Cloud-based Verification IDE

Bachelor Thesis Project Description

Dennis Buitendijk

Supervised by Prof. Dr. Peter Müller, Linard Arquint, João Perreira
Department of Computer Science, ETH Zurich

March 30, 2021

1 Introduction

Bugs in systems can have a significant negative impact, such as system crashes or loss of data. These problems can lead to losses in revenue or users switching to a competitor's product. Because of that, testing techniques have become a common practice in software engineering. However, testing can only show the existence of bugs by executing the software with a specific input. Hence, testing is ill-suited to prove the absence of bugs. To do so, every possible combination of inputs along with every interleaving of the program would have to be explored under testing, which is only feasible for very simple programs. To prove the absence of bugs, companies are adopting formal verification of programs. In formal verification, mathematical proofs are constructed to ensure safety and functional properties.

For this reason, the Programming Methodology Group at ETH Zürich has developed Viper[4], an intermediate verification language with associated verification tools to prove safety and functional properties on Viper programs. Viper allows users to specify methods with pre- and post-conditions, which specify valid states when the method can be called and the expected state when the method returns, respectively. This enables Viper to modularly verify that (1) every method call is performed in a state satisfying the callee's precondition and (2) executing a method in every state satisfying the precondition results in a state satisfying the postcondition.

Viper is an intermediate verification language. As such, Viper can be used to verify programs written in mainstream programming languages, as long as they can be encoded in Viper. For this reason multiple front-ends have been developed. For example, Gobra is a Viper front-end for the Go programming language. It takes as input a Go program annotated with specifications. Gobra encodes the original Go program into a Viper program. If the generated Viper program verifies, verification success is reported. Otherwise, Gobra reports the lines of code that cause the verification failure. In addition, extensions for Microsoft Visual Studio Code (VS Code)[5] are available for several front-ends that improve usability. They offer the ability to verify programs directly from the IDE and highlight the lines that cause verification to fail.

2 Motivation

Formal verification tools typically demand an extra effort from the users to set up their programming environment and to write the necessary specification and proof annotations in their programs. This can lead to potential users losing interest in the tools. As such, we believe that providing a ready to use environment to try out verification tools significantly lowers their barrier of entry. This has been partially achieved by providing VS Code extensions, such as Viper-IDE[3] and Gobra-IDE[6] for Viper and Go respectively, which provide feedback to the users directly in the code editor.

Although the VS Code extensions are already a huge improvement in terms of usability over a command line interface, they still require users to install VS Code and the corresponding extensions and dependencies. Potential users might not be willing to carry out the necessary installation efforts or they might have an unsupported environment, such as an old Java installation or a 32-bit machine. To address these issues, we want to provide a web-based IDE that comes with all

tools and dependencies ready to verify programs, such that the required installation efforts drop to zero. Users simply open the webpage and can immediately try out the verifier.

Another shortcoming of the current solution is that programs are locally verified on users' machines. As these machines have limited computational capabilities, a verification might take several minutes for larger programs. The cloud infrastructure developed as part of this thesis aims to distribute the computational load across multiple servers. Deploying the system to powerful servers together with a good utilization of parallelism in the verification tools could result in verification times that are significantly lower than when verifying the same program locally.

3 Project Goals

3.1 Core Goals

1. Provide a web-based IDE, based on VS Code, that can be accessed through a modern browser. The goal is to create a container-based system which provides this functionality. The container-based system should be a generic base in the sense that it can be instantiated for various frontends by equipping the containers with the necessary VS Code extensions and dependencies. This container should have the following features:

   - As the service will be publicly available, isolation among users is very important. The computational power available to a single user should be limited such that a denial of service (DOS) attack has limited impact and does not block other users from accessing the service. This could be done by having one container per user or having a certain number of containers and spread the users among these containers. In the latter case, operating system user permissions would be used to separate users from each other.
   - A strict boundary between the server and the users has to be enforced such that the users can only interact with the system as intended, i.e. to write and verify programs written in Viper or in one of its front-ends, but have no way to misuse it. This could be achieved by giving users the least amount of operating system permissions and block access to the internet.
   - The solution should be able to handle an arbitrary number of users. This would require a load balancer to evenly spread the load among the servers. Two possible ways of solving this are depicted in section 4. Given that the host servers have limited storage available we do not want users to store files on the servers. Instead, we could store the files in cookies on the user side, which they send to the system when they connect.
   - The system should be accompanied by a large test-suit comprising unit-, regression-, and integration tests such that issues can be quickly identified.

2. Evaluate the performance of the system and analyse the cost of deploying the service to an external provider, such as AWS, Cloudflare or Google cloud, as opposed to hosting it on an ETH server.

3. Deploy the system to best suitable provider found in goal 2.

3.2 Extension Goals

1. Provide extra features for authenticated users. These features could include additional computational power, longer verification timeouts, and priority over unauthenticated users.

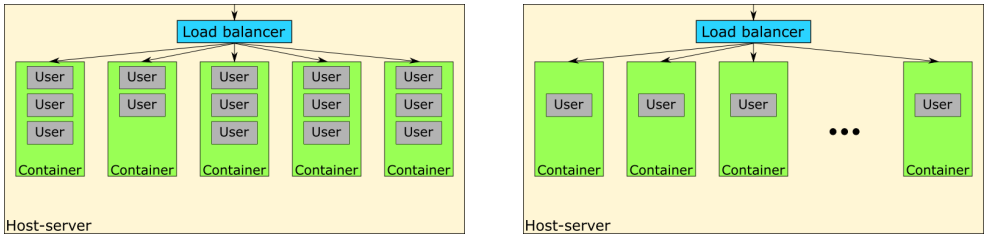
2. Share cache among verification jobs. Having verification results cached can increase performance significantly. To maximize the effect of caching, a single cache could be shared among all servers, containers, and users. However, users should still be isolated from each other. Thus, it has to be evaluated whether sharing the cache results in an acceptable weakening of the isolation.

3. Provide monitoring features and collect statistics to ease maintenance such as the status of the server, the number of accesses to the service, together with location and timing information, and how many accesses failed. If an issue with the system is identified, the maintainer of the system should be notified.

4. Collect verified programs for future analysis. Currently there is no data on what types of programs are verified. Creating a dataset of verified programs enables future research in the area and could lead to improvements in the verification process. Users should have the option of opting-out.

4 Possible Implementations

We plan to build our service on Code-Server[1]. Code-Server is an open source technologie that allows a user to access a VS Code instance through a browser. However, Code-Server is intended for trustworthy users (e.g. developers in a company) and thus requires adaptations to make it suitable for our applications. Code-Server is currently distributed in a binary form or as a Docker[2] image. For reproducability and ease of deployement we plan to build up on the Docker image with one of the following two architectures.



(a) Fixed number of containers (b) One container per user

Figure 1: Two possible architectures

Figure 1 shows two possible approaches for the system developed as part of this thesis. Figure 1a shows the approach where we have a fixed number of containers running on the server. When a new user connects to the system, the load balancer assigns the user to the container with the lowest load. This approach facilitates resource management as we can assign a static amount of resources (CPU time and memory) to each container and the load balancer, regardless of the amount of users each system has. However, the challenge with this approach is achieving isolation between users. Since multiple users share a container, they also share the filesystem of the container. A possible solution would be to give every user a subdirectory to which only that user has access permissions. Hence, no user can interact with files another user has created.

Alternatively, figure 1b shows an approach where every user has a dedicated container. When a new user joins the system, the load balancer creates a new container and assigns the user to it. In this approach we would have to limit the amount of resources each container would be able to use. If the system is getting overloaded the load balancer might have to reduce the allocated resources evenly among the containers. However, this approach solves the isolation between users as a user cannot access anything outside of their assigned container, and as such, cannot interact with any other user. This approach simply builds on the isolation provided by the underlying container manager, which is Docker in this project.

References

[1] *Code-Server*. URL: https://github.com/cdr/code-server.

[2] *Docker*. URL: https://www.docker.com/.

[3] Ruben Kälin. "Advanced Features for an Integrated Verification Environment". MA thesis. ETH Zürich, 2016.

[4] P. Müller, M. Schwerhoff, and A. J. Summers. "Viper: A Verification Infrastructure for Permission-Based Reasoning". In: *Verification, Model Checking, and Abstract Interpretation (VMCAI)*. Ed. by B. Jobstmann and K. R. M. Leino. Vol. 9583. LNCS. Springer-Verlag, 2016, pp. 41–62.

[5] *Visual Studio Code*. URL: https://code.visualstudio.com/.

[6] Silas Walker. "IDE Support for a Golang Verifier". Bachelor's Thesis. ETH Zürich, 2020.