THE UNIVERSITY of EDINBURGH
informatics

Benchmarking, Analysis, and Optimization of Serverless Function Snapshots

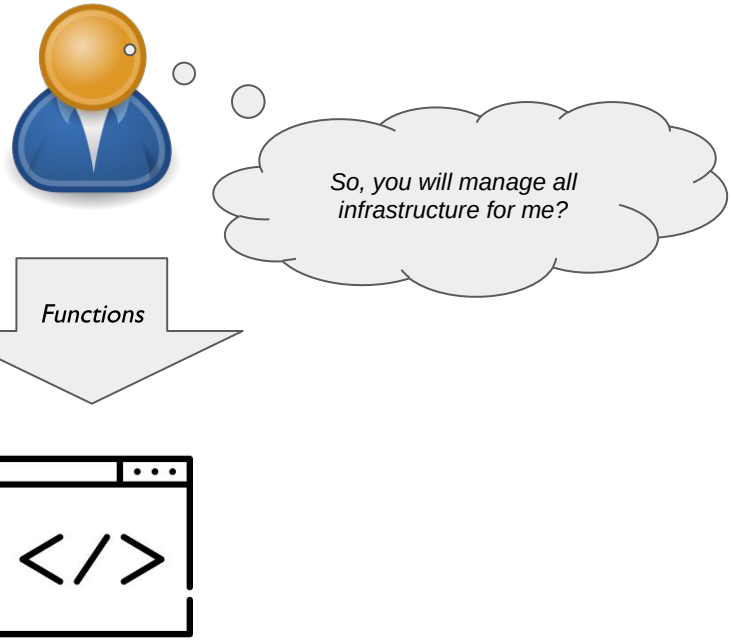
Dmitrii Ustiugov

Plamen Petrov (now at Huawei), Marios Kogias (Microsoft Research), Edouard Bugnion (EPFL), Boris Grot

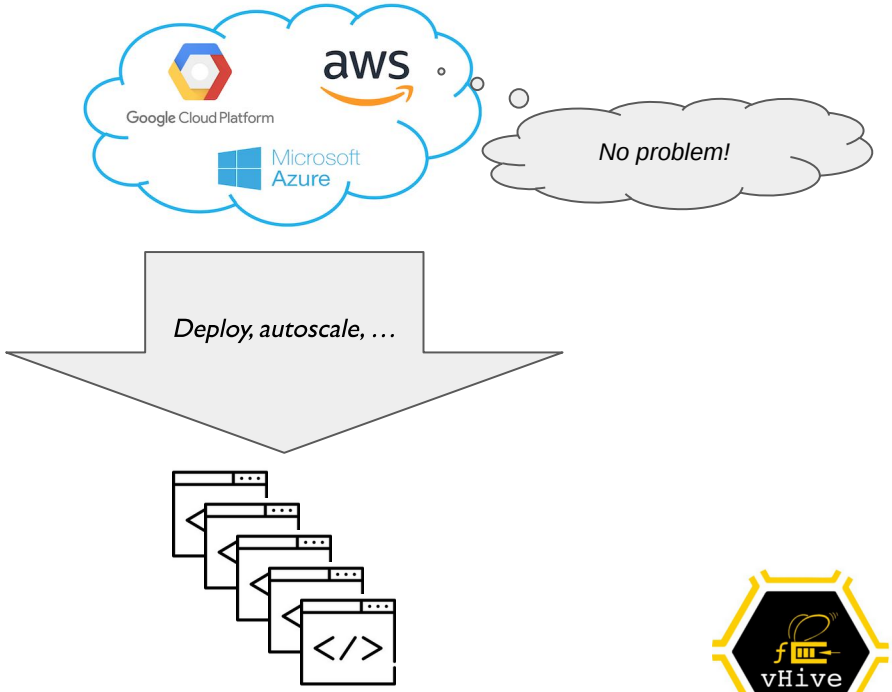
EASE lab at the University of Edinburgh

Why Users Love Serverless

Happy serverless user

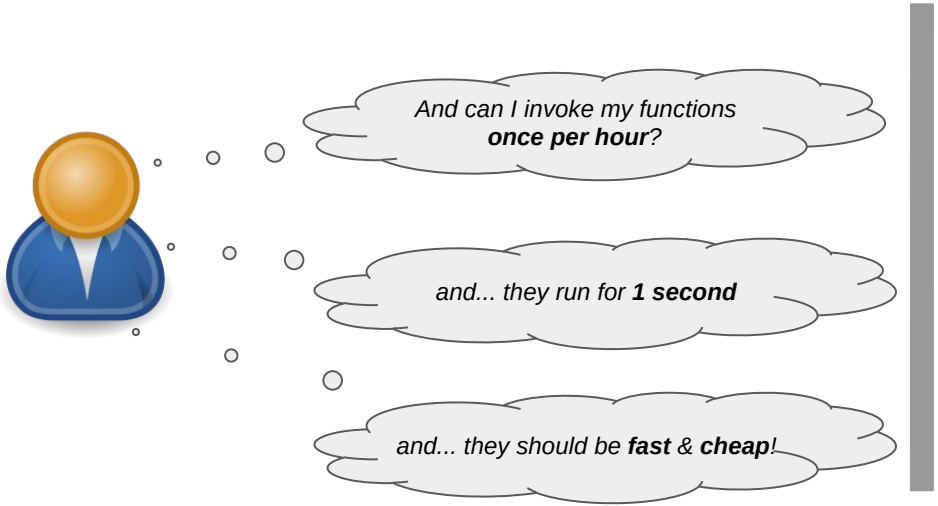


Serverless providers



Why Providers… Struggle with Serverless

Happy serverless user



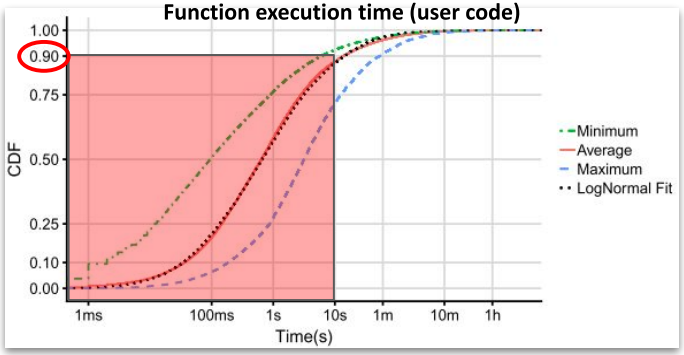
Serverless providers

Is this common in real clouds?

FaaS Characteristics [Azure, ATC'20]

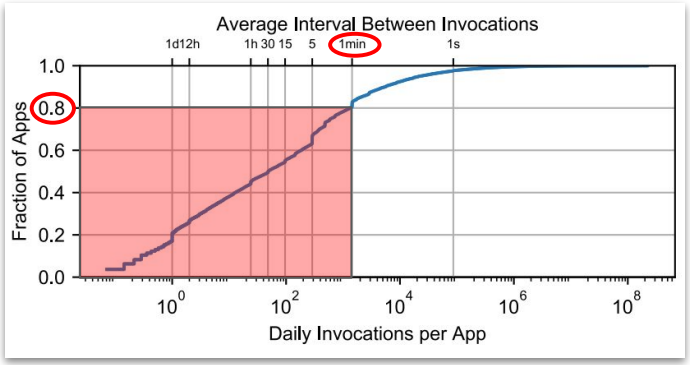
Functions are **short** (user code)

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



Majority of functions are **cold**

- 80% invoked less than once per minute



Short & cold functions are dominant

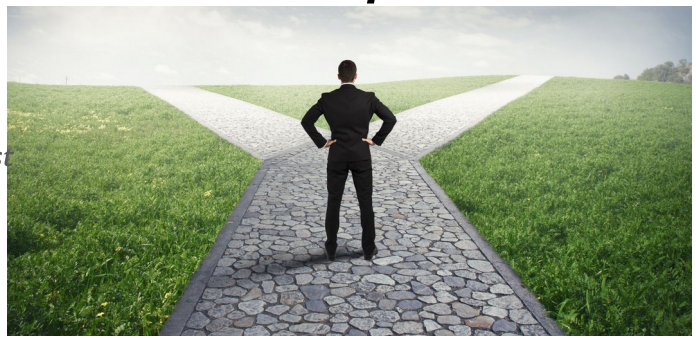
Cold Functions Dilemma: Keep-Alive or Shutdown?

Keep-alive

Idle instances = Expensive!

Memory is up to 50% of a modern server's cost

Serverless providers



Cold starts

Cold starts = Slow!

Cold start ≈ mean execution time

Cold start reduction is key for low-cost & fast serverless

Dmitrii Ustiugov - EASE Lab

How to Investigate Cold Starts in Academia?

Academics have no access to real deployments

Serverless cloud infrastructure is complex

- Deep & distributed software stack

Performance overheads are various

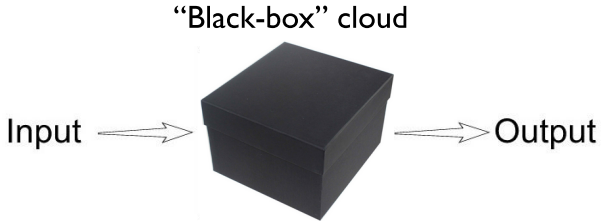
- Networking delays
- Cluster management delays
- Function isolation (sandbox) delays
- …

Research in serverless requires a *holistic* benchmarking framework

Studying Serverless: What Tools We Have Today

Academia: Incomplete or non-representative

- Single-component (e.g., hypervisor, scheduling)
- Support Docker only (e.g., OpenLambda)



Leading providers: Proprietary infrastructure

- Examples: AWS Lambda, Azure Functions

All: No holistic and per-component performance analysis

Serverless in the Age of Open Source

Kubernetes

Knative

Serverless & cluster orchestrator (Google & CNCF)

Firecracker

MicroVM (AWS Lambda)

gRPC

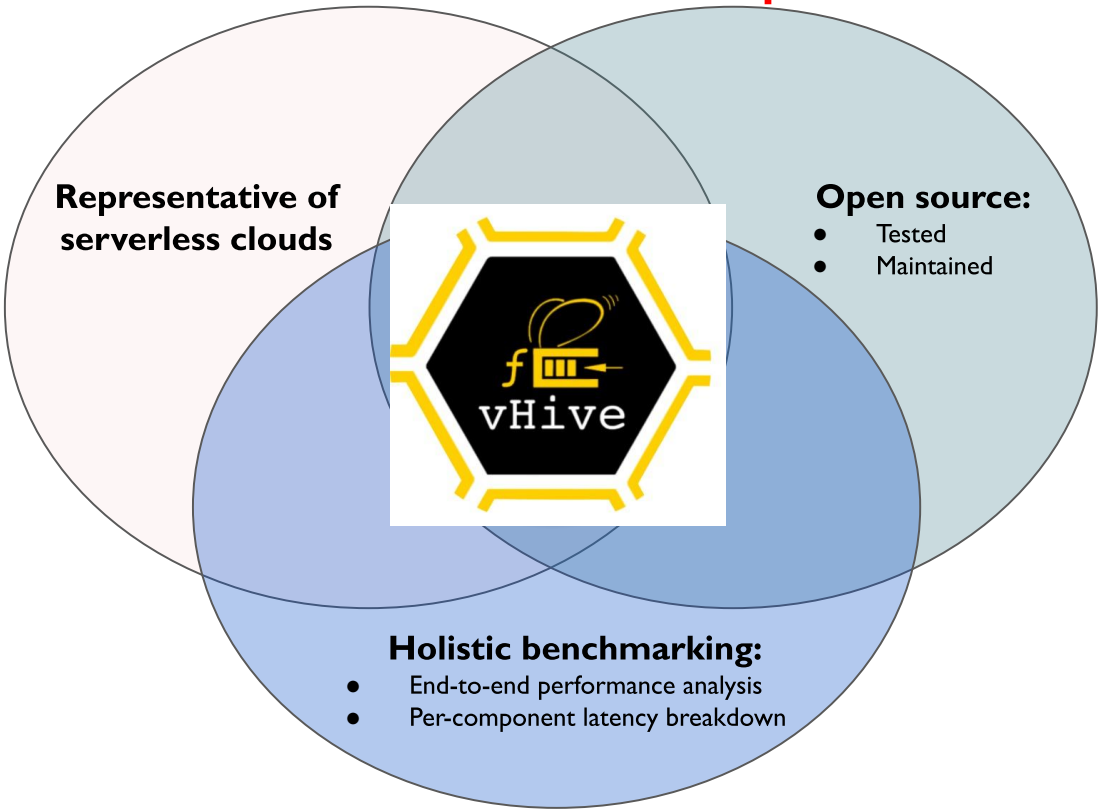
Communication fabric (Google)

containerd

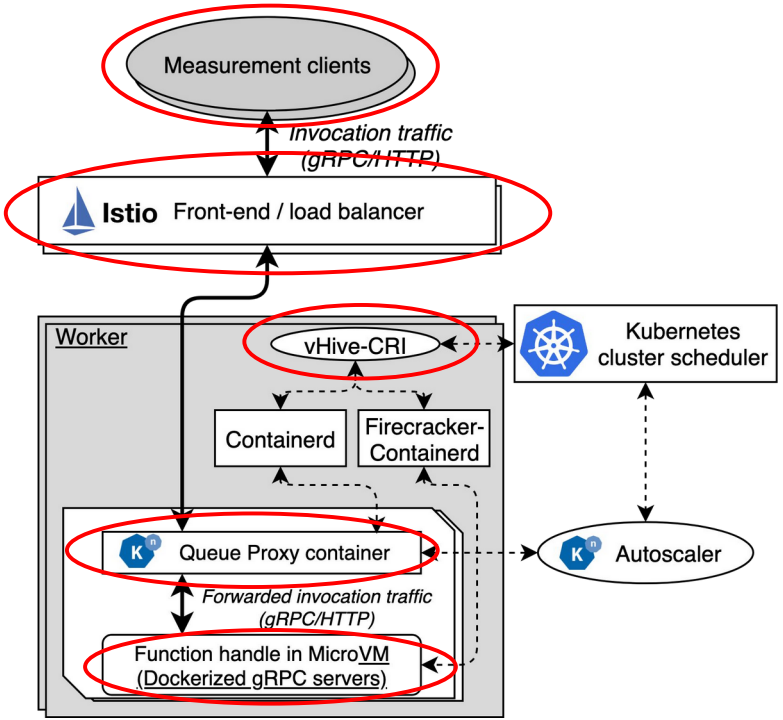
Host management (CNCF)

Dmitrii Ustiugov - EASE Lab

vHive: Framework for Serverless Experimentation



vHive-CRI Integration



Load and latency measurement clients

Istio as a load balancer / front-end service

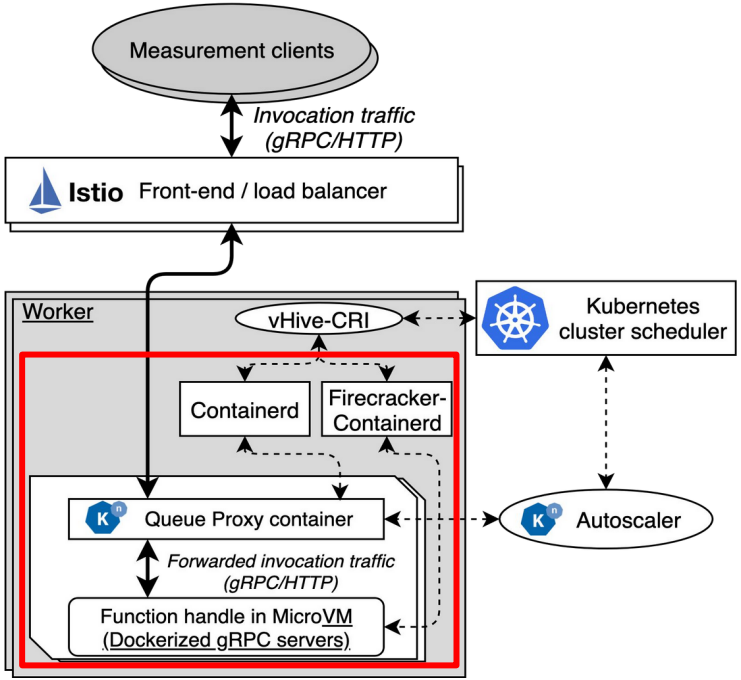
A function instance deployed as a Kubernetes pod, including

- **Queue-proxy** container (per-function instance)
  - Monitors per-instance queue depth
  - Drives function autoscaling
- Function in a Firecracker **MicroVM**
  - With snapshotting support

vHive to enable innovation across entire serverless infrastructure

Characterizing Cold Starts with vHive

Why Cold Starts are Slow?



Cluster delays are low (<10ms)

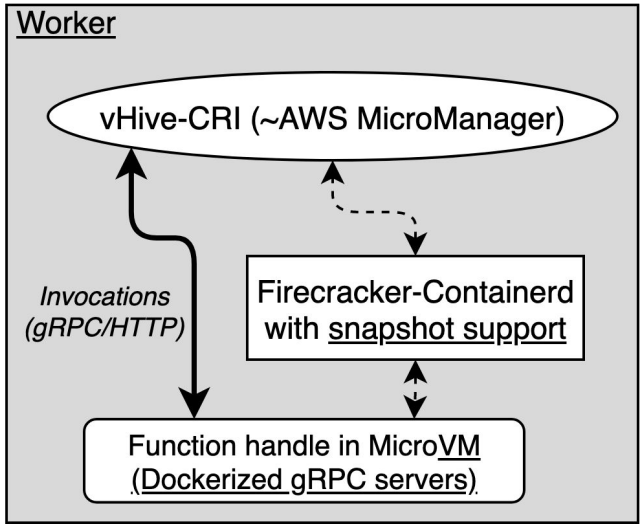
- Corroborating [Firecracker, NSDI'20]

Worker-internal delays dominate (helloworld function)

- 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 modeling of a single worker, similar to AWS Lambda

- MicroManager terminates connections to MicroVMs & Front-end

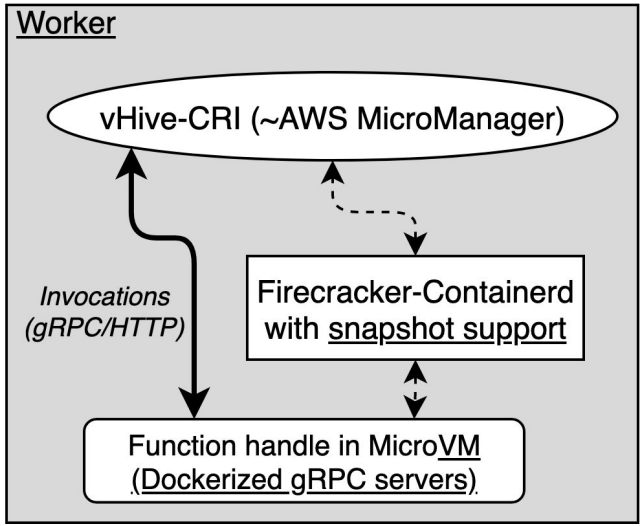
vHive single-node configuration

- Invocation traffic injected by MicroManager

Emulating cold invocations

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

Firecracker Snapshotting Support



Function instance snapshotted when it is ready to serve invocations

On a cold start, a function instance is loaded from a snapshot

1. Loads and restores the VMM state
2. Maps the guest memory file into memory **without** populating its contents
3. Restores MicroManager-to-Function connection

How fast is Firecracker snapshotting for cold functions?

Methodology: Serverless Characterization w/ vHive

Host specs

- 48-core Haswell Xeon
- Ubuntu 18, v4.15
- Snapshots stored on a **local SSD** (SATA3 850MB/sec)
- Large inputs (e.g., images) stored in MinIO S3

MicroVM specs: Alpine, v4.14, 1 vCPU, 256MB RAM

Functions adopted from FunctionBench [SoCC'19]

- Dockerized as gRPC servers

Evaluated functions

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)

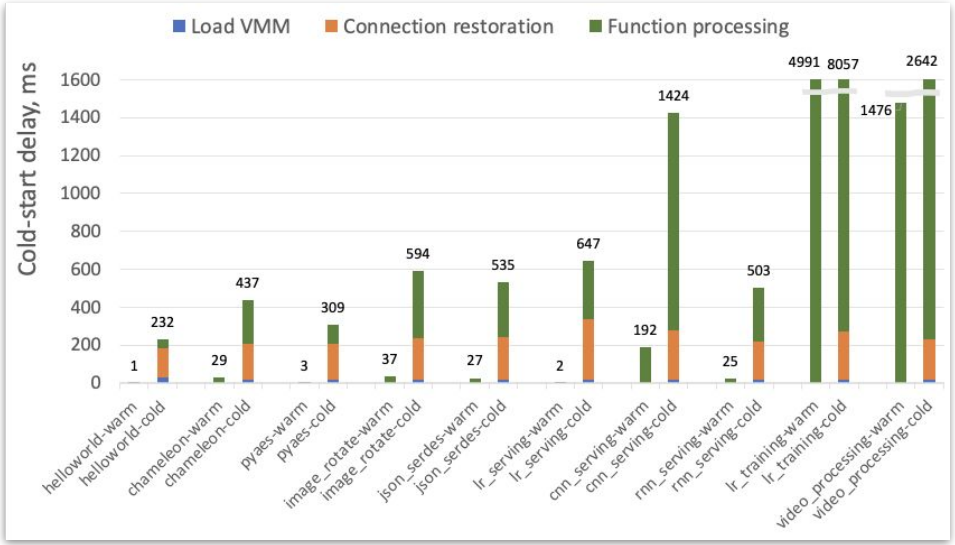
Cold Invocation Delay: Snapshot-Based

Cold start delays dominated by:

- Connection restoration
- Useful function processing

Key: Cold function processing is ~**10x** slower than for warm invocation

Function instance warm (left bars) and cold start latency (right bars)



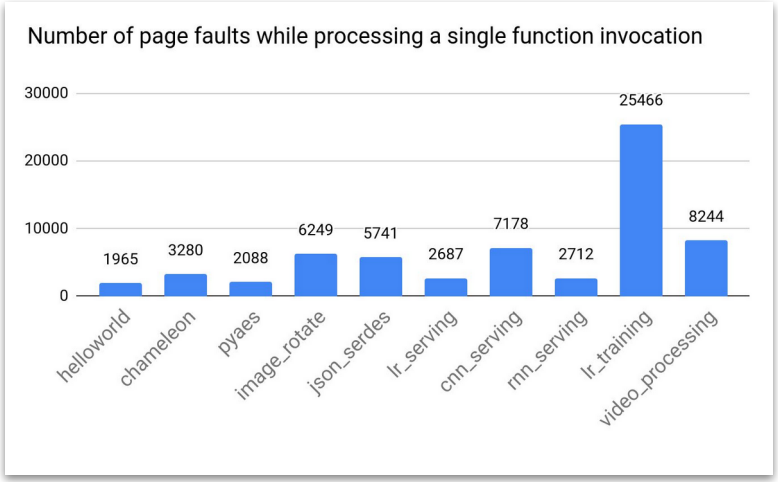
What slows function processing down?

Dmitrii Ustiugov - EASE Lab

Function Memory Usage Characterization

Functions use non-negligible amount of memory

- Libraries and modules of high-level languages
- High infrastructure tax
  - E.g., gRPC fabric, guest OS network code



Recall: Snapshots rely on **lazy paging**

- Guest memory (file) is mapped but **not populated** with its contents
- **95%**, on average, of the cold start delay is in serving page faults
  - **Serial** and mostly **major** page faults

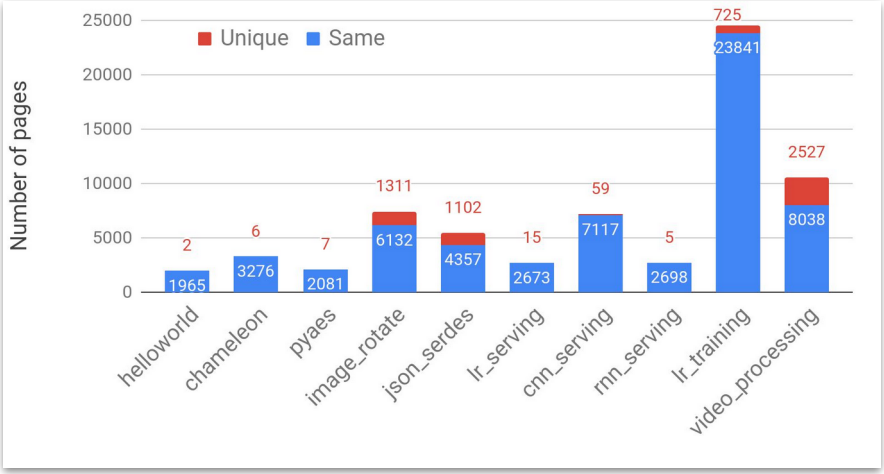
Page faults dominate snapshot-based cold invocation latency

Key Insight: Function Working Sets are Stable

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

Memory footprint

- Functions touch 8-99MB upon each invocation
- 76-99% of pages are the **same across** function invocations (with different input)



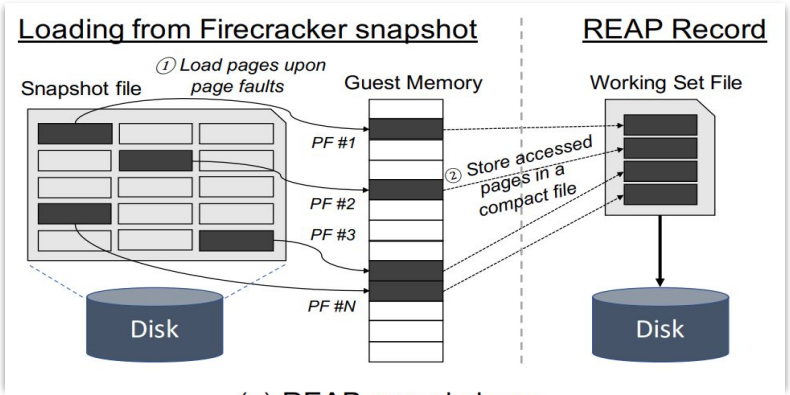
Idea: Record and prefetch the working set pages

Dmitrii Ustiugov - EASE Lab

REcord-And-Prefetch (REAP)

Record phase (1st invocation)

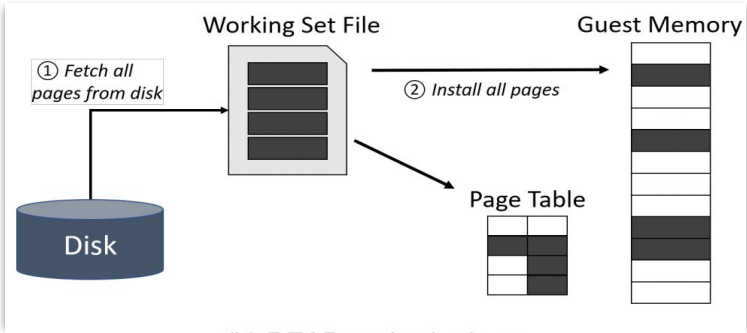
- Intercept individual page faults
- Store working set (WS) in a compact file
  - To maximize SSD read throughput upon load



(a) REAP record phase.

Prefetch phase (2nd and further invocations)

- Prefetch **all WS pages** into guest memory
- Install page mappings into the host page tables
- Install missing (non-WS) pages **on demand**



(b) REAP prefetch phase.

REAP accelerates cold invocations from 2nd invocation and on

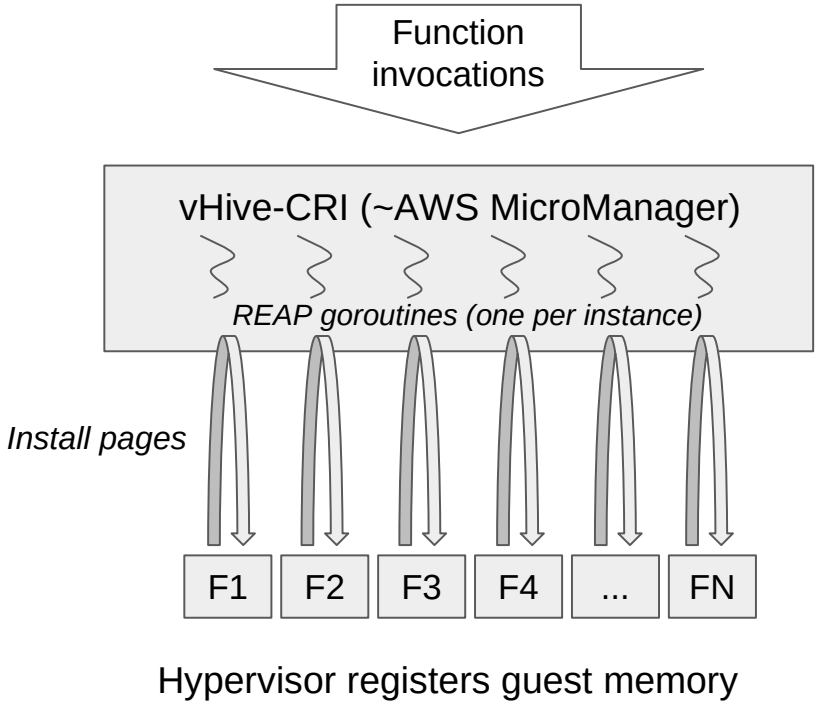
REAP Implementation

Design goals

- Independent of serverless infrastructure
- No modification to host & guest kernel
- Scale to 100s of cold-starting VMs

REAP mechanisms implemented in the MicroManager

- Hypervisor registers guest memory for user page faults
- MicroManager spawns a goroutine per function instance
  - These goroutines poll and serve user page faults

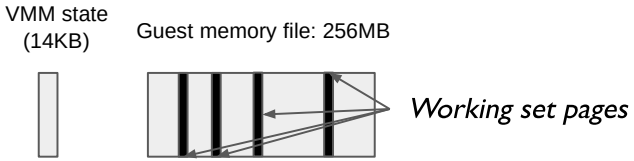


<250 LoC changes to Firecracker & Containerd (discounting public crates)

REAP Goroutine Operation

Dmitrii Ustiugov - EASE Lab

Firecracker VM (host process)



1. Load VMM state
2. mmap(PRIVATE|ANONYMOUS)
3. Register for user page faults, get a file descriptor (fd)
4. Pass the fd over Unix socket

REAP goroutine (per function instance)

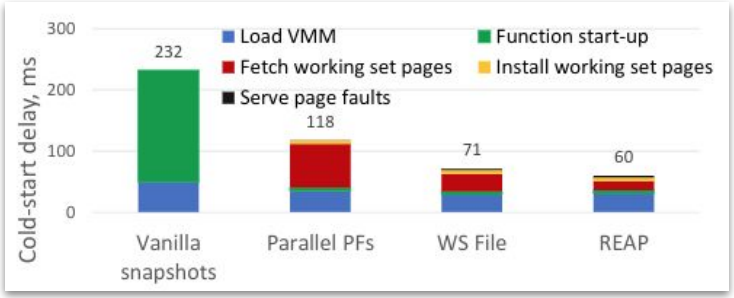
5. Receive the fd over socket
6. Epoll on the fd

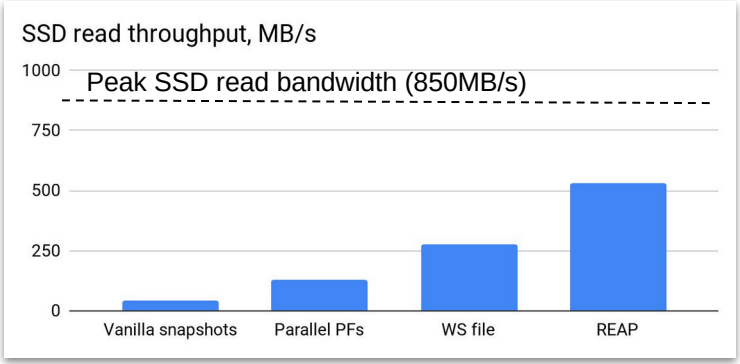
7-∞.

A. Install pages via IOCTL(src, dst, len)
B. Wake up the vCPU

Evaluation: Optimization Steps (helloworld)

Single cold function invocation (prefetch phase)





Vanilla snapshots: Load VMM and serial page fault processing

- Serial major page faults are slow

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

- Many SSD accesses to scattered locations in SSD

WS file: Fetch WS pages from a compact WS file

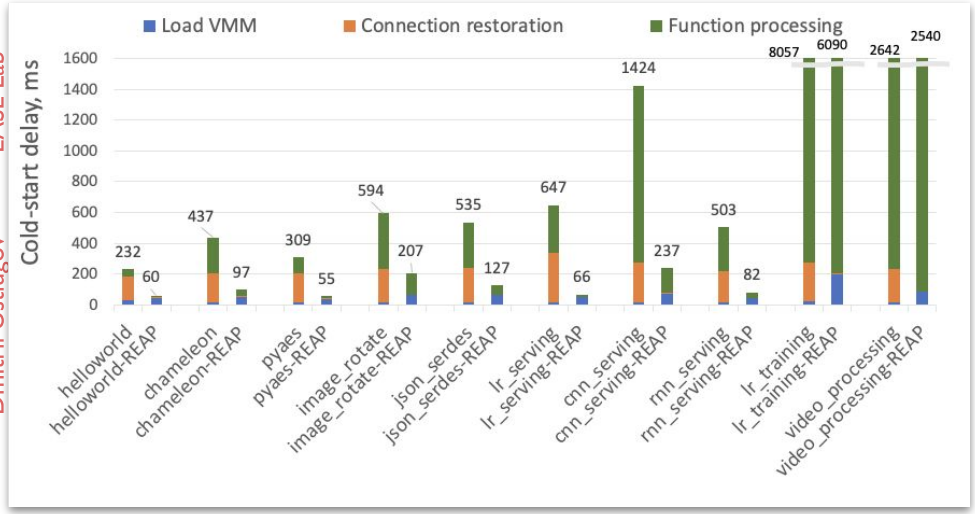
- Host filesystem limits SSD read bandwidth

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

Evaluation: FunctionBench [SOCC'19]

Single function cold start

(left bars: Firecracker snaps, right bars: REAP)



REAP slashes connection restoration by **45x** to 4-5ms

- Efficient prefetching of gRPC & network stack

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

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

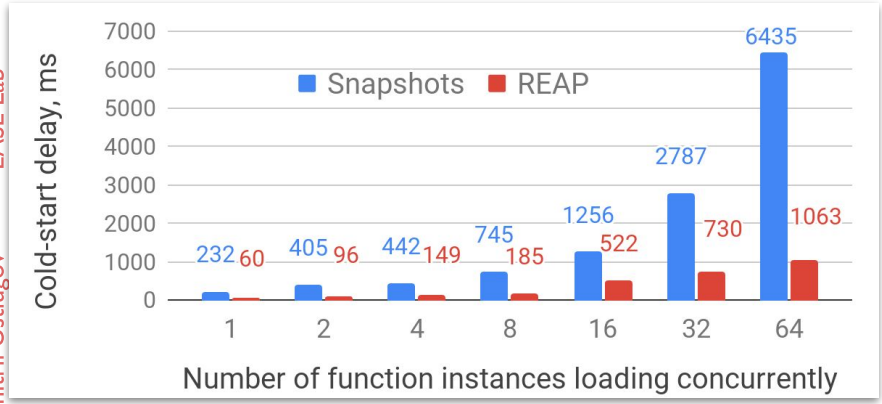
Note: Warm invocations do not impact cold start delays

- Results within 5% with 20 co-runnng hot functions

On average, **3.7x** faster cold invocations

Evaluation: Concurrent Cold Invocations (helloworld)

Average start latency of concurrently starting instances



REAP cold start delay grow sublinearly with concurrency

REAP is faster at scale as it extracts higher SSD throughput

- Firecracker snapshots: 30-80 MB/sec
- REAP: 120-500 MB/sec

REAP becomes SSD-bandwidth bound with >16 instances

REAP shows better scalability **and** lower latency

Takeaways

Most functions are invoked **sporadically**, resulting in **cold invocations**

We build **vHive** benchmarking platform to enable **holistic** serverless research

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

We introduce **REcord-And-Prefetch (REAP)** technique

- Record working set (WS) pages upon 1st invocation, prefetch upon further invocations
  - **3.7x** cold invocation speedup, on average
- Implementation: Userspace, infrastructure agnostic, scalable

Thank you!

On behalf of the vHive team at EASE lab at University of Edinburgh

<u>Supervisor</u>

Prof. Boris Grot

<u>PhD student & leader</u>

Dmitrii Ustiugov

<u>Undergrad students at Uni of Edinburgh</u>

Theodor Amariucai Yuchen Neu

<u>Full-time intern</u>

Shyam Jesalpura, BITS, India

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

Backup

Programming with KNative Functions (in vHive)

To deploy a function, user provides:

- OCI (Docker) image
  - Template gRPC server & function-specific handle
- YAML file for function configuration

```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  namespace: default
spec:
  template:
    spec:
      containers:
        - image: crccheck/hello-world:latest # Stub image. See https://github.com/ease-lab/vhive/issues/68
          ports:
            - name: h2c # For GRPC support
              containerPort: 50051
          env:
            - name: GUEST_PORT # Port on which the firecracker-containerd container is accepting requests
              value: "50051"
            - name: GUEST_IMAGE # Container image to use for firecracker-containerd container
              value: "vhiveease/helloworld:var_workload"
```

KNative provides a URL handle per function

Easy to compose functions and regular services (e.g., S3)

Towards REAP Adoption in Public Cloud

Dmitrii Ustiugov - EASE Lab

Security implications

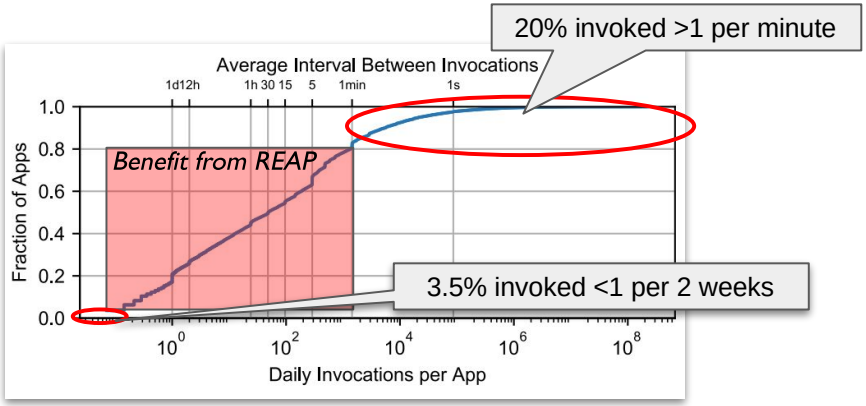
(Any) snapshotting impairs

- Entropy of random generators [Firecracker repo]
- Guest-OS ASLR

Mitigation strategies for ASLR:

- Option 1: Periodic snapshot re-generation (repeating record phases)
- Option 2: Run-time guest memory randomization
  - Prefetch WS pages but to randomized locations
  - Requires guest page table rewriting upon cold start

Applicability to real-world functions [Azure, ATC'20]



REAP should accelerate **>77%** of functions

Note: Functions with complex control flow may not benefit