

**WEBASSEMBLY AND V8 ISOLATE-BASED
APPROACH FOR SERVERLESS COMPUTING
ARCHITECTURE**

Asanka Lakmal Basnayake

209314P

Master of Science in Computer Science

Department of Computer Science and Engineering

Faculty of Engineering

University of Moratuwa

Sri Lanka

July 2023

WEBASSEMBLY AND V8 ISOLATE-BASED APPROACH FOR SERVERLESS COMPUTING ARCHITECTURE

Asanka Lakmal Basnayake

209314P

Thesis submitted in partial fulfillment of the requirements for the degree
Master of Science in Computer Science

Department of Computer Science and Engineering

Faculty of Engineering

University of Moratuwa

Sri Lanka

July 2023

DECLARATION

I declare that this is my own work and this thesis/dissertation does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other University or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text. I retain the right to use this content in whole or part in future works (such as articles or books)

Name: B.M.A.L.Basnayake

Signature:

Date: 2023/07/27

The supervisor/s should certify the thesis/dissertation with the following declaration. The above candidate has carried out research for the Masters thesis under my supervision. I confirm that the declaration made above by the student is true and correct.

Name of the Supervisor: Prof. Indika Perera

Signature of the supervisor:

Date: 27/07/2023

ACKNOWLEDGEMENT

First and foremost, I would like to express my heartfelt gratitude to my supervisor, Prof. Indika Perera, for providing abundant guidance, support, and encouragement throughout this research. Also, I would like to thank my colleagues for sharing knowledge, providing support, and providing constant encouragement. Also, I would like to thank tech communities and tech articles for sharing knowledge, providing support, and providing constant encouragement.

ABSTRACT

User have the flexibility to execute short-lived applications with the help of serverless computing frameworks without concern about load balancing by scaling zero to large-scale traffic and further without considering configuring platform and runtime software, etc. Stateless containers are used in existing serverless platforms to isolate functions by preventing memory sharing directly with other serverless functions. Duplicating and serialising data repeatedly per new user request hit on the server by invoking VMs and containers leads to high performance and resource usage. The necessity of a light-weighted isolation approach is required to overcome these issues. We introduce WebAssembly and Chrome V8 isolate based, a new isolation mechanism for serverless computing. This isolates the memory of each function using linear memory with software fault isolation (SFI), which is a portable low-level bytecode called WebAssembly (which major browser vendors have collaboratively designed). The proposed runtime for serverless functions uses Chrome V8 Isolate-based isolation and virtualizes file systems with a similar approach to POSIX host interfaces using WebAssembly System Interface (WASI). WebAssembly (Wasm) and Chrome V8-based software fault isolation have been proposed in the study as an alternative method for conventional container-based serverless approaches, with near-native speed, a small memory footprint compared to containers, and optimised invocation time. WebAssembly-based isolation is compared against the container-based approach to validate its applicability within the study.

Keywords: WebAssembly, Serverless, Containers, V8

TABLE OF CONTENT

DECLARATION	i
ACKNOWLEDGEMENT	ii
ABSTRACT	iii
LIST OF TABLES	vii
LIST OF ABBREVIATIONS	viii
LIST OF APPENDICES	ix
CHAPTER 1 - INTRODUCTION	1
1.1. Serverless computing	1
1.2 WebAssembly	2
1.3 Research problem	2
1.4 Research objectives	3
CHAPTER 2 - LITERATURE REVIEW	4
2.1 AWS lambda	5
2.2 Serverless challenges due to containers	6
2.3 Serverless open-source implementations	6
2.4 WebAssembly	6
2.5 Compiler and toolchain components	7
2.6 Jailing	8
2.7. Network virtualization	9
2.8. Warm-up containers to reduce cold starts	10
2.9. Stateful serverless approach	11
2.10 Solutions exist	12
2.11 Gaps identified	13
CHAPTER 3 - METHODOLOGY	15
3.1 Introduction	15
3.2 Strong isolation.	16
Module	17
3.3 Embedding and interoperability	17
3.4 Runtime	17
3.5 Runtime limitations	18
3.8 Resource provisioning.	18
3.10 Implementation architecture	19

3.10.1 Isolated-vm	20
3.10.2 Isolation via worker threads	20
3.11 Isolation	21
3.11.1 Filesystem isolation	21
3.11.2 Memory Isolation	21
3.11.3 Network isolation	21
3. 12 Scalability	22
CHAPTER 4 - IMPLEMENTATION	23
4.1 WAFUZ runtime architecture	24
4.2 Scalability and load balancing	24
4.3 Compilation of C++ code inside the code builder section	25
4.4 Generating WebAssembly code out of the Rust codes	25
4.5 Code compilation steps;	25
4.6 Network/API call implementation	26
4.7 Communication between host memory and WebAssembly memory	26
4.8 Disk sandboxing/isolation	27
4.9 Network request handling	28
4.10 Possible vulnerabilities	28
CHAPTER 5 - EVALUATION	29
5.1 Setup container-based serverless functions	29
5.2 Resource and response time evaluation using jMeter	30
5.3 Serverless function's response time evaluation	31
5.4 API HTTP comparison	33
5.5 Memory usage	34
5.6 Response time comparison with WebAssembly-based solutions	37
5.7 Evaluation in a simulated distributed environment	38
CHAPTER 6 - CONCLUSION	39
6.1 Future work	39
REFERENCES	41
[Appendix I: WAFUZ Network Function Implementation]	46
[Appendix II: Openwhisk User Interface]	47

LIST OF FIGURES

Figure	Description	Page
Fig. 1.1	Serverless platform architecture	2
Fig. 2.1	Virtualization techniques.	4
Fig. 2.2	KVM network I/O architecture	9
Fig. 2.3	Auto scaling with warm-up containers architecture	10
Fig. 2.4	Shredder isolation architecture	11
Fig. 2.5	Shredder in-memory state architecture	12
Fig. 2.6	Serverless Functions execution architecture for the Edge	12
Fig. 3.1	Isolation approach in proposed architecture	14
Fig. 3.2	WebAssembly code generation process	17
Fig. 3.3	Google V8 engine architecture	18
Fig. 3.4	Node.JS worker thread architecture	19
Fig. 4.1	AWS lambda-based web service architecture	21
Fig 4.2	WAFUZ architecture	22
Fig. 4.3	WASM linear memory and function table	25
Fig. 4.4	WAFUZ network request handling architecture	26
Fig. 5.1	Openwhisk serverless function deployment architecture	27
Fig. 5.2	Setup jMeter thread groups	28
Fig. 5.3	Open Lambda serverless startup time	29
Fig. 5.4	Average response time comparison	30
Fig. 5.5	Response time percentile	31
Fig. 5.6	Inside the V8	33
Fig. 5.7	Memory usage per thread in WebAssembly-based functions and container-based functions	34
Fig. 5.8	Response comparison with wasmtime-spine and WAFUZ	38

LIST OF TABLES

Table	Description	Page
Table 2.1	Isolation Techniques Undergone By Existing Serverless Architectures	8
Table 3.1	Node Js Based Isolation Libraries And Approaches For Software-Based Isolation Using Webassembly And V8 Isolates.	15
Table 5.1	Client Network Calls Inside A Container-Based And Webassembly-Based Environments	32
Table 5.2	Inspect Container Details	32

LIST OF ABBREVIATIONS

Abbreviation	Description
WASM	WebAssembly
API	Application Programming Interface
WASI	Webassembly System Interface
JS	JavaScript
OS	Operating System
POSIX	Portable Operating System Interface
IPC	Inter-Process Communication
BSD	Berkeley Software Distribution
NPM	Node Module Package
VM	Virtual Machine
SFI	Software Fault Isolation
MB	Megabyte
KB	Kilobyte
I/O	Input/Output

LIST OF APPENDICES

Appendix	Description	Page
Appendix – I	Wafuz Network Handler Function Implementation	46
Appendix – II	Openwhisk User Interface	47