HPS
https://valerius.me/

Valerius Mattfeld

Result Presentation: Cloud Infrastructure with Go

Overhead Management in Invocations of Serverless Functions for Small Workloads

Table of contents

About Serverless Functions I

- Serverless functions are only executed when they are needed, in response to specific events or triggers. Schall et al., "Lukewarm Serverless Functions: Characterization and Optimization"
- The cloud provider automatically scales the resources up or down based on demand; infrastructure is provided. Roy, Patel, and Tiwari, "IceBreaker: Warming Serverless Functions Better with Heterogeneity"
- Cost-Effective: Only resources during the execution are billed Roy, Patel, and Tiwari, "IceBreaker: Warming Serverless Functions Better with Heterogeneity"

About Serverless Functions II

- Typically, functions reside within container images. (Brooker et al., *On-demand Container Loading in AWS Lambda*)
- Functions can be implemented using a wide range of programming languages. (Jackson and Clynch, “An Investigation of the Impact of Language Runtime on the Performance and Cost of Serverless Functions”)

Self-hostable Platforms for Serverless Functions

- Kubernetes (written in Go) will be the base-platform for this topic. (“Kubernetes - GitHub Repository”)
- Notable examples for Kubernetes-based self-hostable platforms are:
  - knative.dev (supporting languages like Go, Elixir, Java, etc.), “Knative Documentation”
  - nuclio.io (completely written in Go), “Nuclio - "Serverless" for Real-Time Events and Data Processing”
  - openfaas.com (also using Go), “Openfaas/Faas: OpenFaaS - Serverless Functions Made Simple”
  - fission.io (built with Go), “Fission/Fission: Fast and Simple Serverless Functions for Kubernetes”
  - openwhisk.apache.org (implemented in Scala), “OpenWhisk”

Serverless Functions on HPC I

- Serverless computing is gaining interest in the field of scientific computing for High-Performance Cluster (HPC) applications. (Malawski and Balis, “Serverless Computing for Scientific Applications”)
- However, Function-as-a-Service (FaaS) platforms often impose restrictions on available hardware resources. (Decker, Kasprzak, and Kunkel, “Performance Evaluation of Open-Source Serverless Platforms for Kubernetes”)
- On the other hand, serverless architecture offers a more granular and efficient approach to resource reservations. Qu, Calheiros, and Buyya, “A Reliable and Cost-Efficient Auto-Scaling System for Web Applications Using Heterogeneous Spot Instances”

Serverless Functions on HPC II

- Core Question: *Can serverless open-source software (OSS) in Go meet the performance requirements of High-Performance Computing (HPC)*,
- The particular area of focus of this project lies in optimizing the I/O of function invocations for small workloads (Decker, *The Potential of Serverless Kubernetes-Based FaaS Platforms for Scientific Computing Workloads* and Decker, Kasprzak, and Kunkel, “Performance Evaluation of Open-Source Serverless Platforms for Kubernetes”)

decker$_p$*erformance*$_2$022

Related Work

`Decker, Kasprzak, and Kunkel, "Performance Evaluation of Open-Source Serverless Platforms for Kubernetes"`

- Testing open-source platforms OpenFaaS and Nuclio on top of Kubernetes
- Serverless platforms - not an alternative for classic HPC:
  - Problematic parallelization of I/O
  - User unawareness of available hardware resources
  - Platform provider unawareness of user function resource requirements
  - Possible vendor lock-in

Setup

- The environment setup will use OpenStack GWDG VMs, as well as locally
- VMs exist in one location, Göttingen
- VM constellation
  - Message Service and Load Tester instance (HTTP Handler instance)
  - Node instance, which will host a function (RPC Server instance)
  - Emitter instance, which hosts the load-balancer
- A test PNG file is deployed on the first VM.
- The environment is reset after each benchmark

Software

- One executable is able to be a Node or Emmitter
- Executables are wrapped in Docker containers.
- Emitter-Node-Communication is done via TCP
- The message service communicates inputs and results on separate channels
- A load-tester (
) is deployed and sends HTTP requests to each implemented endpoint

“valerius21/yabt”

Web Frameworks

- `net/http`: The standard library
- Gin: The most popular repository
- Echo: Barebones Web Framework for Go
- Iris: Ergonomic Web Framework for Go
- Fiber: Express-inspired, ergonomic implementation of fastHTTP, which is used in nuclio.io

Endpoints

- Empty Endpoint: Function is empty
- Math Endpoint: Approximates Pi with the Monte-Carlo method
- Sleeper Endpoint: Blocks the invocation for one second
- I/O Endpoint: Applies image transformations and read-write operations on the Node VM

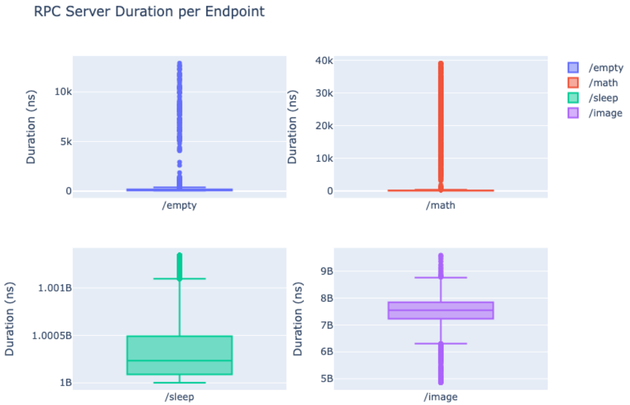
Load Testing

- Tool: “`valerius21/yabt`”
- Configuration:
  - Each endpoint is tested separately for every framework
  - Each endpoint is tested 100.000 times
  - Exception: `/image` with 1.000 times
  - Image used for the image endpoint is in the repository, `https://github.com/valerius21/scap-2024`
  - N=1000 for the Math Endpoint

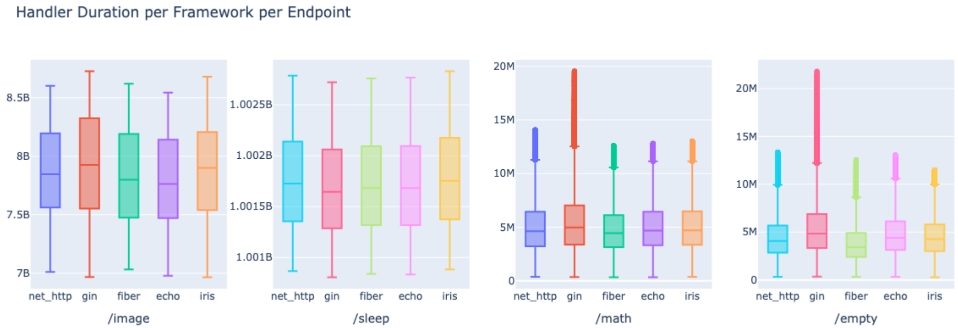
Pure Executions (ns)

- Empty function: **80 ns** on average
- Math (n=1000) function: **110 ns** on average
- Sleep Function: **20 ns (delta to 1 sec)** on average
- Image Function: **1sec** on average

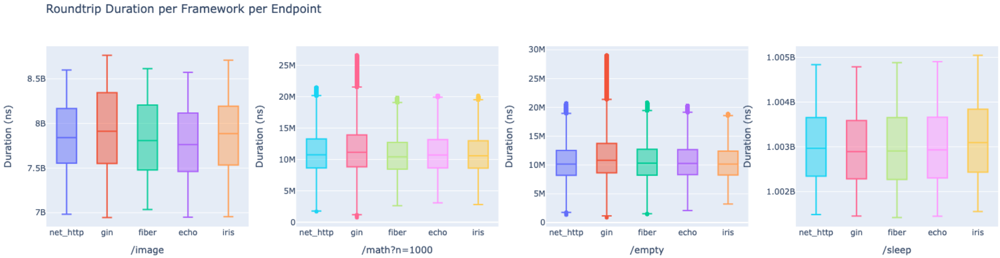
TCP Executions (ns)



In-Handler Executions (ns)



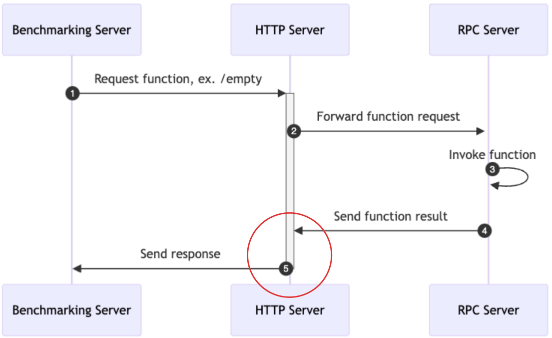
Request Roundtrip (ns)



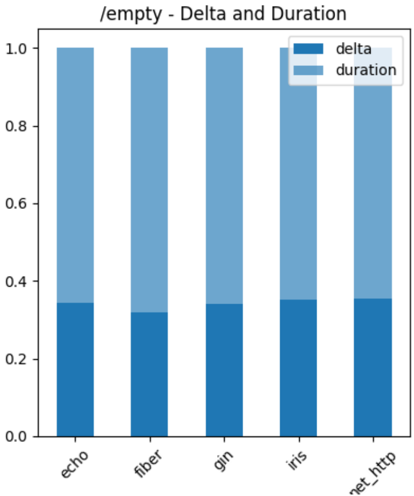
Performance Impressions

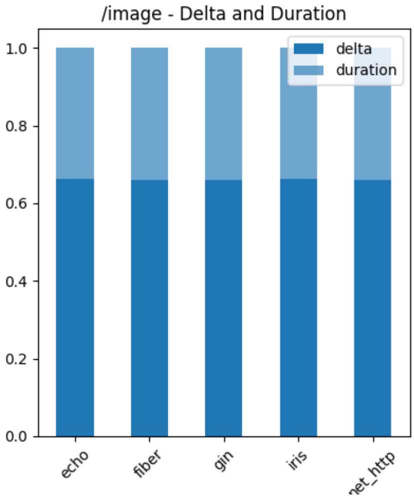
- the TCP / message communications take up a lot of time
- the handler performances vary vastly depending on the task
- in overall performances, Iris and net/http fall back

Bottleneck location

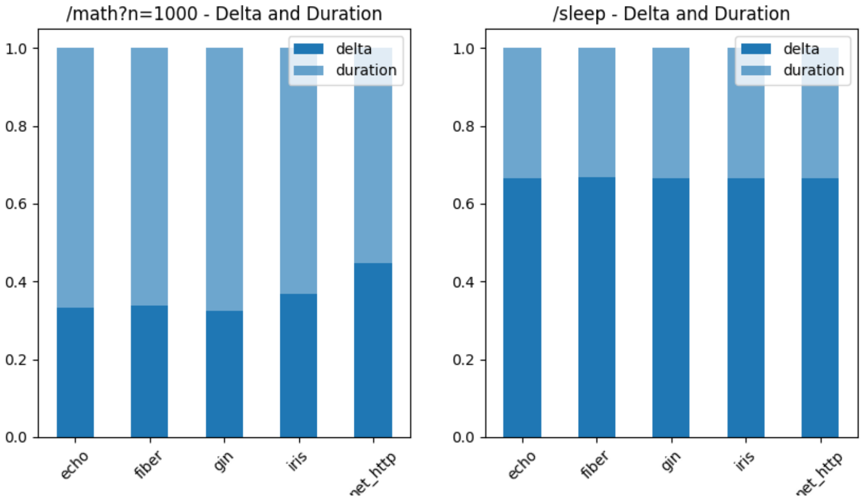


Average Delta per Endpoint I





Average Delta per Endpoint II



Conclusion

- Almost all frameworks perform similarly
- Gin has the worst performance on average (in-handler)
- `net/rpc` is used to parse function requests
- the bottleneck occupies one to two-thirds of the roundtrip time

References

Brooker, Marc et al. *On-demand Container Loading in AWS Lambda*. 2023. arXiv: 2305.13162 [cs.DC].

Decker, Jonathan. *The Potential of Serverless Kubernetes-Based FaaS Platforms for Scientific Computing Workloads*. Version V1. 2022. DOI: 10.25625/6GSJSE. URL: https://doi.org/10.25625/6GSJSE.

Decker, Jonathan, Piotr Kasprzak, and Julian Martin Kunkel. "Performance Evaluation of Open-Source Serverless Platforms for Kubernetes". In: *Algorithms* 15.7 (2022). ISSN: 1999-4893. DOI: 10.3390/a15070234. URL: https://www.mdpi.com/1999-4893/15/7/234.

"Fission/Fission: Fast and Simple Serverless Functions for Kubernetes". In: (2023). URL: https://github.com/fission/fission (visited on 05/31/2023).

Jackson, David and Gary Clynch. "An Investigation of the Impact of Language Runtime on the Performance and Cost of Serverless Functions". In: *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. 2018, pp. 154–160. DOI: 10.1109/UCC-Companion.2018.00050.

"Knative Documentation". In: (May 2023). URL: https://github.com/knative/docs (visited on 05/31/2023).

"Kubernetes - GitHub Repository". In: (May 2023). URL: https://github.com/kubernetes/kubernetes (visited on 05/31/2023).

Malawski, Maciej and Bartosz Balis. "Serverless Computing for Scientific Applications". In: *IEEE Internet Computing* 26.4 (2022), pp. 53–58. DOI: 10.1109/MIC.2022.3168810.

"Nuclio - "Serverless" for Real-Time Events and Data Processing". In: (May 2023). URL: https://github.com/nuclio/nuclio (visited on 05/31/2023).

"Openfaas/Faas: OpenFaaS - Serverless Functions Made Simple". In: (2023). URL: https://github.com/openfaas/faas (visited on 05/31/2023).