Containers in Cloud

Swaroop Kotni

Outline

Container use cases

Docker Containers

Docker vs VMs

Case study:
Containers for FaaS

- What is FaaS?
- Deconstructing container performance
- Special purpose containers for FaaS

Containers – Who uses them?

- Cloud providers
  - Light-weight solution to isolate applications running on a single machine
  - Example -- Function as a Service (FaaS)/Serverless computing
- Developers
  - Focus on application logic rather than chasing version dependencies
  - Agnostic to application environment

Container Creation

Without Docker

```
# Create a minimal root installation

sh1$ wget http://dl-cdn.alpinelinux.org/alpine/v3.10/releases/x86_64/alpine-minirootfs-3.10.1-x86_64.tar.gz

# Create rootfs directory, for new_root argument to pivot_root syscall

sh1$ mkdir rootfs

# Untar the contents to rootfs directory

sh1$ tar -xzf alpine-minirootfs-3.10.1-x86_64.tar.gz -C rootfs
```

1

```
# Create new (u)ts and (m)ount namespaces

$ PS1 ='sh2% ' sudo unshare -um bash

# Make rootfs directory as new root

sh2% mount --bind rootfs rootfs

sh2% cd rootfs

sh2% mkdir put_old

sh2% pivot_root . put_old

sh2% cd /

# Unmount put_old

sh2% unmount -l put_old

# Change hostname

sh2% hostname subzero
```

2

With Docker

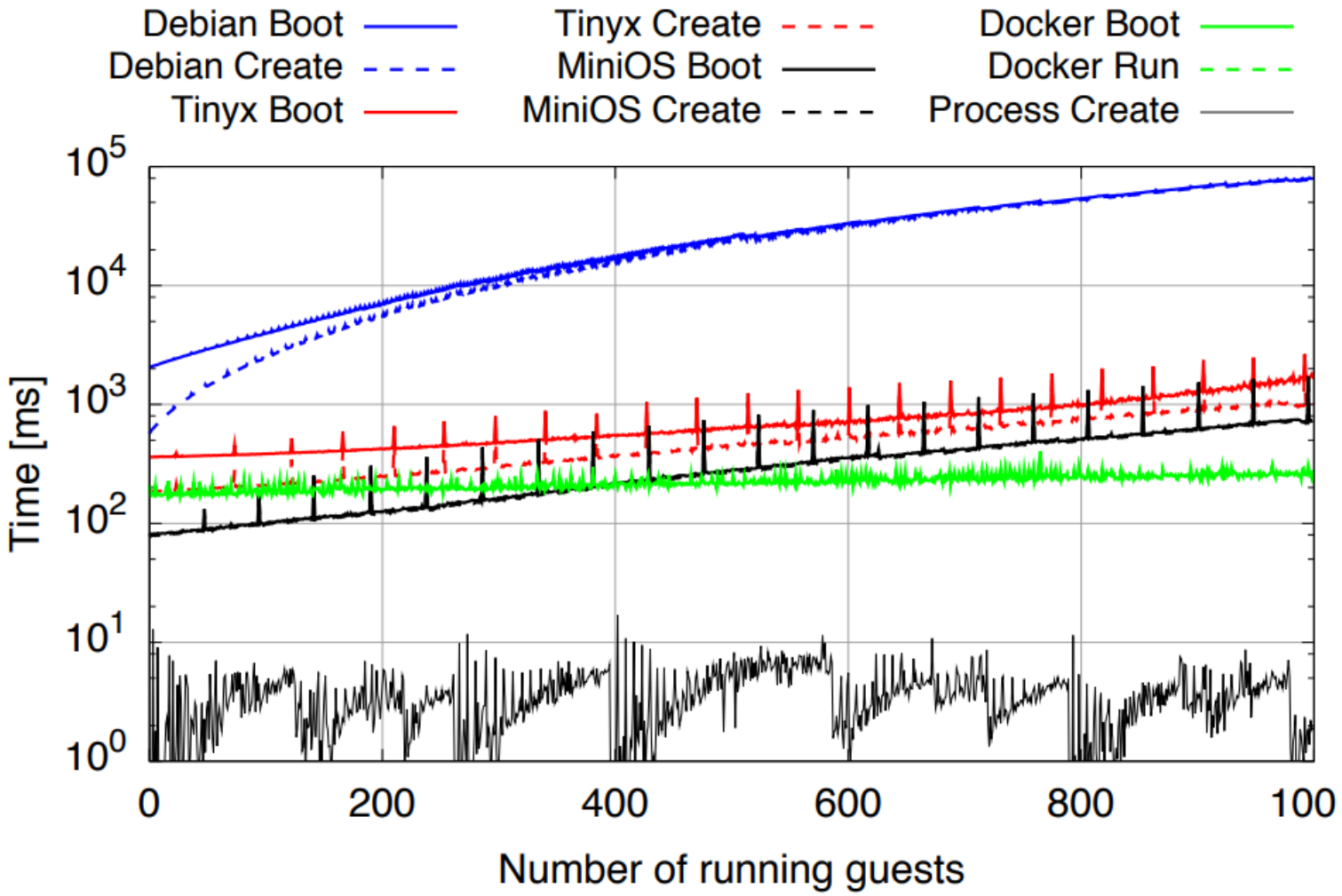
```
# Launch container using docker

sh1$ docker run -dt --hostname=subzero --memory 512MB --name demo alpine:3.10


# Debug docker container

sh1$ docker exec -it demo bash
```

Easy to launch, easy to debug

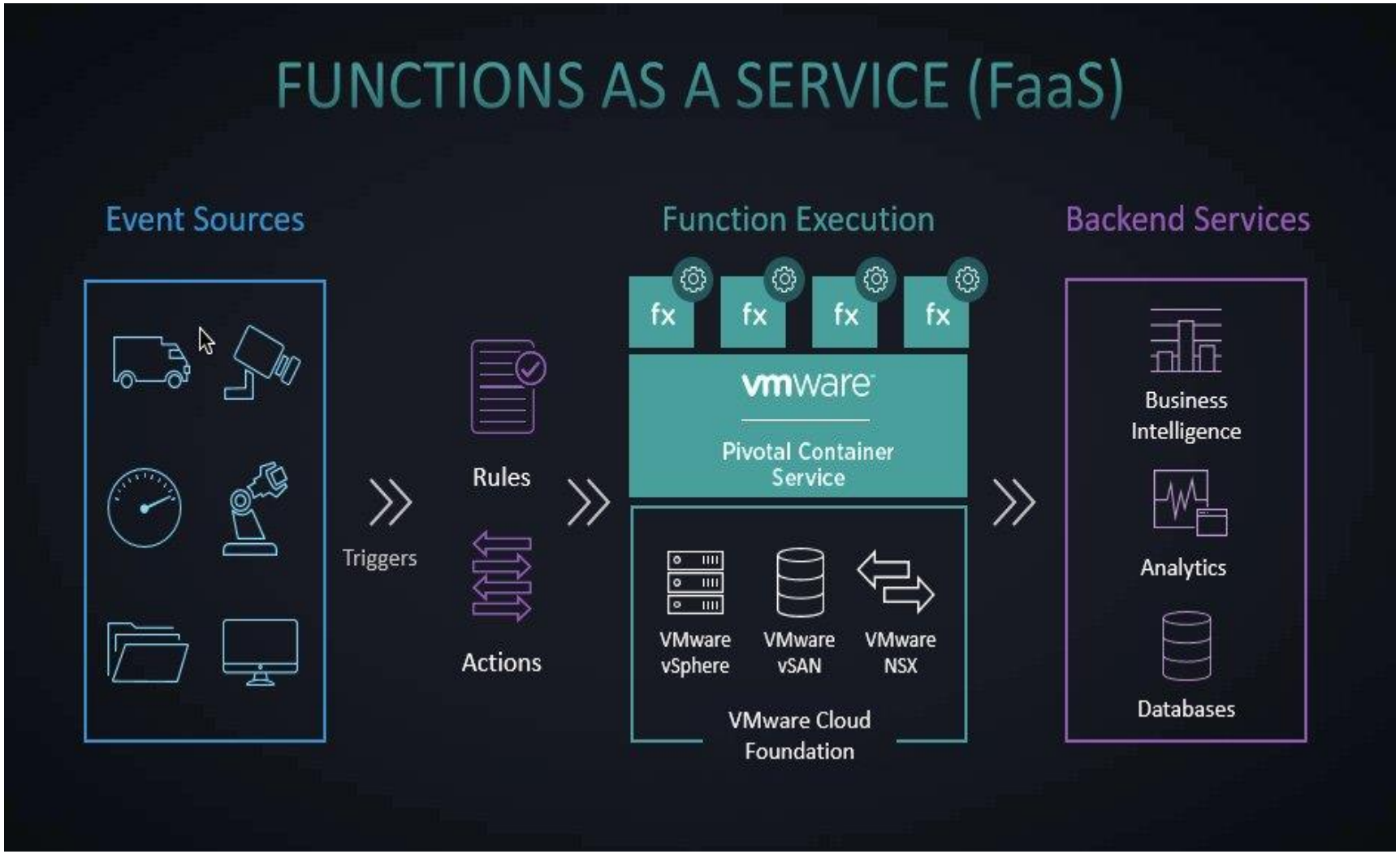


Docker vs VMs

Source: **My VM is Lighter (and Safer) than your Container [SOSP '17]**

Case Study: Containers for FaaS

Interlude: What is FaaS?



Features

- Event-driven computing for stateless functions
- Functions execute in containers
- Platform provides auto-scaling
- Pay per use cost model
- Developers can focus on business logic

Landscape

- Function execution time: 1-60 s
- Triggers: HTTP, Timer, Queue, Storage (S3)
- Languages: Node.js, Python
- Popular services: AWS Lambda, Azure Functions, Google Cloud Functions

Source: [VMWare Twitter Handle]

Deconstructing Container Startup

Container Creation
1. Allocate Storage
2. Namespaces
3. Cgroups

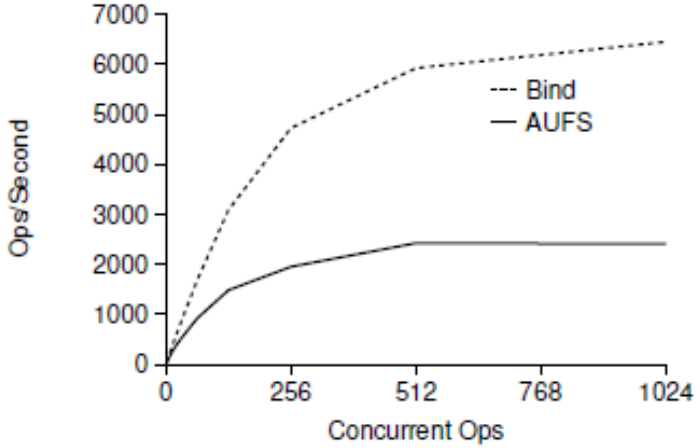


Figure 1. Storage Primitives. *The performance of mounting and unmounting AUFS file systems is compared to the performance of doing the same with bind mounts.*

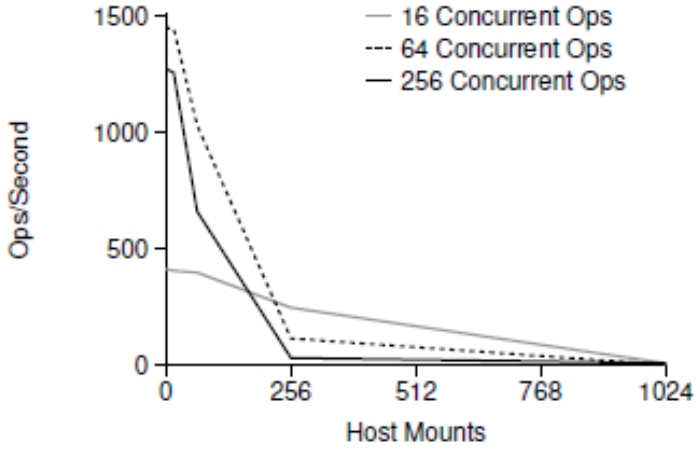


Figure 2. Mount Scalability. *This shows the rate at which processes can be unshared into new mount namespaces (y-axis) as the number of existing mounts varies (x-axis).*

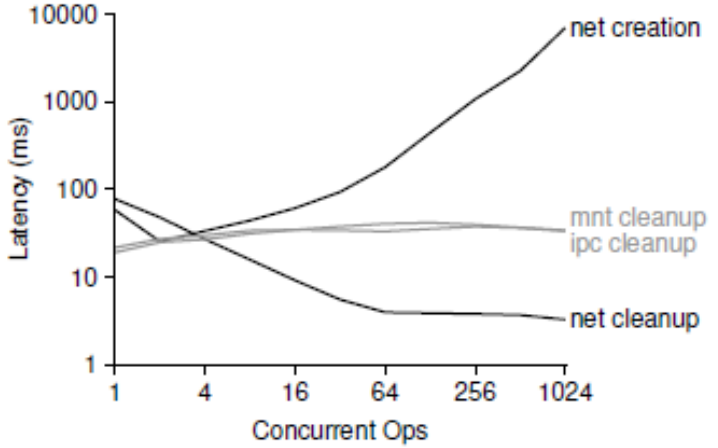


Figure 3. Namespace Primitives. *Lines show the latencies for copy_net_ns, put_mnt_ns and put_ipc_ns, and the average time spent by the network cleanup task per namespace.*

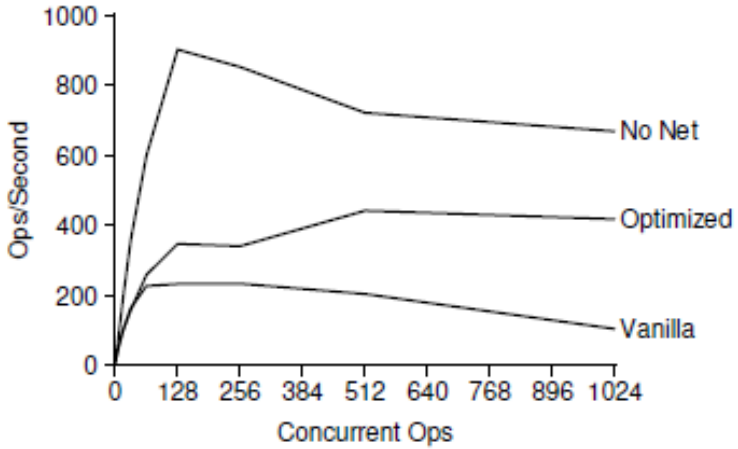


Figure 4. Network Namespace Performance. *A given number of containers (x-axis) are created and deleted in parallel with multiple network namespace configurations.*

Source: SOCK - Rapid Task Provisioning with Serverless-Optimized Containers [ATC '18]

Optimizations for FaaS

- Use bind mounts
- Use chroot
- No network namespaces
- cgroup pools

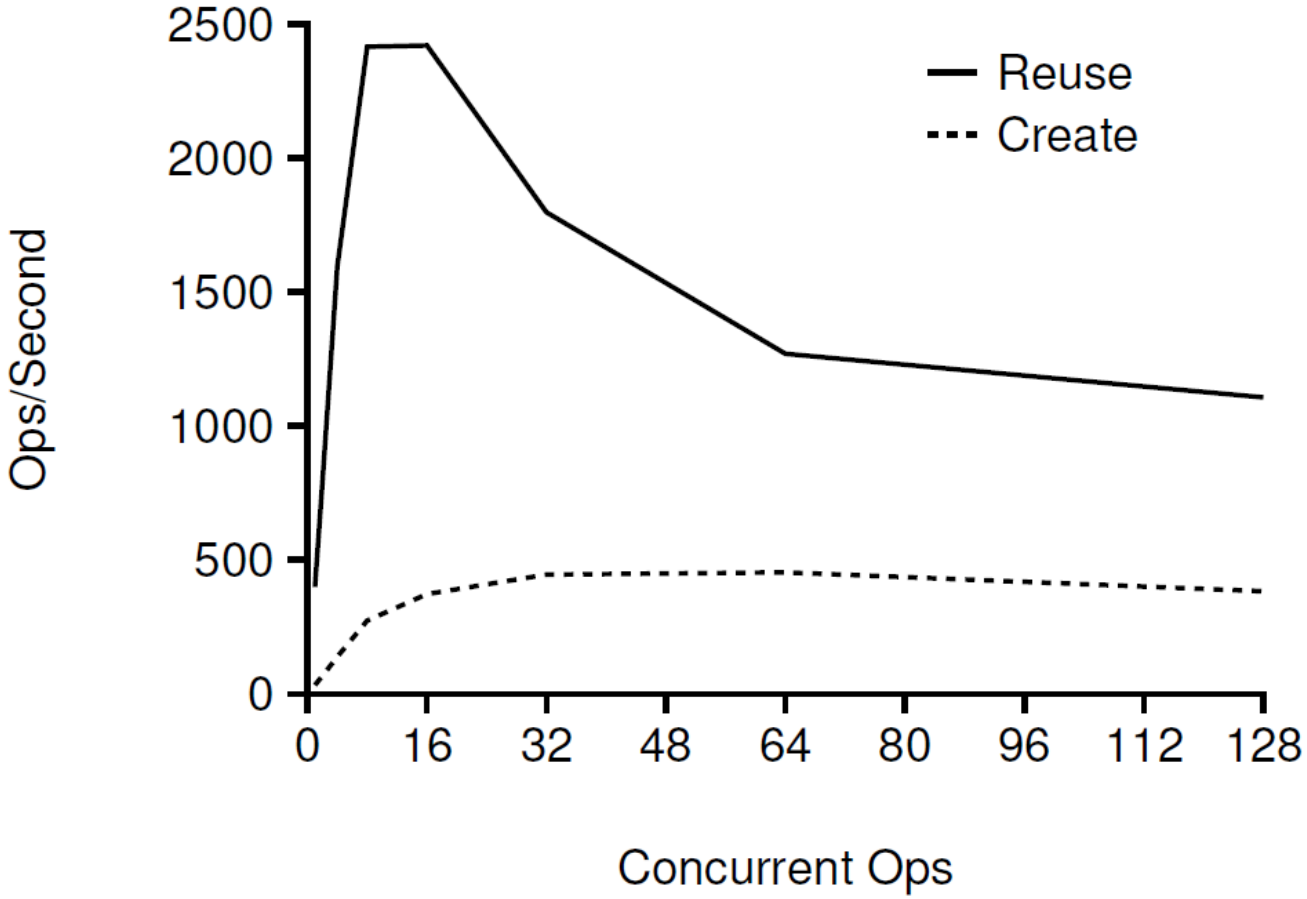


Figure 5. Cgroup Primitives. *Cgroup performance is shown for reuse and fresh-creation patterns.*

Source: SOCK - Rapid Task Provisioning with Serverless-Optimized Containers [ATC '18]

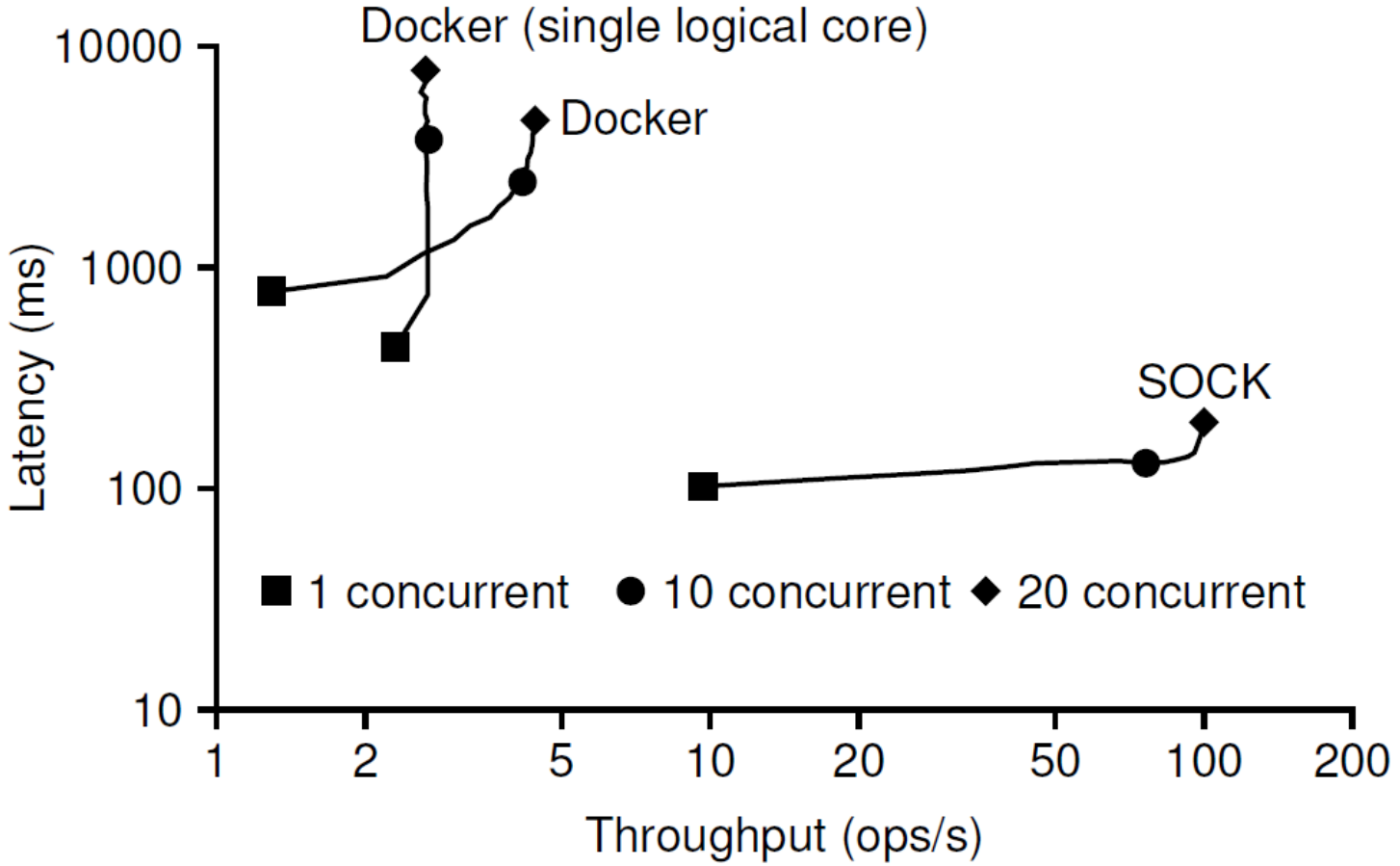


Figure 14. Docker vs. SOCK. *Request throughput (x-axis) and latency (y-axis) are shown for SOCK (without Zygotes) and Docker for varying concurrency.*

SOCK vs Docker

Source: **SOCK - Rapid Task Provisioning with Serverless-Optimized Containers [ATC '18]**