Nebula

Comparing two waves of cloud compute

Marius Nilsen Kluften

Thesis submitted for the degree of

Master in Informatics: Programming and System Architecture

60 credits

Institute of Informatics

Faculty of mathematics and natural sciences

University of Oslo

2024

Nebula

Comparing two waves of cloud compute

Marius Nilsen Kluften

Nebula

https://duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

The ever increasing demand for cloud services has resulted in the expansion of energy-intensive data centers, the ICT industry accounts for about 1 % of global electricity use, highlighting a need for sustainable options in cloud computing architectures.

This project investigates WebAssembly, a technology originally intended for running in the browser, as a potential contender in the space of technologies to consider in cloud native applications. Leveraging the inherent efficiency, portability and lower startup times of WebAssembly modules, this thesis presents an approach that aligns with green energy principles, while maintaining performance and scalability, essential for cloud services.

Preliminary findings suggest that programs compiled to WebAssembly modules have reduced startup and runtimes, which hopefully leads to less energy consumption and offering a viable pathway towards a more sustainable cloud.

Acknowledgments

The idea for the topic for this thesis appeared in an episode of the podcast ”Rustacean station”. Matt Butcher, the CEO of Fermyon, told the story of his journey through the different waves of cloud computing, and why Fermyon decided to bet on WebAssembly for the next big wave of cloud compute.

The capabilities of WebAssembly running on the server, with the aid of the WebAssembly System Interface (WASI) project, caught my interest and started the snowball that ended up as the avalanche that is this thesis.

I'd like to thank Matt Butcher and the people over at Fermyon for inadvertently inspiring my topic.

Furthermore I'd like to thank my two supervisors Joachim T. Kristensen and Marcus Kirkedal Thomsen, whom I somehow managed to convinced to help guide me through such a cutting edge topic Their guidance and insight have been invaluable the past semesters.

Contents

Abstract **i**

Acknowledgements **ii**

Contents **iii**

List of Figures **v**

List of Tables **vi**

I Overview **1**

1 Introduction **3**

1.1 Motivation 4

1.2 Problem Statement 4

1.3 Outline 4

2 Background **5**

2.1 The Evolution of Cloud Computing 5

2.1.1 The First Wave: Virtualization 5

2.1.2 The Second Wave: Containerization 6

2.1.3 The Third Wave: WebAssembly modules 6

2.2 Cloud Computing: An quick summary 6

2.3 Serverless and Function-as-a-Service (FaaS) 7

2.4 Major vendors in Serverless 8

2.5 WebAssembly 8

II Project **11**

3 Approach **12**

4 Analysis **13**

5 Design **14**

6 Implementation **15**

6.1 Tech stack 15

III Results **16**

7 Evaulation **17**

8 Discussion **18**

9 Conclusion **19**

10 References **20**

List of Figures

List of Tables

Part I

Overview

If WASM+WASI existed in 2008, we wouldn't have needed to created Docker. That's how important it is. Webassembly on the server is the future of computing. A standardized system interface was the missing link. Let's hope WASI is up to the task!

—Solomon Hykes, *Founder of Docker*

Chapter 1

Introduction

In the digital age, cloud computing has emerged as a foundational technology in the technological landscape, driving innovation and increased efficiency across various sectors. Its growth over the past decade has not only transformed how consumers store, process, and access data, but also raised environmental concerns. The Information and Communication Technology (ICT) industry, with cloud computing at its core, accounts for an estimated 2.1% to 3.9% of global greenhouse gas emissions. Data centers, the backbone of cloud computing infrastructures are responsible for about 200 TWh/yr, or about 1% of the global electricity consumption, a figure projected to escalate, potentially reaching 15% to 30% of electricity consumption in some countries by 2030. (Freitag et al., 2021)

The sustainability of cloud computing is thus under scrutiny, and while some vendors strive to achieve a net-zero carbon footprint for their cloud computing services, many data centers still rely on electricity generated by fossil fuels, a leading contributor to climate change. (Mytton, D. 2021) This reality underscores an urgent need to explore alternative technologies that promise enhanced energy efficiency while meeting customers demands. In this vein, serverless computing has emerged as a compelling paradigm, offering scalability and flexibility by enabling functions to execute in response to requests, rather than having a server running all the time. However, the inherent startup latency associated with containerized serverless functions pose a challenge, particularly for on-demand applications.

This thesis proposes exploring WebAssembly with WebAssembly System Interface (WASI) as an innovative choice for deploying functions to the cloud, through developing a prototype Functions-as-a-Service platform named Nebula. This platform will run functions compiled to WebAssembly, originally designed for high-performance tasks in web browsers, which coupled with WASI, allows us to give WebAssembly programs access to the underlying system. This holds potential for a more efficient way to package and deploy functions, potentially reducing the startup latency and the overhead associated with traditional serverless platforms. WebAssembly and WASI offers a pathway where the demands of today is met, while reducing the carbon footprint for cloud applications.

1.1 Motivation

The environmental footprint of cloud computing, particularly the energy demands of data centers, is a pressing issue. As the digital landscape continues to evolve, the quest for sustainable solutions has never been more critical. This thesis is motivated by the need to reconcile the growing demand for cloud services with the pressing need for environmental sustainability. Through the lens of WebAssembly and WASI, this thesis aims to investigate innovative deployment methods that promise to reduce energy consumption without sacrificing performance, thereby contributing to the development of a more sustainable cloud computing ecosystem.

1.2 Problem Statement

This thesis focuses on two primary objectives:

1. Assessing the potential for optimizing cloud service deployments using WebAssembly, with an eye towards the trade-offs in energy consumption associated with different deployment strategies.
2. Evaluate the potential energy savings and potential performance gains by employing WebAssembly for function execution in cloud environments.

By addressing these objectives, this thesis seeks to shed light on the feasibility and implications of adopting WebAssembly and WASI for a more energy-efficient cloud computing world.

1.3 Outline

The thesis has five chapters; this introduction, a chapter that goes through the background for how cloud computing got to this point, a chapter dedicated to the process of building Nebula, a chapter for discussing the results from the experiments, and finally a chapter suggesting future works.

Chapter 2

Background

The evolution of cloud computing represents a transformative quest, driven by the relentless pursuit for efficiency, scalability and innovation. This chapter introduces the concept of cloud, then goes back in time to what can be deemed as "the dawn of cloud" and navigates the evolution through introducing the concept of "Three waves of cloud computing", coined by the WebAssembly community. (TODO: SRC, Matt B?) Some benefits and challenges to cloud are discussed, ultimately leading to the exploration of WebAssembly as a potential solution for driving down the carbon footprint of cloud services.

2.1 The Evolution of Cloud Computing

As cloud computing matured, its journey first unfolded in two significant waves, each marking a new age of innovation and addressing the dynamic needs and challenges of digital infrastructure. This evolution reflects the industry's efforts to optimize resource efficiency, reducing operational costs, and minimize environmental impacts.

2.1.1 The First Wave: Virtualization

The dawn of cloud computing can be traced back to the push for virtualization, a response to the costly and complex nature of managin traditional, on-premise data centers. During the mid-2000s, Amazon launched its subsidiary, Amazon Web Services (AWS), who in turn launched Amazon S3 in March 2006, followed by Elastic Compute Cloud (EC2) in August the same year.(Barr, J. 2006) With these services, AWS positioned itself as a pioneer in this space, marking a major turning point in application development and deployment, and popularized cloud computing. EC2, as an Infrastructure-as-a-Service (IaaS) platform, empowered developers to run virtual machines remotely. While similar services existed before 2006, with Amazon's existing large customer base helped them gain significant traction, and ushered in a the first era, or wave, of *cloud computing*.

2.1.2 The Second Wave: Containerization

As we entered the 2010s, the focus shifted from Virtual Machines to containers, largely due to the limitations of VMs in efficiency, resource utilization, and application deployment speed. Containers, being a lightweight alternative to VMs, designed to overcome these hurdles (Sharma, et al. 2016).

In contrast to VMs, which require installation of resource-intensive operating systems and minutes to start up, containers along with their required OS components, could start up in seconds. Typically managed by orchestration tools like Kubernetes, containers enabled applications to package alongside their required OS components, facilitating scalability in response to varying service loads. Consequently, an increasing number of companies have since established platform teams to build orchestrated developers platforms, thereby simplifying application development in Kubernetes clusters.

2.1.3 The Third Wave: WebAssembly modules

2.2 Cloud Computing: An quick summary

Cloud computing, more commonly known as "*the cloud*", refers to the delivery of computing services, such as storage, processing power, network, and software, served over the internet, instead of running on locally owned hardware (on-premise). For companies, this has proved to be a super power, where businesses can focus on deploying their own applications and services to their users without worrying about the underlying infrastructure.

Some benefits include:

- *Reduced total cost of ownership*: Cloud computing has enabled companies to take their computing needs to the next level. Startups who can't afford neither the cost or time required to build their own infrastructure, and larger companies that want to iterate faster and decrease their lead time from idea to production (Thomas, Dave. 2009).
- *Scalability* is one of the most significant benefits of cloud computing. As organizations expand, so do their customer needs and the complexity of their

application infrastructure. Each additional feature brings with it additional costs, highlighting the importance of efficient resource management to optimize hardware investments in a cloud computing environment (Thomas, Dave. 2008).

However, some challenges persist:

- *Cost management*: Managing the cost of cloud computing is an ongoing challenge, with Gartner predicting that through 2024, 60% of infrastructure and operations (I&O) leaders will encounter cloud costs that are higher than budgeted for (Rimol, M. 2021).
- *Energy consumption*: The challenge of energy usage in cloud computing is a significant concern, with data centers alone accounting for approximately 1% of the world's electricity consumption (Freitag et al. 2021). This staggering statistic highlights the need to explore strategies and measures to mitigate energy usage in data centers. By reducing energy consumption, not only can costs be reduced, but also the carbon footprint associated with running the cloud can be lessened. Decreasing energy usage in data centers is expected to yield cost savings and contribute to the overall sustainability goals by reducing the cloud's environmental impact.

2.3 Serverless and Function-as-a-Service (FaaS)

Building your own developer platform on top of Kubernetes, much like building your own infrastructure, also entails a significant cost. Often, developers wish to launch specialized smaller services, without having to grapple with complicated orchestration. This led to the emergence of the Serverless model. Despite its somewhat misleading name, serverless doesn't imply the absence of a server. Instead, it means that the responsibility of server management has shifted from the developer to a third party provider.

From the advancements of serverless, we get its subset, Functions-as-a-Service, or FaaS. Companies already in the cloud game decided to develop their own FaaS platforms to attract developers interested in just writing their functions and running them, and not worry about anything underneath.

2.4 Major vendors in Serverless

The concept of "the cloud" isn't owned by any single organization, but rather, through the collective effort of industry players including Amazon, Microsoft, Google, Alibaba and DigitalOcean, among others. This essay delves into some challenges faced by the biggest three vendors: Amazon, Google and Microsoft.

Amazon Web Services (AWS) provides AWS Lambdas, a technology that hinges on their proprietary Firecracker - a streamlined virtualization technology for executing functions. Interestingly, for this thesis, is that Amazon's Prime Video streaming service transitioned recently from a serverless architecture to a monolithic system to meet specific service demands. One might question whether this reflects the suitability of serverless systems for cloud computing, or for specific use cases like theirs (Kolny, M. 2023. Accessed 29.05.23). Some discussions suggest that their need to process videos frame by frame led to astronomical costs on their sibling company's FaaS, Amazon Lambda.

Google provides Google Cloud Functions, which allow developers to write and execute functions in languages such as Node.js, Python, Go and execute them in response to events. Google's approach to function execution centeres around container technology (Wayner, P. 2018. Accessed 29.05.23).

Microsoft's Azure Functions is a Faas platform that enables developers to create and execute functions written in languages like C#, JavaScript, Python. Similar to Google, they also harness the power of containers to execute these functions.

2.5 WebAssembly

WebAssembly, originally designed for running demanding computations in web browsers, present a promising technology that could help reduce the energy consumption of cloud services. It offers an interesting option for packaging functions with its compact binary format and fast execution time. This has the potiential to significantly reduce startup latency and resource overhead associated with traditional serverless platforms. This increased efficiency could lead to a direct decrease in energy consumption for cloud services, which in turn could motivate the industry to adopt alternative technology that enable a more sustainable cloud.

WebAssembly (Wasm) is a binary instruction format designed as a stack-based virtual machine. It aims to be a portable target for the compilation of high-level languages like Rust, C++, Go and many others, enabling deployment on the web for client and server applications. Originally designed and developed to complement

JavaScript in the browser, it now expands its scope to server-side applications, thanks to projects like WebAssembly System Interface (WASI), which provides a standarized interface for WebAssembly modules to interface with a system.

WebAssembly's design provides advantages over traditional deployments methods in the context of cloud native applications:

Efficiency and Speed: Wasm was designed to be fast, enabling near-native performance. Its binary format is compact and designed for quick decoding, contributing to quicker startup times, an important aspect for server-side applications. The performance gains could lead to less CPU usage, thereby improving energy efficiency.

Safety and Security: WebAssembly is designed to be safe and sandboxed. Each WebAssembly module executes within a confined environment without direct access to the host system's resources. This isolation of processes is inherent in WebAssembly's design, promoting secure practices.

Portability: WebAssembly's platform-agnostic design makes it highly portable. It can run across a variety of different system architectures. For cloud native applications, this means WebAssembly modules, once compiled, can run anywhere - from the edge to the server, irrespective of the environment.

Language Support: A large amount of programming langauages can already target WebAssembly. This means developers are not restricted to a particular language when developing applications intended to be deployed as WebAssembly modules. This provides greater flexibility to leverage the most suitable languages for particular tasks.

In contrast, traditional methods such as deployment with containers or VMs can be resource-intensive, slower to boot up, less secure due to a larger surface attack area, and less efficient. Given these, WebAssembly, with its efficiency, security, and portability, can potentially offer an attractive alternative deployment method for building and running cloud native applications, like the "Academemes" service we will explore in this essay.

2.5.0.1 WASM+WASI: Towards Energy-efficient FaaS Platforms

WebAssembly (WASM) and WebAssembly System Interface (WASI) present promising choices to traditional ways of deploying and hosting Function as a Service (FaaS) platforms, offering several notable advantages, in terms of startup times and energy efficiency.

Reduced Startup Times: One of the greatest strengths of Wasm is its compact binary format designed for quick decoding and efficient execution. It offers near-native performance, which results in significantly reduced startup times compared to container-based or VM-based solutions. In a FaaS context, where functions need to spin up rapidly in response to events, this attribute is particularly advantageous. This not only contributes to the overall performance but also improves the user experience, as the latency associated with function initialization is minimized.

Improved Energy Efficiency: Wasm's efficiency extends to energy use as well. Thanks to its optimized execution, Wasm can accomplish the same tasks as traditional cloud applications but with less computational effort. The CPU doesn't need to work as hard, which results in less energy consumed. With data centers being responsible for a significant portion of global energy consumption and carbon emissions, adopting Wasm could lead to substantial energy savings and environmental benefits.

Scalability: Wasm's small footprint and fast startup times make it an excellent fit for highly scalable cloud applications. Its efficiency means it can handle many more requests within the same hardware resources, hence reducing the need for additional servers and thus reducing the energy footprint further.

Portability and Flexibility: WASI extends the portability of Wasm outside the browser environment, making it possible to run Wasm modules securely on any WASI-compatible runtime. This means that FaaS platforms can run these modules on any hardware, operating system, or cloud provider that supports WASI. This portability ensures flexibility and mitigates the risk of vendor lock-in.

While runtime efficiency is an important aspect and typically a strength of Wasm, it might not be the primary focus of this thesis. That being said, it is worth mentioning that the efficient execution of Wasm modules does contribute to the overall operational efficiency and energy savings of Wasm-based FaaS platforms.

In summary, introducing WASM+WASI as a component for deploying and hosting FaaS platforms can offer significant benefits. Focusing on energy efficiency and reduced startup times, this approach could pave the way for more sustainable, efficient, and responsive cloud services. In the context of our "Academemes" service, this could lead to a scalable, performant, and environmentally friendly platform.

Part II

Project

Chapter 3

Approach

Chapter 4

Analysis

Chapter 5

Design

Chapter 6

Implementation

This is the chapter on implementing Nebula.

6.1 Tech stack

Rust/Docker/Etc.

Part III

Results

Chapter 7

Evaulation

Chapter 8

Discussion

Chapter 9

Conclusion

Chapter 10

References