Distributed architecture for autograding system

R Elsen*, A Latifah and A Sutedi
Department of Informatics, Sekolah Tinggi Teknologi Garut, Garut, Indonesia

*rickardelsen@sttgarut.ac.id

Abstract. Autograder used to measure source code quality based on the reference implementation. It will compare code execution results with reference implementation like a test case and generate grade based on the comparison result. College can use Autograder to help lecturer to test student’s skill in programming. Laboratory or online course can apply Autograder so students can use it to test their skill. Most of autograder use centralized architecture. It just executes and grades only one request at a time so if students submit their code to test, it will create a long queue and can risk highly to cause timeout because of a long queue and unexecuted request. In this paper, we propose to apply distributed architecture based on Service Oriented Architecture (SOA) for autograder to reduce queue and lower the risk for unexecuted requests that can cause an error in grading.

1. Introduction
Autograder used for grading code assignment to make sure that assigned code has the same behaviour with the given reference implementation [1]. If the assigned code has the same behaviour, it means code pass the test and considered correct. Reference implementation must be more than one to maintain the integrity of behaviour. For example, if autograder just use only one reference implementation, assigned code with the only use of print function may pass the test. It will show the correct result, but not correct behaviour.

Autograder can be used on college for grading student code assignment [2]. It can help lecturer to evaluate and marking student work. Student can use it to practice their programming skill. Autograder can show execution result and show the grade to the student so the student can know if they make a mistake when doing the task. Autograder can help student practice faster and precise because it grades the code based on reference implementation so the student can know if the code passes all behaviour.

Most autograder use centralized architecture [3-6]. It only has one backend to compile and execute code. With this architecture, autograder will make a queue for all code assignment and execute it one by one. The more requests come in, the longer the execution time will be. It can cause server overload because of the long queue of request and if it reaches max execution time defined by the system, the code cannot be graded and autograder cannot show the result of code execution.

2. Methodology
In this paper, we use blackbox and whitebox as software testing methodology to measure performance difference. Blackbox testing is a software-testing method to test the behaviour of software [7]. It named blackbox because this method does not test the code. Tester does not have any knowledge about code structure and design [8]. It just tests the function of the software to make sure that software can work correctly. Blackbox used to test the following categories:
• Incorrect or missing functions
• Interface errors
• Errors in data structures or external database access
• Behaviour or performance errors
• Initialization and termination errors

Blackbox testing needs prepared data to test software functions. Prepared data must be various so it can cover all functional test. Blackbox has some techniques for prepare testing data:

• Equivalence partitioning, prepare correct data and false data with the same portion.
• Boundary value analysis, prepare handled and unhandled data.
• Cause-effect graphing, prepare test case and check software behaviour based on the comparison between execution result and test case.

Whitebox testing is a software-testing method to test code [9]. Different from blackbox that tests the functionality of the software, whitebox test code behaviour. It checks the path of execution based on code structure so it can detect the location of the software problem. The tester must have knowledge about code structure and design [8]. This method usually used for tracking bugs and checking code structure.

Most of autograder use blackbox method [1–6,10]. It uses the cause-effect graphing technique. Autograder will grade code assignment based on the prepared test case. It also called as functional testing [11]. All possible behaviour must be covered in the test case so autograder can grade code assignment precisely and show the correct result to students. But we also found one Autograder using whitebox [12]. It not just grade behaviour of code but also check bugs/flaws in code.

3. Result and discussion

3.1. How autograder works

![Figure 1. Autograder process.](image)

As shown in figure 1, autograder grade code assignment with executing the code and comparing the result with the test case. Autograder can grade in two ways:

• Execute code assignment and compare it with the test case [2]. This way is done by prepare some test file contains correct expected result and compares it with the code assignment execution result.
- Execute code assignment and compare it with the sample execution result [1]. This way is done with prepare correct executable file, execute it, and then comparing the result with code assignment execution result.

Based on the explanation above, there are three phases for grading code assignment:

- Expected result preparation. In this phase, lecturer prepare test case for comparing with code assignment to test execution result correctness from code assignment.
- Code assignment execution. It may start with compile code first to create executable files. Execution result will be compared with the expected result.
- Grading code assignment by comparing code execution result with prepared expected execution result. A comparison result will determine grades of code submitted by students.

3.2. Autograder with centralized architecture

As shown in Figure 2, centralized architecture only uses one backend to handle all request. In the case of autograder, it only has one backend to handle code assignment execution and grading execution results. It will create a long queue if autograder used by many students. If the number of autograder backend \( A \) handle some requests \( R \) and the queue represented as an array, so it will create this queue:

\[
A_1 = [R_1, R_2, R_3, R_4, R_5, R_6 \ldots R_n]
\]

\( A \) must execute \( R \) one by one. If one of \( R \) was stuck because of error like infinite loop or execution error, the rest \( R \) may not be executed and stay in the queue forever. Applying maximum execution time per \( R \) can solve this problem so if \( R \) failed to execute and reach maximum execution time, the next \( R \) will be executed. However, it still makes the rest of \( R \) must wait until fail \( R \) reach max execution time. This will result in longer grading process.

![Figure 2. Autograder with centralized architecture.](image-url)
3.3. Autograder with distributed architecture

In a distributed architecture, autograder can use multiple backends as shown in Figure 3. Based on SOA, autograder can use multiple backends for loose coupling and flexibility purpose [13]. With the help of a load balancing system [14], $R$ will be distributed in a balanced manner. There are two methods for load balancing the request:

- **Backend queueing distribution.** With this method, $A$ will be queued in load balancer so each $A$ will get balance $R$. For example, with 4 $A$ and 13 $R$, it will result in this queue:
  
  $A_1 = [R_1, R_5, R_9, R_{13}]$
  $A_2 = [R_2, R_6, R_{10}]$
  $A_3 = [R_3, R_7, R_{11}]$
  $A_4 = [R_4, R_8, R_{12}]$

  By using this load balancing method, autograder can execute $R$ in parallel method based on the number of $A$. If some $R$ got an error and stuck, it will affect only to $A$ which handle that execution and will not affect other $A$ so grading can still run. The advantages of this method are fast in $R$ distribution. Load balancer just sends $R$ to the next $A$. Disadvantages of this method are still have chance to $A$ got stuck because of $R$ execution fails. Other $R$ in $A$ with fail $R$ still have to wait for fail $R$ to reach maximum execution time.

- **Backend idle distribution.** With this method, $R$ will be queued in the load balancer. $R$ will be distributed to idle $A$. Load balancer will check $A$ state and determine which $A$ is in idle state. If load balancer found idle $A$, $R$ will be distributed to this $A$. Advantages of this method is if there is a fail in executing $R$, it only affects to one $A$. Other $R$ do not have to wait for fail $R$ to reach maximum execution time because load balancer can distribute other $R$ to idle $A$. Disadvantages of this method is slower to distribute $R$ because it must check each $A$ to make sure $A$ is in idle state.

We have conducted a preliminary test for applying distributed architecture by using backend idle distribution and test a correct code to determine oddly and even digit. In this test, we compare both centralized and distributed architecture. In addition, the result show increased performance using distributed architecture. Still, need further testing to test overall performance in a distributed architecture.

4. Conclusion

Applying distributed architecture to autograder can reduce the risk of a system failure while grading code assignment. This architecture support autograder to use multiple backends so theoretically it will increase performance. Autograder can distribute code assignment request to multiple backends. If the
request fails, only one backend will be affected and another backend can continue their work. We conducted a preliminary test and show increased performance compared with centralized architecture. For test overall performance, need further testing with the various test case.

Acknowledgment
The Authors acknowledge Sekolah Tinggi Teknologi Garut for supporting and funding this paper.

References
[1] Liu X, Wang S, Wang P and Wu D 2019 Automatic Grading of Programming Assignments: An Approach Based on Formal Semantics Proceedings of the 41st ACM/IEEE International Conference on Software Engineering (ICSE 2019), the Software Engineering Education and Training (SEET) track
[2] Danutama K and Liem I 2013 Scalable Autograder and LMS Integration Procedia Technology 11 388-395
[3] Syaifudin Y 2019 yusufsyaiduridin/grader-library [Online] Retrieved from: https://github.com/yusufsyaiduridin/grader-library Accessed on 2019 April 11
[4] skuhl/autograder 2019 [Online] Retrieved from: https://github.com/skuhl/autograder 2019 April 11
[5] BilalZaib/AutoGrader 2019 [Online] Retrieved from: https://github.com/BilalZaib/AutoGrader Accessed on 2019 Apr 11
[6] AgileVentures/AutoGraderExamples 2019 [Online] Retrieved from: https://github.com/AgileVentures/AutoGraderExamples Accessed on 2019 Apr 11
[7] Black Box Testing 2018 Software Testing Fundamentals [Online] Retrieved from: http://softwaretestingfundamentals.com/black-box-testing/ Accessed on 2019 May 1
[8] Sneha K and Malle G M 2017 Research on Software Testing Techniques and Software Automation Testing Tools 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)
[9] White Box Testing 2018 Software Testing Fundamentals [Online] Retrieved from: http://softwaretestingfundamentals.com/white-box-testing/ Accessed on 2019 May 1
[10] Matthews K, Janicki T, He L and Patterson L 2012 Implementation of an Automated Grading System with an Adaptive Learning Component to Affect Student Feedback and Response Time Journal of Information Systems Education (JISE) 23 1 71-148
[11] Mustafa K M, Al-Qutaish R E and Muhairat M I 2009 Classification of Software Testing Tools Based on the Software Testing Methods 2009 Second International Conference on Computer and Electrical Engineering
[12] Yulianto S and Liem I 2014 Automatic Grader For Programming Assignment Using Source Code Analyzer International Conference on Data and Software Engineering (ICODSE)
[13] Li H and Wu Z 2009 Research on Distributed Architecture Based on SOA International Conference on Communication Software and Networks
[14] Kaur S and Sharma T 2018 Efficient load balancing using improved central load balancing technique 2nd International Conference on Inventive Systems and Control (ICISC)