|  |
| --- |
| CS2510-Computer Operating Systems |
| mini Google |
| PROJECT 2 – FINAL REPORT |
|  |
| **ANKITA MOHAPATRA and DONALD KLINE Jr.** |
| **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.

**TINYGOOGLE in SOCKETS**

1. **TASKS OF tiny-Google**

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

## 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.

1. **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.

* 1. **SYSTEM COMPONENTS:**

**Helper(Mapper/Reducer):**

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 search for a particular keyword in the list of documents provided by the client. Helpers can act as both mappers and reducers.

**Master/NameNode/Scheduler:**

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 NameNode 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.

**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 etc.

**NameServer or TinyGoogle server:**

As in the case of RPC, the Nameserver is responsible for initialization of the namenode.

* 1. **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 layer 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. nameserver or the tinyGoogle server, namenode or the 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.

1. **SYSTEM ARCHITECTURE**

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

* 1. **DETAILS OF SYSTEM COMPONENTS:**

**Name Server (TinyGoogle Server):**

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 name server first. The NameServer initializes the Master (NameNode). In addition, clients can ask the nameserver, the IP Address and port numbers for the Master that can provide the client with the requested operation.

**Master (NameNode):**

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 helper table in its primary memory. This table stores the IP address and Port Number of the helper. 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 (Mapper and Reducer):**

Helpers get 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.

.

* 1. **COMPONENT LIFE CYCLE**

**4.2.1 Bootstrapping NameServer**

Server\_ip.txt

Nameserver

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

Step 2: As we are not hard-coding IP address and Port Number of any component to improve portability, NameServer makes a system call to get its IP and port number. It then sends the details to namenode and the client.

Step 3: Initialize Namenode.

**­**

**4.2.2 Bootstrapping Master/NameNode**

Namenode\_ip.txt

NameNode

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

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

Step 3: Wait for the client to connect. NameNode is passive after the initialization.

**4.2.3 Bootstrapping Helper and Registering at NameNode**

NameNode

HELPER

Helper\_Table

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 its helper table.

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



**4.2.4 Interaction between Client and Master**

Namenode\_ip.txt

READ

Namenode

CONNECT

CLIENT

Step 1: The Client connects to the namenode by reading the port and IP address from the namenode\_ip.txt file.

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

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

1. *Indexing the document:*

Instead of sending the entire document to the master for indexing, the client sends the location of the directory where the documents are stored. In addition the client mentions the size of the file segment that needs to be sent to each map and reduce task (splits). The master accepts all these details and sends it to the helper.

1. *Search the Query based on the term:*

Client sends the keywords that form the search query to the master. The master then assigns each word of the search string to each of the helpers. The helpers then go on to search for the particular keywords in the master inverted index. The helpers then send their partial results to the master and the master aggregates those results and sends the name of the documents that contain all of the keywords in the search phrase to the client.

**4.2.5 Interaction between Master and Helper**

MASTER

HELPER

Inverted Index

Step 1: The Helper’s connect to the master 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 helpers that are free to do the task The Master then connects to those helpers and divides the work among few of the free workers while keeping some of the helpers on the bay so that they would be available to address any further requests from the client while the indexing request is being fulfilled. The helper runs through the document word by word. It finds the count of every word in the document. It then writes the word and count details to the inverted index word by word.

1. *Search the query based on the document name:*

Client sends the keywords that form the search query to the master. The master then assigns each word of the search string to each of the helpers. The helpers then go on to search for the particular keywords in the master inverted index. The helpers then send their partial results to the master and the master aggregates those results and sends the name of the documents that contain all of the keywords in the search phrase to the client.

1. **IMPLEMENTATION**

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

## Indexing the Document

The client sends the request to the master to index the document or set of documents. The client sends the location of the document to the master. The master gets the documents that are to be indexed from that location. The master then assigns the documents to the helpers based on their availability. The helpers then index the document by finding the word count for every word present in the document. The word and consequent word count details are written into the master inverted index.

txt1

HELPER

txt2

HELPER

txt3

HELPER

CLIENT

MASTER

txt4

HELPER

HELPER

txt5

## Simple Search

Client sends the keywords that form the search query to the master. The master then assigns each word of the search string to each of the helpers. The helpers then go on to search for the particular keywords in the master inverted index. The helpers then send their partial results to the master and the master aggregates those results and sends the name of the documents that contain all of the keywords in the search phrase to the client. Based on the word frequency in each file, all the files are sorted in descending order (ranked). The filenames with highest frequency sort order are sent back to the Client.

1. **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.