Password Cracker

Changhao Liang(U16843909), Jun Xiao (U85900288), Qi Yin (U31787103), Yuang Liu (U99473611)

GitHub link: https://github.com/FeixLiu/cs655_2020fall_mini_project
Video demo: BU CS655 2020 Fall GENI Mini Project Demo (with CC)
Project on GENI and any public link:

- *Name of GENI slice:* PC-yq
- *Project:* CS-655-Fall2020
- *Rspec file:* https://raw.githubusercontent.com/FeixLiu/cs655_2020fall_mini_project/main/rspec.txt
- *Web interface:* http://pcvm3-8.instageni.cenic.net/

1. Introduction / Problem Statement

Definition:

The project is to design and implement a distributed system to crack passwords encrypted by md5. Users can submit the md5 hash of 5-character passwords to the system through a web interface. The system will parse requests with a management service. The management service distributes cracking jobs to different workers and scales (add/delete workers) automatically when it gets new requests. The workers will use a brute force approach to crack the password.

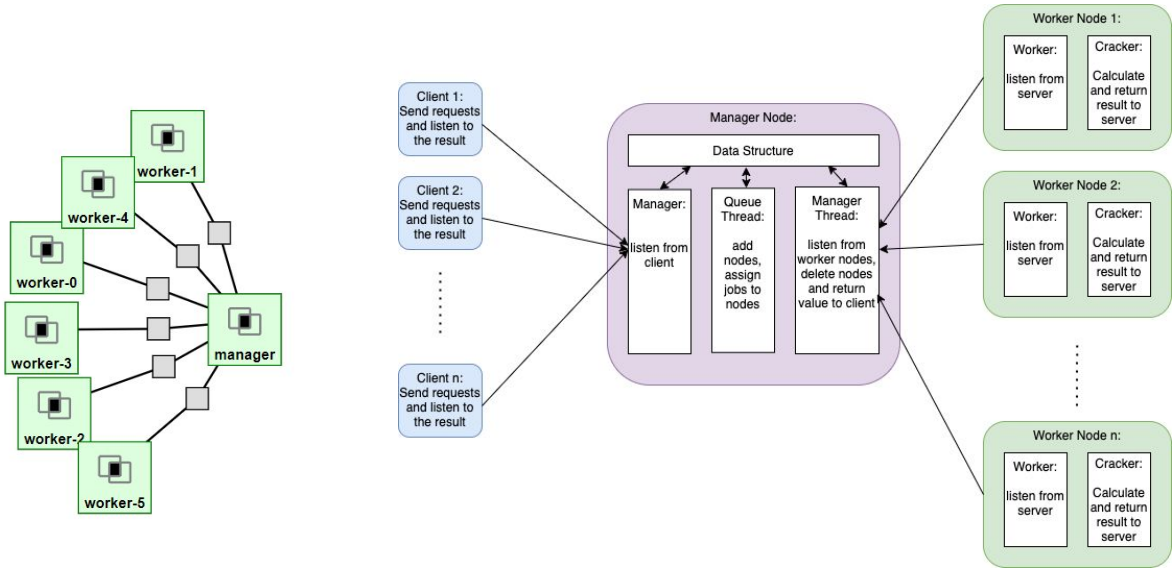
Motivation / Learning outcomes:

Through this project, we would learn socket programming, multi-threads programming, md5 hashing, web front-end, brute force method to crack password, distribution system and GENI testbed.

2. Design / Setup

Setup diagram:

The structure of the slice on the GENI and the workflow of the system are shown in the figure:



Our basic idea is to construct a manager node and multiple worker nodes, every worker is linked to the manager. A thread is created to listen to the client requests on the manager service. Once a new request arrived, it would be added to the queue. The other thread is used for communication with the workers, which would assign the jobs to different workers, manipulate the addition and deletion of workers dynamically, and get cracking results for each request. On each worker node, there are also two threads, one is used to listen to the messages from the server, the other is used to run a brute-force method to compute the md5 and crack the password. When the result is found, the worker stops its job and returns the result to the manager, then it would be sent back to the client and displayed on the web interface.

Environment / resources:

The environment of the experiment is as following:

Linux version	Linux version 4.15.0-121-generic
Java version	Jdk 11.0.9.1 / Jre 11.0.9.1
Nginx version	nginx/1.14.0 (Ubuntu)

The assumption of our system is as following:

maximum number of requests that can handle in parallel	6
maximum number of workers	≤6
user cannot resend or remove requests if the status is pending	

Based on the assumptions, we designed experiments to evaluate the performance of the system. We would test in turn:

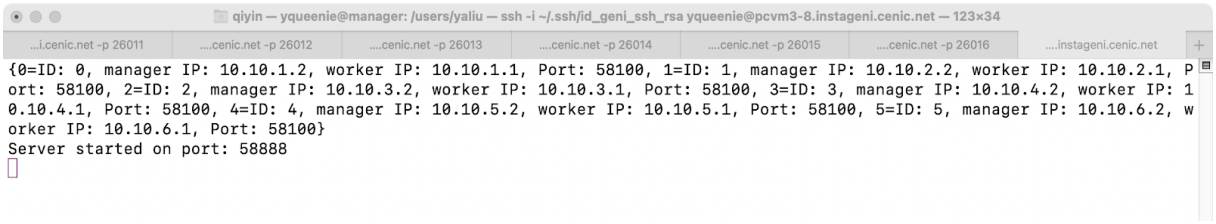
- When the number of users is fixed, the influence of different numbers of workers on the system transmission time and password calculation time;
- When the number of workers is fixed, the influence of different users on the system transmission time and password calculation time;
- The influence of password cracking difficulty on the transmission time and cracking time.

3. Execution / Results

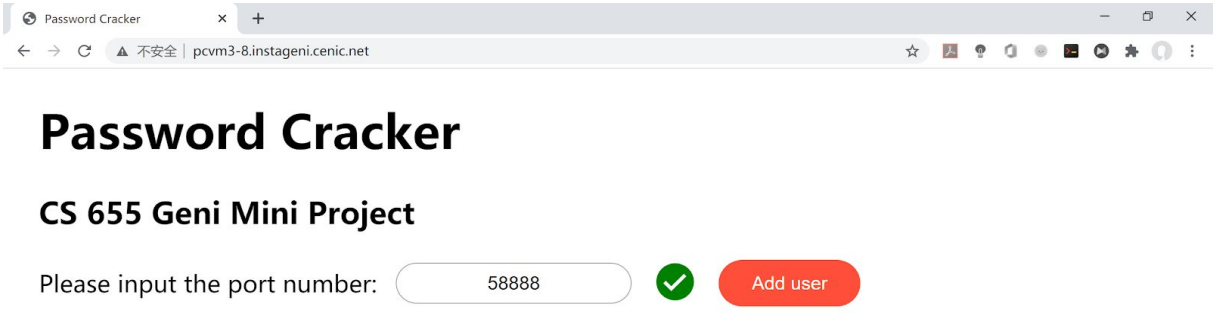
NOTE: we leave the port 58888 for the bash scripts. If you want to reproduce these experiments without running the scripts, please try port *58111* instead of 58888.

3.1. Configuration / Usage Instructions

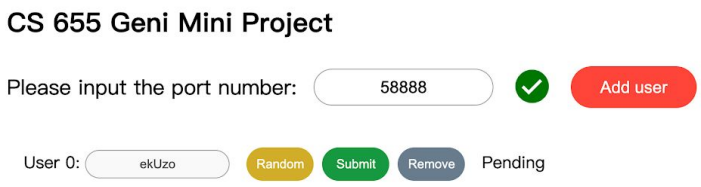
First login to the GENI nodes, then use wget to download the .sh file from github, then run the .sh file on the server node and the worker nodes follow the instructions on the github link.



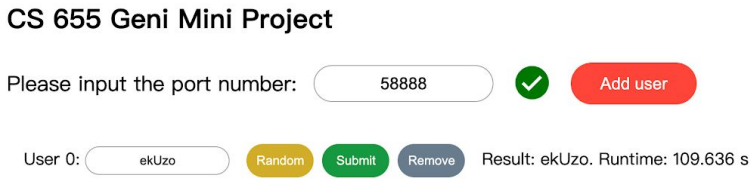
Enter the url http://pcvm3-8.instageni.cenic.net/ in the browser, you would see the page as following:



Enter the port number 58888 in the textfield, click the “add user” button, it will pop out the input field for each user. Users can input their 5-character password (a-z,A-Z) and click “submit”. After then, the system would assign the cracking job to the workers and show “pending” on the page. You may also “random” a password in order to test the efficiency.



The buttons of “sumit” and “remove” are disabled when the corresponding cracking task is pending, otherwise the users can be added or removed in order to submit multiple cracking requests at the same time.

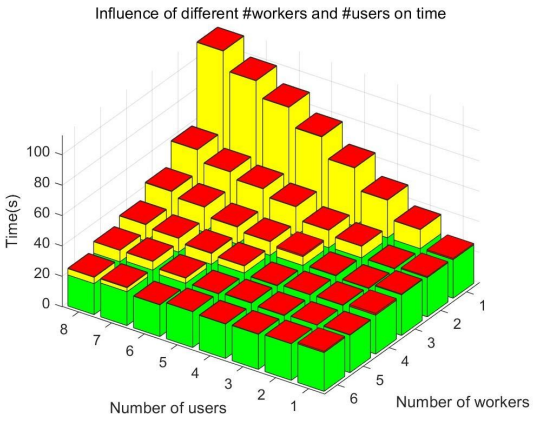


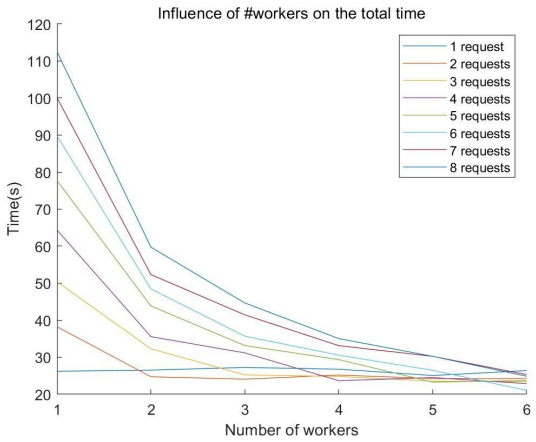
After cracking, the worker would return the results to the server and the server would display it to the website. The result and the total time (including queueing, computing, and transmission time) are shown next to the request.

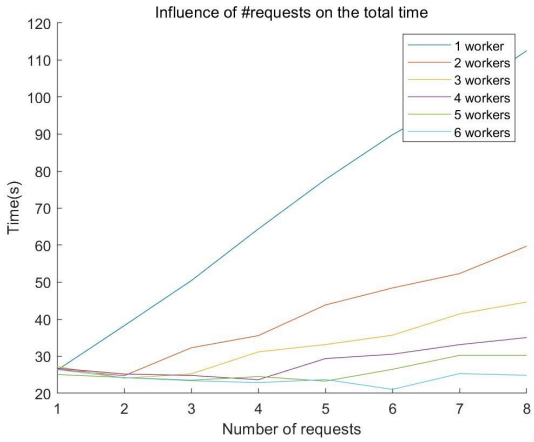
3.2. Metrics, graphs and analysis

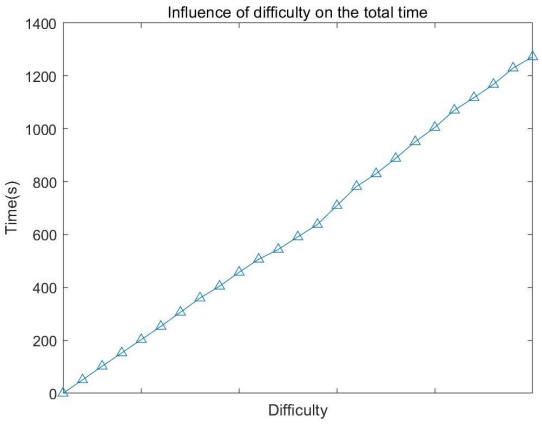
Result of experiments:

We performed the following experiments to verify the efficiency of our system, and the results are shown in the chart. In the upper-left graph, the green part indicates the average cracking time, the yellow part indicates the average queueing time, the red part indicates the average transmission time.









Analysis:

According to the experiments, it is obvious that generally, the average cracking time and the average transmission time remain unchanged with the change of the number of workers and the number of requests, while the average queueing time becomes larger when the number of workers decreases and the number of requests increases. Based on the results, we could generally conclude that the total running time of the system is mainly determined by the queueing time. Since we have a limitation of total number of workers, if all worker nodes are occupied, the remaining cracking requests should wait in the queue until one of the workers finishes its job, and this is why the curve has an inflection point when the number of workers reaches the upper limitation. Besides, as the difficulty of password cracking increases, the total time also increases linearly.

4. Conclusion

Summary:

Generally speaking, through this mini project, we implemented a system with the web-interface as well as the management service that will dedicate jobs to worker nodes deployed on different machines. The system is available for multiple users to submit their cracking requests through the webpage. The request will be sent to the manager 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 performed several corresponding experiments to test the transmission and cracking time with the configuration of different numbers of users and workers. The result shows that the total time (including queueing time, transmission time, and cracking time) becomes larger with the increase of the number of users and the difficulty of passwords, or with the decrease of the number of worker nodes.

Possible extensions:

For future extensions, we may try to add more nodes to this system or enable worker nodes distributed in different sites on the GENI server. In this way, more resources could be used for cracking the password in order to improve the overall speed of computing. Another possible extension is that we could try different queueing and assigning strategies to let the workers work in a different way, so that the efficiency of our system may also be improved.

5. Division of Labor

Changhao Liang (U16843909)	Design the logical structure of the system, constructs the web interface of the system, deploys the web server on the corresponding nodes, configures the environment on the GENI nodes, writes the documents
Jun Xiao (U85900288)	Design the logical structure of the system, writes the cracking part of the worker nodes, performs experiments on the system and creates figures, configures the environment on the GENI nodes, writes the documents
Qi Yin (U31787103)	Design the logical structure of the system, writes the multi-thread tasks on the manager and the worker nodes, performs multiple tests on the system, configures the environment on the GENI, writes the documents
Yuang Liu (U99473611)	Design the logical structure of the system, writes the back-end and the strategy of assigning tasks to the workers, create the executable shell file, configures the environment on the GENI nodes, writes the documents

6. Reproducibility

Please refer to the github link and the readme file for detailed instructions to reproduce the experiment.