Password Cracker Report

Lanfeng Liu(UID: U36953824), Yinpei Su(UID: U46394059), Yuchen Guo(UID: U86016090), Junfei Huang(UID: U83700679)

GitHub link: https://github.com/SimpleMD5Craker/MD5Craker
Project on GENI and any public link: http://204.102.244.50/

1. Introduction / Problem Statement

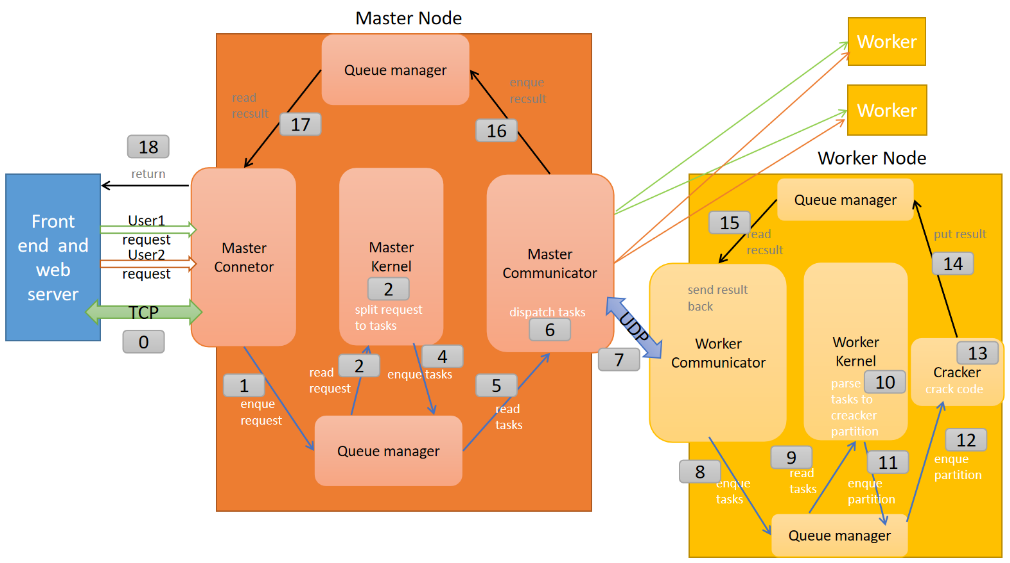
Definition: For this project, our goal is to design a distributed system to crack encrypted passwords by md5. Every user can submit the md5 hash of a 5-character password to the system through a web interface. After that, the system will analyze requests with a master communicator service. The MasterNode will dedicate jobs to different worker nodes and scales automatically when it receives new requests. The workers will crack the password by a brute force approach.
Motivation / Learning outcomes: With Password Cracker project, we are able to learn socket programming, multi-threads programming, web front-end programming, brute force method to crack password, designing distribution system, teamwork and communication.

2. Experimental Methodology

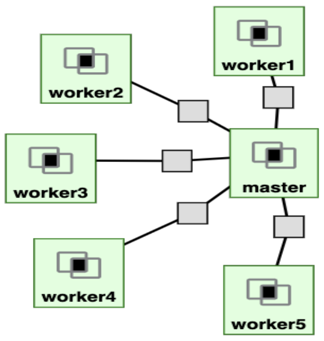
2.1 Design of Password Cracker

The cracker can be divided into 3 parts:

- ***User web interface(Frontend)***.
  1. In the front end, we use html to display the content and jQuery to dynamically show the results.
  2. When you visit this link, the request will be first direct to the Apache deployed on the master node, then the web server will retrieve our homepage and return it to the visitors' browsers.
  3. Visitors will be granted an unique identifier for each user they created. After that, you can input the hd5 hash of a 5-character password and click the submit button.
  4. Then, a POST message which contains the user id and the input will be sent to the 31000 port of the master node.This port is listened to by the flask. Once it receives a POST message, the flask will use a socket to communicate with the backend and return the answer back to the visitors' browsers.
- ***Master and worker Framework(Backend)***.
  1. Front end passes the user input to "master connector" through TCP connection.
  2. The "master connector" will put any request it receives into the "queue manager" and periodically poll "queue manager" for results of users' requests.
  3. The "master kernel" will periodically poll the "queue manager" for the new request, and split it into multiple tasks. Besides, the "master kernel" will keep track of the tasks and workers in two lists. Since works communicate with the master by UDP, we defined our custom message type over the UDP.
  4. The message can be divided into 2 types: **heartbeat** and **assignment.** The **heartbeat** is sent by workers and can carry the task result. A worker can join the cluster by sending heartbeat messages to the master. Workers will periodically send heartbeat to master. When a worker hasn't sent any heartbeat for some time, the master will remove the worker and redispatch the tasks assigned to the worker. The **assignment** message is sent by the master to dispatch tasks.
  5. The picture below shows the workflow of the framework.



- ***Cracker(Backend)***.
  We use a brute force approach to find the password. The method will search all the 5-character passwords, and compute the hash value. Then it will compare it with the input hash value: if it is the same, it will stop and return the password we found, else it will continue to search.



The structure of the slice on the *GENI*:

Linux version	**Ubuntu 18.04.1 LTS**
Java version	**Java8**
Python version	**2.7.**
Flask version	**1.1.2**

2.2 Experimental Design

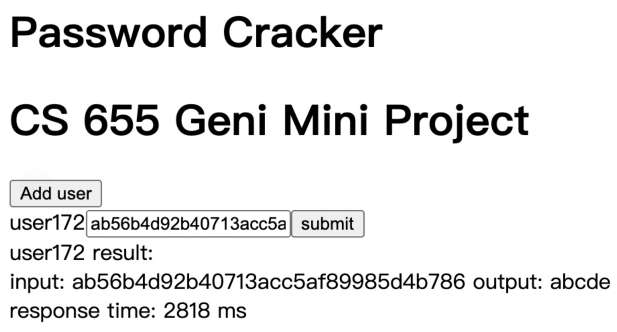
- To test the scalability of our system, we will set different number of workers(If the resource of GENI is sufficient) to see whether the more the workers, the better the performance(Since we only have one master node, too much workers may slow down the master)
- Test the robustness of our system, by manually disconnecting one or more workers, to see whether the application can still get a correct result.
- Multiple requests being processed simultaneously.

3. Results

3.1 Usage Instructions

Usage instructions and a video in the github will show how to use the cracker.

Web UI:



Password Cracker

CS 655 Geni Mini Project

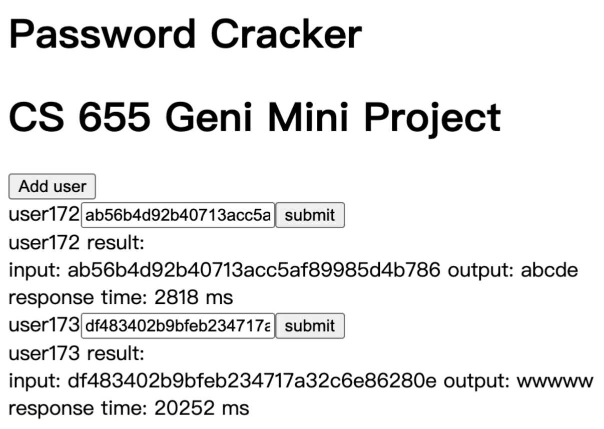
Add user

user172 ab56b4d92b40713acc5a submit

user172 result:

input: ab56b4d92b40713acc5af89985d4b786 output: abcde

response time: 2818 ms



Password Cracker

CS 655 Geni Mini Project

Add user

user172 ab56b4d92b40713acc5a submit

user172 result:

input: ab56b4d92b40713acc5af89985d4b786 output: abcde

response time: 2818 ms

user173 df483402b9bfeb234717a submit

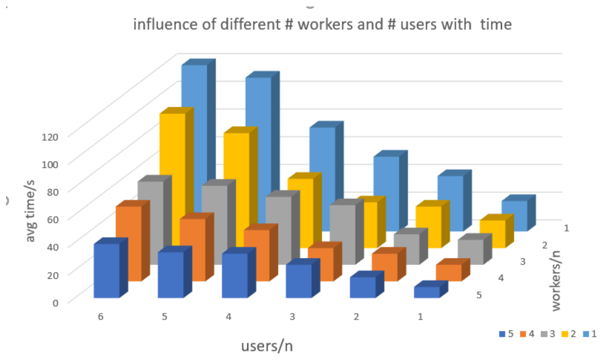
user173 result:

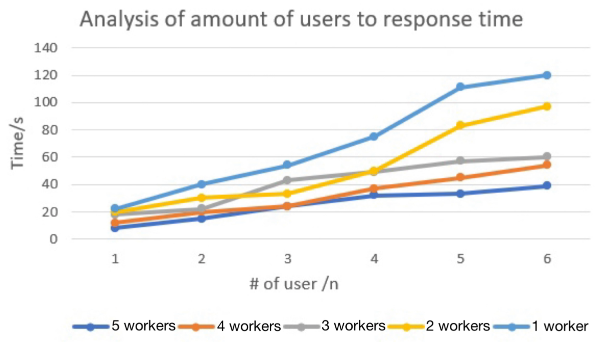
input: df483402b9bfeb234717a32c6e86280e output: wwwww

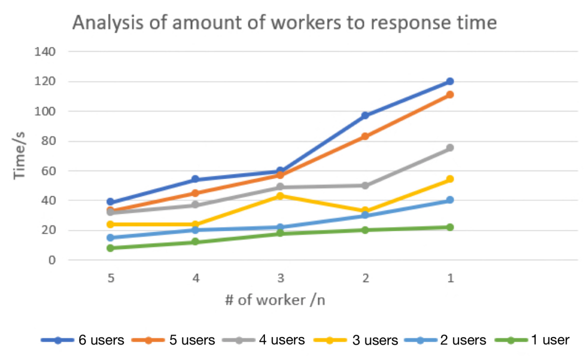
response time: 20252 ms

3.2 Analysis

Result of experiments: We performed the following experiments to verify the efficiency of our system, and to check how the number of workers can improve the performance of the cluster and how the amount of users can reduce system efficiency.







Metrics: We tested the response time with respect to different numbers of workers and users. ***Analysis:***According to the statistics, we can find how the response time is taken when we vary the number of users and workers. The response time becomes longer as the amount of users grows. It is clear that the response time also becomes longer when the number of workers decreases. Besides, the average response time remains unchanged with a fixed number of workers and users. In conclusion, the total running time of the password cracker system is determined by the number of users and workers. Since we have a limitation of total number of workers, if all worker nodes are occupied, the remaining password cracking requests should wait in the queue until one of the workers finishes its job, most of the response time is queueing time.

4. Conclusion

Summary: We designed a system with a web-interface, a master-worker communication framework over UDP, and a cracker function for cracking MD5 passwords. The system is available for multiple users to submit their cracking requests through the webpage. The request will be sent to the master node and then assigned to different workers following our designed strategies. Once the password is cracked, it would return to the server and display the result on the webpage. We also have several corresponding experiments to analyze the response time with different numbers of users and workers.

Possible extensions: One possible extension is to use a cache in the web server, thus for some requests, we may not need to crack. Another extension is to explore a good strategy to partition the cracker job according to the number of current workers, e.g. if there is only 1 worker, no partition is needed.

5. Division of Labor

Describe the contribution of each group member.

Name	Division of Labor
Lanfeng Liu	The design of the master-worker framework, implements the master(except the connector part) and the worker(except the cracker part) code.
Yuchen Guo	The design of the frontend, implements the web page, part of the flask and master connector. Deploy the Apache web server to make the website publicly viewable.
Yinpei Su	The implementation of the cracker part, a baseline for python flask, and a part of the connector.
Junfei Huang	Implement part of master connector, analyze result and write the document