: MiniGoogle

DEBARUN DAS
PITT ID: DED59@PITT.EDU

UNIVERSITY OF PITTSBURGH |

Index

Introduction

This project is aimed at designing a simple search engine, referred to as tiny-Google, to retrieve documents relevant to simple search queries submitted by users. The primary objective is toimplement a client-server based model, using sockets in an Internet domain, to support the operations of tiny-Google. In this model, the client process provides a simple interface to its users to submit indexing requests and search queries. The details of the design and implementation are discussed in the sections below.

Components Of The System:

- **User/Client:** The client is responsible for sending three types of requests to the master node. First, it can upload files to the master server and request for the files to be indexed. Secondly, the client can give a query consisting of one or several words to see the relevant documents where the query occurs. The result obtained will be in ranked order with the documents having the words in the query most frequently appearing first. Of all the document names that the client receives as result on giving a query, the client can choose one of the document and request for retrieving that document from the master server.

- **Master Node:**The master accepts all the requests from the client. If the client requests for indexing a document, then the master chooses a worker node (from the list of worker nodes that it maintains) in a round robin fashion and sends the document to that worker node for indexing the document. Also, the master maintains an Inverted Index that will store the indexing results in a ranked fashion. If the client makes a query to the master, then the master node will search the Inverted Index that it maintains and returns the name of documents in which the words of the query occur, in a ranked order. If the client requests the master to retrieve a document, then then master returns the IP address and Port of the worker node which has the document requested for. For this, the master maintains a list of all the worker nodes which register themselves in the master node.

- **Worker Nodes:** Worker nodes initially register themselves in the master node by sending their port number and IP address to the master node. After registering to the master node, the worker nodes keep accepting requests from the master node to index the document that it receives from the master. Also, after receiving the IP address and port number of the worker that contains the document (that it requests for retrieval from the master, client connects to that specific worker node. On receiving the retrieval request from the client, the worker node sends the appropriate document to the client.

Design And Implementation

The project is implemented using C++. There are three main components – the master node, the worker node and the client. Also, I discuss about how I assign document ID to a document received from the client.

Client: In the tinyGoogle interface, the client has three options to choose from – Uploading a file, Sending a Query and Retrieving a document. If the client chooses to upload a file, the client is given the opportunity to upload as many number of files (which he indicates) as he wants to upload at a time. If a filename entered by the client is not found, then the client sees a message indicating that no such file exists. However, if the file exists, then the client sends a message to the master indicating that it wants to send a file for indexing. The master then check if any worker node is available for indexing the file. If a worker node is available, then the master asks the client to send the file. A client can also choose to give a query and see the search results based on the query. If at least a single word of the query matches with any of the indexed words in the Inverted index maintained by the master, then the client receives the appropriate result in a ranked order (with the Document ID where the query or portions of it matches most frequently being the top result followed by other Documents Ids with lesser frequency of matched words). From the result the client receives on giving a query, the client can choose one of the documents for retrieval from the master node.

Master Node: The master node maintains the following main data structures:

- **List Of Registered Workers (workerList):** The master node maintains a list of the workers who have registered themselves with the master node. This is implemented by using a map<key, value> where the key is the worker id and the value is a structure containing IP address and Port No. of the worker node.

- **List Documents Indexed By The Worker Nodes (docList):** This list indicates which document belongs to which worker node. This is also implemented by using a map<key, value> where the key is the document id and the value is structure containing the IP address and the Port No. of the worker node.

- **Inverted Index:** The inverted index stores the result after a worker node sends an indexed result to the master. This is again implemented by a map<key, value> where the key is the word and the value is another sub map <key, value> with key being the Document id where the word occurs and the value being the frequency of that word in the document.

The master node communicates with the worker nodes as well as with the client. When the worker node sends a message to the master node indicating that it wants to register itself with the master node, the master node adds the port number and address of the worker node in the *workerList*map. Now, when the client sends a request to upload a file for indexing, the master first checks if a worker node is available is the *workerList* map. If so, then the master chooses one of the workers from the *workerList* and sends the file to that worker for the purpose of indexing that file. Now, when it sends a document to worker, the master also makes the appropriate entry in the *docList*, such that it knows which documents were sent to which worker. After the master receives the indexed result from a worker node, it updates the Inverted Index accordingly. On receiving a query, the master node searches the inverted index for the matched words and sorts the documents for each of the matched words in a separate container (a vector), according to their count/ frequency values. The master then returns the ranked result to the client in the form of a character array. On receiving a request for document retrieval from the client, the master node checks the map *docList* to see if the document id requested by the client has been assigned to any of the worker nodes. If yes, then the master node sends the IP address and Port number of the worker node which has the document. If no, then the master sends a message to the client indicating that no such document was found.

Worker Nodes: The worker nodes are initially responsible for registering themselves with the master node. By registration, it means that the worker node will first send a request for registration to the master and if master agrees, then it will send its IP address and port number to the master node. After registration, the worker node will wait to accept request for indexing by the master or for request of document retrieval by the client. On receiving the request for indexing by the master, the worker sends a message to the master indicating to send the file. On receiving the file, it performs indexing on it as described below.

- Indexing Document: A map<key, value> is maintained for the purpose of indexing a document. The key is a new word encountered in the document and the value is the count of that word. The contents of this map are then stored in a character array and sent to the master.

Also, the worker node receives a request for document retrieval from the client. The client contacts the worker node after receiving its address and port no. from the master node. The worker node will send the appropriate document to the client.

Assigning Document ID: A document received from the client by the master node, is assigned a unique document id before distributing it to the client. In the document id is taken to be a simple unsigned long variable whose value increments on receiving a document.

Block Diagram Of The Design Implemented

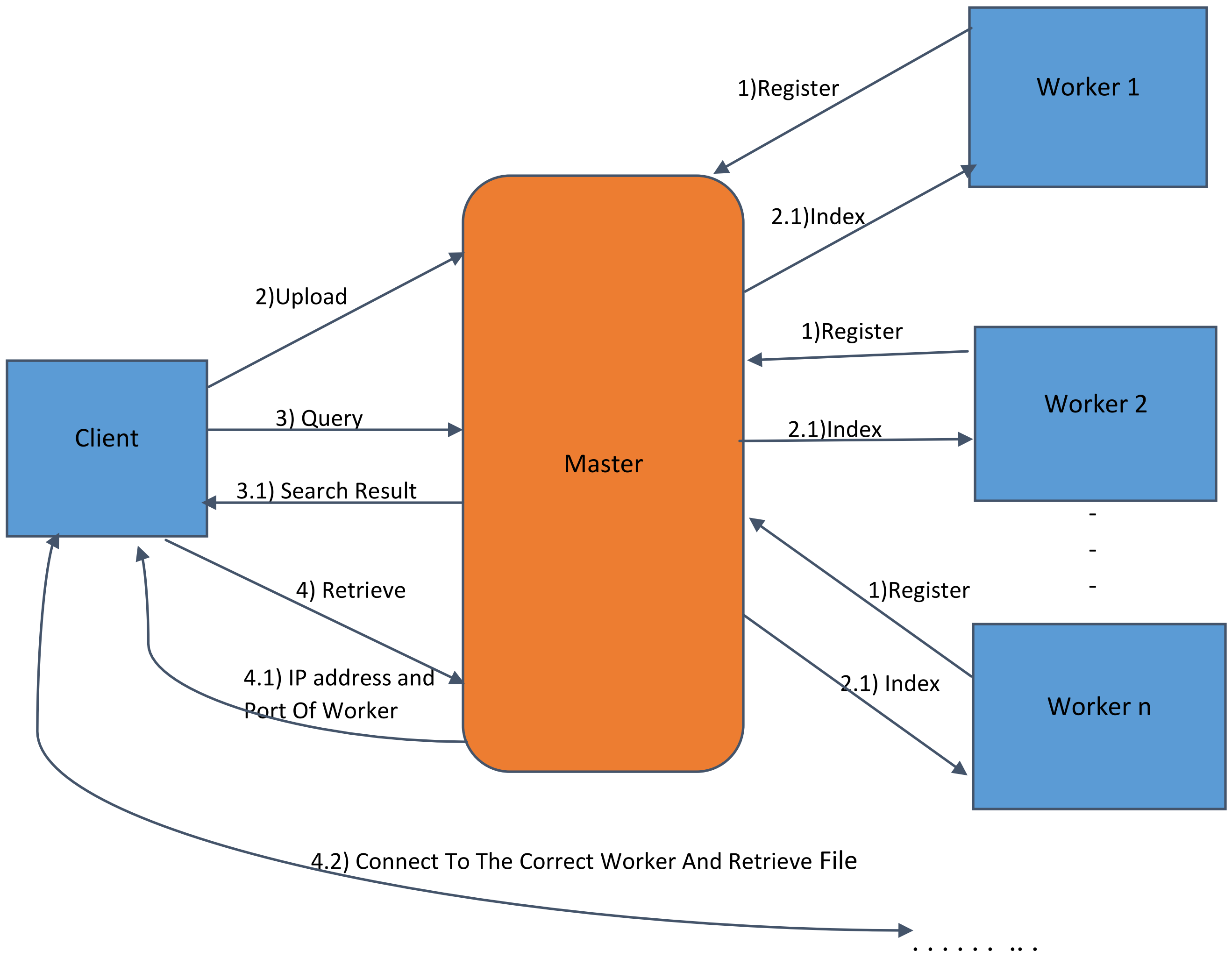

Figure 1: Basic Block Diagram Of The Design Implemented. In addition to this, in my design, there can be multiple clients that can connect to the master node. The details of the steps shown in this structure is explained previously in Design And Implementation Section.

Conclusion

This project helps me to gain a basic idea of indexing large documents, distributing work among several worker nodes and in overall, gives an idea of how a basic search engine would work.