Development of Autograder For Competitive Programming Using Contestant PC As Worker

Jauhar Arifin
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia
jauhararifin10@gmail.com

Riza Satria Perdana
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia
riza@informatika.org

***Abstract*—Competitive programming is a computer science competition where the contestants compete to solve computer science problems by writing a program which satisfies the problem constraints. Autograder is used to grade contestant solutions automatically in real-time. Usually, autograder is deployed in many computers to increase grading performance. In this work, contestant computers are used as a worker to run autograder. By using contestant computers as a worker, the number of workers will proportional to the number of contestant submissions, thus increasing grading performance. Every contestant computers have different specification and can affect grading fairness. To keep grading fairness, contestant's solution and jury's solution executed in contestant worker and compared to check whether contestant's solution satisfies problem constraints. This work tested by simulating grading process in contestant computers. The testing result indicates that using contestant computers as worker gives performance improvement in the grading process.**

***Index Terms*—competitive programming, online judge, autograder.**

I. Introduction

Competitive programming is one of the most popular competitions in the computer science field. In competitive programming competition, contestants are asked to solve computer science problems correctly and as fast as possible. Some institute and organization often organize competitive programming competition periodically. Some big companies like Google and Facebook organize competitive programming competition annually. Competitive programming competition supported by an online judge system. Usually, online judge system is a web-based application where contestants can read the problems, create clarifications, submit their solution, and watch the scoreboard. Currently, the most popular online judges are Codeforces, URI Online Judge [1], Uva, and SPOJ.

In order to grade contestant submissions, online judge system has a subsystem called autograder. Contestant submissions which are source code in a certain programming language will be graded by the autograding system by compiling the program and executes compiled program using test-cases that have been prepared by juries or problem setter. According to [7], this grading method is called black-box grading. By using the autograding system, the grading process can be done automatically and juries don't have to evaluate contestant submissions manually. In order to increase the number of contestant submissions in a certain amount of time, juries usually deploy autograder in many computers. In order to run autograder in many computers, the juries need to prepare many computers with the same specification to keep the fairness of the grading process.

Currently, almost every competitive programming competition use online judge system to support the competition and use many computers to run autograder in order to improve grading performance. Every computer that runs autograder is called a worker. Grading performance is defined as the number of submissions graded in a certain amount of time. Even though autograder deployed in many computers, the grading performance often not enough to evaluate contestant solutions in real-time because the number of contestants is increasing. Furthermore, the number of computers to deploy autograder affect the procurement cost that juries need to be incurred. Therefore, a new grading system is needed to increase grading performance without increasing procurement cost that needs to be incurred.

When competing in a competitive programming competition, the contestants usually use their personal computer to write the solutions. Every contestant computers usually have sufficient specification to compile and execute contestant submissions. Therefore, contestant computers have the ability to run the autograder program and evaluate contestant submissions.

II. Related Work

There are some popular online judge systems that have been used to organize competitive programming competition. Some online judges offer other additional features like discussion forum, training gate, and rating system. Nowadays, most of the online judge deployed as a web application and use a specific computer to grade contestant submissions. There are many types of competitive programming competition rules. Most of the online judge only support a specific competition rule.

One of the most popular competitive programming competition is ACM-ICPC competition where the competition uses ICPC rules. In ACM-ICPC competition, contestants compete in a group of three people. Every group has the same amount of problems to solve, and the score is determined by the number of solved problem and time penalty. There are many online judges that support this competitive programming rule. The most popular online judge that support this type of competition is DOMJudge. DOMJudge is very popular because it has many useful features, easy to use and open source.

Currently, most of the online judge use autograder to evaluate contestant submissions. According to [4], autograder is a system that compiles, executes and evaluate source code. Evaluates source code manually takes three minutes while using autograder only need ten seconds. By using autograder, contestants can receive grading feedback faster and reduce the work that needs to be done by juries. Basically, the autograder evaluate the source code by compiles the source code and executes the compiled program using predefined test-cases. The output of contestant's solution program compared to the predefined test-case and then graded. This evaluation method is called black-box grading [7].

We need to consider the security aspect when developing an autograding system. Competitive programming contestants might submit source code that contains dangerous code. According to [2], there are several attacks that can be made by contestants, such as submitting compile bomb code, submitting code that destroys the autograder environment and submitting code that access forbidden computer's resource. In order to prevent such things, an autograding system must execute contestant's code in an isolated environment called sandbox.

There are several ways to create an isolated environment such as using virtual machine and containerization. By using a virtual machine, one can create an isolated operating system inside the host computer. The virtual machine uses the hypervisor to emulate computer hardware and run its own operating system. In Linux based operating system, there is a feature called KVM that makes it possible to create hypervisor as a process in Linux [3]. We can use a virtual machine to isolate the execution of contestant's code. However, by using the virtual machine, we need to boot a new operating system each time we evaluate contestant's submission. This booting process takes so much computer's resource and reduces grading performance.

Another way to create an isolated environment is by using a container. A container is different from a virtual machine. A Container gives isolation in software level without hypervisor [5]. By using a container, we can isolate process execution without boot a new operating system. The container works by using some Linux features such as chroot, namespace and cgroup. By using chroot, we can isolate the filesystem of Linux process [6]. The container uses a Linux feature called cgroup to limit process resources. By using cgroup, we can limit memory usage, CPU usage and disk IO usage of running process [3]. In order to hide other processes, user, and network from a running process, the container uses a Linux feature called namespace. By using a namespace, we can isolate a Linux process from other processes, so the running process not aware of the existence of other processes. Instead of a virtual machine, most of the autograding system use container to isolate the grading process. This method is used because it is lighter and faster than the virtual machine.

III. Methodology

A. Time and Memory Measurement

In competitive programming competition, every problem has time and memory constraints. Contestants are asked to solve the problem by writing a program that satisfies the constraints. According to ICPC rules, when contestant's solution exceeds the problem constraints, the solution is rejected and the contestant gets a time penalty. Thus, the autograding system needs to tell whether contestant's solution exceeds the constraints or not. The problem is, every computer has a different performance when executing a program. The same program might run faster in one computer and slower in another computer. Currently, most of the online judge system uses identical computers to evaluate contestant submissions. By using identical computers, the same program is expected to run at the same duration and consume the same amount of memories.

Contestant computers have different specification and performance. We cannot force the contestants to have an identical computer. Thus, we need a new time and memory measurement method. The performance of a computer is determined by its specifications such as CPU clock speed and operating system. We can tell the computer performance by its specifications. However, besides CPU clock speed and operating system, there are so many other factors that affect computer performance such as temperature, cache size, the program itself, and many other factors. Thus, determining computer performance by its specification is very hard and not feasible to do.

$$R_{contestant} < (1 + \gamma) \times R_{jury} \tag{1}$$

In this work, we compare contestant's solution and jury's solution. Contestant's solution considered satisfying the problem constraints when the memory and CPU usage of contestant's solution not exceeding jury's solution. Sometimes, contestant solution's memory or CPU exceeds the jury's solution a little bit because the implementation is different. We add a tolerance factor (γ) to tolerate this problem. Contestant's solution is considered satisfying problem constraints when the Equation 1 is fulfilled. When evaluating contestant's solution, the worker compiles and executes contestant's solution and jury's solution. After executes the solutions, the worker compares the memory and CPU usage of the execution and determine whether the contestant's solution satisfies problem constraints.

B. Load Balancing

We need to distribute grading jobs to worker evenly in order to increase grading performance. There are several load balancing method that we can use to distribute grading jobs evenly. The easiest way to distribute the jobs is by making each contestant to grade their own submissions. In this work, we didn't use this method because this method creates security vulnerabilities.

The other way distribute grading jobs is by using push-based load balancing method. In push-based load balancing, the online judge server chooses the worker to do the grading jobs. By using this method, every contestant doesn't know which submission they grade, thus can increase system security. To improve system security, we can also grade every submission multiple times. In order to implement this method, we need the status of all workers. This method is pretty hard to implement because we need to poll the worker status periodically and choose the best worker to evaluate the contestant submissions.

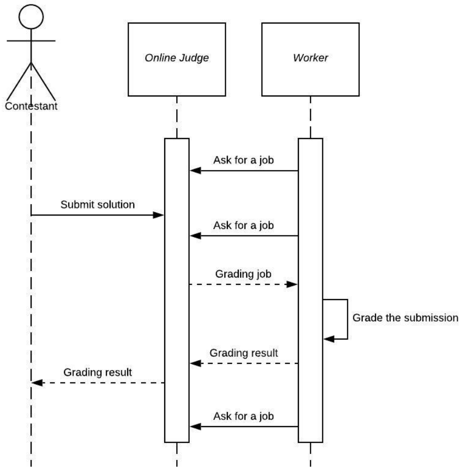

Fig. 1. Pull-based Load Balancing.

In this work, we use a pull-based load balancing method. By using this method, we don't have to poll the worker status periodically. The worker will poll the jobs from the server periodically and evaluate them when there is a ready job to be evaluated. To improve system security, we grade every submission multiple times. We define a grading size of the system as the number of times we need to grade each submission. Fig. 1 depicts this method mechanism. This work is easier to implements than push-based load balancing method and has the same advantages.

C. Evaluating Submission In Isolated Environment

In order to isolate the evaluation process, we use containerization method. We compile and execute contestant's submission inside a lightweight container. This container is created by using several Linux features and system calls such as cgroup, namespace, chroot and setrlimit. We use cgroup to monitor and limit the CPU and memory consumption of the running process. The worker kills the program when the CPU or memory usage is exceeding the limit. In order to isolate the host's user, process and network, we use Linux feature called namespace. By using a namespace, the processes inside an isolated environment will not aware of any process, user or network in the outside of its own environment. In order to isolate filesystem of a process, we use the chroot system call. Besides isolating the filesystem, we also need to bring some files from the host environment into an isolated environment. We bring some of the host filesystems into the isolated environment by using the mount system call. We also limit some of processes resources by using the setrlimit system call. By using setrlimit system call, we limit some process resources such as the number of child processes, the stack size, the number of open files and the size of the created file of that process. Fig. 2 depicts how an isolated environment is created.

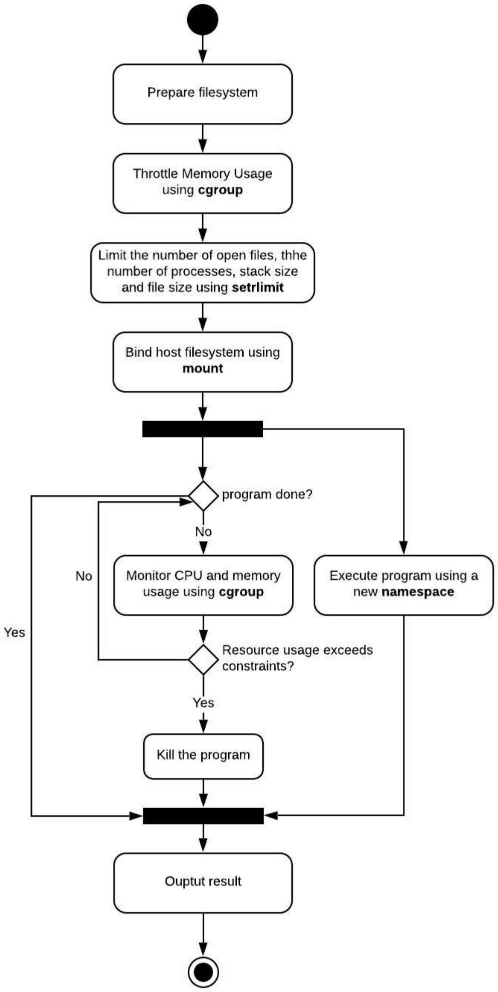
Prepare filesystem
Throttle Memory Usage using cgroup
Limit the number of open files, thhe number of processes, stack size and file size using setrlimit
Bind host filesystem using mount
program done?
No
No
Yes
Monitor CPU and memory usage using cgroup
Execute program using a new namespace
Resource usage exceeds constraints?
Yes
Kill the program
Ouptut result

Fig. 2. Sandbox Activity Diagram.

D. Program Compilation

In order to evaluate contestant's submissions, we need to compile and execute the submitted source code. To compile the source code, we need a specific compiler. Every worker should have the same version of the compiler. The same compiler with different version can produce different output. In order to keep the fairness of the grading process, we need to distribute the same version of the compiler to all workers in the system. In this work, we distribute the compiler in the autograder program to every worker. Every worker compiles the source code inside an isolated filesystem that has been installed with the same version of compilers.

E. Obtaining Problem Test-cases

Every worker needs test-cases to evaluate contestant solutions. A test-case is a pair of text files that contain input and output of the problem. Usually, the test-case files are generated by a test-case generator that created by problem setters. The worker evaluates the contestant solution by executing the solution using the input test-case and compare the output with the output test-case. In order to evaluate the contestant solutions, we need to bring the problem test-cases into the worker. To bring the test-cases into the worker is not easy because the test-case file can be a very large file and confidential.

In this work, we don't transfer the raw test-case files to all the workers. Transferring raw test-case files need huge bandwidth and can reduce grading performance. Instead of transferring raw test-case file, we transfer the test-case generator program to generate test-case files. The worker receives the test-case generator source code from the server and compiles it. The compiled test-case generator program then executed to generates input test-case. The worker uses the jury's solution to generates output test-case files. The worker also needs a checker program to compare contestant's output with jury's output. The checker program receives contestant's output and jury's output as its input and gives the verdict of the submission.

In order to hide the test-case information, we encrypt the test-case generator program and the generated test-case files. Even though the test-case is encrypted, the contestants might decrypt it if they know the encryption key. The contestant can attack the autograder program using reverse engineering technique to obtain the encryption key. In this work, we didn't handle this problem. Further research is needed to solve this problem.

F. Handling Reverse Engineering Attacks

Because the autograder programs are deployed in the contestant computers, the contestants can do everything to the program. The contestant can attack the autograder program on their computer. There are many kinds of attacks that the contestant can do such as changing the compiler program, sending the false verdict to the server and refuse to grade other contestant submissions. In this work, we didn't handle this kind of attacks, but there is a way to handle this kind of attacks.

We can prevent a contestant to refuse to do the grading process by monitoring the grading process. When we find out that there is a worker that not asks for a job for a long time or not sends the grading result for a long time, we can tell that there is a contestant that attacks the autograder program in their computer. Then, we can warn or disqualify the contestant for attacking the autograder program.

By grading every submission multiple times, we can prevent a worker from being attacked that cause it gives the wrong verdict after grading. When we grade a submission multiple times and there is a contestant that attacks the autograder in their computer, we can tell by investigating the grading result. If the grading result of a worker is different from the rest of the workers, we can tell that the contestant attacks the autograder program in that worker. Then, we can warn or disqualify that contestant for attacking the autograder program.

There are many more types of attacks that we didn't handle in this work such as stealing jury's solution, other contestant's solution, and test-case generator. Even though we didn't handle some type of attacks, this system can still work in several competitions such as ACM-ICPC. In ACM-ICPC, every team is given a computer by the juries to solve the problems. The juries can install the autograder program inside contestant's computer using the root user and not giving the root access to the contestant. By using this mechanism, the contestant can't attack the autograder program inside their computer.

IV. Implementation

We named this system UGrade and implement this by developing five programs, i.e., UGServer, UGDesktop, UGCtl, UGJob, and UGSbox. UGServer handles everything related to competition such as authentication, contest creation, problem creation, etc. We can consider UGServer as a server of all the workers in the system. UGServer stores all the problems, submissions and the grading results. We develop UGServer using Python with Django and use GraphQL for the API. We use Django because of its ease of use.

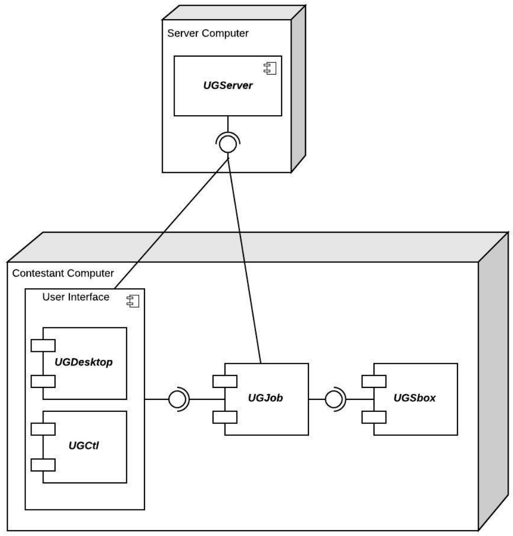

Fig. 3. Component Diagram of UGrade

To create an isolated environment, we create a sandbox utility program called UGSbox. By using UGSbox, one can execute a program inside an isolated environment and limit its resources. We develop UGSbox using Golang because of its ease of use. UGSbox is used by autograder program to compiles and executes the program for the grading process.

In this work, we named the autograder program as UGJob. UGJob asks UGServer for a job and does the grading process. UGJob uses UGSbox to compiles and executes the source code it receives from UGServer. We develop UGJob using

Golang because of its ease of use. The UGJob is deployed in every contestant computers. Every installation of the UGJob is equipped with the same compiler to keep the fairness of the grading process.

The contestants can interact with the system using user interface programs, e.g., UGDesktop and UGCtl. UGDesktop is a graphical user interface program to interact with this system. Every contestant uses UGDesktop to read the problem and submit their solutions. UGDesktop will execute UGJob periodically to grade the available job in the server. UGCtl is a text-based user interface program to interact with this system. Basically, UGDesktop and UGCtl have the same role in this system. We develop UGCtl just to automate the testing process. Fig. 3 depicts the relationship of the components we have developed.

V. Result and Analysis

We compare the grading performance between UGrade and DOMJudge. We chose DOMJudge as the comparison because it was the most popular open source online judge system for ICPC competition. We deployed UGrade and DOMJudge in a computer with the same specification. Then we deployed fifteen computers to simulating the contestant actions. For DOMJudge, we deployed two more computers to work as the autograder worker. For UGrade, the fifteen computers worked as workers.

We created a dummy competition with a single problem, e.g., polynomial multiplication. This problem was expected to be solved using $O(NlogN)$ solution, but there are many other solutions with different complexity. We use four types of solution to simulating contestant submission. The first and second type of solution have $O(N^2)$ complexity but have a different implementation. The third solution has $O(N^{log_2 3})$ complexity. The fourth solution has $O(NlogN)$ complexity and considered as the correct solution.

TABLE I
Average Time to Grade a Single Submission.

System	Time (seconds)
DOMJudge	198.0845733
UGrade with one as grading size	12.3706562
UGrade with two as grading size	11.42285704
UGrade with five as grading size	14.21330876

The fifteen computers were submitting each type of solution one by one two times. We logged every event that happens on the system and analyzed the result. In this work, we grade every submission multiple times for UGrade system. We tested UGrade nine times using three different grading size, e.g., one, two and five. We calculated the average time to grade a single submission in DOMJudge and UGrade. Table V shows the comparison between the average time to grade a single submission in each test. Based on Table V, we know that the UGrade grades the submission faster than DOMJudge. We also know that the grading size of UGrade affects the grading performance a little bit. By increasing the grading size of UGrade, we will grade each submission slower because each submission needs to be graded many times when the grading size is high. In the other hand, increasing the grading size will increase the system security because the submission graded in many workers. The juries need to adjust the grading size so it is fast enough to grade a single submission and secure enough to prevent any attacks from contestants.

VI. Lesson Learned

Based on the development and testing we have done, there are some lessons we learned.

1) There is a high grading performance improvement in UGrade compared to many online judges that are currently popular. This performance improvement is caused by the number of the worker in the UGrade which follows the number of contestants in the competition.
2) The grading performance of UGrade is not affected by the number of contestants but affected by the grading size number of UGrade. The high number of contestant causes a high number of worker too, so that doesn't affect the grading performance. However, the grading size number causes the number of submission doubles, thus affects the grading performance.
3) The high number of grading size reduce grading performance but increase system security. The low number of grading size increase grading performance but reduce system security. The juries need to adjust the grading size to achieve the best result.
4) The security of UGrade needs further research. However, we can use UGrade to organize some types of online judge competition where the contestant don't have root access to their computer such as in ACM-ICPC.
5) UGrade can keep the fairness of grading process even if the worker computers have different specification and performance. UGrade achieves this by comparing the jury's solution and contestant's solution in the same computer.
6) Pull-based load balancing of UGrade causes the UGServer load to become very huge. The load balancing method of this system needs further research.

References

[1] J. L. Bez, N. A. Tonin, and P. R. Rodegheri, "URI Online Judge Academic: A tool for algorithms and programming classes," *2014 9th International Conference on Computer Science & Education*, 2014.

[2] S. Wasik, M. Antczak, J. Badura, A. Laskowski, and T. Sternal, "A Survey on Online Judge Systems and Their Applications," *ACM Computing Surveys*, vol. 51, no. 1, pp. 134, 2018.

[3] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2015.

[4] K. Danutama and I. Liem, "Scalable Autograder and LMS Integration," *Procedia Technology*, vol. 11, pp. 388395, 2013.

[5] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux J.*, vol. 2014, no. 239, Mar. 2014.

[6] P. Lessard, "InfoSec reading room report: Linux process containment - a practical Look at chroot and user mode Linux," *SANS Institute*, 2003.

[7] J. Fernando and I. Liem, "Components and Architectural Design of an Autograder System Family," *Olympiads in Informatics*, vol. 8, pp. 6979, Jan. 2014.