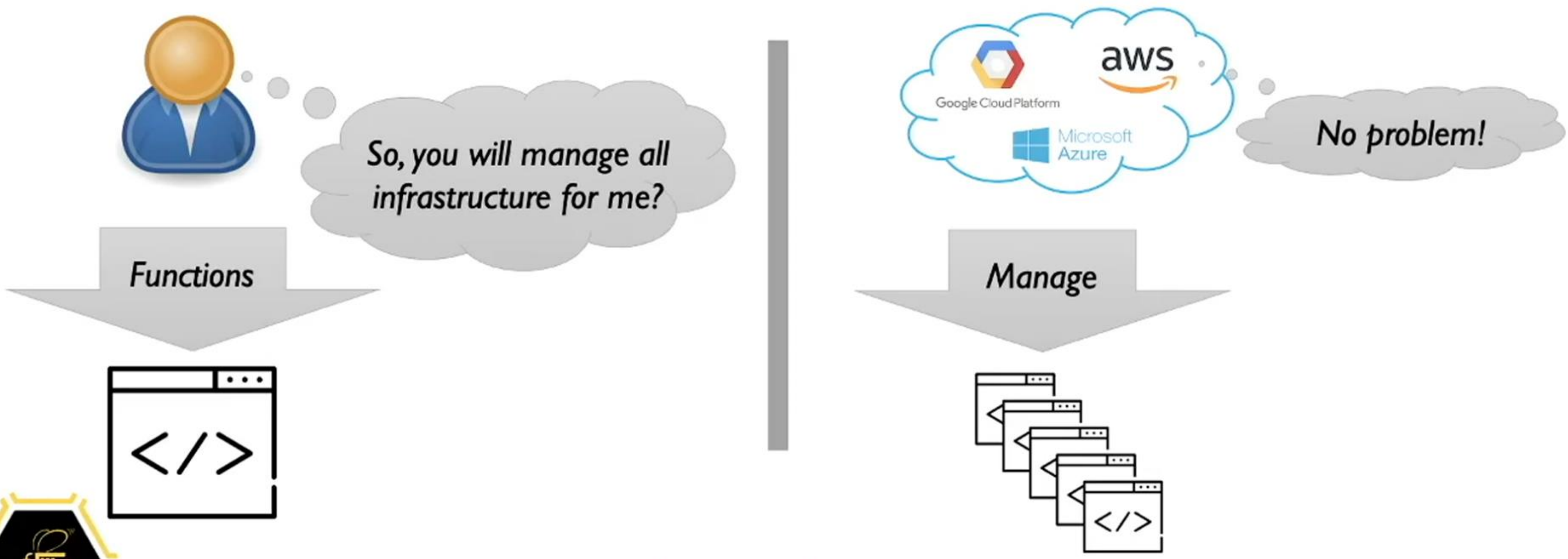
Benchmarking, Analysis, and Optimization of Serverless Function Snapshots

Dmitrii Ustiugov,

Plamen Petrov, Marios Kogias, Edouard Bugnion, Boris Grot

Why Users Love Serverless

Happy serverless user | **Serverless providers**



How to study serverless systems?

Studying Serverless: State-of-the-Art Frameworks

THE UNIVERSITY of EDINBURGH informatics

Bleeding-edge but **proprietary** systems
- Complex distributed software stack

Incomplete or **non-representative**
- Single component, e.g., hypervisor
- Container isolation only (e.g., OpenWhisk, OpenLambda)
  - but >70% of providers (AWS, Azure, Google) rely on VMs

vHive

Need for a complete open-source framework for serverless research

Serverless in the Age of Open Source

THE UNIVERSITY *of* EDINBURGH
informatics

Kubernetes + **Knative**

Cluster scheduler & Function-as-a-Service API (Google & CNCF)

containerd

Host management (CNCF)

Firecracker

MicroVM (AWS Lambda)

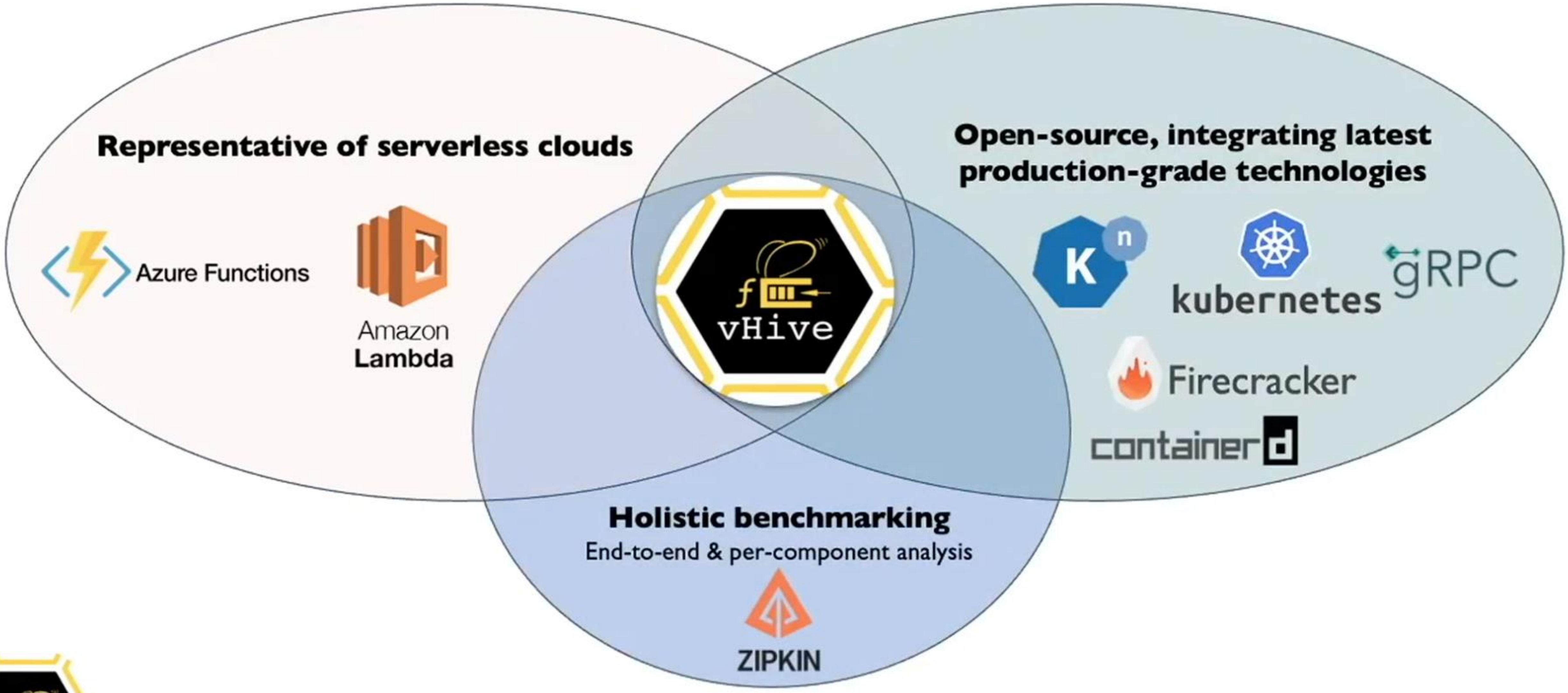
gRPC

Communication fabric (Google)

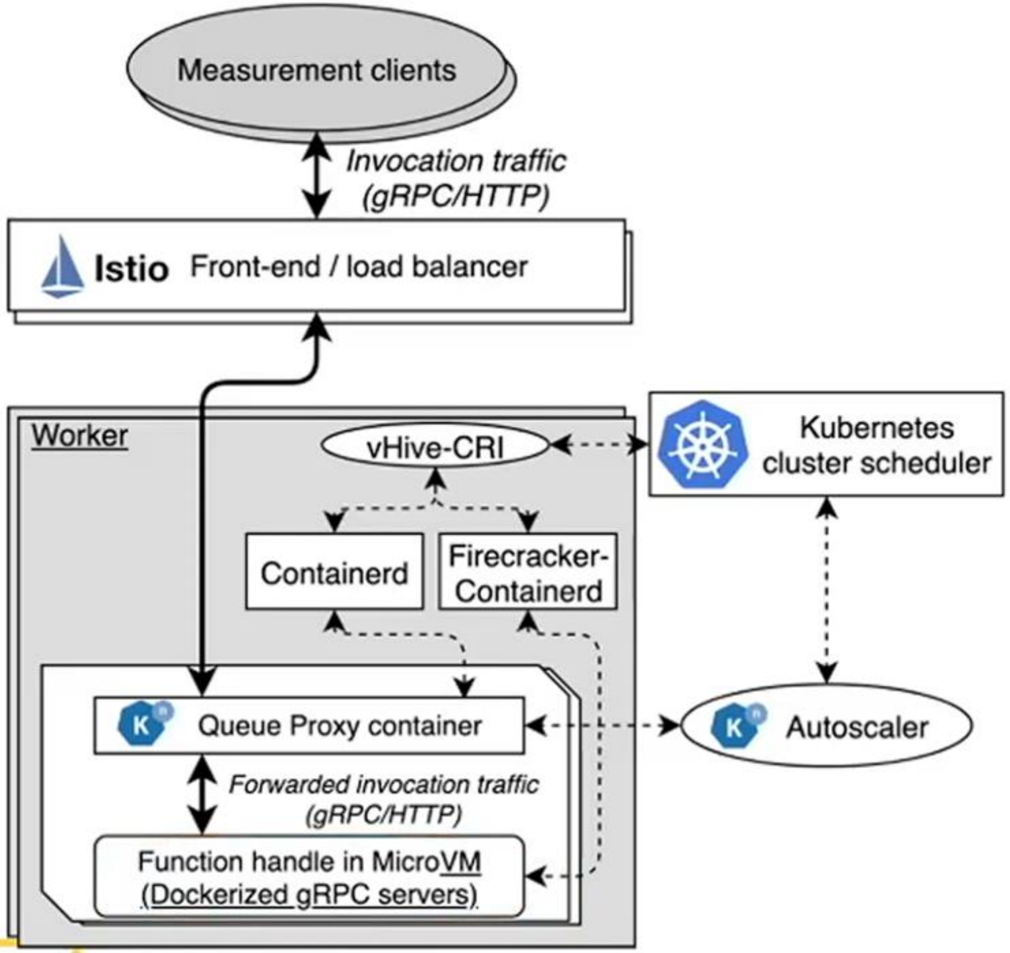
vHive

vHive: Framework for Serverless Experimentation

THE UNIVERSITY of EDINBURGH
informatics



vHive-CRI Integration



Load and **latency** measurement clients

Istio as load balancer

Kubernetes cluster scheduler

A function instance deployed as a Kubernetes pod, including

- **Queue-proxy** container (per function instance)
  - Monitors per-instance queue depth
  - Drives function autoscaling
- Firecracker **MicroVM** with a function handle

First to support snapshotting at scale

vHive integrates all serverless components in an open-source research framework

Characterizing Cold Starts with vHive

Why Providers… Struggle with Serverless

THE UNIVERSITY *of* EDINBURGH **informatics**

Happy serverless user

Can I invoke my functions ***once per hour****?*

and… they run for ***1 second***

and… they should be ***fast*** *&* ***cheap****!*

Serverless providers

vHive

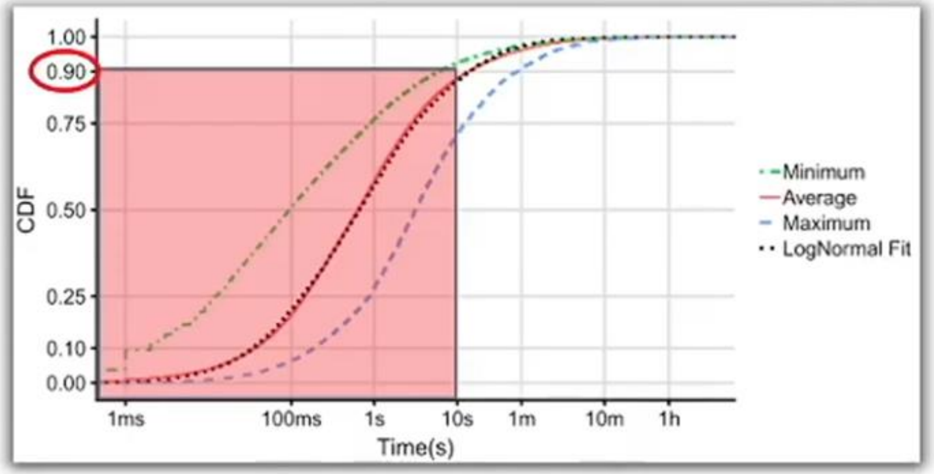
How common are rare and short function invocations in serverless?

FaaS Characteristics [Azure Functions, ATC'20]

Functions are **short** (user code)

- 670ms on average
- 90% execute for <10 seconds

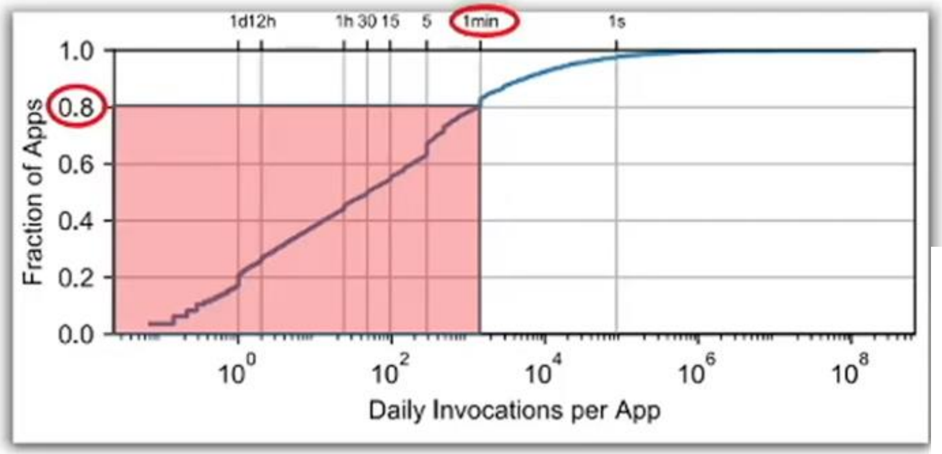
Function execution time (user code)



The majority of functions are rare ("**cold**")

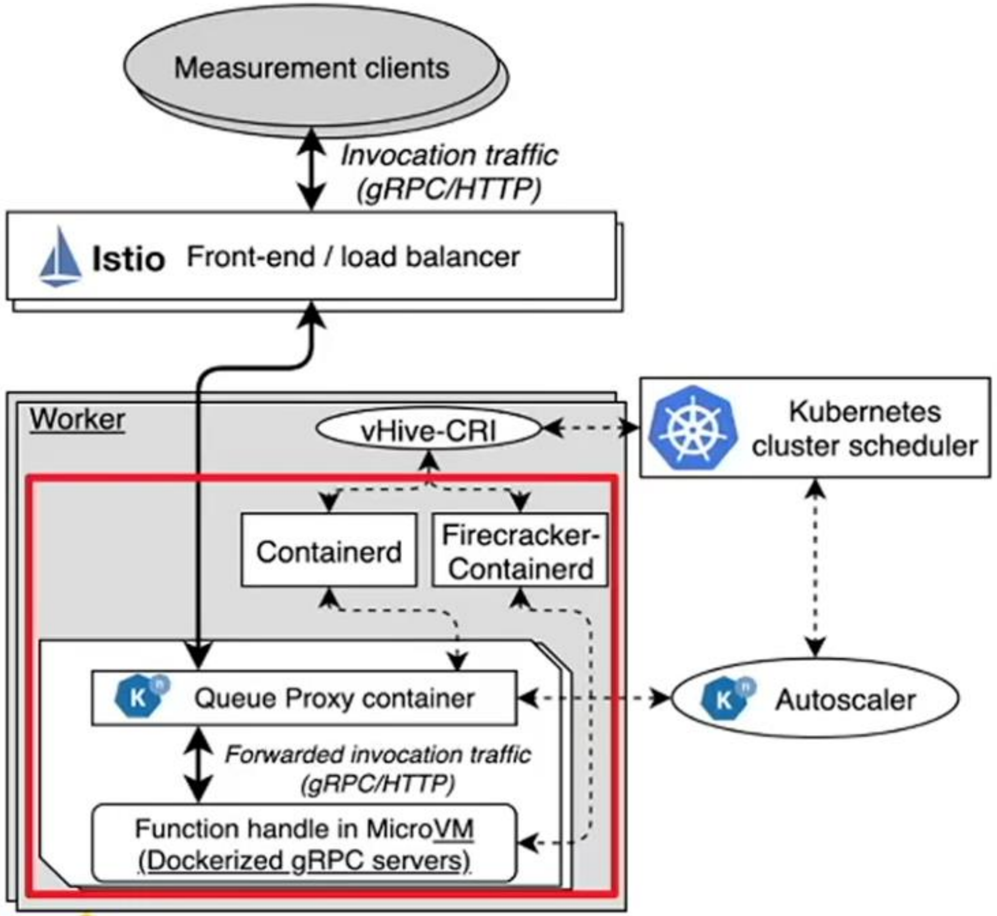
- 80% invoked less than once per minute

Average interval between function invocations



Short and cold functions are dominant

Why Cold Starts are Slow?



Cluster delays are low (<20ms)

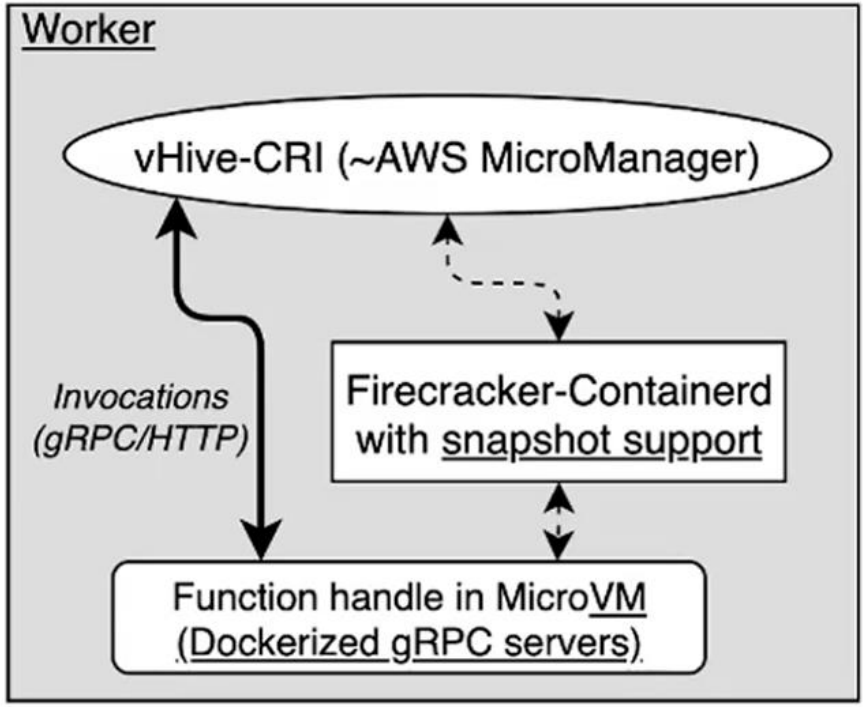
- Corroborating [Firecracker, NSDI'20]

Worker-internal delays dominate (helloworld, Python)

- Boot-based cold start: >2 seconds
- Firecracker snapshots: 100s of milliseconds

Cold start delays dominated by internal worker delays

Evaluating Worker-Internal Delays



Goal: Careful modelling of a single worker, similar to AWS Lambda

- MicroManager terminates connections to MicroVMs & Front-end

vHive single-node configuration

- MicroManager injects the invocation traffic to function instances

Extended Firecracker-Containerd to support VM snapshots

Firecracker Snapshotting Support

Function instance is snapshotted **after** function server initialization

Firecracker snapshots implementation follows Catalyzer [ASPLOS'20]

The procedure of loading a VM from a snapshot includes:

1. **Loads** the state of the VM monitor (VMM), virtual NICs and disks
2. **Mmaps** the guest memory file **without** populating its contents
3. **Resumes** function execution from the point of snapshotting
4. **Restores the connection** between the function server and the MicroManager

How fast is Firecracker snapshotting for cold functions?

Methodology: Serverless Characterization with vHive

THE UNIVERSITY of EDINBURGH informatics

Host specs

- 48-core Haswell Xeon, Linux v4.15 (Ubuntu 18)
- Snapshots stored on a local SSD (SATA3 850MB/sec)
- Large inputs (e.g., videos) stored in a MinIO object store

MicroVM specs

- Linux v4.14 (Alpine), 1 vCPU, 256MB RAM

Functions adopted from FunctionBench [SoCC'19]

- Wide range of single-function serverless workloads

Emulating cold invocations

- **Assumption**: guest memory pages evicted from memory
- **Modelling**: flush the host-OS' page cache after invocation

Evaluated functions from FunctionBench [SoCC'19]

Name	Description
helloworld	Minimal function
chameleon	HTML table rendering
pyaes	Text encryption with an AES block-cipher
image_rotate	JPEG image rotation
json_serdes	JSON serialization and de-serialization
lr_serving	Review analysis, serving (logistic regr., Scikit)
cnn_serving	Image classification (CNN, TensorFlow)
rnn_serving	Names sequence generation (RNN, PyTorch)
lr_training	Review analysis, training (logistic regr., Scikit)
video_processing	Applies gray-scale effect (OpenCV)

Function Memory Usage Characterization

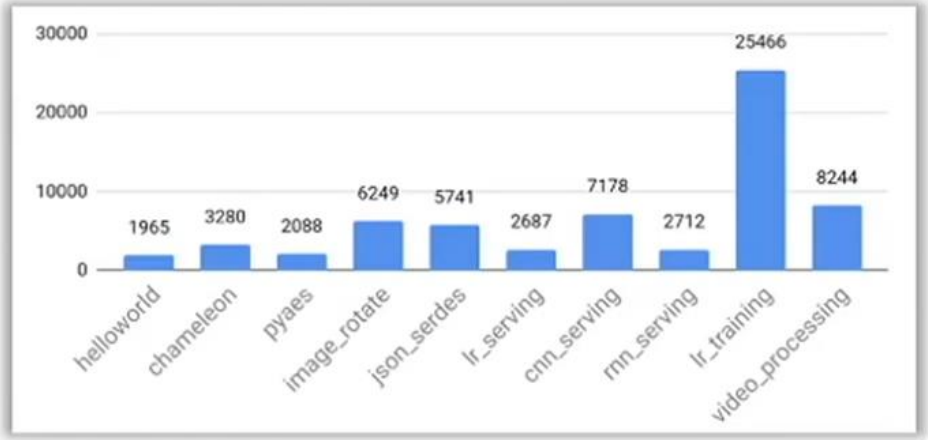
Functions have a **non-negligible** memory footprint

- High-level languages: Libraries and modules
- High infrastructure tax: gRPC fabric, kernel code, …

Recall: Snapshots rely on **lazy paging**

- Guest memory (file) is mapped but not populated with contents
- Page faults result in **20x slowdown** (avg)
    - **Serial**: Page faults occur one at a time
    - **No spatial locality**: Pages are scattered across the guest memory

Number of page faults during a single invocation



Observation: Serial & sparse disk accesses slow down function execution

- Linux run-ahead prefetching is inefficient due to the lack of locality

Page faults dominate snapshot-based cold invocation latency

Key Insight: Function Working Sets are Stable

THE UNIVERSITY of EDINBURGH
informatics

Study: Trace page faults with `userfaultfd`
(stock Linux user-level page fault handling mechanism)

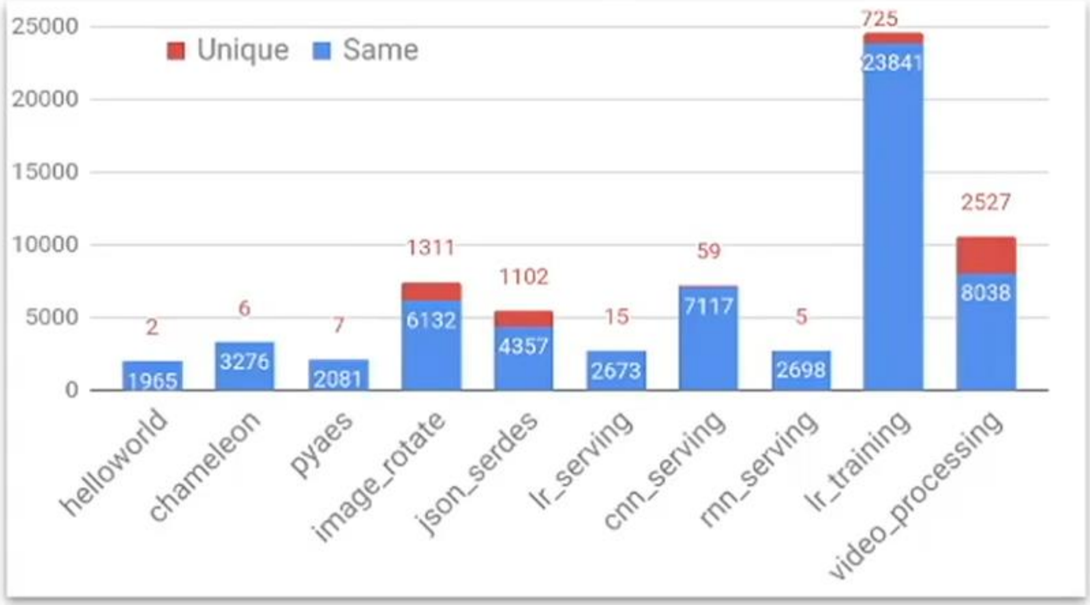
Memory footprint is **non-trivial**

- Functions touch **8-99MB** upon each invocation

Key: Function working sets are **stable** across invocations

- Same language runtime, libraries, guest networking stack, …
- **76-99%** of pages are the same, even with different inputs!

Memory footprint, number of pages



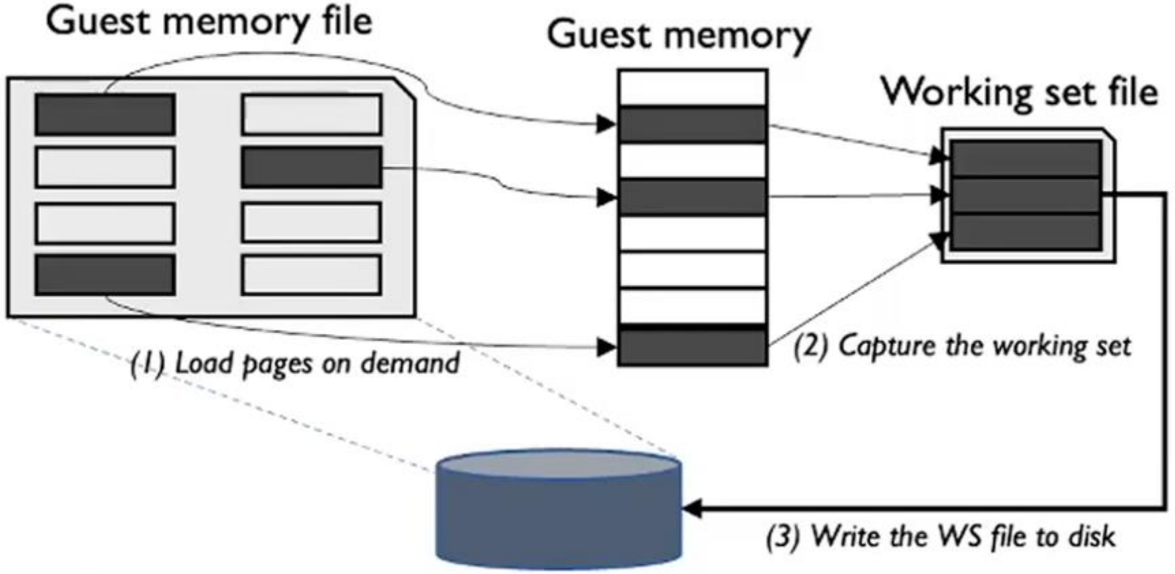
vHive

Idea: Record and prefetch the working set pages

REcord-And-Prefetch (REAP) Snapshots

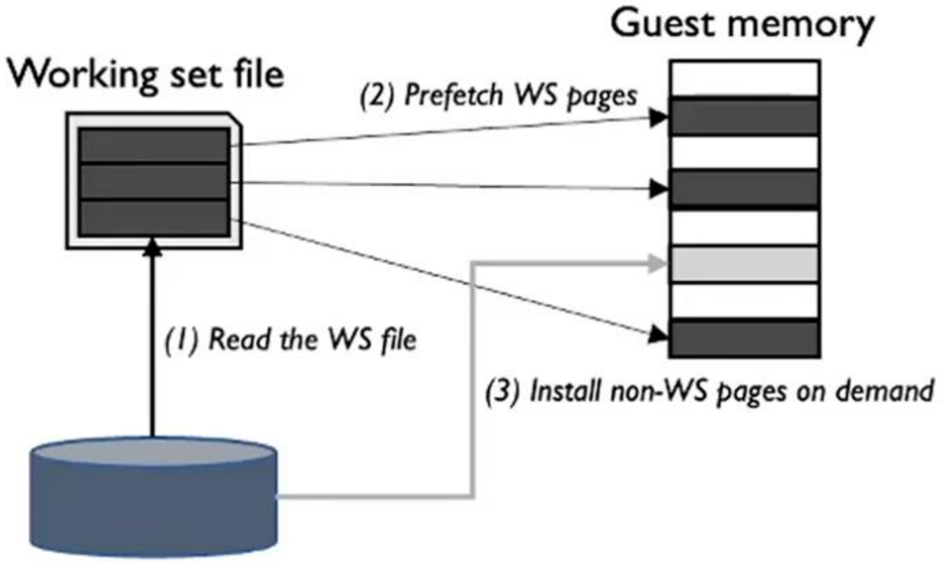
Record phase (1st invocation)

1. Intercept page faults with Linux `userfaultfd`
2. Capture working set (WS) pages in a compact file
3. Write the WS file to disk (SSD, HDD, AWS S3, …)



Prefetch phase (2^{nd} and future invocations)

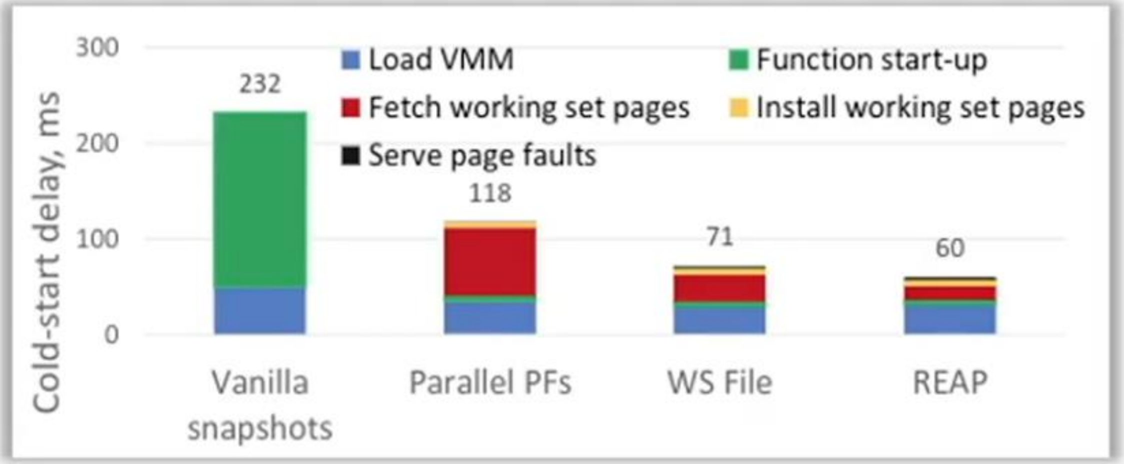
1. Read the WS file from the disk
2. Prefetch **all** WS pages into the guest memory
   - Also, install the page mappings into the host page tables
3. Install **missing,** non-WS, pages **on demand**



REAP trades off a little extra storage for faster cold starts

Evaluation: Optimization Steps (helloworld)

Single cold function invocation latency (prefetch phase)



Vanilla snapshots: Load VMM and serial page fault processing

- Serial major page faults are slow, extracting **<5%** of the peak SSD BW

Parallel page faults: Fetch WS pages from large guest memory file

- Many accesses to scattered locations in the SSD

WS file: Fetch WS pages from a compact WS file

- Host filesystem limits SSD read bandwidth

REAP: Fetch from a WS file & bypass host OS page cache

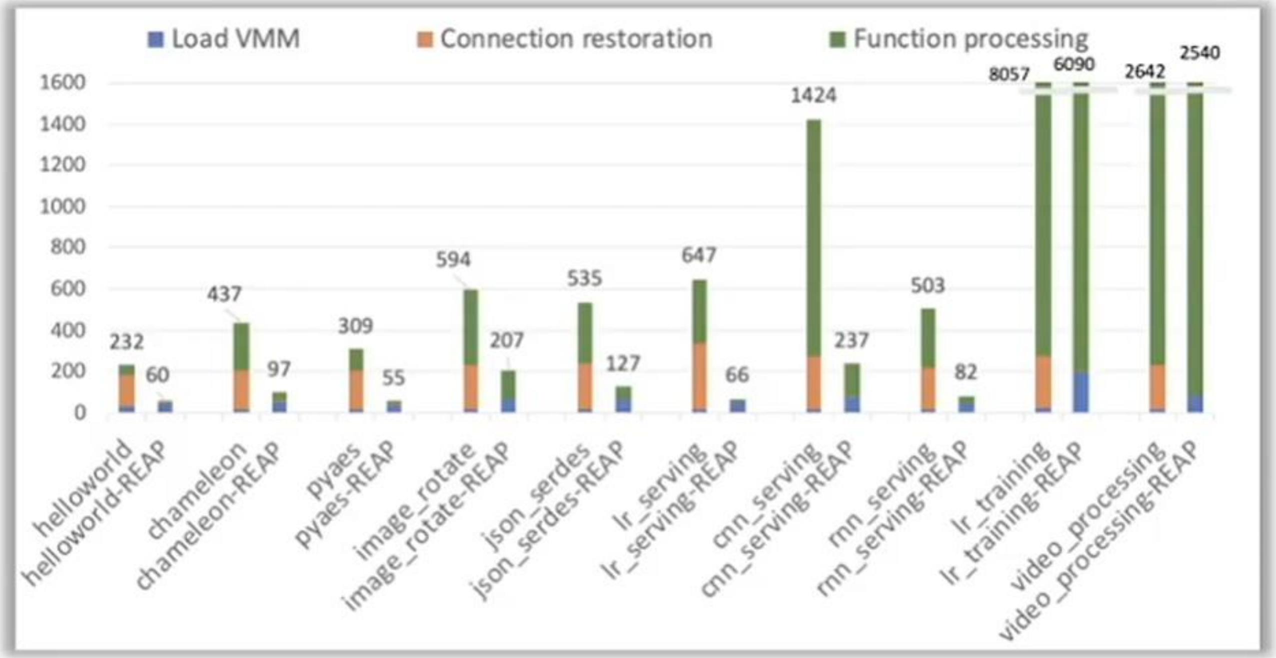
- Achieving **63%** of the peak SSD bandwidth

REAP accelerates cold starts by optimizing SSD accesses

Evaluation: FunctionBench [SOCC'19]

THE UNIVERSITY of EDINBURGH informatics

Single function cold start latency, ms
(left bars: Firecracker snapshots, right bars: REAP)



REAP slashes connection restoration by **45x**

- Efficient prefetching of gRPC & network stack

Function processing reduced by **4.5x** (avg)

- Exception: video_processing, likely due to OpenCV's memory allocation depending on video aspect ratio

3.7x faster cold invocations, on average

Takeaways

We introduce the open-source **vHive** framework for serverless experimentation

Key insight: A function uses the **same** guest memory pages **across** invocations

We introduce **<u>RE</u>cord-<u>A</u>nd-<u>P</u>refetch (REAP)** technique

- Record working set (WS) pages upon 1^{st} invocation, prefetch upon future invocations
  - Reduces the cold-start latency by **3.7x** (avg), by eliminating **97%** of page faults
- **Seamless** integration with Firecracker and Containerd (<250LoC)
  - Entirely in user space and infrastructure agnostic

Join the vHive Open-Source Community

https://github.com/ease-lab/vhive

Slack: firecracker-microvm.slack.com, channel: #firecracker-vhive-research

Academic contributors:

Industrial collaborators: