SALP: A Scalable Autograder System for Learning Programming - A Work in Progress

Diego Calderón, Erick Petersen, Oscar Rodas
Universidad Galileo, diegocdl, ptrsen, orodas@galileo.edu

Abstract - **Programming courses can be hard for students, but also for teachers, because of the huge amount of time that takes to manually grade each student's assignment and the different kind of valid solutions. Moreover, there are other problems related to manually grade assignments such as completely objective and homogeneous grading. In consequence, both students and teachers don't get feedback as fast as they should in order to take action and reinforce the topics with lower performance on each assignment. Finally, the increasing popularity of MOOCs makes manually grading no longer viable. To this aim, a scalable autograder system is proposed in order to provide students with faster feedback and help teachers with the evaluation of assignments. Our proposal can be used for learning different programming languages like Java, Python, C, C# and Ruby.**

Index Terms - automated grading, containers, plagarism, programming assignments, online grading

I. Introduction

In the study of science, technology, engineering, and mathematics (STEM) careers it's necessary to learn computer programming. Especially because the computers are getting more powerful and are very common tools for conducting studies or solve problems nowadays. For instance programming is used for mathematics, electronic engineers, biomedical devices, artificial intelligence, data science, etc.

Moreover in programming courses it is usual that the teacher assigns hands-on laboratories weekly to practice the topics that were covered in the lectures. Usually assignments contain multiple exercises designed to apply one or more theory concepts. Even so the students face different problems such as syntax errors, logic errors or misunderstood a topic making difficult to finish all the exercises and to grasp all the knowledge. In fact, student start to struggle finishing an exercise and can not complete the laboratory. Hence in the next laboratory there are new exercises that probably require the knowledge of the past exercises which the student didn't complete, making even harder to solve these new problems.

Furthermore, students start getting frustrated resulting in dropping the class or trying to understand the next assignments, but unfortunately, it's not easy to recover and approve the class in most of the cases. In the past 5 years, we have seen a medium approval rate in our first programming course at Universidad Galileo and a high rate of students retaking the course.

On the other hand, grading the assignments is a time consuming task and is hard to keep giving feedback to the students before the next assignment. As a result the teachers and the students don't have fast feedback to take actions on reinforcing the topics where the students have problems. Besides the time that it takes, there are several problems when the teacher or the teacher assistants (TAs) manually grade the assignments. We can mention a couple of these problems:

- It's very hard to be homogeneous and completely objective, especially when you have to grade dozens or hundreds of assignments.
- Provide very detailed feedback is not always possible.
- Difficult to replicate the same test on all submissions.
- Can be tricky to test all the possible scenarios on a specific exercise.
- Hard to detect plagiarism.
- More TAs have to be hired and teach them the appropriate manner to conduct the evaluation.

Furthermore, in recent years STEM has been promoted worldwide and creating more interest in studying STEM-related careers. As a consequence, we can see an increment of students that are taking programming courses in universities and people enrolling in massive open online courses (MOOCs) [1]. Hence, in MOOCs usually there are thousands of students that make impossible to manually grade the assignments. For this reason an automated tool for grading assignments is needed.

In fact there are some options to attempt to solve the problem like local scripts or cloud based solutions. For instance, local scripts can be written to provide feedback and have the advantage of easy distribution and students can run them easily on their computers. On the other hand, cloud based solutions are more complex to create but the test can run in a more controlled environment making them more reliable for the teachers. Since students are sending their solutions to be grade, this gives an opportunity to have a dashboard available for students and teachers in order to show the progress in real time.

Moreover there are some specific work related to make automated tools that help to solve this problem [2]-[6] and those tools use different approaches to grade the assignments. For example, some tools evaluate the output given based on some input. This option has several limitations because the student can solve the problem in an incorrect way (i.e. discover the test cases and just output for the test inputs or use loops when the exercise required a recursive algorithm). With

this option we can not assure people are learning to program with the correct fundamentals, we are only grading courses in the capability of students to produce the right output. On the other hand, other solutions are proposed like using semantic formals to evaluate a reference solution against the students solutions to determine if it is correct.

Specifically some work related to autograders has been done in some of the introductory computer science courses in our university with the goal of provide fast feedback to the students. First local scripts were done and the students can run a pre-test in their assignments to help them to check if they were covering all the requirements. Although those scripts help the students, the teachers still didn't have prompt feedback of the progress. Hence, a cloud based solution has started combining the ideas of the scripts and a very simple dashboard where the students and the teachers were able to see the results. Nevertheless, the implementation of this solution was very simple and some problems started to show up which allows us to learn about what should be taken into consideration for building a better solution.

For this reason, in this paper we present a proposal of a cloud-based scalable autograder system for learning programming assignments. The next sections in this paper are organized as follows: Section II describes the challenges that need to be face during the implementation and in Section III the proposed architecture to be used. In Section IV our preliminary conclusions.

II. Challenges to Face

There are many things to consider while designing and implementing a tool for automated grading assignments and the requirements can vary between the course needs:

- **Isolation:** The tests should be executed in an environment controlling the network, memory, disk access and the CPU time. For example, the most common case is when a student makes an infinite loop and the program never stops, so it's necessary to kill the program after some define time. Moreover, the network should be restricted in assignments where it's not needed, to avoid cheating with the use of some API to get the answer to the problem. Finally, the system that receives the submissions must be isolated from the running tests subsystem so the system still accepts submissions besides a case of a failure in the grading side of the system.
- **Scalable:** Taking into account the fact that MOOCs have a very extensive audience, the system should be capable to be scaled up in order to handle all students. Moreover, the workloads are not even, teacher will assign the laboratory and in the first following days a high workload will be demanded to the infrastructure and will decrease in the next days. In this case, the system should be scaled up just for those beginning days.
- **Recovery:** It is important to be prepared in case something fails. Certain precautions have been taken into account but it is hard to cover all possible errors or scenarios. Therefore the system should have a recovery method that restarts all the unfinished grading tests that were executing before the failure.
- **Modular:** The system should be capable of being flexible in order to support different programming languages e.g. Java, Python, C, C#, Ruby, etc. Furthermore, accept submissions through different platforms such as a bash script, git remote repository (i.e. GitHub) and a website.
- **Reports and Analytics:** Knowledge of the progress of assignments is beneficial for both students and teachers, so the system should provide reports in real time. Moreover, some students get motivated through competition, so a leader board that shows the top 10 helps to encourage the students to solve more quickly the assignments should be implemented. Furthermore, using the information collected, gamification concepts can be applied and can help to improve the interest of the students in working and finishing the course assignments [7]. Finally, data analysis can be implemented to show more detailed information and start predicting the performance of the students.
- **Plagiarism:** One of the more important problems to handle in programming courses is the plagiarism and it is necessary to detect it as soon as possible. When a student starts committing this kind of actions, information should be deliver to the teacher so he/she can talk to the student and persuade him/her to stop this plagiarism actions [8]. Moreover, this is one of the most important features of our proposal in order to have accurate reports and grades. If plagiarism is not detected soon, information will show that students are performing well in the course and learning a lot.

III. System Architecture of SALP

Taking into account all the challenges that we described in Section II, the following architecture is proposed as illustrated in Figure I:

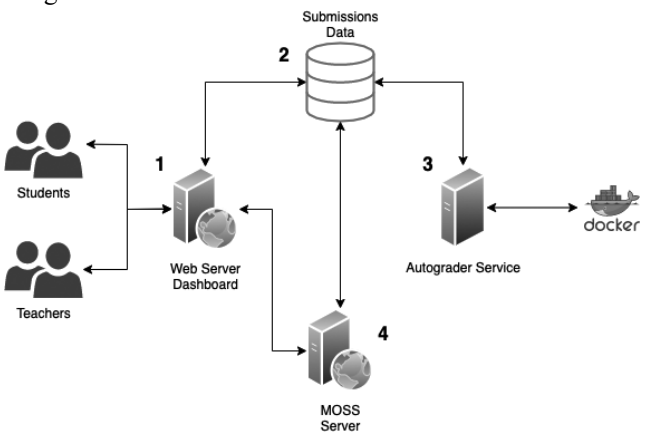

FIGURE I
THE PROPOSED SYSTEM ARCHITECTURE

1) Web Dashboard: The primary method to interact with the system should be a website that allows access control with

authentication for multiple roles such as student, teacher and teacher assistant. For instance, teachers will have the option to create assignments and define tests for grading the exercises in a determined class. On the other hand, students will have the option to view the assignments of the classes which they are enrolled and submit answers to be autograded. Moreover, the student should have the option to view the feedback after the autograder service processes his submission.

In addition, it is important to provide the teacher with the option to manually edit the grade of a particular student because something went wrong with the test cases or the teacher decides the solution does not complains with the specifications of the assignment. Additionally, the teacher should have the option to update the grading tests and re-run the last submission of all students using new tests.

Furthermore, reports and information that the system could provide will help the students and teachers to perform and interact better in the course. For instance, gamification techniques could be applied using leader boards to show which students completed the exercises faster and give them badges by achievements e.g. first in completing the laboratory, the sent solution was right since the first submission, etc. In other outreach activities and courses in Universidad Galileo the use of the website of badgr.io has been helpful to foment a better engagement of students in finishing their assignments.

We have also considered that more experienced students may want to use a command line tool to send files to the system. For this reason, a script should be provided which use an API endpoint in the dashboard to send the files.

2) Data: Since the implementation of the system is distributed, it is necessary to communicate the dashboard, that acts as a front-end that accepts submissions, and en-queue them to be processed by the autograder service. Files are stored in the file system and PostgreSQL will be used for implementing databases. For example, the information about the submissions should be saved, especially the state which will be used by the system to know which submissions are waiting, running, finished or stopped because a timeout. A state machine diagram is shown in Figure II to describe the transitions between the states.

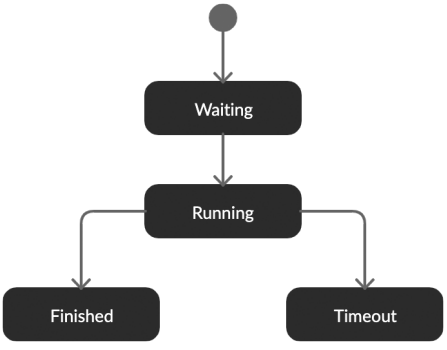

FIGURE II
STATE MACHINE OF A SUBMISSION

3) Autograder Master Service: This is the most complex part of the system. In this subsystem all the submissions that were en-queue will be processed. In fact, this is the part that most of the challenges described in Section II were considered in the design and implementation. Therefore, this service would be in charge of processing the submissions from the queue and store the results of the tests in the database that will be displayed in the dashboard.

An implementation with containers was chosen in order to isolate the running tests, they are lightweight and allow isolation, we can limit the amount of RAM and CPU and also, they can be stopped at a determined time. Specifically, Docker [9] containers with Kubernentes [10] were selected as a solution to implement the system because their popularity, great community and because they provide a nice API to programmatically control the execution of the containers.

Moreover, Docker images are defined in the dashboard by the teachers for grading the assignments. This gives the possibility of using different versions of programming languages making the system more flexible. Additionally, API clients are provided for different languages to be used by the teachers to submit feedback from the docker container to the Autograder Master. For instance, inside a Docker image the teacher could use different methods to evaluate a student's solution and provide feedback for the errors e.g. black box testing, unit testing, static analyzers, memory leak analyzers, semantic formals.

Furthermore, in order to show the status of the service in the dashboard, an API is needed to provide information about status: number of queue submissions, number of executing submissions, configuration of execution limit, etc.

In case of a failure and the system is restarted, the system should be capable of handling this situation without losing information. Hence all the process will have a status that allows to know which of them were processing before the failure. Moreover the system will start again all the processes that didn't finish before the failure.

4) MOSS: The plagiarism detection is done using MOSS (Measure Of Software Similarity). In the dashboard there should be an option to send all the submission files from an assignment for evaluation in MOSS. As a result, a report of similarity would be obtained and will be showed in the dashboard so the teacher can review each case to determine if it is a plagiarism or not.

IV. PRELIMINARY CONCLUSIONS

Finally, some of the challenges to face in the design and implementation of a tool for autograding were presented and taking into account to determine a proposal for the architecture of an autograder system. The tool is still under development and it is expected to be tested during the second half of this year. Specifically, in the course of *"Object Oriented Programming and Data Structures"* where we are looking to validate the functionality and other opportunities for improvement. Additionally, we will evaluate the user experience and the perception of the students while using the system. We can infer this autograder system will have an

impact in the learning process of students and the approval rate will increase.

V. Acknowledgments

We thank the staff of the courses "Ciencias de la Computación I, II and III" who provided their insight and their ideas during this research. We also thank Andres Castellanos for his comments and recommendations in this paper. Additionally, we thank Universidad Galileo, for fomenting in teachers and their staff, the use of technology to embrace excellence and better ways of teaching.

References

[1] L. Yuan and S. Powell, "Moocs and open education: Implications for higher education," 2013.

[2] Head, A., Glassman, E., Soares, G., et al. 2017, April. Writing reusable code feedback at scale with mixed-initiative program synthesis. In Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale (pp. 89-98).

[3] Yulianto, S. V., & Liem, I. 2014, November. Automatic grader for programming assignment using source code analyzer. In 2014 International Conference on Data and Software Engineering (ICODSE) (pp. 1-4). IEEE.

[4] Morris, D. S. 2002, November. Automatically grading Java programming assignments via reflection, inheritance, and regular expressions. In 32nd Annual Frontiers in Education (Vol. 1, p. T3G). IEEE.

[5] Peveler, M., Maicus, E., & Cutler, B. 2019, February. Comparing Jailed Sandboxes vs Containers within an Autograding System. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education (pp. 139-145).

[6] Jackson, D. 1996. A software system for grading student computer programs. Computers & Education, 27(3-4), 171-180.

[7] Khaleel, F. L., Ashaari, N. S., Meriam, T. S., Wook, et al. 2015, January. The study of gamification application architecture for programming language course. In Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication (pp. 1-5).

[8] Fonseca, N. G., Macedo, L., & Mendes, A. J. 2018, September. Using early plagiarism detection in programming classes to address the student's difficulties. In 2018 International Symposium on Computers in Education (SIIE) (pp. 1-6). IEEE.

[9] "Docker" docker.com Web. Accessed: January 28, 2020.

[10] "Kubernetes" kubernetes.io Web. Accessed: January 28, 2020.

Author Information

Diego Calderón, Research Assistant, Turing Lab, Universidad Galileo, Guatemala, Guatemala.

Erick Petersen, Research Assistant, RLICT, Universidad Galileo , Guatemala , Guatemala

Oscar Rodas, Ph.D., EE Career Director, STEM Outreach Program Director and Tesla Lab Director, Universidad Galileo, Guatemala, Guatemala.