Hardware accelerated applications on unikernels for Serverless Computing

Anastassios Nanos, Charalampos Mainas
Overview

- Serverless computing
- Unikernels as the basis for lightweight function execution
- Current state & missing pieces
  - "echo" demo on OpenFaaS with solo5
- ML workloads, hardware acceleration & unikernels
  - Image classification demo on OpenFaaS with unikraft & vAccel
Serverless Computing

- Managed infrastructure orchestration by the service provider
- Effortless scaling (scale-out)
- Focus on Business logic
- Deploy code without provisioning the infrastructure
Serverless Computing

- Code deployed as a function with its dependencies
- Event-driven execution
- Billing model: actual resource usage vs idle
- Stateless execution oriented to:
  - microservices
  - triggered actions
Serverless Computing

- mostly deployed on Cloud infrastructure
- mode of execution seems useful for Edge workloads as well
  - e.g. ML inference for fast decision making
- currently backed by containers
Serverless Computing - common workflow

- control plane:
  - API gateway/scheduler/queue worker

- functions:
  - main init function / endpoint (provider)
    - setup environment (interface/endpoint init)
    - setup handler (to trigger user code)
  - handler function (user)
    - spawned on invocation (via endpoint trigger)
    - actual code execution

- bundled in container images:
  - spawned (sandboxed or plain)
  - listen to events via the endpoint/gateway
Serverless Computing - OpenFaaS in k8s
Serverless Computing - containers

● currently backed by containers: multi-tenancy issues (security/data leaks)

● current solution: sandbox containers using VMs (hardware extensions to isolate workloads).

● But, VMs:
  ○ exhibit non-negligible overhead (mem/mgt footprint)
  ○ do not facilitate hardware access / device sharing
Serverless Computing - containers

- Currently backed by containers: multi-tenancy issues (security/data leaks)

- Current solution: sandbox containers using VMs (hardware extensions to isolate workloads).

- But, VMs:
  - Exhibit non-negligible overhead (mem/mgt footprint)
  - Do not facilitate hardware access / device sharing
Serverless Computing - sandboxed containers

- overhead associated with VM sandboxing:
  - boot time (cold boot) vs warm boot / invocation (checkpointing)
  - memory footprint (Edge devices)
  - VM lifecycle / state (VMM, dependencies)

- complicated stack:
  - VMM, kernel/ramdisk/rootfs/libcontainerd etc.
Serverless Computing - unikernels

- how about we try something more elegant as the basis for serverless execution: **unikernels**!

- unikernels offer:
  - fast boot (inherently, serverless functions have no state)
  - low mem/mgt footprint
  - increased security (sandbox with hardware extensions + minimal attack surface)

- but unikernels lack:
  - function/code compatibility (interoperability)
  - runtime support (orchestration/process spawning)
Serverless Computing - unikernels

- Serverless frameworks are designed for containers:
  - based on container runtimes/operators

- Unikernels are not containers:
  - their management interface (+ I/O) resembles the one of VMs
  - the application is bundled in a single bootable binary
  - there is limited orchestration support
Serverless Computing - unikernels

unikernels for Serverless:

- container image & runtime flows:
  - bundle the unikernel binary & dependencies in a container image
  - tweak a container runtime to spawn a unikernel along with its monitor/sandbox

- invocation triggers
  - endpoint setup
  - interface with the serverless gateway
Serverless Computing - unikernels

● integrate unikernels in modern orchestrators:
  ○ build a compatible runtime able to spawn a unikernel (WiP)

● using the above runtime on a Serverless Framework is straightforward:
  ○ instead of spawning a container on function invocation, the system will spawn a unikernel -> no change needed on the serverless workflow.
Serverless Computing - OpenFaaS & unikernels
Serverless Computing - unikernels

- as a first step, we take a hybrid approach where:
  - we keep the container for the interface & the endpoint setup and
  - spawn the unikernel for the actual code execution

- we use OpenFaaS as the serverless framework, on a generic k8s cluster:
  - `faas-netes` & `gateway` (control plane)
  - `function pods` -> `generic containers` with `fwatchdog` to exec user function

- we use solo5 as the unikernel example
Deploy an echo function with solo5 & OpenFaaS

For more info: [https://github.com/nubificus/solo5-faas](https://github.com/nubificus/solo5-faas)
Serverless Computing - containers

- currently backed by containers: multi-tenancy issues (security/data leaks)
- current solution: sandbox containers using VMs (hardware extensions to isolate workloads).
- But, VMs:
  - exhibit non-negligible overhead (mem/mgt footprint) -> unikernels!
  - do not facilitate hardware access / device sharing
Serverless Computing - containers

- currently backed by containers: multi-tenancy issues (security/data leaks)
- current solution: sandbox containers using VMs (hardware extensions to isolate workloads).
- But, VMs:
  - exhibit non-negligible overhead (mem/mgt footprint) -> unikernels!
  - do not facilitate hardware access / device sharing
Serverless Computing - ML/AI

- Workloads that need device access in serverless execution are ML/AI applications
  - Edge instant decision making based on sensor data
  - Image processing, information extraction based on specific models
- How do we combine unikernel execution with hardware device access?
Unikernels for ML/AI

Unikernels are not a good fit for ML/AI workloads (at least not yet..)

- ML frameworks come in contrast with Unikernel architecture
  - ML frameworks dynamically link dependent libraries
  - ML frameworks have a lot of dependencies
  - Porting such a framework on a unikernel requires huge engineering effort

- No support for accelerated devices
  - hardware passthrough requires porting device drivers – not a good idea
  - the generic paravirtualization solution is almost non-existent
Proposed solution: vAccel

Components:
- user-facing API (accelerate-able functions)
- vAccelRT (dispatch)
- plugins (hardware, acceleration frameworks & transport)

Features:
- Hardware-agnostic API
  - Generic API at function granularity
  - Hardware-specific logic in vAccel, not in unikernel
- Portability
  - Binary compatibility for functions
  - Integration with high-level frameworks (Tensorflow, PyTorch, etc)
  - Multiple execution environments (host/container, VMs, unikernels)
- Security:
  - User code does not access directly the (shared) accelerator
  - Support for execution in virtualized guests
vAccel on Unikernels

- One Transport (VirtIO) plugin
- Offload acceleration requests to host

Unikernel

- vAccelRT linked with VMM or standalone handler (virtio-pci or vsock)
- Receives acceleration requests
- Hardware execution

Host

Unikernel

- vAccelRT linked with VMM or standalone handler (virtio-pci or vsock)
- Receives acceleration requests
- Hardware execution
vAccel: Current state

Unikernel frameworks
- Unikraft
- Rumprun

Programming interface
- C/C++ API
- Rust & Python

Framework integration
- Initial integration with TensorFlow
- Support for BLAS operations
Image classification with Unikraft & OpenFaaS
Summary

● Serverless execution based on unikernels
  ○ reduce cold boot times
  ○ reduce attack surface

● we use vAccel to expose hardware acceleration semantics to unikernels
  ○ function-based hardware acceleration
  ○ multi-framework support

● next step: develop a pure unikernel runtime for upper-layer orchestrators
This work is partly funded as part of the European Union’s Horizon 2020 research and innovation programme under Grant Agreements no 871900 (5G-COMPLETE) & 101017168 (SERRANO)
Source Code & Demo info:
- vAccel: [https://vaccel.org](https://vaccel.org) & [https://docs.vaccel.org](https://docs.vaccel.org)
- vAccelRT: [https://github.com/cloudkernels/vaccelrt](https://github.com/cloudkernels/vaccelrt)
- vAccel unikraft: [https://github.com/nubificus/unikraft-vaccel](https://github.com/nubificus/unikraft-vaccel)
- openfaas-solo5: [https://github.com/nubificus/solo5-faas](https://github.com/nubificus/solo5-faas)
- openfaas-vaccel: [https://github.com/nubificus/unikraft-vaccel-faas](https://github.com/nubificus/unikraft-vaccel-faas)

Thanks!