Austin Joseph
Gao Xiangshuai
Timothy Yuen
Yehonathan Litman

CSE 331
Team 4

Introduction

Team 4's members are Austin Joseph, Gao Xiangshuai, Timothy Yuen, and Yehonathan Litman. We were assigned Project 4 to create a distributed typosquatter detector. Our project was written in Python, utilizing Bootstrap and jQuery to create a visually appealing user interface, Flask to both serve this interface to the user and handle the requests to run typosquatting detection on a specific URL, and a MySQL database server to serve as our task queue and final list of all alive/dead domains.

Application Design

Our system is designed for that each of the slave nodes get their tasks from our database. The web server serves as our master node, it receives input of a URL from the user and adds that URL to the submittedUrls table in our database for the typo generating slave node to process. Contrary to the professor's initial design we have two different slaves nodes, a typo generating slave node and a URL querying slave node. Our typo generating slave nodes assign themselves a URL to process and update the database to show that this URL is currently being processed. The newly generated URLs are added to the generatedUrls table on the typo generator which then process the URL and get the entry in submittedUrls table to updated to show that its been processed already. The URL querying slave nodes then take over the process of processing URLs, each URL querying slave chooses a generated URL from the generatedUrls table that hasn't been processed. It updates the database to say that that URL has been assigned and then processes that URL. Once it finishes it will store the response code

and the image gotten from the database in the database and report update the database to say it has finished it's assignment before grabbing another.

For the master node/web server when it receives a POST request to the /view endpoint containing a URL. It returns all of the information that we currently have on this URL. If the URL is new to our system(not present in the submittedUrls table), it adds the URL to the table and returns status code 1. This status code means that the URL is newly submitted so the server should wait a moment and perform the query again in a moment. This happens behind the scenes, and the user merely sees a loading screen.

If the URL has already been submitted but the slave nodes are still processing it then we return status code two, along with whatever data we have at the moment. Status code two indicates that we the URLs already exists in the database but the system is currently still processing the URL. So the client should repeat the request in a moment.

Status code 0 is reserved for when a URL has been completely processed the request then contains all of our generated information and the client can stop querying.

Why This Design?

We chose this three segment design as it allows us to both separate the code we have for each task into different files and its own processes, similar to microservices, each of which can run concurrently. During our testing, we were able to create different numbers of the slave

processes as needed for each task easily without impacting the functioning of the rest of the application.

We also divide the work required for each team member into discrete files that each member could update and work on without merging issues on our remote repository. This design also allows to keep the code modular, maintainable, and easy to modify without breaking other components.

Work Distribution

- Austin Joseph
    - Application design
    - Database interfacing
    - Web server implementation
- Gao Xiangshuai
    - Website design
    - Application deployment
    - Database deployment
- Timothy Yuen
    - URL generation algorithm implementation
- Yehonathan Litman
    - URL querying implementation

Austin Joseph
Gao Xiangshuai
Timothy Yuen
Yehonathan Litman

CSE 331
Team 4

Setup Instructions

- Requirements
    - Python 3.7.4
        - Flask (pip install Flask)
        - MySQL Connector for Python (pip install mysql-connector-python)
        - Selenium (pip install Selenium)
            - You may also need to install chromedriver from https://chromedriver.chromium.org/ if on windows. (There is a version in the repo that should work but other systems can be weird)
            - Note that on linux this can be simply done with ‘sudo apt install chromium-chromedriver’
    - MySQL 8.0.18 Community Server (https://dev.mysql.com/downloads/mysql/)
        - The scripts to setup auth and tables can be found in the mysql dir
- Execution
    - Webserver.py, typogenerator.py and urlverifier.py are the three parts of the application. Any number of instances of each program can be run but we recommend only one instance of webserver.py to avoid port routing issues.
    - Each program takes one argument to run the config.json file that is in the src dir(next to each of these files)
    - Run each program with the command python command, program location, config file location
    - Examples
        - python3 webserver.py config.json
        - python3 typogenerator.py config.json
        - python3 urlverifier.py config.json
    - You can then reach the webserver at localhost:3000