Think Outside the Container:
Address Space Caching for Serverless Performance

Presenter: Tommy Unger
James Cadden, Han Dong, Yara Awad, Jonathan Appavoo, Orran Krieger
A FaaS System

Client API Gateway

Blob Store

Controller

Key Val

Messenger

Invoker

Docker

Docker

Docker

Docker

Docker

Docker

Docker

Docker

Docker

Docker

run f on x!
The Challenges

• Isolated Execution Environment Creation
• Initialize High Level Language Interpreters
• Do both fast!
8192 Fn Executions on 16 Cores

- Runtime (mins)
- # Unique Fns

The graph shows the runtime in minutes as the number of unique functions increases. The runtime remains relatively stable until a certain point, after which it increases dramatically.
Container Creation Time

- Container creation is 1000x slower than fork()

- We get bit by the first head before we can even worry about init times
Existing Mechanisms

- Isolation
- Performance
- Processes
- Containers
- VMs
Research Directions (not to scale!)

- Processes
- Containers
- SOCK (Oakes, ’18)
- VMs

Unikernels as Processes (Williams, ’18)
My VM … (Manco, ’17)
Research Directions (not to scale!)

- Processes
- Containers
- VMs
- SOCK (Oakes, '18)
- Unikernels as Processes (Williams, '18)
- My VM ... (Manco, '17)

Axes:
- Performance
- Isolation
Remember Ali’s Unikernels?

Standard Processes

- Application
- Application
- Application

| Memory Management | Needed Device Drivers | Network Stack |

Hardware

Unikernel

- Application
- Memory Management
- Needed Device Drivers
- Network Stack

Hardware
Teaser Isolation Argument

Standard Processes

Unikernel Monitor

Unikernels as Processes
(Williams, SoCC ’18)
Our Goal

Performance

Processes

SOCK (Oakes, ’18)

Containers

VMs

Isolation

Unikernels as Processes (Williams, ’18)

My VM ... (Manco, ’17)
Key Observation

• Look at all that consolidated state, wouldn't it be easy to take a *snapshot*?
Key Observation

• Look at all that consolidated state, wouldn’t it be easy to take a *snapshot*?

• State bound in single address space

• No serialization of kernel data structures

• Clearly defined quiescent points
Snapshot Caching
Snapshot Caching

- Boot Execution Environment
- Bringup Interpreter **
- Receive Fn Code
- Import Libraries ***
- Compile to Bytecode ***
- Receive Fn Args
- Execute Function
Snapshot Caching

Boot Execution Environment
Bringup Interpreter **
Receive Fn Code
Import Libraries ***
Compile to Bytecode ***
Receive Fn Args
Execute Function

Generic
Language Specific
Function Specific
Arg Specific
Snapshot Caching

- Boot Execution Environment
- Bringup Interpreter **
- Receive Fn Code
- Import Libraries ***
- Compile to Bytecode ***
- Receive Fn Args
- Execute Function

- Generic
- Language Specific
- Function Specific
- Arg Specific
Speedup Using Cached Snapshots

- Cold
- Warm
- Hot
Speedup Using Cached Snapshots

Cold

Warm

Hot

run f on x!
Speedup Using Cached Snapshots

- Cold
- Warm
- Hot
Speedup Using Cached Snapshots

- Cold
- Warm
- Hot

- Boot Execution Environment
- Bringup Interpreter **
- Receive Fn Code
- Import Libraries ***
- Compile to Bytecode ***
- Receive Fn Args
- Execute Function
Speedup Using Cached Snapshots

Cold

- Boot Execution Environment
- Bringup Interpreter **
- Receive Fn Code
- Import Libraries ***
- Compile to Bytecode ***
- Receive Fn Args
- Execute Function

Warm

Hot
Speedup Using Cached Snapshots

Cold
- Boot Execution Environment
- Bringup Interpreter **
- Receive Fn Code
- Import Libraries ***
- Compile to Bytecode ***
- Receive Fn Args
- Execute Function

Warm
- Boot Execution Environment
- Bringup Interpreter **
- Receive Fn Code
- Import Libraries ***
- Compile to Bytecode ***
- Receive Fn Args
- Execute Function

Hot
Speedup Using Cached Snapshots

Cold
- Boot Execution Environment
- Bringup Interpreter **
- Receive Fn Code
- Import Libraries ***
- Compile to Bytecode ***
- Receive Fn Args
- Execute Function

Warm
- Boot Execution Environment
- Bringup Interpreter **
- Receive Fn Code
- Import Libraries ***
- Compile to Bytecode ***
- Receive Fn Args
- Execute Function

Hot
- Boot Execution Environment
- Bringup Interpreter **
- Receive Fn Code
- Import Libraries ***
- Compile to Bytecode ***
- Receive Fn Args
- Execute Function
Fn Execution Runtime vs # Unique Functions

- Seuss-16 (cold) [8192, 1U, 1F, 24RC]
- LinuxVM-512 (cold) [8192, 1U, 1F, 24RC]
Cache Miss Latency vs # Unique Functions
Current Work

- Adding performant I/O
- Enabling Hardware Isolation:
  - Page Tables Access Bits
  - Ring Transitions
- Building Benchmarking Tools
Future Directions

• Multi Node & Distributed Snapshot Cache
• Overlay Precompiled Include Libraries -> DLLs
• Client Snapshot & Rollback
• Densify Snapshots ->
  • implement snapshot by single memcpy()
• Hardware Acceleration: RDMA
Think Outside the Container: Address Space Caching for Serverless Performance

Presenter: Tommy Unger

James Cadden, Han Dong, Yara Awad, Jonathan Appavoo, Orran Krieger
Snapshotting for Parallel Distribution

run \( f \) on \([x_1, x_2 \ldots x_n]\)!

run \( g \) on \( y \)!
Snapshot Deployment for Parallel Execution

Locate Language Specific Snapshot

JS Interpreter
Snapshot Deployment for Parallel Execution

JS Interpreter

Overlay Fn Specific Snapshot

f
g
Snapshot Deployment for Parallel Execution

JS Interpreter

f

COW Forks

f(x1) f(x2) f(xn)

COW Fork

g

g(y)
Backups
FaaS Promises Users…

- Massive Elastic Concurrency
- Rich Function Composition, Fan In/Out
- à la carte Execution
- No Software Provisioning
• Reduce \#Syscalls from > 300 -> 10

• Touches Fewer Unique Kernel Functions than Process

• Touches Fewer Unique Kernel Functions than VM!

Table 1: The entire set of ukvm hypercalls and the system call/resource pair that correspond to them.

<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>-</td>
<td>-</td>
</tr>
<tr>
<td>blkwrite</td>
<td>pwrite64</td>
<td>blk_fd</td>
</tr>
<tr>
<td>blkread</td>
<td>pread64</td>
<td>blk_fd</td>
</tr>
<tr>
<td>netinfo</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>netwrite</td>
<td>write</td>
<td>net_fd</td>
</tr>
<tr>
<td>netread</td>
<td>read</td>
<td>net_fd</td>
</tr>
<tr>
<td>halt</td>
<td>exit_group</td>
<td>-</td>
</tr>
</tbody>
</table>
Our Port

Node 1

Lightweight Distributed OS

Node n
Remember the Challenges?

- Isolated Execution Environment Creation
- Interpreter Initialization & Imports
- Bytecode Pre-compilation
Remember the Challenges?

- Isolated Execution Environment Creation
- Interpreter Initialization & Imports
- Bytecode Pre-compilation

Unified Snapshotting Solution
run f on x!
run f on x!
Thanks!
Standard Process

Node

APP1

APP2

Net
FS
Linux
IPC ...
Sched

Syscalls

HW
Moving Functionality to Userspace

Node

APP1

Net

APP2

Net

FS

Linux

IPC ...

Sched

HW
Moving Functionality to Userspace
Moving Functionality to Userspace

Node

Linux

HW

APP1

FS

Net

Sched

IPC ...

APP2

FS

Net

Sched

IPC ...

Node