Challenges in Impleneting client side compilation using WebAssembly

VJS PRANAVASRI, International Institute of Information Technology, India

WebAssembly compilers are tools that allow users to compile their code into the WebAssembly format. This format can be run on web browsers and other platforms that support the WebAssembly standard. We look at practical application of a webassembly compiler running on client side and study the challenges that arise in doing so. We present a prototype implementation of a webassembly compiler that runs on client side and discuss the challenges that we faced in implementing it. We also present a solution to the challenges that we faced.

1 INTRODUCTION

WebAssembly (Wasm)[5] is a web standard that defines a binary format and a corresponding assembly-like text format for executable code in web pages. The Wasm text format is designed as a portable, size-efficient, and embeddable format that runs with near-native performance and provides languages with low-level memory models such as C++ and Rust with a compilation target so that they can run on the web. A webassembly compiler is a tool that converts a code written in a high-level language like C++ or Rust into webassembly code. This code can be run on a web browser or on a web server.

The main focus of this paper would be a study on such web compilers which themselves are written in webassembly. We first start with looking at the available solutions for the same. We then go on with pyodide as a base for our implementation. We then discuss the challenges that we faced in implementing a webassembly compiler on client side. We will broadly try to validate the study on WebAssembly compiler bugs[9]

We will use Virtual Labs as a platform, where we test out our implementation for validaing our study.

We ask the following Research Questions to guide our study:

- **RQ1:** How will the implementation of a webassembly compiler help in the development of web applications?
- **RQ2:** What are the challenges in implementing a webassembly compiler on client side?
- **RQ3:** How big of an impact do bugs in WebAssembly compilers have on the implementation requirement of Virtual Labs?

Author's address: VJS Pranavasri, vjs.pranavasri@research.iiit.ac.in, International Institute of Information Technology, Professor CR Rao Road, Gachibowli, Hyderabad, Telangana, India, 500032.

Unpublished working draft. Not for distribution.

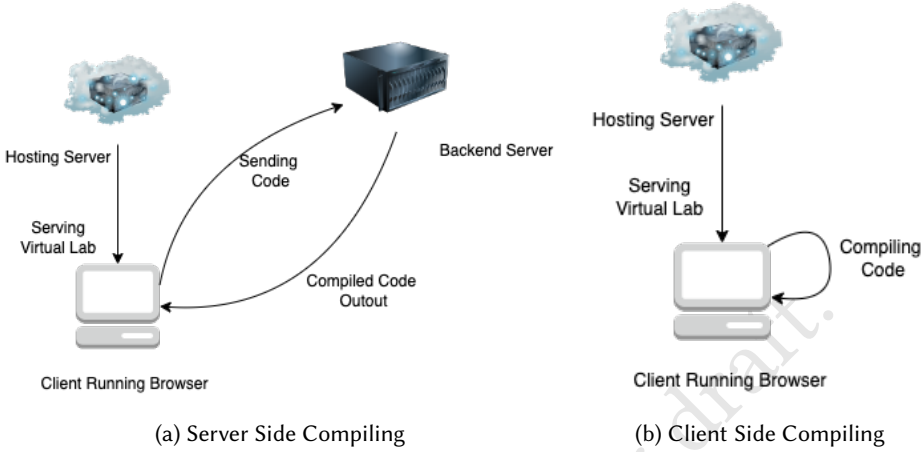(a) Server Side Compiling   (b) Client Side Compiling

Fig. 1. Server Side Model vs Client Side Model

2 VIRTUAL LABS

Virtual Labs is an initiative by the Ministry of Human Resource Development (MHRD), Government of India, under the aegis of the National Mission on Education through Information and Communication Technology (NMEICT). This is coordinated by 12 participating institutes of which IIIT Hyderabad is a part. The main goal is to make any lab or education available independently of the platform. Virtual Labs mainly help professors and students who do not have proper lab facilities available by virtually emulating the scenario.

One of the current challenges faced by Virtual Labs is the ability to run code. This is mainly because of the requirement of a backend server to run the code. As the number of students using Virtual Labs increases, the load on the backend server increases. This is mainly because of the fact that the code will be run on the server and the results are sent back to the client. This would take a toll on the resources of the server.

The solution to this problem is to run the code on the client side. This would eliminate the need for a backend server, and possibly also allow interacting with the lab without any network after the first time.

For the scope of this paper, we would be working on a Python based Virtual Lab and we will be using pyodide as a base for our study and implementation.

3 RELATED WORK

There is a lot of active study happening in the area of webassembly compilers, however we specifically focus on the ones that themseleves are written in webassembly that allows for client side compilations of other high/low level languages.

WebAssembly

We take a lot of motivation from the paper that initially introduced webassembly to the world[5]. The paper explains the gap in web devlopment which cannot inherenly be solved by using javascript. This in depth describes major components WebAssembly as a language and a tool. One of the main goal being able to run cross language codes.

Study on WebAssembly

With the introduction of the new byte code format, a lot of analysis in terms of performance and security has been done. The paper [12] dicusses about the mechanisms of webassembly and how they verify the correctness of WASM. Then there ar multiple studies that go on to talk about performance of webassembly applications.

The paper [14] talks about how performance of web assembly based applications are and how different parameters like JIT(Just In Time) compilation and other factors affect the performance. Further studies focus on performance comparisions and energy consumption of webassembly applications compared to native applications[2]. An intersting study was one that used WebAssembly to study the memory model of javascript and propose changes to improve consistency[13].

A final research area is mainly geared towards security. The paper *Everything Old is New Again*[6] talks about binary security of web assembly, as we are again now moving towards assembly format for the web. We see a paper that talks about the security in place for web assembly compilers[11]. We would briefly refer to this to talk about security concerns that we find, and finally use the paper that does a performance analysis of webassembly compilers[8] as basis for our performance analysis.

WebAssembly Applications & Compilers

We look at some existing applications and compiler implementations to gain a better understanding. One such is an implementation of the e-commerce website ebay using WASM[7]. Then we look at an implementation of tensorflow backend which has been ported to web through WASM[10], this allows for running models on web through client. Before heading further we look at one of the most popular compilers that is written for C, emscripten[15]. Emscripten is a compiler that compiles C/C++ to WASM. It is one of the most popular compilers for WASM.

Bugs in WebAssembly Compilers

As we are focused more on the compilers that compile to WebAssembly, we use this paper as a base for our study[9]. This paper discusses the bugs that are present in the WebAssembly compilers. We use this to compare and study its effects the implementation of Virtual Labs.

We see that there isn't that major of work being done in fields of web assembly compilers, and nothing specific to running them on client side. This would vouch for the novelty of the paper and we use the remotely similar work as a base for our study. For this paper we will be specifically focussing on pyodide, a python distribution that runs on browser[1]. We will be using this as a base for our study and implementation. The source code for the same is available on github.[1]

4 METHODOLOGY

Initially we do a collection of bugs from Pyodide repo. We then go ahead to analyse these bugs and try to compare them to the bugs proosed by Romano et al. [9].

4.1 Bugs Collection

Our first goal is data collection. We follow the methods proposed followed by [9] and use Github Search API[3] and Github REST API[4] to collect all issues and pull requests. The project has a total of 880 Closed and 277 Open issues at the time of collection. For our use case, we filter out all the issues with label bug. This brings the total issues down tp 149 (closed) + 43 (open). We go through a detailed analysis of each issue and seggregate them based on the categories as shown in table 1. For the scope of the research course, we look at only the closed issues and filter out all

---

[1]https://github.com/pyodide/pyodide

Category	Number of Unique Issues
Asyncify Synchronous Code	1
Incompatible Data Type	3
Memory Model Differences	7
Other Infrastructure Bug	12
Emulating Native Environment	1
Supporting Web API's	6
Pyodide Specific	21

Table 1. Categories of issues taken from [9]

duplicate bugs, won't-fix bugs and any bugs in the build time of pyodide. We majorly look at issues that are posed which are more focused with respect to pyodide as a framework itself or one of it's dependency CPython, emscripten or WebAssembly amongst others[2].

4.2 Analysis

5 IMPLEMENTATION

6 DISCUSSION

7 CONCLUSION

REFERENCES

[1] Pyodide contributors. 2021. Version 0.21.3. https://pyodide.org/en/stable/

[2] João De Macedo, Rui Abreu, Rui Pereira, and João Saraiva. 2022. WebAssembly versus JavaScript: Energy and Runtime Performance. In *2022 International Conference on ICT for Sustainability (ICT4S)*. 24–34. https://doi.org/10.1109/ICT4S55073.2022.00014

[3] Search Github Docs. 2020. Github search api. https://docs.github.com/en/rest/reference/search

[4] Github. 2019. Github REST API. https://developer.github.com/v3/

[5] Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. 2017. Bringing the Web up to Speed with WebAssembly. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Barcelona, Spain) *(PLDI 2017)*. Association for Computing Machinery, New York, NY, USA, 185–200. https://doi.org/10.1145/3062341.3062363

[6] Daniel Lehmann, Johannes Kinder, and Michael Pradel. 2020. Everything Old is New Again: Binary Security of WebAssembly. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 217–234. https://www.usenix.org/conference/usenixsecurity20/presentation/lehmann

[7] Senthil Padmanabhan and Pranav Jha. 2020. Webassembly at ebay: A real-world use case.

[8] Micha Reiser and Luc Bläser. 2017. Accelerate JavaScript applications by cross-compiling to WebAssembly. 10–17. https://doi.org/10.1145/3141871.3141873

[9] Alan Romano, Xinyue Liu, Yonghwi Kwon, and Weihang Wang. 2021. An Empirical Study of Bugs in WebAssembly Compilers. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, Piscataway, NJ, USA, 42–54. https://doi.org/10.1109/ASE51524.2021.9678776

[10] Daniel Smilkov, Nikhil Thorat, and Ann Yuan. 2020. Introducing the WebAssembly backend for TensorFlow. js. *Retrieved February* 3 (2020), 2022.

[11] Quentin Stiévenart, Coen De Roover, and Mohammad Ghafari. 2021. The Security Risk of Lacking Compiler Protection in WebAssembly. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*. 132–139. https://doi.org/10.1109/QRS54544.2021.00024

[12] Conrad Watt. 2018. Mechanising and Verifying the WebAssembly Specification. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs* (Los Angeles, CA, USA) *(CPP 2018)*. Association for Computing Machinery, New York, NY, USA, 53–65. https://doi.org/10.1145/3167082

[13] Conrad Watt, Andreas Rossberg, and Jean Pichon-Pharabod. 2019. Weakening WebAssembly. *Proc. ACM Program. Lang.* 3, OOPSLA, Article 133 (oct 2019), 28 pages. https://doi.org/10.1145/3360559

---

[2]The complete list of bugs is avaialble on sheets https://stagbin.tk/pyodide_issues

[14] Yutian Yan, Tengfei Tu, Lijian Zhao, Yuchen Zhou, and Weihang Wang. 2021. Understanding the Performance of Webassembly Applications. In *Proceedings of the 21st ACM Internet Measurement Conference* (Virtual Event) *(IMC '21)*. Association for Computing Machinery, New York, NY, USA, 533–549. https://doi.org/10.1145/3487552.3487827

[15] Alon Zakai. 2011. Emscripten: an LLVM-to-JavaScript compiler. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion.* 301–312.