SEUSS: Skip Redundant Paths to Make Serverless Fast

James Cadden, Thomas Unger, Yara Awad, Han Dong, Orran Krieger, Jonathan Appavoo

Department of Computer Science
Boston University

The Proceedings of EuroSys, 2020
April 29th, 2020
Serverless Computing

1. **Function-as-a-Service (FaaS):** on-demand execution of a client code snippet (*functions*)

2. Applications are deployed and scaled *automatically*

3. Function **start time** is dominated by deterministic setup & install paths
Serverless Computing

1. event

Platform API

Application database

f₁.js  f₂.py  f₃.rb  ...  fₙ.js

Isolated Execution Environments

E₁

2. Run!

Compute Resources

4. Functions deploy quickly using a **pre-initialized** environment!
5. FaaS platform utility becomes a matter of cache efficiency!

6. Mechanism of the system-level defines the security, cache density, and responsiveness
FaaS Environment Caching

Cache Primitive:
Node.js “launcher” provides a REST API to import and run an arbitrary JavaScript function

Machine Density | Start Time
--- | ---
Linux Process | 4200 | 0.3 s
Docker Container | 3000 | 0.5 to 4 s
Container in a MicroVM | 450 | 1 to 7 s

Linux v4.15; Docker v18.09; [Xeon 2.20 GHz; 88GB]
Is there... a method to better enable reuse across the entire memory footprint of the function?

We want to...

1. Shorten the setup time of new function invocations
2. Improve cache density for fast repeat invocations
In SEUSS, functions are deployed inside of dedicated *unikernels*

1. Unikernels support strong isolation semantics

2. Enable “black box” capture of environment’s memory footprint into an *snapshot* (object)

3. **Page-level sharing** can be applied ubiquitously across the application and kernel layers
Environment Snapshots

Function invocation times are dominated by deterministic import & initialization procedures

Snapshots captured at strategic points in time can be used as templates for deploying execution
Environment Snapshots

Immutable snapshot images acts as a reusable launch point for new function invocations.

Runtime snapshot used for new invocations.
Environment Snapshots

Function-specific snapshots provide the near-immediate execution of function bytecode.

- **Function snapshot used** for repeat invocations.
Page-level sharing & copy-on-write (COW) can be applied to drastically reduce replicated state.
Snapshot Lineages

Child snapshots contain only a memory ‘diff’ of written pages

Anticipatory optimization enabled by accumulating state within the origin snapshot
SEUSS OS

- Foundation event-driven multi-core kernel (x86_64 native)
- Per-core job scheduler & network (NAT) layer
- In-memory snapshot cache
- Unprivileged unikernel guest:
  - POSIX0ish unikernel (Rumprun)
  - Minimal domain interface (Solo5)
FaaS Performance

Benchmarks

• 3-node OpenWhisk cluster
• 12-core, 88GB nodes
• custom benchmark tool
Sequential invocation requests to an Apache OpenWhisk compute node

Report the average start time

(cache thrashing)

(cache limit)

(Linux v4.15; Xeon 2.20 GHz; 88GB)
FaaS Platform Cache

Sequential invocation requests to an Apache OpenWhisk compute node

Report the average start time
FaaS Platform Throughput

- 64 concurrent requests
- NOP ('hello world) invocations
Resiliency to Traffic *Bursts*

**32-second Burst Intervals**

- **blue/purple**: Blocking IO requests to an external HTTP host (~250ms)
- **red**: 128 concurrent CPU-bound functions (~150 ms)

**16-second Burst Intervals**

![Graphs showing resiliency to traffic bursts with different intervals and colors for blocking IO requests and CPU-bound functions.](image-url)
Final Thoughts

- Unikernel snapshots promote reuse in a safe, simple, and effective way.

- Prototype demonstrates a major advantage for serverless applications models.

- In the end, high-performance cloud computing will continue to challenged our infrastructure software in new ways.

- It will be the operating system (design, mechanisms & techniques) that will address challenge and enable new workloads.