How to make the most out of computers

Lixiang Ao

OS Meetup

Aug 6 2022

OSes provide…

Abstraction

Multiplexing

OSes provide…

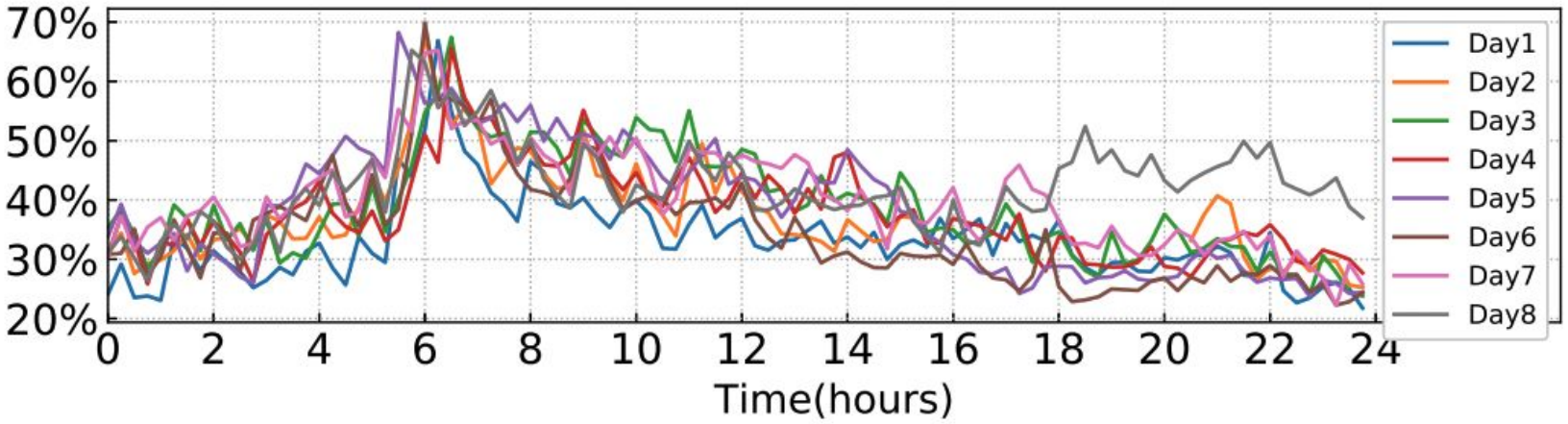
Abstraction: make life easier for applications

Multiplexing: do more things at the same time

Computers do not have enough things to do

PCs/servers are idle most of the time. Average CPU utilization as low as 20%.

Computing power is reserved for **peak demand**, which happens very occasionally



Alibaba Datacenter Trace CPU utilization [IWQoS '19]

Computers do not have enough things to do

What if you can rent out your computer time when you're not using them?

What if you run a huge e-commerce company that uses thousands of servers.

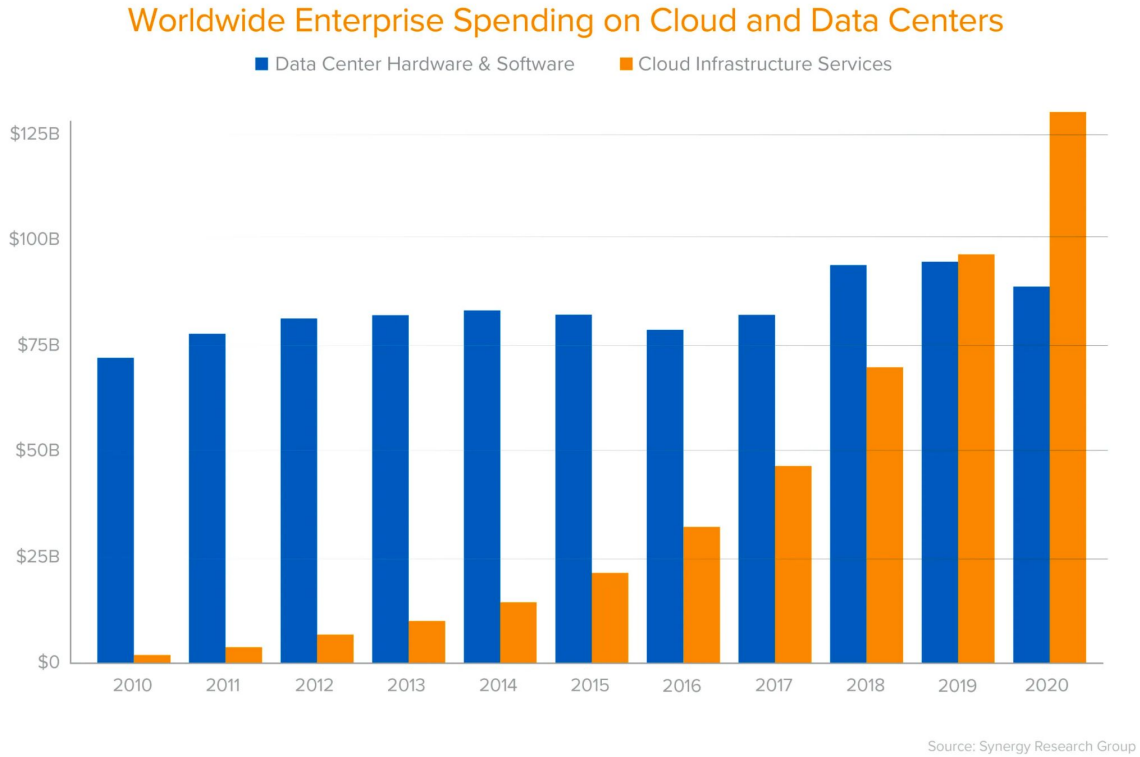
Just like landlords, you charge your "server tenants" by the time they use your servers.

Cloud computing

In 2006, Amazon launches EC2 (IaaS), renting out their idle servers that are sometimes only 10% utilized.

People were skeptical in the beginning.

Now IaaS is a $120 b / yr market.



Wait a minute…

Don’t you worry your tenants would mess up your computers?

Tenants interfere with each other.

Viruses, worms, backdoors, ransomware…

You need **isolation**.

Don’t OSes Already Provide Isolation?

Process abstraction provides CPU isolation

Virtual memory provides memory isolation

chroot provides file system isolation

Not enough for multi-tenant env:

Many other resources are still shared: IPC, network stack, users, etc.

Performance isolation: noisy neighbors

OS bugs!

Need Stronger Isolation Mechanisms

Virtual machines

Containers (probably not for multi-tenant)

Sandboxed containers

Each tenant is in a isolated **sandbox.**

(FreeBSD Jails)

Problem Solved?

Cloud providers can happily rent out computers.

Users still struggle to manage the amount of resources

- too little: can't handle workload bursts
- too much: idling, wasted $$$

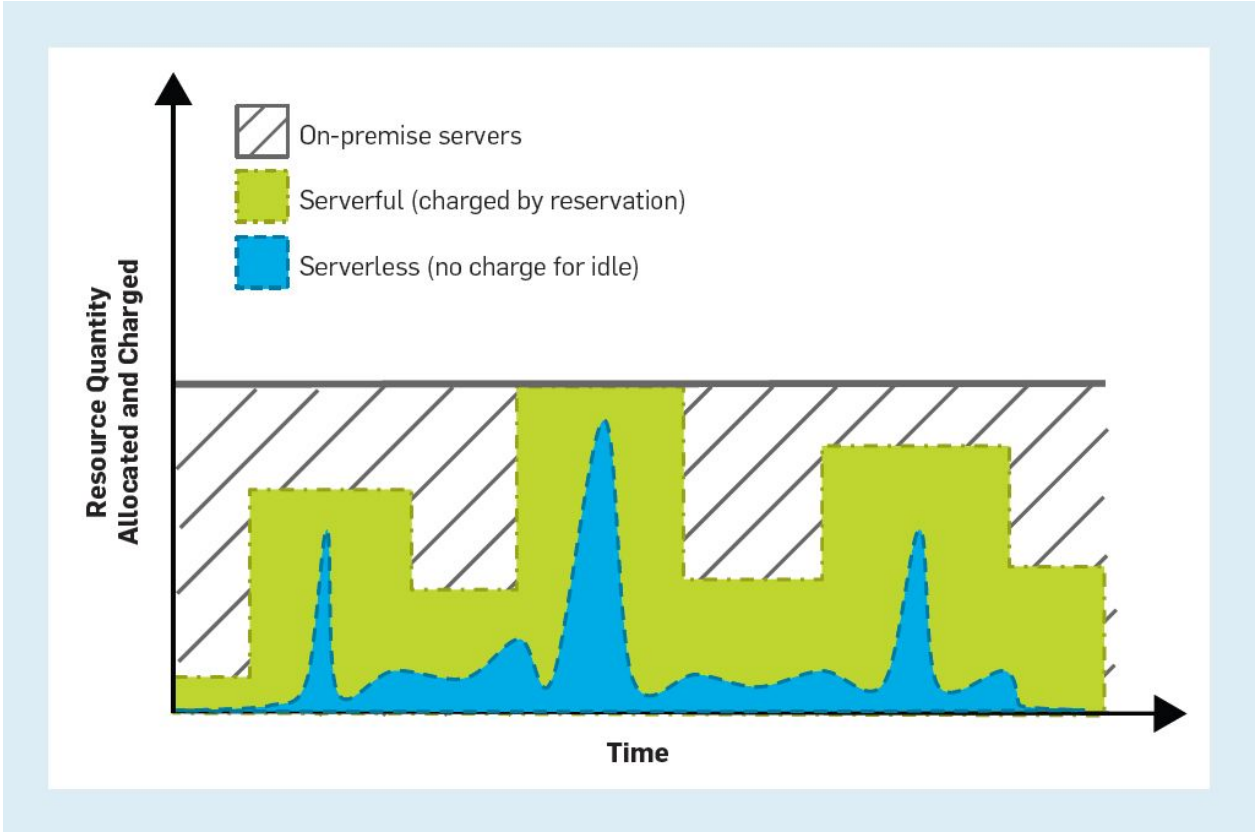
Serverless computing: fine-grained short-lived cloud resources

Instead of renting servers, users only rent CPU time, billed in sub-second

Provide code (function), which is triggered to run when receiving a request.

Users only pay for the CPU time they actually use.

Function as a Service (FaaS)



What Serverless Computing Is and Should Become: The Next Phase of Cloud Computing [CACM '21]

Costs of isolation and short-livedness

Since tenants require strong isolation, we can't allow the them to share resources like runtime and libraries.

To start a serverless function:

1. Boot VM/container (second to minutes)
2. Init runtime/library (seconds to minutes)
3. Load application/warm cache/JIT compilation (up to minutes)

Cold start can take much **longer time** than actual function execution (average a few seconds)

"Cold start problem". Bad for both latency and utilization.

Cold start mitigations

Booting time mitigation

Mirage [ASPLOS'13]

LightVM [SOSP'17]

Firecracker [NSDI'20]

Memory state reuse

Keep-alive (warm start)
Potemkin [SOSP'05]
Snowflock [EuroSys'09]
SEUSS [EuroSys'20]

Snapshots

Catalyzer [ASPLOS'20]

Firecracker snapshots

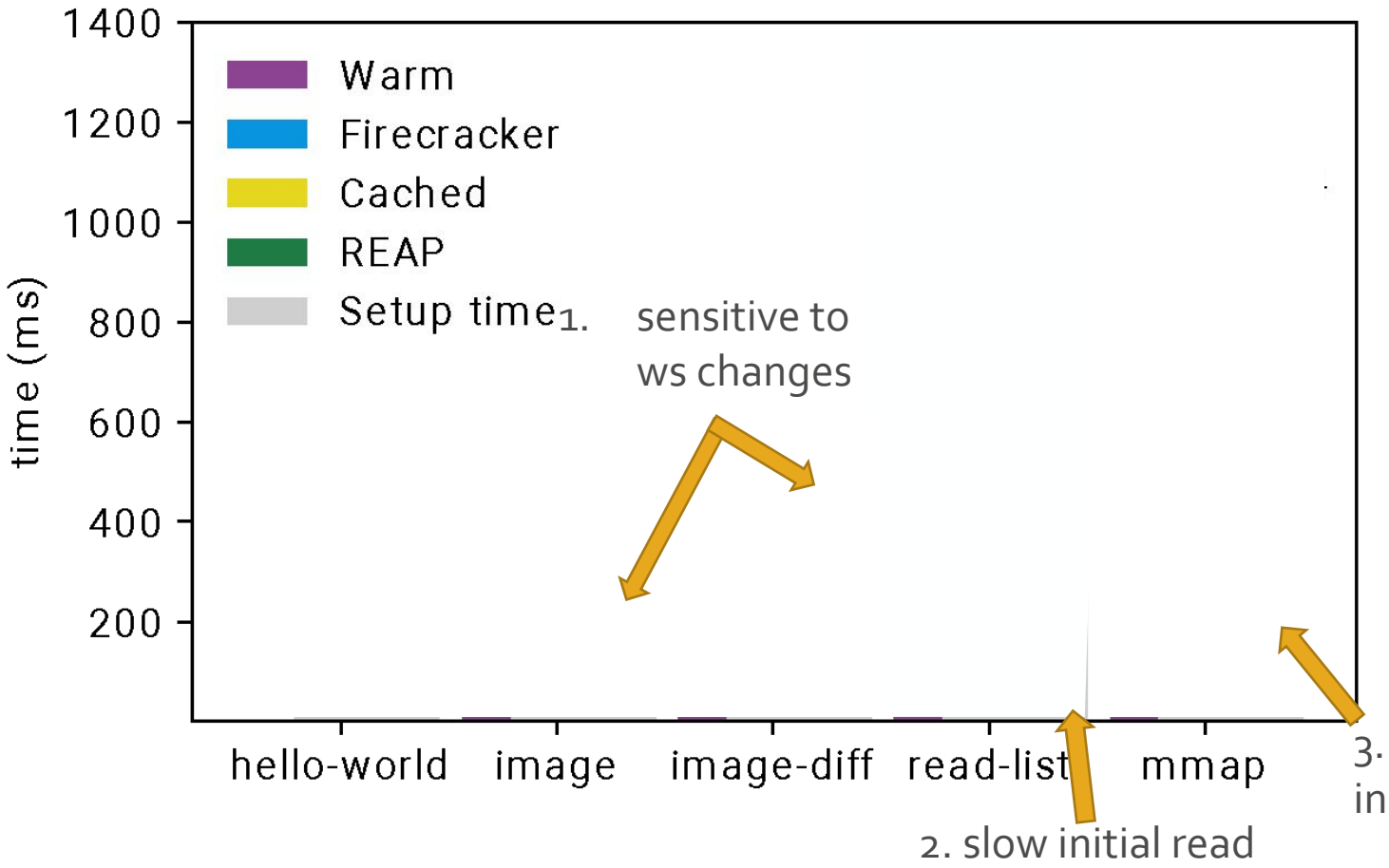
- REAP [ASPLOS'21]
- **FaaSnap**

Firecracker snapshots

- Firecracker is a lightweight kvm-based VM tailored for serverless. Used in AWS Lambda, Fargate.
- Snapshots supported since June 2020
- *Fast* to start
  - `mmap` memory file (previous snapshot of guest) as guest memory
  - Resume VM immediately
  - Load guest pages from disk on demand (demand paging)
- *Slow* to finish
  - Guest accessing pages not in memory causes major page faults
  - Context switches, scheduling
  - Slow down execution

State of the art: REAP [ASPLOS'21]

- Reduce major page faults by prefetching working set
- Phase 1: Record guest working set and save to a compact file
- Phase 2 (invoke): Prefetch working set from disk
- `userfaultfd` to handle page faults in userspace
- Prefetch stable working set -> fewer guest major page faults -> faster function execution



1. Sensitive to working set changes
   - Pagefaults outside of working set handled slowly
2. Blocking initial prefetch
   - Working set needs to be fully loaded to start the guest VM
3. Pagefault semantic gap
   - Host does not know the cause of page faults

Hello-world: minimal function
Image: image rotating
Read-list: reading large memory region
Mmap: allocating and writing memory region

EC2 c5d.metal 1vCPU, 2GB RAM / function

FaaSnap : FaaS Made Fast Using Snapshot-based VMs

Lixiang Ao, George Porter, and Geoffrey M. Voelker

UC San Diego

EuroSys 2022

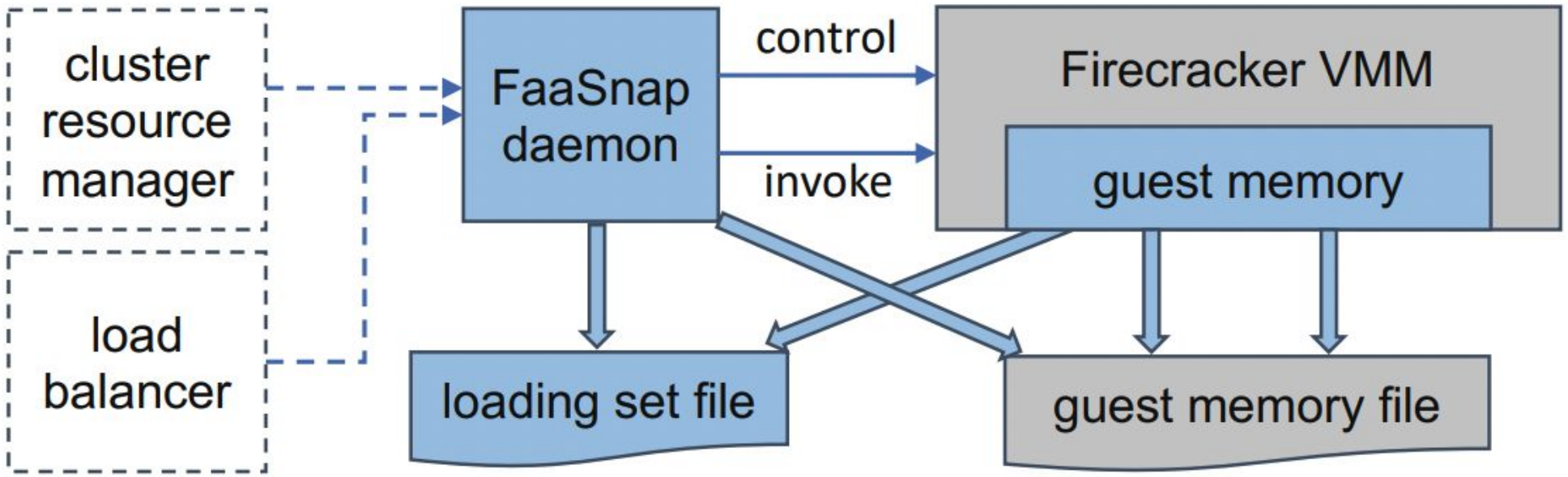
Goal

Reducing time spent on blocking prefetching and page faults

Approach

Serving the *right pages* at the *right time* in the *right way*

Overview


cluster resource manager
load balancer
FaaSnap daemon
control
invoke
Firecracker VMM
guest memory
loading set file
guest memory file

Right pages: redefine the working set

- REAP
  - Accessed pages
  - Works well for stable working set
  - Sensitive to working set changes

- FaaSnap
  - Present pages: accessed pages + readahead pages
  - Readahead is a prediction of future accesses
  - Present pages improves tolerance of working set changes

Right time: blocking vs nonblocking

- REAP
  - Blocking initial prefetch
  - Guarantee working set are loaded before guest starts
  - Long wait time

- FaaSnap
  - Start guest immediately
  - Concurrent paging
  - Guest and FaaSnap fetch pages concurrently to page cache, reducing guest major pagefaults opportunistically
  - Fetch pages in same order as previous invocation

Right way: uniform vs non-uniform

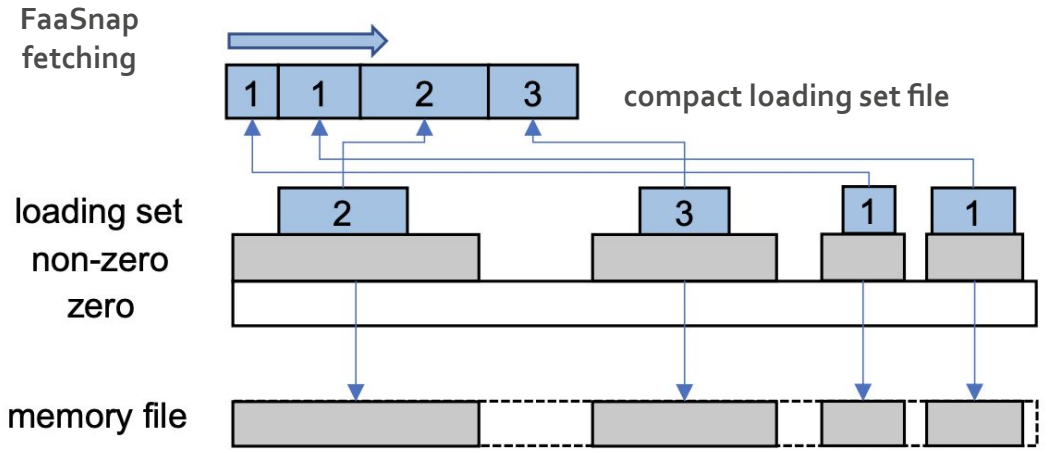
- REAP
  - Whole guest memory are mapped the same way
  - Any guest page needs to be read from disk

- FaaSnap
  - Per-region mapping
  - Anonymous, memory file, loading set file depending on page type (see figure next slide)
  - Only semantically useful pages are read from disk
  - Improve page fault serving efficiency

Per-region mapping

Type	Non-zero	Working set	Mapping
Loading set	Y	Y	loading set file
Cold set	Y	N	memory file
Released set	N	Y	anonymous
Unused set	N	N	

modify guest kernel
to write-zero
released set



Overlay mappings

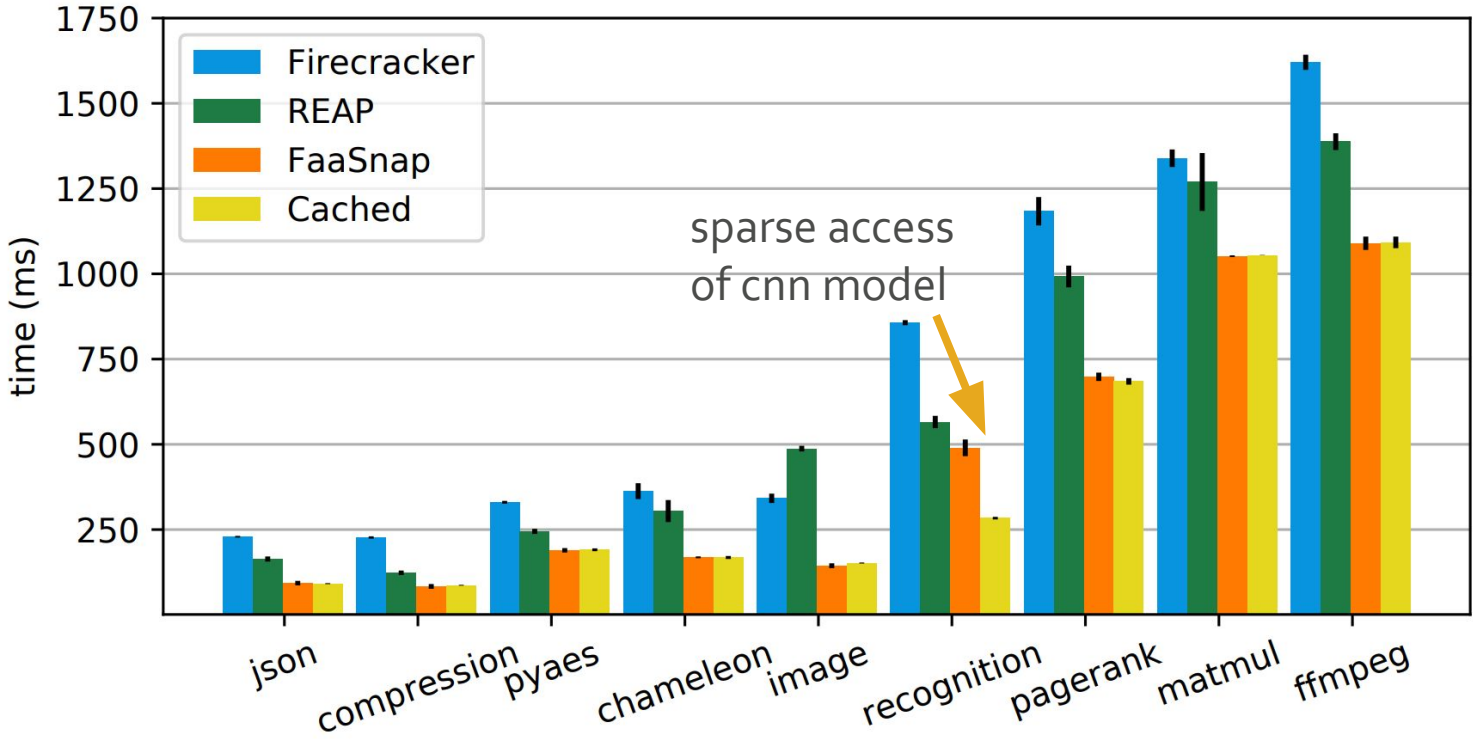
Evaluation

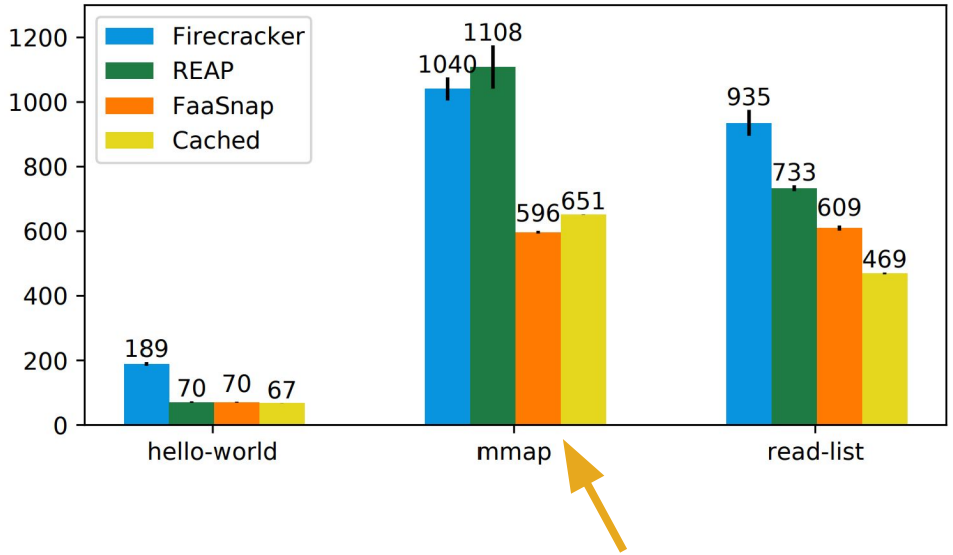
- Variety of applications
- Different inputs for recording phase and testing phase
- AWS c5d.metal instance
  - 96 vCPU at 3.00 GHz, 192 GB of memory
- NVMe SSD
  - 1589 MB/s, 285k IOPS
- 2vCPUs, 2GB RAM per function

	Description	Input A	Input B	Working Set A	Working Set B
Hello-world	a minimal function	n/a	n/a	11.8 MB	11.8 MB
Read-list	read an 512 MB Python list	n/a	n/a	526 MB	526 MB
Mmap	allocate anonymous memory	512 MB	512 MB	536 MB	536 MB
Image	rotate a JPEG image	101 KB JPEG	103 KB JPEG	20.6 MB	32.6 MB
Json	deserialize and serialize json	13 KB json	148 KB json	12.7 MB	14.4 MB
Pyaes	AES encryption	string of 20k	string of 22k	12.6 MB	13.2 MB
Chameleon	render HTML table	table size 30k	table size 40k	22.9 MB	25.1 MB
Matmul	matrix multiplication	matrix size 2000	matrix size 2200	113 MB	133 MB
FFmpeg	apply grayscale filter	1-sec 480p video, 338KB	1-sec 480p video, 381KB	179 MB	178 MB
Compression	file compression	13 KB file	148 KB file	15.3 MB	15.8 MB
Recognition	PyTorch ResNet image recognition	ResNet-50 cnn, 101 KB JPEG	ResNet-50 cnn, 103 KB JPEG	230 MB	234 MB
PageRank	igraph PageRank	graph size 90k	graph size 100k	104 MB	114 MB

Functions from FunctionBench [Cloud'19], Sprocket [SoCC'18], and SeBS [Middleware'21]

Function execution time





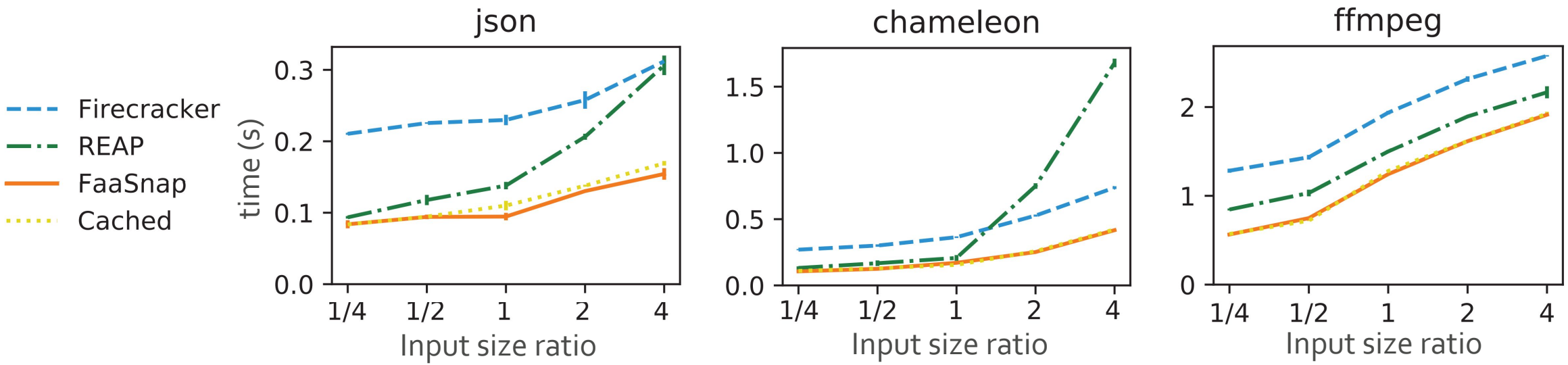
per-region mapping
closes the semantic gap

FaaSnap is **2.0x** faster than Firecracker

Faasnap is **1.36x** faster than REAP

By reducing time on the critical path, FaaSnap using SSD is only **3.5%** slower than snapshots cached in memory.

Resilience to changing working set



REAP can't handle working set changes well for some functions

FaaSnap performs similarly to Cached for most functions

FaaSnap is resilient to working set changes thanks to **present page recording** & **per-region mapping**

Summary

- OSes enable multiplexing, which helps improving utilization
- Fluctuations of workloads and the need for better utilization gives rise to cloud computing including serverless computing
- Strong isolation requirements cause cold start problem
- FaaSnap can solve serverless cold start problem – reducing latency, improving utilization, *making the most out of computers*

References

[ASPLOS'13] Anil Madhavapeddy, et al. Unikernels: Library Operating Systems for the Cloud. In Proceedings of the Eighteenth ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'13). ACM, Houston, TX, USA, 461–472.

[ASPLOS'20] Dong Du, et al. 2020. Catalyzer: Sub-millisecond Startup for Serverless Computing with Initialization-less Booting. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20). ACM, Lausanne, Switzerland, 467–481.

[ASPLOS'21] Dmitrii Ustiugov, Plamen Petrov, Marios Kogias, Edouard Bugnion, and Boris Grot. 2021. Benchmarking, Analysis, and Optimization of Serverless Function Snapshots. In Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'21). ACM, Virtual, 559–572.

[CACM'21] Schleier-Smith, Johann, et al. "What serverless computing is and should become: The next phase of cloud computing." Communications of the ACM 64.5 (2021): 76-84.

[CLOUD'19] Jeongchul Kim and Kyungyong Lee. 2019. FunctionBench: A Suite of Workloads for Serverless Cloud Function Service. In 2019 IEEE 12th International Conference on Cloud Computing (CLOUD'19). IEEE, Milan, Italy, 502–504

[EuroSys'09] Horacio Andrés Lagar-Cavilla, et al. Snowflock: Rapid Virtual Machine Cloning for Cloud Computing. In Proceedings of the 4th ACM European Conference on Computer Systems (EuroSys'09). ACM, Nuremberg, Germany, 1–12.

[EuroSys'20] James Cadden, et al. 2020. SEUSS: Skip Redundant Paths to Make Serverless Fast. In Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys'20). ACM, Heraklion, Crete, Greece, 1–15.

[IWQoS'19] Guo, Jing, et al. "Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces." Proceedings of the International Symposium on Quality of Service. 2019.

[Middleware'21] Marcin Copik, Grzegorz Kwasniewski, Maciej Besta, Michal Podstawski, and Torsten Hoefler. 2021. SeBS: A Serverless Benchmark Suite for Function-as-a-Service Computing. In Proceedings of the 22nd International Middleware Conference (Middleware'21). ACM, New York, NY, USA, 64–78.

[NSDI'20] Alexandru Agache, et al. Firecracker: Lightweight Virtualization for Serverless Applications. In Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI'20). USENIX Association, Santa Clara, CA, 419–434.

[SOCC'18] Lixiang Ao, Liz Izhikevich, Geoffrey M. Voelker, and George Porter. 2018. Sprocket: A Serverless Video Processing Framework. In Proceedings of the ACM Symposium on Cloud Computing (SoCC'18). ACM, Carlsbad, CA, 263–274.

[SOSP'05] Michael Vrable, et al. Scalability, Fidelity, and Containment in the Potemkin Virtual Honeyfarm. In Proceedings of the Twentieth ACM Symposium on Operating Systems Principles. ACM, Brighton, UK, 148–162.

[SOSP'17] Filipe Manco, et al. My VM is Lighter (and Safer) than your Container. In Proceedings of the 26th Symposium on Operating Systems Principles (SOSP'17). ACM, Shanghai, China, 218–233.

Thank you!

Q & A