COMP4561 Project

# Report on a Distributed Project Data Analysis Using Edge Computing With Visual Studio Code Plugin

Group:
Group 16

Members:
Aleksandr Sergeev (21158862)
Wong Hiu Tung (20859316)

Project repository:
https://github.com/pseusys/dspdavscp

# 1. Introduction

When completing coding assignments, the only insight obtained would be whether the project could be compiled, run, and pass the pre-defined tests. However, as information on the development process would be disregarded, details on the development process and time spent might be lost. These data potentially could bring more insights into analyzing an individual's or a group's coding struggles. Furthermore, since code grading is only performed as a batch process, it is impossible to have a real-time analysis of the project's performance.

Therefore, this project aims to develop a system that can process and analyze the coding habits and statistics on a shared project on demand using edge computing frameworks. This allows the data to be analyzed closer to the source, thus using less bandwidth when transmitting to the server. It is then aggregated and further summarized on an endpoint for better visualization.

# 2. Design

## 2.1. System design

The project is designed with the user case of a coding assignment in mind, in which the students would be provided with a starting codebase. Therefore, for the program, we have taken the following assumptions:

1. The file names of the scripts and programs cannot be changed. This is to maintain unity within all the submissions.
2. The program would only be executed using the integrated terminal.

For the system design, concerning the client-side extension, we would use VS Code as the targeted platform to host the extension. Since VS Code has an established extension eco-system, as well as being a popular IDE for students' coding assignments. There should also be a server hosted to receive the requests from the extension, as well as a dashboard to display the analyzed result for better viewing experience..

## 2.2. Data analysis design

As we would like to track the performance of the coding progress, we would need to consider what data could be extracted, and what information to be analyzed and reported. Therefore, to report on the performance, we chose the following metrics to be collected:

1. The time the project is opened
2. The time the code was run
3. The line number and the modification per line
4. Terminal output of the errors

5. Total time the file was saved

The following would be the information to be extracted and analyzed:

1. The average time all users spent on the project. This gives an estimation as to how long an average student needs to use to complete this task, and for better adjustment for future deadlines.
2. The average time all users spent running the code
3. The top 5 most common error messages encountered. The number of errors most often encountered can show where the students might have difficulties, and better adjust the curriculum to cover the weakness.
4. The top 10 lines of code students that have the most modification. This might hint at where the students find most difficult since the intuitive thought is that if a line is challenging, they might spend a lot of time trying different solutions. Thus, lines that are most modified are often those that encounter the most error.

Apart from direct analysis of the code in these matrices, we would also like to explore whether some other information could be extracted, such as how hardworking a student was, or whether a student possibly uses much copy and paste operations in their code.

For hardworkingness, we define that a student would be hardworking if they are spending a lot of time on the work actively coding and modifying. Therefore, data such as the coding time, run time and the total number of lines could help model the hardworkingness. From this idea, we formulate the following model for hardworkingness.

$$Hardworkingness = totalNumLinesModified^{c}[\frac{a(codeTime)}{1000} + \frac{b(runTime)}{1000}]$$

Where a, b, and c are hyperparameters with default values 1, 1, and 2 respectively.

This is formulated since we would like to reward a higher score to students who spend more time modifying code. However, since the code time metric taken is only the time that the project was open, therefore there might exist the case where the project is opened, but no modification was performed. Therefore, we allocated a larger scale to the lines modified. If in cases where the coding time and run-time are equal, the client with more lines modified would achieve a larger ranking. Retrospectively, if a user did not modify any lines in the code, and just purely ran their code or left the project open, they will be given 0. Only when there are lines modified would there be a positive score.

For ranking the possibility of the user using copy and paste operation many times in the project, intuitively we would like to identify large fluctuations and alterations to the codespace in a short period. Since when the paste copy and paste option is invoked, there should be a large change within a short time as if the same copied item was typed by the user, it would take more time. Therefore, with this idea in mind, we formulated the following model to guess whether the user used copy and paste. Furthermore, if the user did paste the content copied from another source, normally they would not need to use any time to think for the solution in the project. Therefore in general the code time would be short, with large modifications in code. Thus, the following model was proposed.

$$Possibility\ score\ = \frac{codingTime}{totalLinesModified}$$

Which calculates the average time spent per line. The lower the score, the higher the possibility that the user uses the copy and pasting option a lot.

# 3. Implementation

## 3.1. The VS Code extension implementation

The client is a VSCode extension, written in TypeScript using VSCode API. The HTTP requests are handled by the client package automatically generated by openapi-generator app. The extension can be configured with VSCode settings and commands, the status of the extension is reflected in a status bar element. Different information about user activity is collected and stored in memory, upon timeout or extension termination it is sent to the server (this can also be triggered manually). The memory storage is cleared afterwards.

In order not to overload the network with large requests, file-related and terminal output information is processed and sorted before sending, only top-25 records are taken.

The file-related information is sorted according to a special `score`:

$$score\ = \frac{FMLN}{FLN} + \frac{FCT}{TCT} + \frac{FSN}{TSN}$$

Where `FMLN` stands for number of lines modified in this file, `FLN` stands for total number of lines in this file, `FCT` stands for the time this file was opened in editor, `TCT` stands for total time all project files were opened in editor, `FSN` stands for the number of times this file was saved and `TSN` stands for the total number of times all project files were saved. For all the selected files, modified lines are sorted by the modification number and top-25 are taken.

A non-max-suppression-like sorting algorithm is applied to the terminal error strings and top-25 of them are taken; the algorithm goes like this:

1. Cumulative levenshtein distance is calculated between all the outputs
2. The outputs are sorted according to this distance, ascending (lowest distance means error lines that are more similar to all the others)
3. For all the lines, first one is taken, then all the other error strings, that are more than 70% similar to the taken one are discarded (similarity is calculated by levenshtein distance again, divided by the string length)

## 3.2. Server implementation

The server was built as a REST API server to receive the requests posted by the client extensions. We decided to use Python as the main language to implement the server. It follows the OpenAPI specifications too for better versioning and enhances future readability. The reports received from the extension are stored in a local SQL database and retrieved and analyzed when the dashboard calls.

A simple dashboard interface is implemented using HTML and can be accessed using a web browser. It displays the summarized report and can be updated by refreshing the webpage. A form is also included within the dashboard to alter the hyper-parameter for the formula modeling hardworkingness.

# 4. Conclusion

All in all, this project utilizes edge computing to enhance data transmission for better data analysis. By using distributed computing frameworks such as edge computing, we are able to bring more insights into scenarios such as coding assignment performance compared with previously, which we are limited to only being able to gather insight on bulk. This allows more responsive actions to be taken in response to these data analyses. Furthermore, by utilizing such a structure, it would be easier to perform big data analysis as the total number of data is less than compared with direct analysis of raw data transmitted from clients since the data received at the server are summarized reports.