Peeking Behind the Curtains of Serverless Platforms

Liang Wang, Uw-madison; Mengyuan Li And Yinqian Zhang, The Ohio State University; Thomas Ristenpart, Cornell Tech; Michael Swift, Uw-madison

Presented by-Ayush Garg

Paper Contributions

- In-depth study of resource management and performance isolation in

- Identify opportunities to improve serverless platforms
  - AWS: Bad performance isolation, function consistency issue, …
  - Azure: Unpredictable performance, tenant isolation issues, …
  - Google: Resource accounting bug, …
- Open-source measurement tool
  - (https://github.com/liangw89/faas_measure)

Presentation Overview

Serverless Introduction

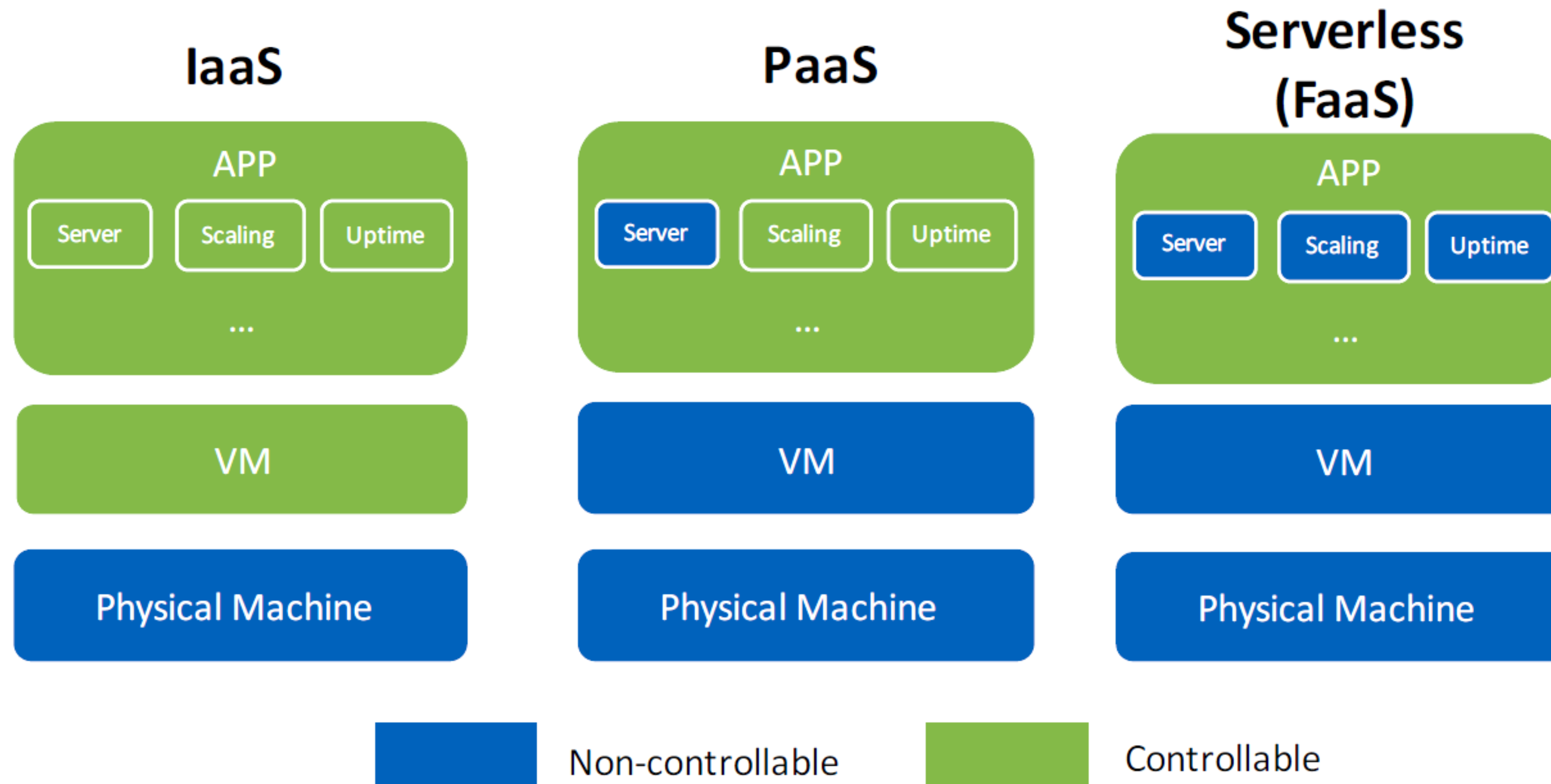
Methodology

Results

- Serverless Architecture
- Resource Scheduling
- Performance Isolation
- Bugs

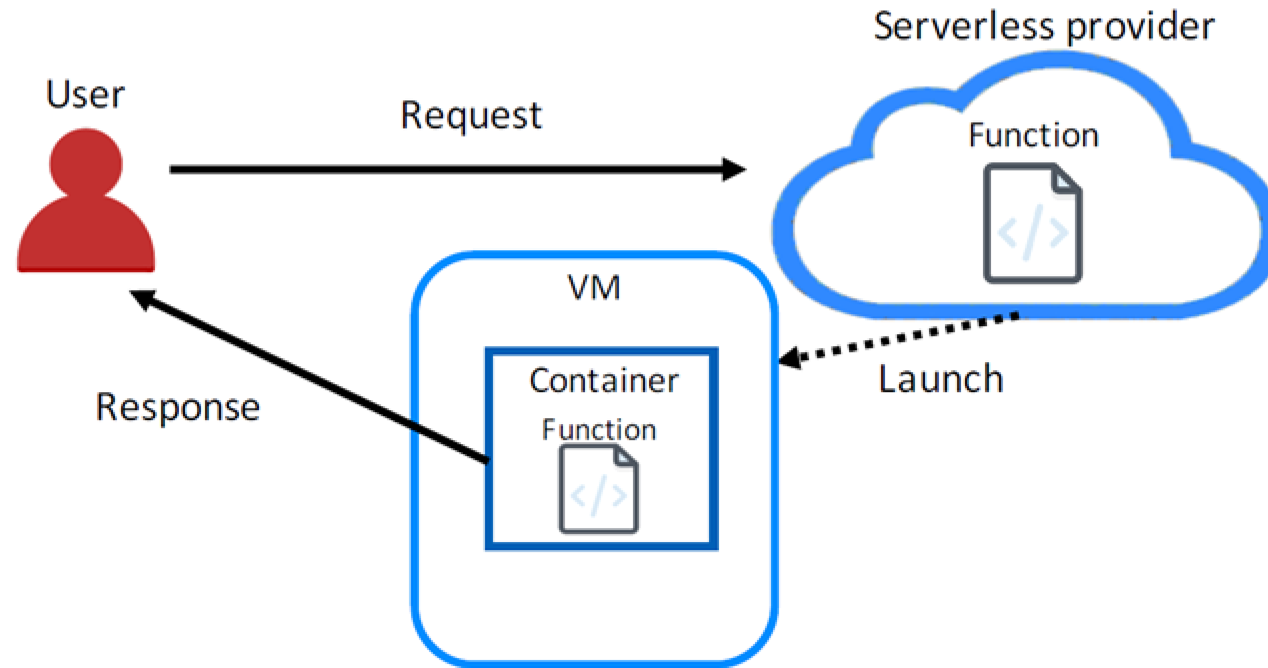
Summary

Providers do more, tenant do less



How Serverless Works

A function runs in a container (**function instance**) launched by the provider with limited CPU/memory/execution time



Serverless Introduction

Methodology

Results

- Serverless Architecture
- Resource Scheduling
- Performance Isolation
- Bugs

Summary

Methodology

Invoke measurement functions many times (50K+) under various settings from vantage points in the same cloud region

Measurement function

- Collect information via procfs/cmd/env
- Execute performance tests

Setting variables:

- Function memory
- Function language
- Request frequency
- Concurrent request

Time:

- July–Dec 2017, May 2018

Serverless Introduction

Methodology

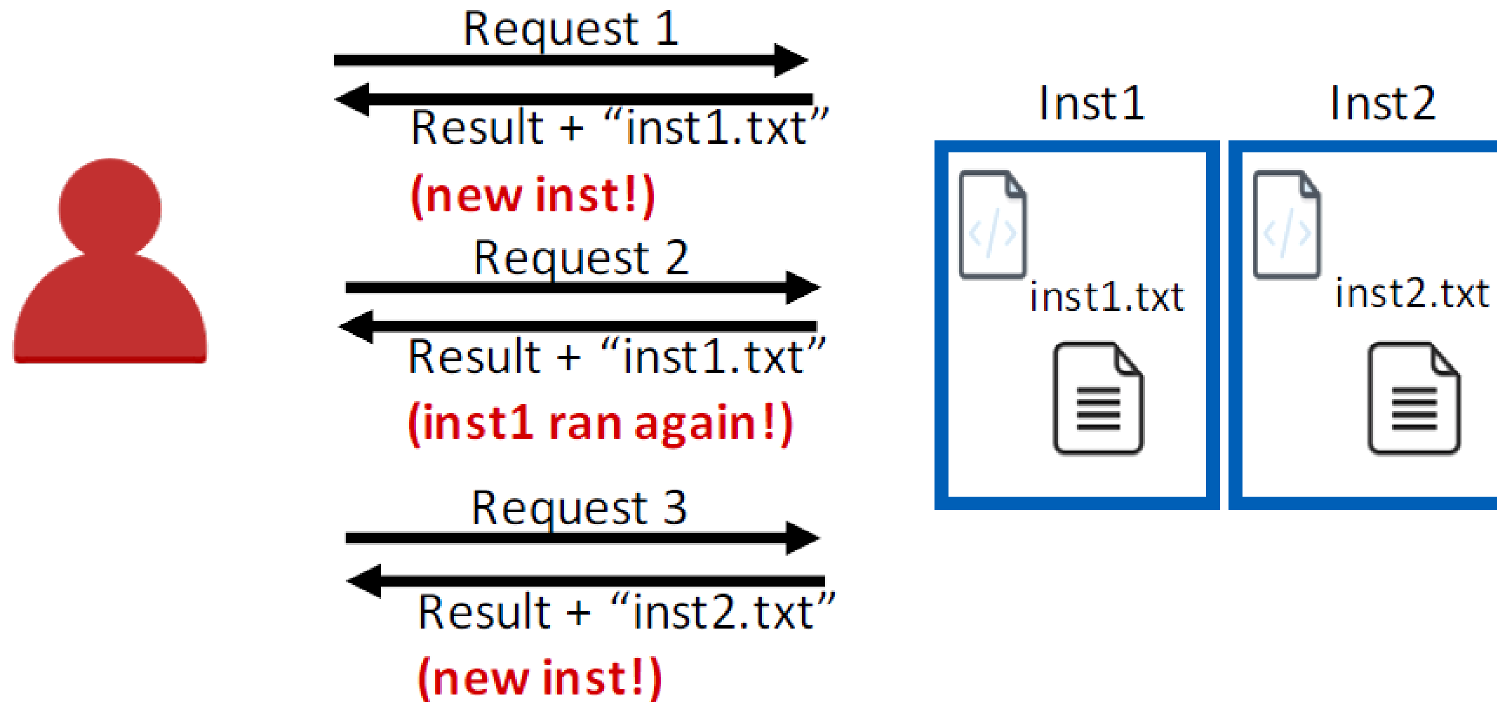
Results

- Serverless Architecture
- Resource Scheduling
- Performance Isolation
- Bugs

Summary

Instance Identification

Write a unique file on /tmp →persistent during instance lifetime



VM Identification

AWS

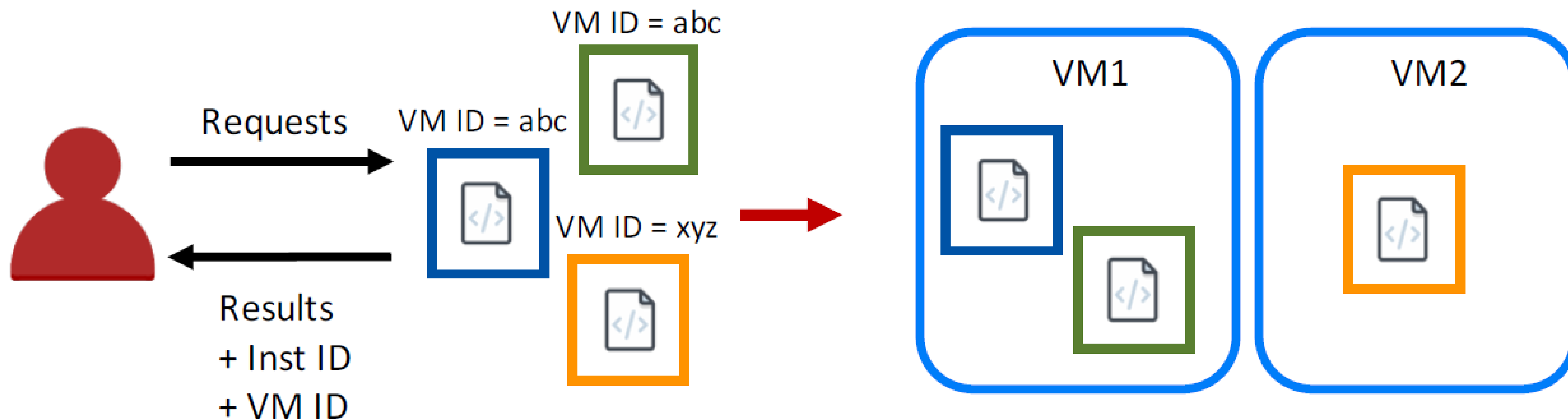
- Entry in /proc/self/cgroup called *"instance root ID"*
- Verified by I/O-based coresidency tests, and
- Have same VM public IP and VM private IP

Azure

- The WEBSITE INSTANCE ID environment variable

Google

- No information
- Procfs does not contain global usage statistics



IO based Coresidency Test

(1) Set up N distinct functions $f_1, \ldots, f_N$ that run the following task upon receiving a `RUN` message: record `/proc/diskstats`, write 20 K – 30 K times to a file (1 byte each time), and record `/proc/diskstats` again.

(2) Invoke each function once without `RUN` message to launch N function instances.

(3) Assuming the instances of $f_1, \ldots, f_k$ (k instances) share the same instance root ID, invoke $f_1, \ldots, f_k$ once each with the `RUN` message and examine I/O statistics of each function instance.

Figure 3: I/O-based coresidency test in **AWS**.

Tenant Isolation

tenant = 1 user account

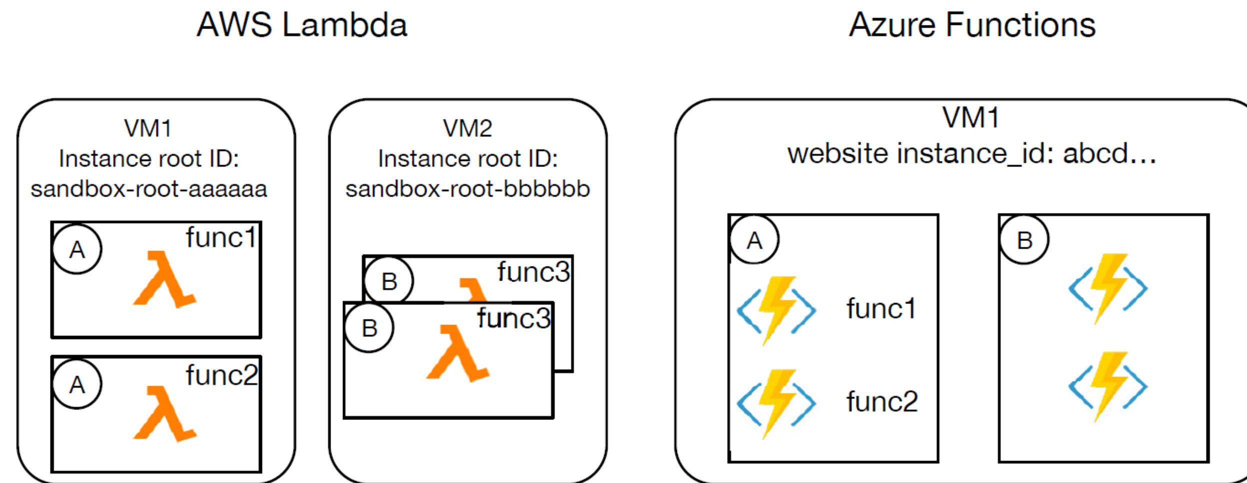


Figure 2: VM and function instance organization in AWS Lambda and Azure Functions. A rectangle represents a function instance. A or B indicates different tenants.

As of May 2018, different tenants have different VM's in Azure

VM Configurations

AWS

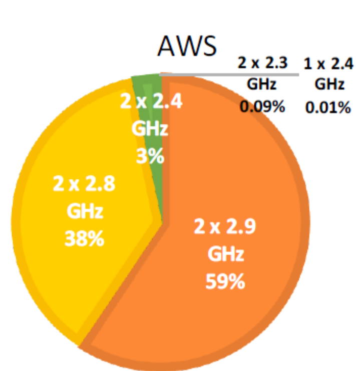
- Use procfs file to read global statistics
- VMs can have 1, 2 or 4 vCPUs

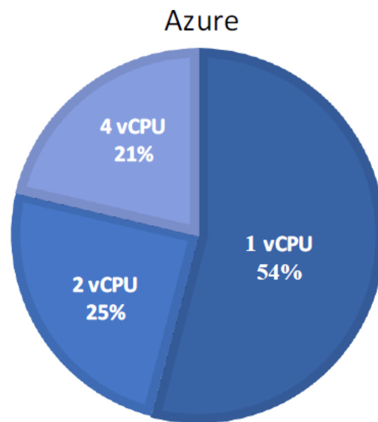
Azure

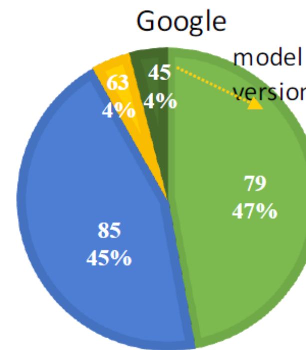
- Environment variables collected suggest the host VMs can have 1, 2 or 4 CPUs.

Google

- Isolates and filters information that can be accessed from procfs
- many system files and syscalls are obscured
- /proc/meminfo and /proc/cpuinfo files suggest a function instance has 2GB RAM and 8 vCPUs







Different types of VMs could result in different instance performance

Serverless Introduction

Methodology

Results

- Serverless Architecture
- Resource Scheduling
- Performance Isolation
- Bugs

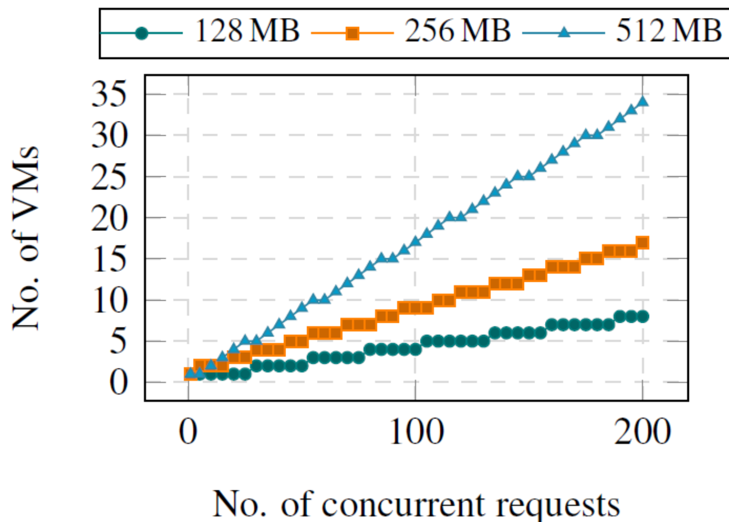
Summary

Can the platforms effectively handle concurrent requests

Methodology: send N concurrent requests and examine the number of instances running concurrently

AWS

- AWS could easily scale up to 200 functions
- Max **3328 MB** memory per VM



Azure

- Only 10 instances
- Most of the functions coresident on 1 vCPU VM
- Vulnerable to attacks (Fixed in May 2018)

Google

- Half of expected instances

How long does it take to launch an instance?

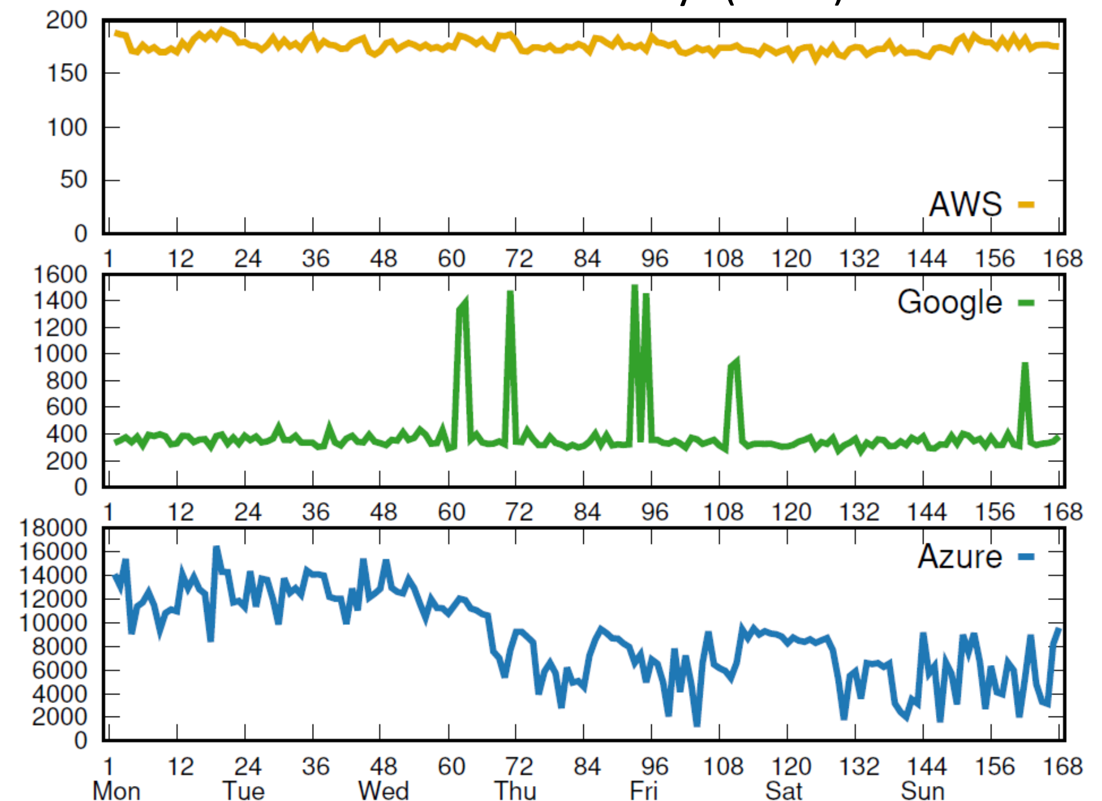
Median coldstart latency of 1000 instances

AWS: 160 ms
has a pool of ready VMs

Google: 500 ms (2017)
→2000 ms (2018)

Azure: 3600 ms (2017)
→ 300 ms (2018)

Median coldstart latency(ms) per hour over 7 days (2017)



Instance lifetime

There is a trade-off between long and short idle time, as maintaining more idle instances is a waste of VM memory resources, while fewer ready-to-serve instances cause more coldstarts.

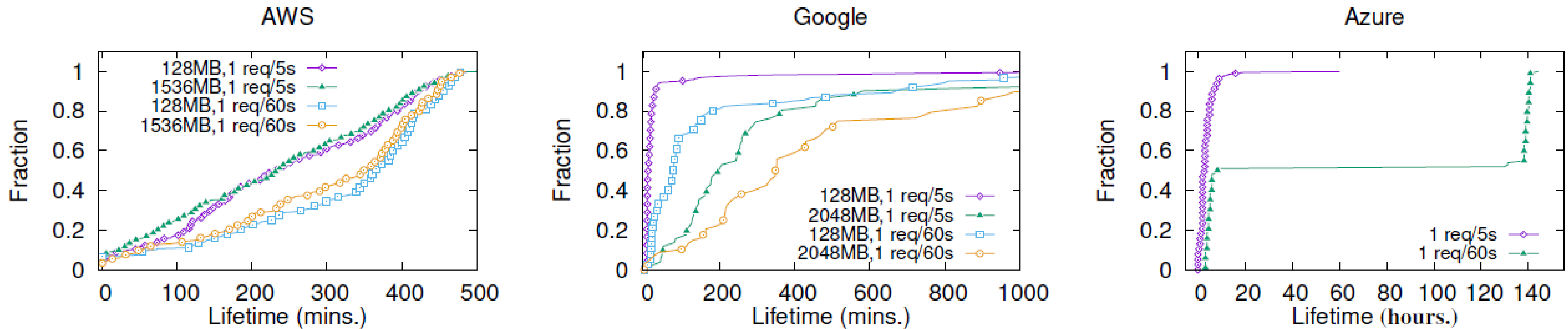


Figure 9: The CDFs of instance lifetime in AWS, Google, and Azure under different memory and request frequency.

- Serverless Introduction
- Methodology
- Results
    - Serverless Architecture
    - Resource Scheduling
    - Performance Isolation
    - Bugs
- Summary

Performance Isolation

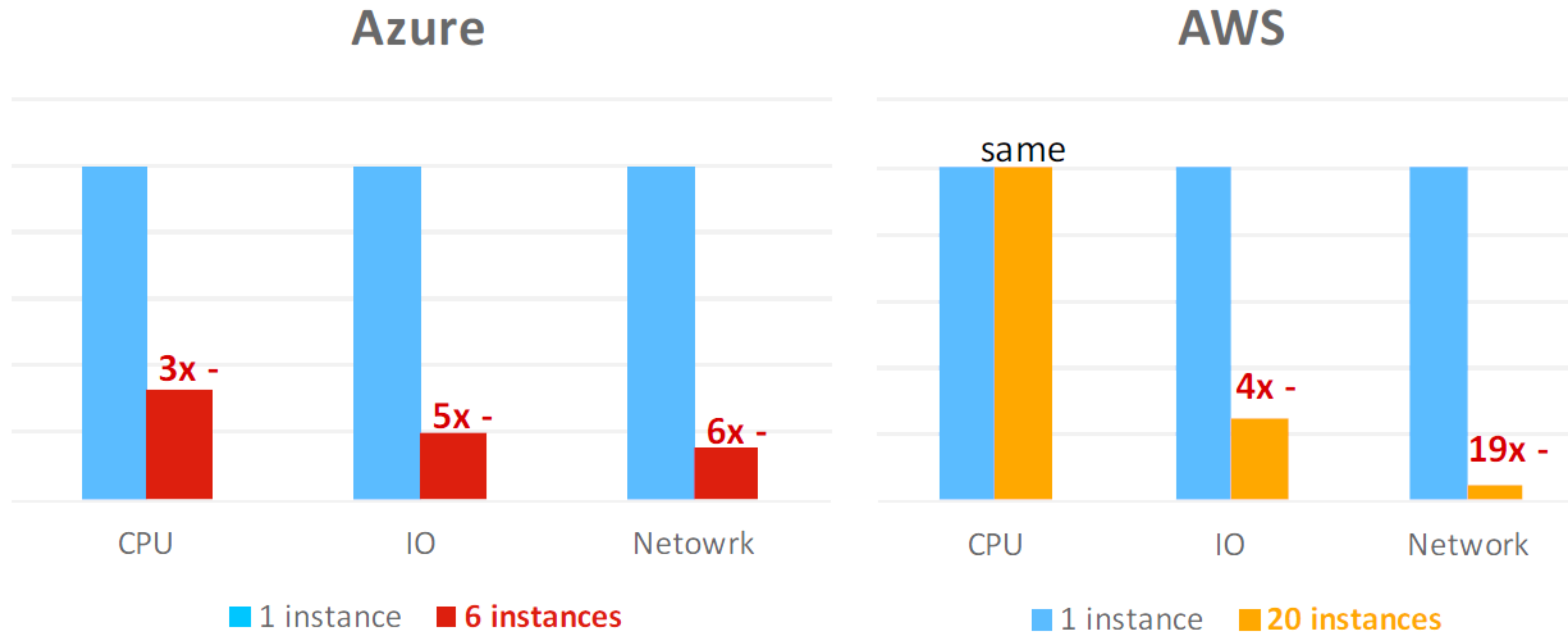
CPU share: Fraction of 1000-ms time period for which the instance can use CPU

IO throughput: Write 512 KB of data to the local disk 1,000 times (via dd or scripts)

Network throughput: Use iperf3 to run the throughput test for 10 seconds

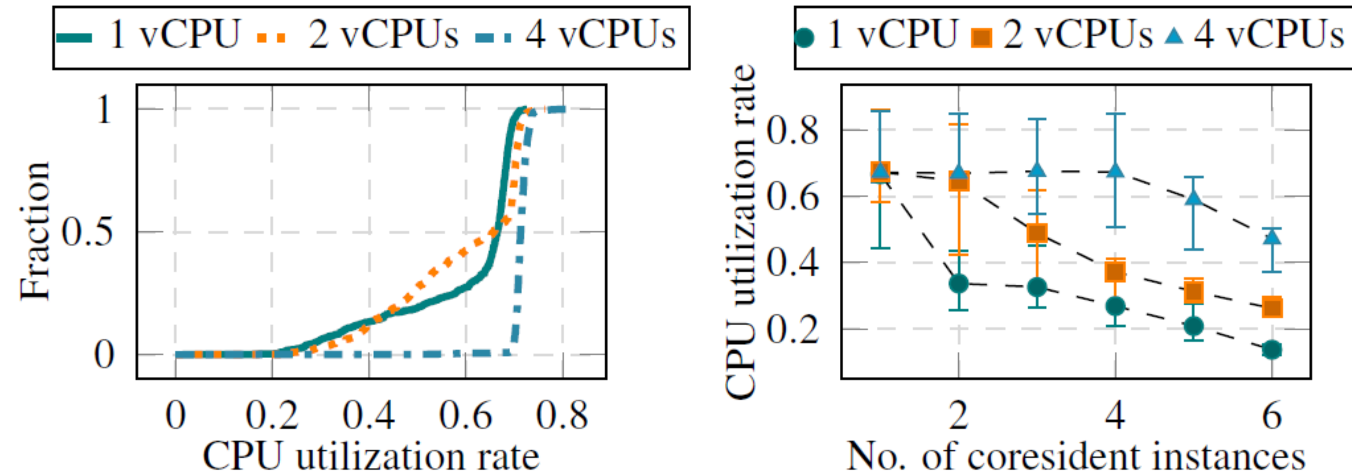
	AWS	Azure	Google
Coresidency	Yes	Yes	Unknown
VM Configuration	No	Yes	No

Coresidency



Resources are allocated per VM More co-residency decreases resources per function

VM configuration (AZURE)



(a) Azure: CPU utilization CDF (b) Azure: CPU vs. coresideny

4-vCPU VMs get **1.5x** IO throughput, **2x** network throughput, and more CPU than other types of VMs

Serverless Introduction

Methodology

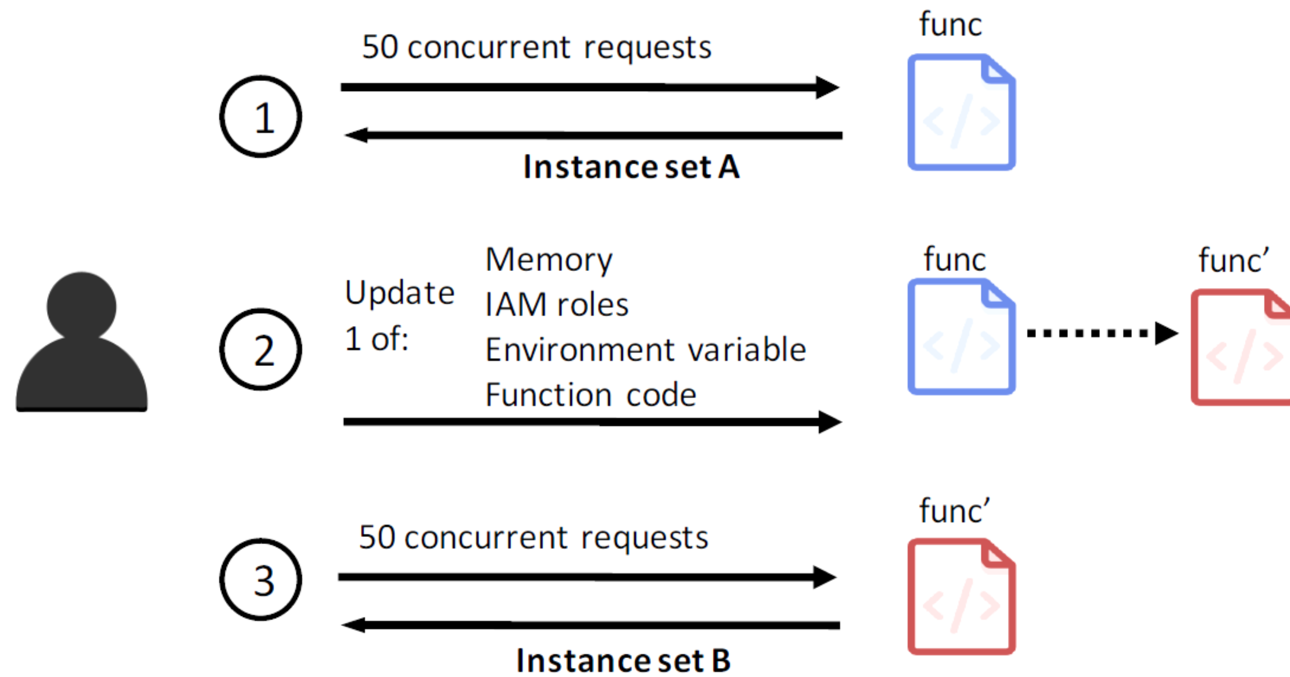
Results

- Serverless Architecture
- Resource Scheduling
- Performance Isolation
- Bugs

Summary

AWS: Function Consistency

Inconsistent behaviour: requests handled by an old version of the function



AWS: Inconsistent function usage

3.8% (out of 20K) ran an inconsistent or outdated function

- Case 1: New instances ran outdated functions (0.1%)
- Case 2: Requests handled by the instances for outdated functions (3.7%)

Inconsistent responses to users

Google: Stealthy background process

Processes can run after function invocation concluded

Nodejs will execute line B without waiting for user_task returns

```
exports.handler = function handler(req, res) {
// run asynchronous task here.
line A: user_task();
// send back results.
line B: res.status(http_code).send(user_data);
}
```

Processes can stay alive for to 21 hours

- No billing → **Use extra resources for free!**

Serverless Introduction

Methodology

Results

- Serverless Architecture
- Resource Scheduling
- Platform Isolation
- Bugs

Summary

Summary

In-depth measurement study that discover various issues in three serverless computing platforms

- Unpredictable performance
- Bad performance isolation
- Consistency issues

Performance baselines and design considerations for future design of serverless platforms

Thank You