Distributed Operating Systems Project II Report

Bhavin Modi, Jose Michael Joseph

# Components:

Our system consists of the following components:

Client: This component contains the UI that will be displayed to the client

Client-Server Directory: This lies in between the client and the server

Server: This is the component that is also known as “tinyGoogle”

Worker: This component is responsible for processing the documents sent to it by the server

# Architecture:

C:\Users\JoseMichael\Desktop\OS Stuff\Project 2 Stuff\osproj2arch.png

*tinyGoogle architecture*

* The client connects to the server via the Client Server Directory
* The Client Server Directory connects to the Server and the Client. It provides the client the IP and port of the Server to which to connect to
* The Server is connected to the Client and the Workers via the Client Server Directory and Worker Directory respectively
* The Workers are registered to the Worker Directory and connected to the Sever as required

# Functions:

## Client:

* Responsible for taking input and displaying information from the user

Client has 3 options:

1. Index

2. Search

3. Exit

Indexing requires that the User enter the path he wants to index:

This path can be the path pointing to a file or an entire directory. If it is the former, then that file may be any regular file. If it is the latter, then only ‘.txt’ files in that directory will be indexed.

Searching requires the user to enter the Keywords to search, there is a restriction of 100 characters on the length on the keyword or keywords taken as input from the user. The client has then the option to enter the name of the file he wants to retrieve and that file is retrieved from the server and stored in the client directory.

The User also has an option to exit, which effectively will exit the code cleanly. He can start the terminal up again at his convenience.

Finally, as an added feature for fault tolerance if the server is taken offline while the User is connected, the User will be notified as and when he interacts with the terminal and will be given an option to continue trying to connect to the server. If the User does decide to continue, we do this checking at every 1-minute interval. The User is connected to the server as and when it becomes available and continues normal working.

## Client Server Directory:

* Responsible for keeping track of the ‘*tinyGoogle*’ server
* Provides server’s IP address to the client and worker
* Maintains a queue that contains all the servers
* If a server disconnects then the server is removed from the queue
* Keeps pinging the server at regular intervals to check if server is alive
* Has the capability to register and deregister the server

## Worker Directory:

* Responsible for keeping track of the worker
* Provides worker’s IP address and port to the server
* Maintains a queue that contains all the workers
* If a worker disconnects then the worker is removed from the queue
* Keeps pinging all workers at regular intervals to check if they are alive

Both the directories work in a similar fashion. The worker directory has the added functionality to notify the server when a worker goes offline. The server then initiates rebuilding of the index.

The directories are also informed when a connection request fails to some node that it returned, so they can remove it and always maintain a consistent state.

## Worker:

* Responsible for processing the documents given to it by the server
* Produces a local index that hashes all the words present in a particular document
* Also stores these files locally
* Maintains a hash table that is a subset of the hash table present in the server
* In case of a server crash, the workers are responsible for providing the new servers with fragments of their hash table

Worker also has the option to deregister itself by giving it the input 1. Even more functions can be added to the console, as and when required.

## Server:

* Maintains the global hash table that contains all the words indexed from every document sent to the server
* Responsible for taking the files to be indexed from the client and delivering it to the workers
* Responsible for coordinating the index merge task from the workers
* Maintains a lock that determines if incoming requests have access to particular resources
* Searches the index maintained in it for keywords provided by the user and returns this information to the client
* Coordinates the task of rebuilding the global index in case of server failures and if any worker is taken offline

Server also has the option to deregister itself by giving it the input 1. Even more functions can be added to the console, as and when required.

Server never remains connected to any workers, they are accessed through the worker directory as and when required. The server with the help of the worker directory distributes the load among workers if batch indexing is done. In case of single file indexing the server sends the file to the directory that has the least number of files stored in it.

It does not maintain or store any files, only the index, so if any worker goes offline, those files are lost with it. We make sure to rebuild the index in such cases, so indicate the loss of information.

# Design & Implementation Constraints:

1. The client can retrieve any file after the searching request, he is not restricted to the files shown as the result of the search.

2. If the server or worker cannot register themselves with directory, then they cannot be started. So without the directory running, the entire system will not function correctly.

3. All Indexing and searching operation are synchronized, and take places sequentially as we have used mutex locks on these operations.

4. Load distributing does not take into account the file size and how much work may be done in indexing it, it distributes the load by dividing the incoming files between the workers in the order in which they are received from the client.

5. Files and the index are sent across the network line by line.

# Indexing:

Indexing is maintained by having three hashes spread over the system. We will discuss the various hashes in detail.

C:\Users\JoseMichael\Desktop\OS Stuff\Project 2 Stuff\oshashes.png

## Local Worker hash:

This is the hash present in the worker and contains the words indexed of the last file sent to the worker. The contents of this file are sent to the server upon completion. This way we ensure that the index being pushed to the server only contains the data associated with one file.

## Mini Global hash:

In case the server fails, when a new server opens up the worker would send its Mini Global Hash to the server. In this way the server can aggregate all the fragments of the global index and recreate the lost index. In this way the worker provides fault tolerance for the server.

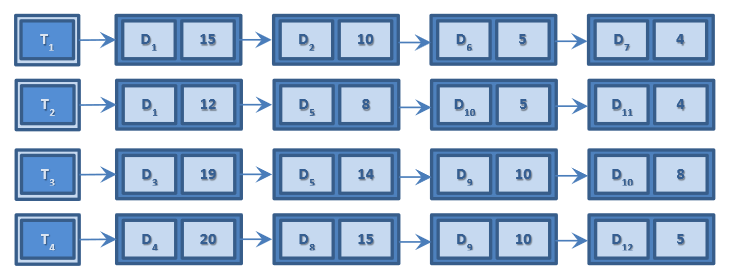
## Global hash:

This is a hash file that is stored in the server. It consists of all the words together with the document they are present in and the number of times they are present in that document.

Upon receiving a search query, the server performs a search on the index present on the server and returns the value associated with those keywords.

## Structure of the hash table:

The hash table is an inverted index.



There is a struct that is hashed which contains two elements a) the word being hashed (which is also the key for the hash table) and b) the root node of the linked list.

This node contains the document name where the word is present as well as the number of occurrences of the word in that document. The node also links to the next node in the linked list.

Thus in this format we have built a linked list where each node contains information associated with the struct with which it is hashed.

## Method of token generation:

We have taken care to ensure that the file is read only one word at a time and we immediately tokenize this word and insert it into our hash table. This is to ensure that we don’t directly pull the whole file into main memory which could cause obvious problems associated with memory limitations.

## Method of index transportation:

The index is “transported” to various components by packing it as a character pointer. Each “packet” would thus contain the word, the document name that contains the word and the number of times the word is contained in that document. This information is then transported to the necessary component where it is consumed and merged into the existing hash table.

## Method of file operations:

All files in our project are accessed one KB at a time. This is to ensure that we don’t consume all our memory by opening a huge file wholly into main memory. This ensures that our code is scalable and can work on systems with even the highest of memory limitations.