Concurrent and Distributed Systems – Course Project

Document Version: 1.0[1]

Deadline (no extension possible!): 29th April, 2018 – 11:55 PM.

Overview:

The project focuses on searching for a password in the list of already known (hacked) passwords. The project idea stems from the enormous list of hacked passwords, more than 500 million passwords – 9GB compressed and more than 30 GB uncompressed file available on this link.

However given the resource constraints, you are not required to work on the list above but rather a subset as provided by CrackStation's Password Cracking Dictionary. There are around 64 million passwords in the list (around 270 MB compressed) to be downloaded from https://crackstation.net/files/crackstation-human-only.txt.gz.

For the project you need to of course download the file and uncompress it. You then need to tokenize the file in smaller files (size of each chunk is your decision but you can base it on performance and the number of nodes in your distributed system). This code https://socketloop.com/tutorials/golang-how-to-split-or-chunking-a-file-to-smaller-pieces can help you do the split. You need to then place the smaller files (chunked files) on multiple nodes of your distributed system. A file may be present on more than one node and you need to set a replication factor of more than 1 i.e. a file is present on 2 or more nodes.

Application Architecture and Components:

The overall application architecture consists of three core components (and corresponding programs): Client, Server and Slaves.

- The client is responsible for making the request, to the Server, to search the password.
- The server program on receiving the request, welcomes the client, analyzes and divides the task into parts and allocates these parts to Slaves, which have already registered with the server.
- The slaves are the workhorses and they register with server and are assigned the tasks. Once they register, they notify the server about the file chunks they have and the server uses this info for scheduling.

[1] Error and omissions expected. Please report any inconsistencies and/or missing points by email and I will update them in the next version.

The Client Program

The client program is responsible for making the request to the Server, to search for the password. It does not take user input but rather uses the command line arguments for the required information from the user. The command line parameters include:

- ***-searchText****=somePassword* – This argument is the compulsory for the user to provide and represent the password that needs to be searched.
- ***-server****=serverIPandPort* – This argument represents the IP address and port for the server, to whom the request is being sent.

Example usage:

- *go run client.go -searchText=hello -server=127.0.0.1:2600*

Once the request has been sent to the server, the client program displays the information to the user and waits for the response from the server. Once the response arrives, it displays it to the user and exits.

The Server Program

The server is the core component and listens on two different ports; one it exposes to Clients to receive jobs (password search requests). The other port is for Slaves to register themselves with the server. Specifically it has following core attributes:

- The ports on which it is listening are provided using command-line arguments and should default to some values if arguments are missing.
- It should be a multi-threaded server and should be able to handle more than one client and their requests simultaneously.
- It maintains a lit of Slaves to whom it can delegate computation. This list is populated by the messages it receives from the Slaves. List is of course dynamic and changes when slaves enter and leave the system.
- In addition to the list of slaves, it also maintains what data chunks (subfiles) a slave is currently storing. It uses this info for scheduling jobs amongst slaves.
- It also serves as the load-balancer and equally distributes the load amongst Slaves and manages different clients and slaves.

The server is the core component and carries significant weightage.

The Slave Program

The Slaves are the workhorses and register with the Server, are assigned the tasks and report the success/failure status and associated information to the Server. Specifically the core attributes of the Slave component are as follows:

- The IP for the server and the port is specified by command line arguments as with the Client program.
- On startup, the Slave program connects to server, on the specified IP and port, to register itself with the server. It also provides a list of sub files it is storing. The following function may help in this context:

```
package main

import (
	"fmt"
	"io/ioutil"
	"log"
)

func main() {
	files, err := ioutil.ReadDir(".")
	if err != nil {
		log.Fatal(err)
	}

	for _, file := range files {
		fmt.Println(file.Name())
	}
}
```

- Once registered, it listens on a port (whose number can only be provided using command line argument) for the tasks assigned from the server.
- Once the task is assigned, they perform the task and send the response back to the Server.
- The slave quits the search once a abort message is received by the server (which signifies another slave has found the password to be searched).

Evaluation Criteria:

The evaluation criteria would be communicated later in a separate post.