FaaSCell: A Case for Intra-node Resource Management

Work-In-Progress

Christos Katsakioris
National Technical University of Athens
ckatsak@cslab.ece.ntua.gr

Chloe Alverti
National Technical University of Athens
xalverti@cslab.ece.ntua.gr

Konstantinos Nikas
National Technical University of Athens
knikas@cslab.ece.ntua.gr

Stratos Psomadakis
National Technical University of Athens
psomas@cslab.ece.ntua.gr

Vasileios Karakostas
University of Athens
vkarakos@di.uoa.gr

Nectarios Koziris
National Technical University of Athens
nkoziris@cslab.ece.ntua.gr

ACM Reference Format:
Christos Katsakioris, Chloe Alverti, Konstantinos Nikas, Stratos Psomadakis, Vasileios Karakostas, and Nectarios Koziris. 2023. FaaSCell: A Case for Intra-node Resource Management: Work-In-Progress. In *The 1st Workshop on SErverless Systems, Applications and MEthodologies (SESAME '23), May 8, 2023, Rome, Italy*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3592533.3592812

1 MOTIVATION

Open-source FaaS platforms have recently shown rapid growth, which is usually manifested as extension or specialization of existing cloud-native components and systems –mainly over Kubernetes– since they are provably capable of standing their ground against production-level needs. Despite its advances, the cloud-native ecosystem has focused mostly on container-based deployments so far. FaaS workloads' need for massive colocation [1] without sacrificing security guarantees, pushes multi-tenancy to its limits.

First, this calls for highly isolated environments, which are traditionally associated with hardware-accelerated virtualization [1, 4, 21, 24, 25, 41]. Second, the fine-grained management of a node's available resources becomes essential, but also challenging because of FaaS workloads' characteristics [35]. Invocations tend to be short-lived, with irregular inter-arrival patterns. Keeping function instances warm becomes too expensive in terms of allocated resources, whereas booting them anew on each invocation imposes prohibitive slowdowns (i.e., cold starts). Lately, both Cloud providers and researchers consider VM snapshots as a viable mitigation [9, 16, 40]. After booting the function sandbox, its whole state –including its memory– is captured and persisted on storage. On subsequent invocations, the sandbox can be restored from the snapshot rather than booted anew, thus drastically improving startup time. Such mechanisms are being adopted by open-source systems [8, 9] and Cloud providers [27].

FaaS platforms are by nature distributed. Nevertheless, research on resource management also entails work on intra-node resource allocation. This is attested by several serverless studies focusing on single-node experiments to either investigate problems or evaluate proposed solutions. Past work generally examines node-local resource management, involving CPU, networking or I/O, often in the form of pre-allocating and caching policies (e.g., [2, 20, 31, 33, 36, 37]) and sandbox snapshotting in particular [5, 16, 29, 39, 42]. For this sort of experimentation, deploying a full-blown Kubernetes stack, including API extensions retrofitting FaaS concepts into cloud-native workflows, might be superfluous.

For that reason, we propose an alternative design which we find simpler to implement, extend and debug in scenarios involving single-node experimentation. We aim to enrich the existing open-source FaaS ecosystem with a versatile serverless system that offers a set of mechanisms to facilitate research and development of local orchestration intelligence. This component should be orthogonal with higher-level platforms that cater for cluster-level orchestration, hence also capable of integrating with them.

SESAME '23, May 8, 2023, Rome, Italy

ACM ISBN 979-8-4007-0185-6/23/05.
https://doi.org/10.1145/3592533.3592812

2 STATE-OF-THE-ART PLATFORMS

We are examining three major open-source systems that enable FaaS deployments with respect to our prior observations.

vHive. [40] It employs firecracker-containerd [3] to boot function instances in Firecracker microVMs rather than containers, also supporting optimized snapshotting. As a CRI [10] implementation, vHive relies on Kubernetes for cluster orchestration, on top of which Knative [11] provides the API primitives necessary for enabling FaaS deployments. The extensive use of cloud-native components makes vHive robust in cases of distributed deployments, but renders the software stack significantly more complex. Tracing function invocations and platform's overheads end-to-end can become cumbersome, especially when such systems are further extended for various research purposes. Moreover, vHive equates idle instances with those restored from snapshots. While the latter have been proved to significantly mitigate cold starts, research in sandbox caching and keep-alive policies is ongoing and orthogonal to snapshotting. This, in conjunction with utter reliance on Kubernetes for CPU and memory management, makes vHive more suitable for research on larger-scale settings.

Apache OpenWhisk. OpenWhisk [6] features a clean, extensible architecture, which is also the reason why many studies rely on it for prototype implementations [2, 20, 28, 31, 32, 34, 35, 38, 43–45]. It is simple to deploy due to comprising fewer moving parts. For instance, it can be deployed independently of Kubernetes, and also in single-node mode. Despite its numerous benefits, OpenWhisk's architecture is entirely focused on using containers as function instance sandboxes. By default, it is integrated with the docker CLI [26] and unaware of sandbox snapshotting. Furthermore, being implemented in Scala raises the engineering effort of integrating it with a high-level container runtime, like firecracker-containerd, due to missing components (e.g., TTRPC [15] compiler for JVM).

Kata Containers. As an OCI [19] runtime, Kata [12] does not provide FaaS semantics by itself. However, through its integration with higher-level container runtimes (e.g., containerd [14]) it can be used as a Kubernetes RuntimeClass[1], thus enabling Kubernetes API-extending FaaS platforms (e.g., Knative [11], OpenFaaS [17]) to seamlessly leverage the benefits of VMs. Its clean, VM-centric architecture makes it a robust solution for production use [30] and for experimentation at scale. However, support for Firecracker is currently unimplemented [18]. Besides, by design, Kata has been primarily versed towards Cloud workload deployments (e.g., via QEMU [13] or Cloud Hypervisor [7]) rather than FaaS.

Research on intra-node resource management could benefit from a simpler software stack. Bootstrapping such an effort would include a clean architectural component on top of a microVM-enabled container manager, free from the complexity of distributed deployments. This should improve overall transparency in the system, thus facilitating research and enabling innovation at lower levels as well. In light of this, we aspire to design a system which:

- considers Firecracker microVMs and their snapshots as first-class function execution units in the system;
- focuses on resource management within a single node rather than cluster-wide orchestration;
- incorporates extensible mechanisms and configurable knobs to enable research and development of a variety of node-local algorithms and policies.

3 DESIGN OF FAASCELL

FaaSCell is responsible for responding to clients' function requests –incoming presumably from an upper-layer entity– by invoking the respective user-defined functions. The latter are sandboxed within microVMs, which:

- may already run (in case of previous invocation(s) and depending on the keep-alive policy in place);
- can be loaded from a microVM snapshot, created during an earlier invocation and persisted on one of the node's available storage devices;
- may need to be booted anew (i.e., cold start).

FaaSCell is therefore responsible for orchestrating these microVMs. It has to track their state and manage their lifecycle, spawning new ones and reaping old ones when needed, aiming for low response latencies and high invocations throughput. Furthermore, FaaSCell needs to control the allocation of the node's resources occupied by those microVMs. While the exact policy for doing so is subject to active research and thus may vary, the system should provide the appropriate mechanisms to facilitate this sort of extensibility.

At the bottom of the stack, similar to vHive, FaaSCell uses firecracker-containerd as the high-level container runtime that communicates with the Firecracker processes. Through its API, system components at higher layers can control each microVM's lifecycle. We extend firecracker-containerd to support microVM snapshotting in a resource-efficient manner.

[1] https://kubernetes.io/docs/concepts/containers/runtime-class/#runtime-class

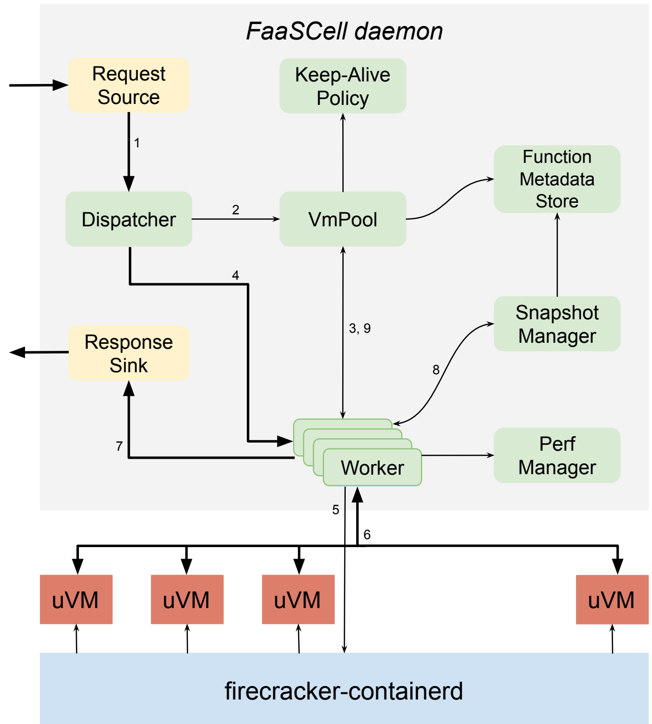

Figure 1: Interaction among some of the principal Actors in FaaSCell during a function invocation.

On top of firecracker-containerd, the FaaSCell daemon is responsible for actually orchestrating the microVMs according to invocation requests. This is where decision-making takes place, hence where robust yet flexible mechanisms should enable innovation through intelligent algorithms.

Similar to Kata, we pick Rust for implementing this layer, for the memory safety and zero-cost abstractions it provides. To cope with the high-concurrency requirements of its role, the daemon is designed after the Actor model [22, 23], similar to OpenWhisk: asynchronous userspace threads, each dedicated to a specific role, communicating with one another via efficient message passing.

Figure 1 roughly illustrates the interaction of some of the system's principal Actors during a function invocation. A request flows into the system through a *Source*, which forwards it to *Dispatcher* ①. *Dispatcher* requests *VmPool* to assign the invocation to a *Worker* ②. To achieve that, *VmPool* examines the system's state and consults any decision-making policy that may be in place, to allocate the desired resources accordingly and possibly to prepare the execution environment (e.g., by restoring or spawning new microVMs and associated *Workers* ③). Subsequently, *Dispatcher* can dispatch the request to the appropriate *Worker* ④, who is responsible for communicating to firecracker-containerd the decisions made earlier by *VmPool* ⑤. When the environment is set (i.e., the microVM is ready to serve the function at hand), *Worker* is responsible for forwarding the client's request to the function ⑥, possibly collecting performance metrics during its execution. The respective response is then channeled to *Sink* ⑦, to be forwarded further to any upper-layer system components. Meanwhile, *Worker* updates

SnapshotManager with any newly collected metrics, possibly querying for information necessary to create a new snapshot for the function instance ⑧. Finally, *Worker* notifies *VmPool* of its state ⑨, enabling the enforcement of additional policies (e.g., keep-alive) that potentially influence decisions in subsequent invocations.

Some of the Actors are designed for extensibility; i.e., based on interfaces that can have multiple implementations according to the needs of the experimentation. For instance, alternative implementations for *Source* and *Sink* make FaaSCell's interface versatile, as long as the actual function request is accessible to *Dispatcher*. Similarly, entities interacting with *VmPool* can be extended, allowing a variety of keep-alive policies, microVM selection algorithms and function metadata storage and retrieval. *Workers* could be extended to use alternative low-level runtimes, and their associated *PerfManagers* to collect different performance metrics during function execution, depending on the research context. *NetworkManager* abstracts away the specifics of allocating and setting up any network resources and rules required to provide connectivity to the microVMs. Finally, *SnapshotManager* can implement several algorithms for arranging sandbox snapshots among storage devices made available.

4 CONCLUSIONS

We advocate and originally design FaaSCell, an intra-node orchestrator for serverless functions. It aims to enable single-node resource management and performance studies, while remaining compatible with the distributed software stack of FaaS. FaaSCell could potentially be integrated with Kubernetes and its ecosystem, or with any other upper-layer component or platform that may be used for cluster-wide orchestration of FaaS deployments.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback. This work was funded by the European Union under the Horizon Europe grant 101092850 (project AERO).

REFERENCES

[1] Alexandru Agache, Marc Brooker, Andreea Florescu, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight Virtualization for Serverless Applications. In *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation* (Santa Clara, CA, USA) *(NSDI'20)*. USENIX Association, USA, 419–434.

[2] Mohamed Alzayat, Jonathan Mace, Peter Druschel, and Deepak Garg. 2022. Groundhog: Efficient Request Isolation in FaaS. https://doi.org/10.48550/ARXIV.2205.11458

[3] Inc. or its affiliates Amazon.com. 2023. *Firecracker Containerd*. Retrieved February 24, 2023 from https://github.com/firecracker-microvm/firecracker-containerd/

[4] Anjali, Tyler Caraza-Harter, and Michael M. Swift. 2020. Blending Containers and Virtual Machines: A Study of Firecracker and GVisor. In *Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (Lausanne, Switzerland) *(VEE '20)*. Association for Computing Machinery, New York, NY, USA, 101–113. https://doi.org/10.1145/3381052.3381315

[5] Lixiang Ao, George Porter, and Geoffrey M. Voelker. 2022. FaaSnap: FaaS Made Fast Using Snapshot-Based VMs. In *Proceedings of the Seventeenth European Conference on Computer Systems* (Rennes, France) *(EuroSys '22)*. Association for Computing Machinery, New York, NY, USA, 730–746. https://doi.org/10.1145/3492321.3524270

[6] Apache Software Foundation (ASF). 2023. *Open Source Serverless Cloud Platform*. Retrieved February 24, 2023 from https://openwhisk.apache.org/

[7] The Cloud Hypervisor Authors. 2023. *Cloud Hypervisor – Run Cloud Virtual Machines Securely and Efficiently*. Linux Foundation. Retrieved February 24, 2023 from https://www.cloudhypervisor.org/

[8] The Cloud-Hypervisor Authors. 2023. *Snapshot and Restore*. Retrieved February 24, 2023 from https://github.com/cloud-hypervisor/cloud-hypervisor/blob/main/docs/snapshot_restore.md

[9] The Firecracker Authors. 2023. *Firecracker Snapshotting*. Retrieved February 24, 2023 from https://github.com/firecracker-microvm/firecracker/blob/main/docs/snapshotting/snapshot-support.md

[10] The Kubernetes Authors. 2023. *Container Runtime Interface (CRI)*. Retrieved February 24, 2023 from https://kubernetes.io/docs/concepts/architecture/cri/

[11] The Knative Authors. 2023. *Knative is an Open-Source Enterprise-level solution to build Serverless and Event Driven Applications*. Retrieved February 24, 2023 from https://knative.dev/

[12] The Kata Containers Authors. 2023. *Kata Containers – Open Source Container Runtime Software*. Retrieved February 24, 2023 from https://katacontainers.io/

[13] Fabrice Bellard. 2005. QEMU, a Fast and Portable Dynamic Translator. In *2005 USENIX Annual Technical Conference (USENIX ATC 05)*. USENIX Association, Anaheim, CA. https://www.usenix.org/conference/2005-usenix-annual-technical-conference/qemu-fast-and-portable-dynamic-translator

[14] The containerd Authors. 2023. *containerd – An industry-standard container runtime with an emphasis on simplicity, robustness and portability*. Retrieved February 24, 2023 from https://containerd.io/

[15] The containerd Authors. 2023. *TTRPC – Protocol Specification*. Retrieved February 24, 2023 from https://github.com/containerd/ttrpc/blob/main/PROTOCOL.md

[16] Dong Du, Tianyi Yu, Yubin Xia, Binyu Zang, Guanglu Yan, Chenggang Qin, Qixuan Wu, and Haibo Chen. 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* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 467–481. https://doi.org/10.1145/3373376.3378512

[17] Alex Ellis and OpenFaaS Ltd. 2023. *OpenFaaS – Serverless Functions, Made Simple*. Retrieved February 24, 2023 from https://www.openfaas.com/

[18] Open Infrastructure Foundation. 2023. *Github Issue #5268 – Implement a runtime-rs hypervisor plugin for Firecracker*. Retrieved February 24, 2023 from https://github.com/kata-containers/kata-containers/issues/5268

[19] The Linux Foundation. 2023. *Open Container Initiative*. Retrieved February 24, 2023 from https://opencontainers.org/

[20] Alexander Fuerst and Prateek Sharma. 2021. FaasCache: Keeping Serverless Computing Alive with Greedy-Dual Caching. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 386–400. https://doi.org/10.1145/3445814.3446757

[21] Google. 2023. *gVisor: an application kernel for containers*. Google. Retrieved February 24, 2023 from https://gvisor.dev/

[22] Carl Hewitt. 2010. Actor Model of Computation: Scalable Robust Information Systems. https://doi.org/10.48550/ARXIV.1008.1459

[23] Carl Hewitt, Peter Bishop, and Richard Steiger. 1973. A Universal Modular ACTOR Formalism for Artificial Intelligence. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence* (Stanford, USA) *(IJCAI'73)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 235–245.

[24] Amazon Web Services Inc. 2023. *AWS Fargate – Serverless compute for containers*. Amazon Web Services Inc. Retrieved February 24, 2023 from https://aws.amazon.com/fargate/

[25] Amazon Web Services Inc. 2023. *AWS Lambda – Run code without thinking about servers or clusters*. Amazon Web Services Inc. Retrieved February 24, 2023 from https://aws.amazon.com/lambda/

[26] Docker Inc. 2023. *Docker run reference*. Retrieved February 24, 2023 from https://docs.docker.com/engine/reference/run/

[27] Eric Johnson. 2022. *Reducing Java cold starts on AWS Lambda functions with SnapStart*. Amazon Web Services Inc. Retrieved February 24, 2023 from https://aws.amazon.com/blogs/compute/reducing-java-cold-starts-on-aws-lambda-functions-with-snapstart/

[28] Kostis Kaffes, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2022. Hermod: Principled and Practical Scheduling for Serverless Functions. In *Proceedings of the 13th Symposium on Cloud Computing* (San Francisco, California) *(SoCC '22)*. Association for Computing Machinery, New York, NY, USA, 289–305. https://doi.org/10.1145/3542929.3563468

[29] Christos Katsakioris, Chloe Alverti, Vasileios Karakostas, Konstantinos Nikas, Georgios Goumas, and Nectarios Koziris. 2022. FaaS in the Age of (Sub-)μs I/O: A Performance Analysis of Snapshotting. In *Proceedings of the 15th ACM International Conference on Systems and Storage* (Haifa, Israel) *(SYSTOR '22)*. Association for Computing Machinery, New York, NY, USA, 13–25. https://doi.org/10.1145/3534056.3534938

[30] Zijun Li, Jiagan Cheng, Quan Chen, Eryu Guan, Zizheng Bian, Yi Tao, Bin Zha, Qiang Wang, Weidong Han, and Minyi Guo. 2022. RunD: A Lightweight Secure Container Runtime for High-density Deployment and High-concurrency Startup in Serverless Computing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 53–68. https://www.usenix.org/conference/atc22/presentation/li-zijun-rund

[31] Anup Mohan, Harshad Sane, Kshitij Doshi, Saikrishna Edupuganti, Naren Nayak, and Vadim Sukhomlinov. 2019. Agile Cold Starts for Scalable Serverless. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*. USENIX

Association, Renton, WA. https://www.usenix.org/conference/hotcloud19/presentation/mohan

[32] Djob Mvondo, Mathieu Bacou, Kevin Nguetchouang, Lucien Ngale, Stéphane Pouget, Josiane Kouam, Renaud Lachaize, Jinho Hwang, Tim Wood, Daniel Hagimont, Noël De Palma, Bernabé Batchakui, and Alain Tchana. 2021. OFC: An Opportunistic Caching System for FaaS Platforms. In *Proceedings of the Sixteenth European Conference on Computer Systems* (Online Event, United Kingdom) *(EuroSys '21)*. Association for Computing Machinery, New York, NY, USA, 228–244. https://doi.org/10.1145/3447786.3456239

[33] Orestis Lagkas Nikolos, Georgios Goumas, and Nectarios Koziris. 2022. Deverlay: Container Snapshots For Virtual Machines. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 11–20. https://doi.org/10.1109/CCGrid54584.2022.00010

[34] Rohan Basu Roy, Tirthak Patel, and Devesh Tiwari. 2022. IceBreaker: Warming Serverless Functions Better with Heterogeneity. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 753–767. https://doi.org/10.1145/3503222.3507750

[35] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 205–218. https://www.usenix.org/conference/atc20/presentation/shahrad

[36] Wonseok Shin, Wook-Hee Kim, and Changwoo Min. 2022. Fireworks: A Fast, Efficient, and Safe Serverless Framework Using VM-Level Post-JIT Snapshot. In *Proceedings of the Seventeenth European Conference on Computer Systems* (Rennes, France) *(EuroSys '22)*. Association for Computing Machinery, New York, NY, USA, 663–677. https://doi.org/10.1145/3492321.3519581

[37] Gaetano Somma, Constantine Ayimba, Paolo Casari, Simon Pietro Romano, and Vincenzo Mancuso. 2020. When Less is More: Core-Restricted Container Provisioning for Serverless Computing. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 1153–1159. https://doi.org/10.1109/INFOCOMWKSHPS50562.2020.9162876

[38] Amoghvarsha Suresh and Anshul Gandhi. 2019. FnSched: An Efficient Scheduler for Serverless Functions. In *Proceedings of the 5th International Workshop on Serverless Computing* (Davis, CA, USA) *(WOSC '19)*. Association for Computing Machinery, New York, NY, USA, 19–24. https://doi.org/10.1145/3366623.3368136

[39] Yue Tan, David Liu, Nanqinqin Li, and Amit Levy. 2021. How Low Can You Go? Practical cold-start performance limits in FaaS. https://doi.org/10.48550/ARXIV.2109.13319

[40] 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* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 559–572. https://doi.org/10.1145/3445814.3446714

[41] Ao Wang, Shuai Chang, Huangshi Tian, Hongqi Wang, Haoran Yang, Huiba Li, Rui Du, and Yue Cheng. 2021. FaaSNet: Scalable and Fast Provisioning of Custom Serverless Container Runtimes at Alibaba Cloud Function Compute. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 443–457. https://www.usenix.org/conference/atc21/presentation/wang-ao

[42] Kai-Ting Amy Wang, Rayson Ho, and Peng Wu. 2019. Replayable Execution Optimized for Page Sharing for a Managed Runtime Environment. In *Proceedings of the Fourteenth EuroSys Conference 2019* (Dresden, Germany) *(EuroSys '19)*. Association for Computing Machinery, New York, NY, USA, Article 39, 16 pages. https://doi.org/10.1145/3302424.3303978

[43] Song Wu, Zhiheng Tao, Hao Fan, Zhuo Huang, Xinmin Zhang, Hai Jin, Chen Yu, and Chun Cao. 2022. Container lifecycle-aware scheduling for serverless computing. *Software: Practice and Experience* 52, 2 (2022), 337–352. https://doi.org/10.1002/spe.3016 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.3016

[44] Hanfei Yu, Athirai A. Irissappane, Hao Wang, and Wes J. Lloyd. 2021. FaaSRank: Learning to Schedule Functions in Serverless Platforms. In *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. 31–40. https://doi.org/10.1109/ACSOS52086.2021.00023

[45] Ziming Zhao, Mingyu Wu, Jiawei Tang, Binyu Zang, Zhaoguo Wang, and Haibo Chen. 2023. BeeHive: Sub-Second Elasticity for Web Services with Semi-FaaS Execution. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 74–87. https://doi.org/10.1145/3575693.3575752