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

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*

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 alternatives in cloud computing architectures.

This thesis investigates WebAssembly, a potential contender in the space of technlogies to consider in cloud native applications, to challenge conventional cloud paradigms through energy-efficient solutions. Leveraging the inherent efficiency, portability and lower startup times of WebAssembly modules, the study presents a novel approach that not only aligns with green energy principles, but also maintains the performance and scalability, essential for cloud services.

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

Acknowledgments

Express gratitude to advisors, colleagues, and others who have supported and helped throughout the research process.

Contents

Introduction

The rapid growth of cloud computing the past decade has led to the cloud consuming enormous amounts of energy. The entire ICT (Information- and Communication Technology) industry emits 2.1% to 3.9% of global green gas emission. Data centers, which are at the core of cloud computing, consume a significant amount of energy, estimated at around 200 TWh/yr or 1% of the worlds electricity (Freitag et al. 2021). This energy consumption could grow to between 15-30% of electricity consumption in some countries by 2030. Allthough data centers strive to reach a net zero sum carbon footprint, there are still a lot of electricity genereated by fossil fuels, a leading contributer to climate change (Mytton, D. 2021).

Important to note is that these measurements and estimates comes with some level of uncertainty, but give us a rough idea of the current and future situation. As demand for cloud services continues to rise, there is a pressing need to explore alternative methods that can help improve energy efficiency, while maintaining the performance, availability and scalability of these services.

This essay investigates the potential for utilizing technologies like Rust and WebAssembly to develop a prototype Function-as-a-Service (FaaS) platform. This prototype will aim to address the energy efficiency problem in cloud computing by serving academic memes through a web service.

Background

Cloud Computing: An Overview

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).

Some challenges include:

- Cost: 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 usage: 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.

Traditional Deployment Methods: Virtual machines and Containers

In the era preceding cloud computing, companies bought, set up and managed their own infrastructure. This necessitated having in-house infrastructure engineers to maintain on-premise data centers or servers, leading to significant cost. Sensing the potential in offering managed infrastructure, Amazon launched its subsidiary, Amazon Web Services, during the mid-2000s.

The launch of AWS's Amazon S3 cloud service in March 2006, followed by Elastic Compute Cloud (EC2) in August the same year (Barr, J. 2006), marked a major turning point in application development and deployment, and popularized cloud computing. EC2, as an Infrastructure-as-a-Service platform, empowered developers to run virtual machines remotely. While similar services existed before 2006, Amazon's large customer base helped them gain significant traction, effectively bringing cloud computing to the mainstream.

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.

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.

Some 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.

WebAssembly: A new paradigm?

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.

Wasm+WASI: Towards Energy-efficient FaaS Platforms

WebAssembly (Wasm) and WebAssembly System Interface (WASI) present promising alternatives to traditional ways of deploying and hosting Function as a Service (FaaS) platforms, offering several notable advantages, especially 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.

Hypothesis Revisited

Some graphs

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------------------------------
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
```

```
## v purrr     1.0.2
## -- Conflicts ------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts
```

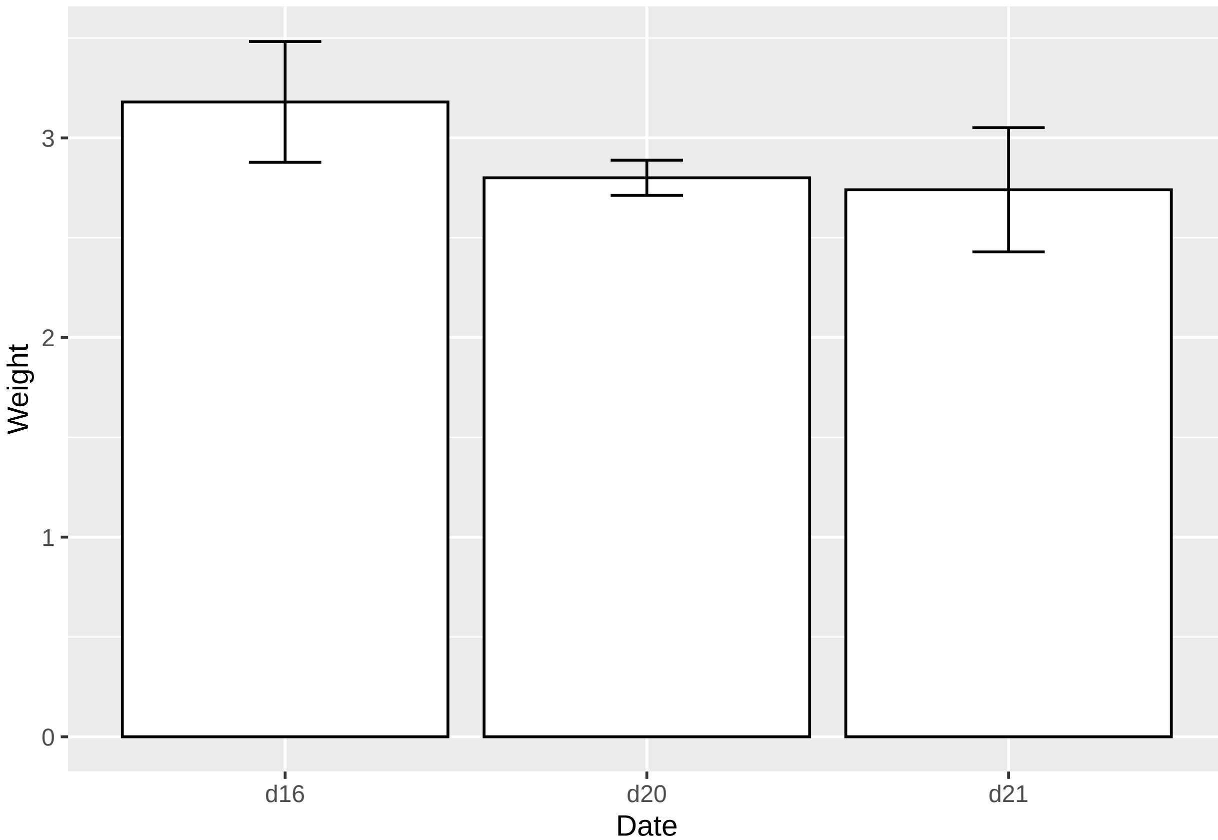
```r
library(gcookbook) # Load gcookbook for the cabbage_exp data set
library(dplyr)

cabbage_exp
```

```
##   Cultivar Date Weight        sd  n         se
## 1      c39  d16   3.18 0.9566144 10 0.30250803
## 2      c39  d20   2.80 0.2788867 10 0.08819171
## 3      c39  d21   2.74 0.9834181 10 0.31098410
## 4      c52  d16   2.26 0.4452215 10 0.14079141
## 5      c52  d20   3.11 0.7908505 10 0.25008887
## 6      c52  d21   1.47 0.2110819 10 0.06674995
```

```r
# Take a subset of the cabbage_exp data for this example
ce_mod <- cabbage_exp %>%
  filter(Cultivar == "c39")

# With a bar graph
ggplot(ce_mod, aes(x = Date, y = Weight)) +
  geom_col(fill = "white", colour = "black") +
  geom_errorbar(aes(ymin = Weight - se, ymax = Weight + se), width = .2)
```



Methodology

References