Unikernel as Processes, Firecracker

Yiying Zhang
Discussion

• As many of the recent virtualization systems are moving away from traditional, full-scale VMs, do you still think VMs can have a good market? Is it still worthwhile spending efforts into improving full-scale VMs?

• Unikernel is from academia, and Firecracker is from the industry. What do you think are some of the different focuses?

• In terms of virtualization, do you think the industry is leading the academia or the other way round? Which way do you think it should be? Give some examples.
Unikernels as Processes

Dan Williams, Ricardo Koller (IBM Research)
Martin Lucina (robur.io/Center for the Cultivation of Technology)
Nikhil Prakash (BITS Pilani)
Virtualization is a mixed bag

- Good for isolation, but...
  - Tooling for VMs not designed for lightweight (e.g., lightVM)
  - How do you debug black-box VMs?
  - Poor VM performance due to vmexits
  - Deployment issues on already-virtualized infrastructure
Why not run unikernels as processes?

• Unikernels are a **single process** anyway!

• Many benefits as a process
  • Better **performance**
  • Common tooling (gdb, perf, etc.)
  • ASLR
  • Memory sharing
  • Architecture independence

• **Isolation** by limiting process interface to host
  • 98% reduction in accessible kernel functions
Unikernel architecture

• ukvm unikernel monitor
  • Userspace process
  • Uses Linux/KVM

• Setup and loading
• Exit handling
Unikernel as process architecture

- **Tender**: modified ukvm unikernel monitor
  - Userspace process
  - Uses `seccomp` to restrict interface

- Setup and loading
- “Exit” handling
Unikernel isolation comes from the interface

- Direct mapping between 10 hypercalls and system call/resource pairs
  - 6 for I/O
    - Network: packet level
    - Storage: block level
- vs. >350 syscalls

<table>
<thead>
<tr>
<th>Hypercall</th>
<th>System Call</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>walltime</td>
<td>clock_gettime</td>
<td></td>
</tr>
<tr>
<td>puts</td>
<td>write</td>
<td>stdout</td>
</tr>
<tr>
<td>poll</td>
<td>ppoll</td>
<td>net_fd</td>
</tr>
<tr>
<td>blkinfo</td>
<td>pwrite64</td>
<td>blk_fd</td>
</tr>
<tr>
<td>blkwrite</td>
<td>pread64</td>
<td>blk_fd</td>
</tr>
<tr>
<td>blkread</td>
<td></td>
<td></td>
</tr>
<tr>
<td>netinfo</td>
<td>write</td>
<td>net_fd</td>
</tr>
<tr>
<td>netwrite</td>
<td>read</td>
<td>net_fd</td>
</tr>
<tr>
<td>netread</td>
<td></td>
<td></td>
</tr>
<tr>
<td>halt</td>
<td>exit_group</td>
<td></td>
</tr>
</tbody>
</table>
Implementation: nabla

- Extended Solo5 unikernel ecosystem and ukvm
- Prototype supports:
  - MirageOS
  - IncludeOS
  - Rumprun
- https://github.com/solo5/solo5
Results

• Unique kernel functions accessed: normal processes and VMs have 5-6x and 2-3x more than Nabla

• Application throughput: Nabla 101% - 245% higher than ukvm

• CPU utilization: Nabla has 12% reduction over ukvm

• Startup time: ukvm takes 30-370% longer
Conclusion

• Library OS (Exokernel) is an old idea aiming to expose more hardware interface directly to applications running in user space

• Unikernels: run app+libOS as VMs on hypervisor
  • Better isolation
  • Much more lightweight  \{ \textit{Better fit for modern cloud environments} \}
  • But need a lot of reimplementation and can’t use existing tooling

• Unikernels as processes: run app+libOS as processes on host OS, limit interface to OS for security
  • Can use existing tooling, more flexible
  • Even more lightweight
Firecracker
Lightweight Virtualization for Serverless Applications

Alexandru Agache, Marc Brooker, Andreea Florescu, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Diana-Maria Popa, and Phil Piwonka

February 2020
Why Firecracker?
EC2 m5.metal instance
384GB of RAM

Smallest Lambda Function
128MB of RAM
AWS Lambda before Firecracker

- Linux containers on VM
  - One container per function
  - One VM per customer (per machine)
- Containers: trading off security and compatibility
- VMs: difficulties of efficiently packing workloads
Requirements for Amazon
Requirements for Amazon

- **Isolation**: It must be safe for multiple functions to run on the same hardware
Requirements for Amazon

• *Isolation*: It must be safe for multiple functions to run on the same hardware

• *Overhead & Density*: Thousands of functions on a single machine
Requirements for Amazon

- **Isolation**: It must be safe for multiple functions to run on the same hardware

- **Overhead & Density**: Thousands of functions on a single machine

- **Performance**: Functions must perform similarly to running natively
Requirements for Amazon

- **Isolation**: It must be safe for multiple functions to run on the same hardware

- **Overhead & Density**: Thousands of functions on a single machine

- **Performance**: Functions must perform similarly to running natively

- **Compatibility**: Arbitrary Linux binaries and libraries. No code changes or recompilation
Requirements for Amazon

- **Isolation**: It must be safe for multiple functions to run on the same hardware.

- **Overhead & Density**: Thousands of functions on a single machine.

- **Performance**: Functions must perform similarly to running natively.

- **Compatibility**: Arbitrary Linux binaries and libraries. No code changes or recompilation.

- **Soft Allocation**: It must be possible to over-commit CPU, memory, and other resources.
What about Existing Virtualization Solutions?

- QEMU/KVM: density and overhead challenges
- Linux containers: isolation and compatibility challenges
- LibOS approaches: compatibility challenges
- Language VM isolation: compatibility and isolation challenges
Firecracker is an open source VMM that is purpose-built for creating and managing secure, multi-tenant container and function-based services.
Firecracker Overview

- A VMM that uses KVM to provide minimal VMs (MicroVMs)
- Supports Linux and OSv guest OS
- Rely on Linux for many functionalities
  - Saving implementation efforts
  - Fits what Amazon’s operators are familiar with
- Started with a branch of Google Chrome’s *crosvm*
  - Removed >50% code (drivers, etc.) and added more
- Written in Rust, open source
What Firecracker does not provide

- No BIOS
- Cannot boot arbitrary kernels (e.g., no Windows)
- No legacy device or PCI support
- No VM migration support
Firecracker Design Details

- Device model
  - Limited emulated devices, virtio for network and block devices

- REST API for configure, manage, start, and stop MicroVMs

- Rate limiters in block and network devices

- Security: apply defenses of hardware-based attacks when deploying

- Jailer: a wrapper around Firecracker to sandbox it (e.g., chroot, pid/network namespaces, seccomp with 24 whitelist syscalls etc.)

The interface b/w VM and hypervisor is the block interface
  => file system in guest OS
  => better security
Firecracker + Lambda

- “Sticky” routing to few workers
- Each worker runs 100s to 1000s of slots (MicroVMs)
- If no available slot, executes “Placement” service
- Shim process in MicroVM communicates with MicroManager

Figure 2: High-level architecture of AWS Lambda event path, showing control path (light lines) and data path (heavy lines)

Figure 3: Architecture of the Lambda worker
Operational Lessons
• In production in AWS Lambda
  • Millions of workloads
  • Trillions of requests/month
Lesson #1: Compatibility is Hard

Just disabling Hyperthreading revealed two bugs in Apache Commons HTTP Client, and one in our own code.

Re-implementing OS components would have been worse.

Performance compatibility too!
Lesson #2: Immutable, Time-Limited Machines

Common systems-administration tools like `rpm` and `dpkg` are non-deterministic.

Limiting max fleet life helps operational hygiene.
Lesson #3: The Job is Never Done

Changing customer needs means that there are always improvements to be made.
Performance
MicroVM start latency (serial)
MicroVM start latency (50 parallel)
QD32 IO Throughput vs Bare Metal

Bandwidth (MB/s)

- metal
- FC
- Cloud HV
- QEMU

4k read  4k write  128k read  128k write
QD1 IO Latency vs Bare Metal
Discussion

• As many of the recent virtualization systems are moving away from traditional, full-scale VMs, do you still think VMs can have a good market? Is it still worthwhile spending efforts into improving full-scale VMs?

• Unikernel is from academia, and Firecracker is from the industry. What do you think are some of the different focuses?

• In terms of virtualization, do you think the industry is leading the academia or the other way round? Which way do you think it should be? Give some examples.