Robust and Scalable Online Code Execution System

Herman Zvonimir Došilović
University of Zagreb
Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
hermanz.dosilovic@gmail.com

Igor Mekterović
University of Zagreb
Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
igor.mekterovic@fer.hr

***Abstract*—In this paper, we present a novel, robust, scalable, and open-source online code execution system called Judge0. It features a modern modular architecture that can be deployed over an arbitrary number of computers and operating systems. We study its design, comment on the various challenges that arise in building such systems, compare it with other available online code execution systems and online judge systems, and finally comment on several scenarios how it can be used to build a wide range of applications varying from competitive programming platforms, educational and recruitment platforms, to online code editors. Though first presented now, Judge0 is in active use since October 2017 and has become a crucial part of several production systems.**

***Index Terms*—online code execution system, online judge system, untrusted code execution**

I. Introduction

Programming skills have been recognized as a core competency skill [1] and the demand for the educational services in this field surpasses the supply. Educational organizations that teach programming can extend the classical human code assessment of students' assignments with an automated assessment that relies on online judges (OJs) [2]. Online judges are systems designed for evaluation of the user's source code on a predefined set of test cases. There are many different categories of online judges which can be classified by use-case varying from competitive programming platforms, e-learning systems, recruitment platforms, etc. Studying the architecture design of many different online judges surveyed in [2] we have identified that their mutual component is a code execution engine (CEE) that uses some sandboxing mechanism for secure execution of the user's source code. Extracting a code execution engine into a standalone component and providing a feature-rich web API for it we were able to create a robust, scalable, and configurable online code execution system (OCES) that can be integrated into many different types of online judges. In section II we explore and define components of the OJ ecosystem. After that in section III we define OCES through functional and non-functional requirements and in section IV we study technical challenges in building an OCES and present modern architecture and execution pipeline of Judge0. Finally, in section V we give a quick overview of related work done in the field of online code execution systems.

II. Online Judge Ecosystem

Figure 1 shows the layered architecture of an online judge ecosystem. This ecosystem includes components commonly associated with OJ system such as platforms built on top of online judges and online compilers, as well as less addressed components that online judges rely on - sandboxes and code execution systems. Decomposing an online judge into separated layers allows us to better understand and define functional and non-functional requirements of each layer. In the remainder of this chapter, we define and comment on each layer to clarify how these layers fit together and how an online code execution system and online judges, in general, are built.

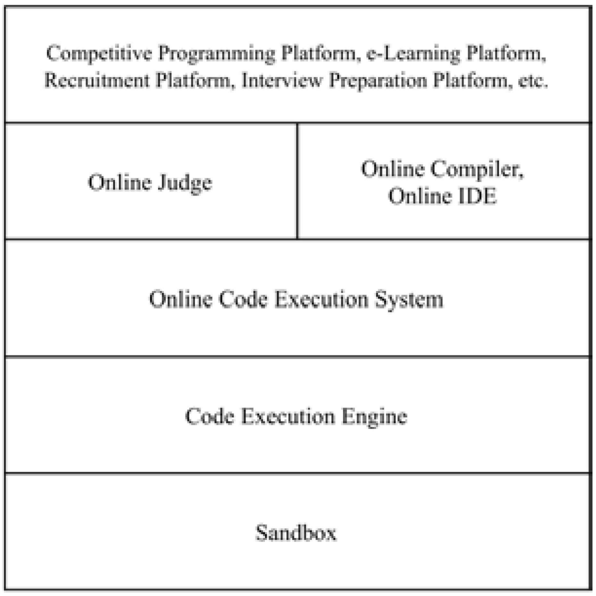
Competitive Programming Platform, e-Learning Platform, Recruitment Platform, Interview Preparation Platform, etc.
Online Judge
Online Compiler, Online IDE
Online Code Execution System
Code Execution Engine
Sandbox

Fig. 1: OJ ecosystem architecture.

A. Sandbox

Sandbox is a security mechanism for separating running programs, often used to execute untrusted source code [3]. In the context of online judges, sandboxes are used for security measures and to enforce limits on resources [4] like memory or CPU. In competitive programming, every challenge has defined time and memory limits which should not be exceeded. Thus, with good test cases and resource limits, authors of a challenge can check if the user solved a challenge with appropriate time and memory complexity. Many sandboxing techniques can be used for isolating untrusted source code with custom resource limitations such as LXC containers [5], Docker [6], virtual machines (VMs) [7] and others [3], [4]. In the context of online judges we expect that with a sandbox we can (1) initialize sandbox environment, (2) copy users code and other files into the sandbox, (3) compile and execute users code with custom resource limitations, (4) collect the output of a program and execution metadata, and (5) clean-up the sandbox environment.

B. *Code Execution Engine*

Code execution engine (CEE) is a component of an online judge that uses a sandbox to (1) compile and run arbitrary source code inside a sandbox and (2) to parse various metadata generated after compilation and execution. Building an online judge implies building a CEE, however, CEEs are rarely commented in the literature on online judges.

C. *Online Code Execution System*

Online code execution systems (OCESs) are (typically distributed) systems that provide a web API for compiling and executing arbitrary source code. They use CEE as a service for compiling and executing the given source code. Figure 2 shows the architecture of OCES. Up until now, OCESs have not received much attention in the literature in the context of online judges because online judges were developed for specific use-case for which OCESs were hidden inside the online judge architecture.

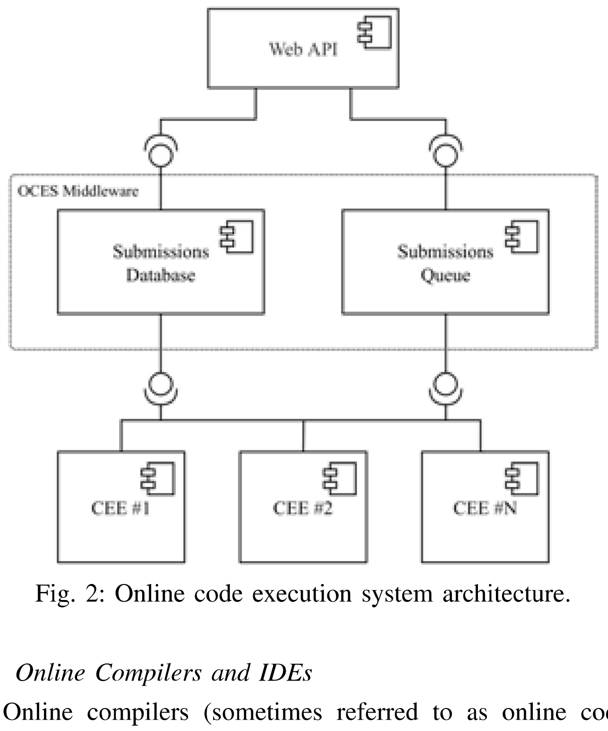

Fig. 2: Online code execution system architecture.

D. *Online Compilers and IDEs*

Online compilers (sometimes referred to as online code editors) like [8]–[11] are online platforms where users can write, compile and execute their source code in one or more programming languages. They usually allow users to write a single-file program, save their snippets, and share it with other users. Online IDEs like [12]–[14] are more advanced online platforms that allow users to develop bigger projects in a cloud. Some online platforms described in II-F have integrated online code editors so that users can write solutions in the browser without the need to install compiler and code editor on their computer. Detailed comparison of various online compilers and IDEs can be found in [2].

E. *Online Judges*

The definition of an online judge (OJ) given by [2] states that an online judge is an online service that performs either submission, assessment, or scoring in a cloud. In the submission phase, the user's code is compiled (if needed) and it is verified that the resulting binary can be successfully executed. In the assessment phase, the resulting binary is run on a given set of test cases and for each of the test cases it is verified that (1) the execution process completed without errors (2) resource limits have not been exceeded and (3) the obtained output complies to the rules described in the problem definition. In the last, scoring phase, the user's score is computed based on the results from the assessment phase. Following the above definition, figure 1 shows the architecture of an online judge and various online platforms that can be built on top of it. Online judges can be realized in many different ways that depend on the use-case, but the common functionality is problem management, assessment, and scoring of a user's solution.

F. *Online Platforms as Online Judges*

Different categories of online judges classified by use-case are described in [2]. Their common goal is the assessment of the user's solution of a given problem, but the user's score is subsequently used for different purposes. The most common types of online judges are competitive programming platforms such as [15]–[17] for organizing programming contests or for solving various programming challenges for practice. These challenges have well-defined input and output format and expect from users to implement the appropriate algorithm using appropriate data structures without exceeding given resource limitations such as time and memory limits. E-learning platforms can be also considered as online judges if they automatically assess students' solutions on a given programming challenge. They are often used in educational institutions and in the form of Massive Open Online Courses (MOOC) [2]. Interview preparation platforms such as [18]–[20] are another type of online judges that users can use to practice challenges, algorithms and data structures that appear in interview questions for software engineering positions. Recruitment platforms such as [20] are online judges that many companies use in their recruitment process to assess candidate skills with various programming challenges.

III. Online Code Execution System

In this paper, we propose that OCES should be distinguished from the OJs and treated a standalone entity - a "black-box" component that can be used to build various applications atop of, OJ being one of them. OCES can be defined through functional and non-functional requirements.

A. Functional Requirements

Online code execution system **must** provide:

- a well-documented web API,
- support for compiling and running arbitrary source code for at least one programming language,
- sandboxed execution, with the predefined default resource limits of compiled binaries,
- support for specifying arbitrary input that will be given as standard input to the program and
- support for returning execution results that include at least the content of the standard output.

Additionally, online code execution system **could** provide support for:

- specifying custom compiler flags,
- specifying custom command-line arguments,
- specifying resource limits such as CPU time limit and memory limit,
- initializing a sandbox environment with multiple files other than source code,
- returning detailed execution results such as CPU time and memory usage, exit signal and content of a standard error,
- returning compilation warnings or errors if they exist,
- batched submissions,
- multiple inputs and
- authenticated requests

Of course, the more functional requirements it satisfies the more use-cases it can cover.

B. Non-Functional Requirements

Online code execution system **should** be:

- scalable,
- configurable,
- secure,
- easily deployable,
- efficient,
- resilient,
- extendable and
- robust.

The more non-functional requirements OCES satisfies it is easier for it to be integrated for specific use-case platforms.

IV. Overview of Judge0

Judge0 is a new and open-source [21] online code execution system that features scalable architecture and a new approach in building online judges (discussed in V). Development of Judge0 started in August 2016 with the goal of creating a new, open-source, robust, scalable, easy to use, and extendable OCES that can be easily integrated into various web applications. Since the academic year 2018./2019. Judge0 has been used by more than 3000 students from the University of Zagreb Faculty of Electrical Engineering and Computing (FER) as an OCES integrated into e-learning platform called Edgar [22]. It has now become an indispensable tool for several courses dealing with a large number of students, today, and in the foreseeable future.

Since October 2017 Judge0 has been publicly [23] available as a free OCES and till June 2020 it has executed more than 8 million programs submitted from all around the world. Most of Judge0 users integrate it into the online judge for specific use-case like e-learning or interview preparation platform, while others integrate it into online compilers like Judge0 IDE [8].

A. Requirements and Technical Challenges

There were many technical challenges that were considered and overcome while developing Judge0 and we study each in the following sections.

1) Scalability: Scalability of Judge0 can be achieved using three strategies. The first one is by scaling a number of CEEs run on a single computer. The number of CEEs determines the number of parallel submissions that can be executed simultaneously. Empirical research shows that the maximum number of CEEs that can be run depends on the computer resources and compilers/interpreters used for running submissions. However, determining the optimal number of CEEs with regards to the computer resources and compiler/interpreter type is still an open research question for us. Current empirical results show that the good choice for the number of CEEs run on a single computer is the same as the number of logical CPU cores. This kind of scaling, by increasing the resources and number of CEEs on a single machine, corresponds to the vertical scaling and is sure to hit a ceiling at a certain point. That is where the second, horizontal scaling strategy can be employed. The second scaling strategy assumes that all components of OCES shown in figure 2 are run on the same computer. Replicating OCES over many computers and abstracting them behind a load balancer achieves horizontal scaling. Note that the second scaling strategy can be used in the combination with the first one, i.e. first determine and run the maximum number of CEEs and then horizontally scale to an arbitrary number of nodes. This type of scaling is used at FER and has shown good results having done more than 210,000 submissions of C and Java code for more than 2000 students in the academic year 2019./2020. alone. The third scaling strategy is more complex but uses computer resources more sparingly. Since Judge0 features modular architecture where each module can be easily deployed as a Docker container, the idea is to scale only parts of the system that might present a bottleneck. For instance, by horizontally scaling just CEEs with web API and middleware being run on the same or different computers. This type of scaling is used for the public instance of Judge0 [23] where web API and middleware are run on one server and CEEs are horizontally scaled (in addition to the first strategy) on two servers. This type of scaling with servers having just 4GB of RAM and 2 vCPUs was able to run, at its peak, around 120,000 submissions per day.

2) Security and Sandboxing: Because Judge0 allows users to run arbitrary source code and binaries on a remote server,

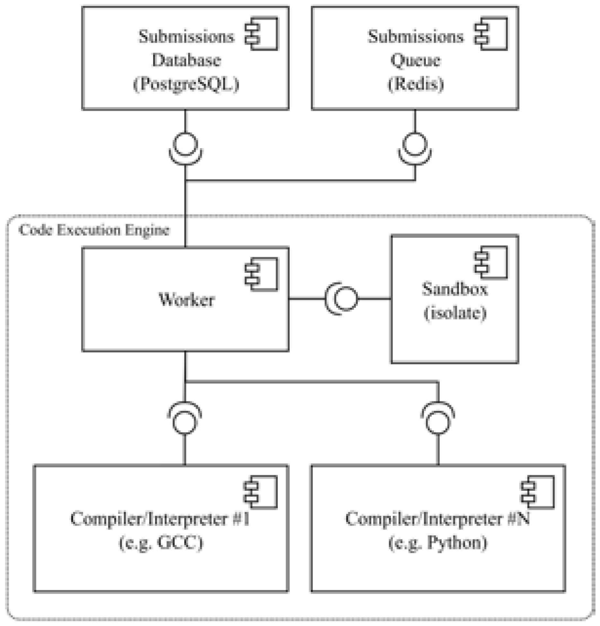
Submissions Database (PostgreSQL)
Submissions Queue (Redis)
Code Execution Engine
Worker
Sandbox (isolate)
Compiler/Interpreter #1 (e.g. GCC)
Compiler/Interpreter #N (e.g. Python)

Fig. 3: Code execution engine architecture.

the attack surface is big and source code or binary for every submission is considered **untrusted** i.e. it must be compiled and run in a sandboxed environment. CEE of Judge0 uses *isolate* [4] (figure 3) for sandboxing compilation and execution. *Isolate* is a well-known and proven sandbox mechanism for popular competitive programming online judge called *CMS* [24], [25] which has been used on large competitive programming contests such as International Olympiad in Informatics (IOI) and Central European Olympiad in Informatics (CEOI). Another aspect of security is connected with wide configuration options for execution such as, for example, setting a custom maximum CPU time limit for the submission. Judge0 features rich configuration options (mostly inherited from *isolate*) for limiting resources (e.g. CPU) by setting the upper and lower bounds which enables Judge0 to support many different use-cases. Note that both compilation and execution should be sandboxed and resource-limited. To better understand this let us examine two code examples given with listings 1 and 2. Listing 1 shows a valid program written in C programming language that includes Linux pseudo-random number device driver which generates an infinite sequence of secure bits [26], resulting in infinite compilation time.

```c
#include </dev/random>
int main() {
    return 0;
}
```

Listing 1: Valid C code with infinite compilation time.

The second example (listing 2) shows a denial of a service attack called *fork bomb* which generates many processes rapidly, thus exhausting the resources of the computer it is running on. As a result, any legitimate process can not start its tasks because the malicious processes are exhausting system resources [27].

```c
#include <unistd.h>
int main() {
    while(1) {
        fork();
    }
    return 0;
}
```

Listing 2: Fork bomb in C programming language.

By using *isolate* as its sandbox environment Judge0 can safely compile and execute arbitrary source code and even binaries that have come from an unknown source. Moreover, parsing execution metadata from *isolate* CEE of Judge0 can save and report the status of every submission and the potential failure reason (for e.g. because of time limit exceeded, memory limit exceeded and etc.).

3) Robustness: Judge0 is designed to withstand any kind of malicious attempts with the following principle in mind: (malicious) submissions may fail, but the system will continue to function intact. This robustness in Judge0 is concerned with two aspects: (1) robustness to recovering from internal errors on program execution and (2) robustness to sending and receiving arbitrary data. The first one, robustness to recovering from internal errors, is achieved with separation of concerns in the CEE component. Malicious or faulty code will most likely cause an internal error in on of the following steps: (i) initializing the sandbox environment, (ii) compiling and executing source code in the sandbox, and (iii) clean-up of the sandbox environment. If CEE catches an undefined or unexpected behavior in any of the steps it will mark that submission with the status ”Internal Error” accompanied with a status message - a reason for the internal error. The second aspect of robustness can be demonstrated with the program shown in listing 3 that outputs an invalid UTF-8 byte sequence. Even though this is a valid C program and it will run without any errors, the result of this program i.e. content of its standard output cannot be sent (serialized) as a raw sequence of bytes in a JSON format. Judge0 allows users to ask for the Base64 encoded data [28] in the JSON format. Respectively, Judge0 allows users to send Base64 encoded data in JSON format which allows them to avoid such problems.

```c
#include <stdio.h>
int main() {
    printf("\xFE");
    return 0;
}
```

Listing 3: Program that outputs content that cannot be sent in a raw format with JSON.

4) Ease of use and deployment: Judge0 features simple JSON web API that has two main endpoints:

- create a new submission for one of the available languages and

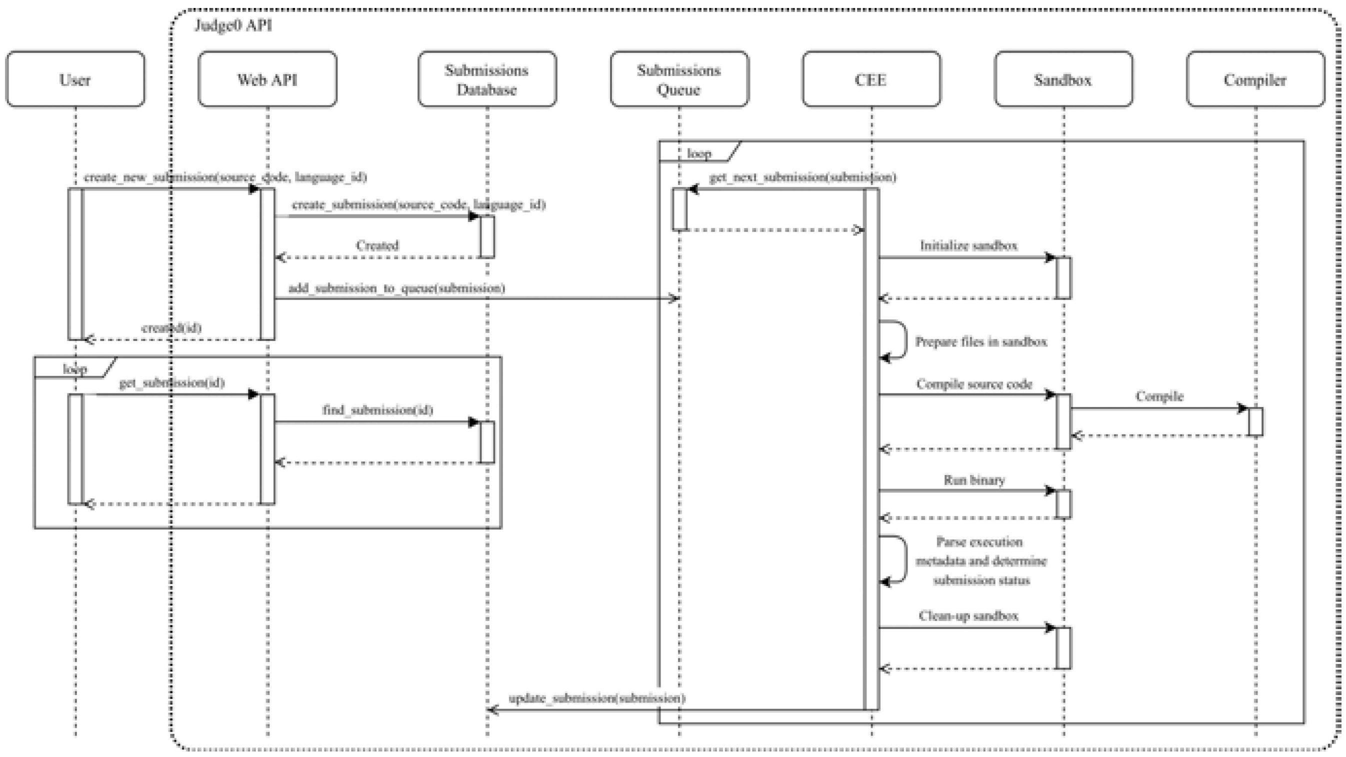
Judge0 API
User
Web API
Submissions Database
Submissions Queue
CEE
Sandbox
Compiler
loop
get_next_submission(submission)
create_new_submission(source_code, language_id)
create_submission(source_code, language_id)
Created
Initialize sandbox
add_submission_to_queue(submission)
created(id)
Prepare files in sandbox
loop
get_submission(id)
find_submission(id)
Compile source code
Compile
Run binary
Parse execution metadata and determine submission status
Clean-up sandbox
update_submission(submission)

Fig. 4: Judge0 code execution sequence diagram.

- fetch submission status.

The first endpoint creates a new submission that must contain (1) source code and (2) unique identifier of a compiler or interpreter that needs to be used for compiling/interpreting given source code. Current version v1.9.0 of Judge0 supports 46 different compilers and interpreters but is not limited to those out-of-the-box languages: adding a new language/-compiler boils down to installing the compiler on the CEE machine and inserting one record (describing the newly added language) into the database. With the second endpoint, the user can fetch submission results i.e. get detailed information about execution results such as the content of standard output and standard error, CPU time, memory usage, etc. With just these two endpoints one can build simple e-learning online judge or online compiler like Judge0 IDE or [11].

Ease of deployment is achieved with Docker [6] that allows users to deploy Judge0 on any platform that has Docker Engine installed. Furthermore, because Judge0 depends on external services such as database (PostgreSQL) and submission queue (Redis) (figure 3) we use Docker Compose [29] for connecting these services which can then all be installed and run with a single command.

B. Architecture and Execution Pipeline

Judge0 features scalable architecture design as seen in figures 2 and 3. In-depth interaction between actors of the system can be seen in figure 4. First, the user creates a request to web API for creating a new submission providing at least source code and desired language. Web API then creates a new submission record in the database and pushes newly created submission to the submission queue. After pushing it to the queue, web API returns the submission unique identifier to the user. With the returned unique identifier user can request for submission results. Meanwhile, CEE continuously tries to pull the next submission from the queue. Upon successful pull it will initialize sandbox, prepare files in the sandbox, compile source code, run compiled binary, parse metadata after the execution and determine submission status, clean-up the sandbox environment and finally update submission in the database. User is expected to frequently poll submission from the web API until it receives the completed submission. We characterize this type of interaction between the user and web API as *async execution* because the user does not know when the submission is complete and, must *asynchronously* pull the submission from the web API. Judge0 also supports another type of interaction - *sync execution*, where the user gets submission result as a response instead of the submission unique identifier. Current empirical results show that *async execution* scales better when the incoming number of submissions is much higher than the number of submissions that can be executed by CEEs in the same time window. On the other hand, sync execution is a simpler paradigm for the client programs: a single request-response is easier to implement.

V. Related Work

A detailed survey on online judge systems and their applications has been done in [2] but not much attention has been given to online code execution systems because online judges are typically developed for specific use-case for which OCESs are hidden inside the online judge architecture. In our research, we found two OCES that were developed with the competitive programming use-case in the main focus. The first one is *Sphere Engine* [30] which is a commercial and closed-source product that provides limited configuration options for each submission, whereas Judge0 exposes almost all sandboxing options *isolate* provides. Another OCES is *camisole* [31] that also uses *isolate* as a sandbox environment. It is open-source and free. However, they do not expose all configuration options that *isolate* provides, but only basic ones that are important for competitive programming. Deployment of *camisole* is done either by downloading a prepared VM image or by manual install. This type of deployment procedure is not easily scalable as the one we propose with Judge0. In conclusion, both of these systems are biased towards competitive programming while Judge0 presents an agnostic, general-purpose solution.

VI. Conclusion

With the rise of demand in computer science education, development of technology and increased connectedness, expansion of distance learning, and online courses the need for online code execution and assessment is considerable and growing. Online judges are typically built with specific use-case in mind such as for competitive programming, e-learning, recruitment, etc. and they typically have an online code execution system hidden inside their architecture which allows them to run users code in a secure and isolated environment. In such architectures, online code execution systems are strongly coupled and developed for a specific use-case. In this paper, we presented a novel online code execution system architecture defined through functional and non-functional requirements and designed as a standalone and multi-purpose entity in the architecture of online judges. Finally, the proposed system is implemented and presented here as Judge0, a generic, robust, configurable, and scalable online code execution system with feature-rich web API that can be used for many different use-cases. It is thoroughly tested and heavily used in many production systems and is available to the general public as open-source software.

References

[1] R. S. Lindberg, T. H. Laine, and L. Haaranen, "Gamifying programming education in k-12: A review of programming curricula in seven countries and programming games," *British Journal of Educational Technology*, vol. 50, no. 4, pp. 1979–1995, 2019.

[2] S. Wasik, M. Antczak, J. Badura, A. Laskowski, and T. Sternal, "A survey on online judge systems and their applications," *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, pp. 1–34, 2018.

[3] C. Yi, S. Feng, and Z. Gong, "A comparison of sandbox technologies used in online judge systems," in *Applied Mechanics and Materials*, vol. 490. Trans Tech Publ, 2014, pp. 1201–1204.

[4] M. Mareš and B. Blackham, "A new contest sandbox." *Olympiads in Informatics*, vol. 6, 2012.

[5] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2015, pp. 171–172.

[6] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[7] J. E. Smith and R. Nair, "The architecture of virtual machines," *Computer*, vol. 38, no. 5, pp. 32–38, 2005.

[8] H. Z. Došilović, "Judge0 IDE - Free and open-source online code editor," accessed 2020-23-02. [Online]. Available: https://ide.judge0.com

[9] S. R. Labs, "Online Compiler and IDE ¿¿ C/C++, Java, PHP, Python, Perl and 70+ other compilers and interpreters - Ideone.com," accessed 2020-23-02. [Online]. Available: https://ideone.com

[10] melpon, "Wandbox - Social Compilation Service," accessed 2020-23-02. [Online]. Available: https://wandbox.org

[11] H. Z. Došilović, "Online compiler written in less than 200 lines of code." accessed 2020-06-30. [Online]. Available: https://online-compiler.dosilovic.com

[12] Codeanywhere, "Codeanywhere · Cross Platform Cloud IDE," accessed 2020-23-02. [Online]. Available: https://www.codeanywhere.com

[13] CodeSandbox, "CodeSandbox: Online IDE for Rapid Web Development," accessed 2020-23-02. [Online]. Available: https://codesandbox.io

[14] Repl.it, "Repl.it - The world's leading online coding platform," accessed 2020-23-02. [Online]. Available: https://repl.it

[15] M. Mirzayanov, "CodeForces," accessed 2020-23-02. [Online]. Available: https://codeforces.com

[16] S. R. Labs, "Sphere Online Judge (SPOJ)," accessed 2020-23-02. [Online]. Available: https://spoj.com

[17] Directi, "CodeChef - A Platform for Aspiring Programmers," accessed 2020-23-02. [Online]. Available: https://codechef.com

[18] AlgoExpert, "AlgoExpert | Ace the Coding Interviews," accessed 2020-23-02. [Online]. Available: https://www.algoexpert.io

[19] LeetCode, "LeetCode - The World's Leading Online Programming Learning Platform," accessed 2020-23-02. [Online]. Available: https://leetcode.com

[20] HackerRank, "HackerRank - Matching developers with great companies," accessed 2020-23-02. [Online]. Available: https://hackerrank.com

[21] GitHub, "GitHub - Judge0 source code," accessed 2020-26-02. [Online]. Available: https://github.com/judge0/api

[22] I. Mekterović, L. Brkić, B. Milašinović, and M. Baranović, "Building a comprehensive automated programming assessment system," *IEEE Access*, vol. 8, pp. 81 154–81 172, 2020.

[23] H. Z. Došilović, "Judge0 - Robust and scalable open-source online code execution system," accessed 2020-26-02. [Online]. Available: https://api.judge0.com

[24] S. Maggiolo and G. Mascellani, "Introducing cms: A contest management system." *Olympiads in Informatics*, vol. 6, 2012.

[25] S. MAGGIOLO, G. MASCELLANI, and L. WEHRSTEDT, "Cms: a growing grading system." *Olympiads in Informatics*, vol. 8, 2014.

[26] Z. Gutterman, B. Pinkas, and T. Reinman, "Analysis of the linux random number generator," in *2006 IEEE Symposium on Security and Privacy (S&P'06)*. IEEE, 2006, pp. 15–pp.

[27] G. Nakagawa and S. Oikawa, "Fork bomb attack mitigation by process resource quarantine," in *2016 Fourth International Symposium on Computing and Networking (CANDAR)*. IEEE, 2016, pp. 691–695.

[28] S. Josefsson *et al.*, "The base16, base32, and base64 data encodings," RFC 4648, October, Tech. Rep., 2006.

[29] Docker, "Overview of Docker Compose," accessed 2020-26-02. [Online]. Available: https://docs.docker.com/compose

[30] S. R. Labs, "Online Compilers and Programming Challenges APIs - Sphere Engine," accessed 2020-27-02. [Online]. Available: https://sphere-engine.com

[31] A. Prologin, "camisole," accessed 2020-27-02. [Online]. Available: https://camisole.prologin.org