No Provisioned Concurrency: Fast RDMA-codesigned Remote Fork for Serverless Computing*

Xingda Wei[1,2] Fangming Lu[1] Tianxia Wang[1] Jinyu Gu[1] Yuhan Yang[1] Rong Chen[1,2] Haibo Chen[1]

[1]Institute of Parallel and Distributed Systems, SEIEE, Shanghai Jiao Tong University

[2]Shanghai AI Laboratory

1 Motivation

Serverless computing is an emerging cloud computing paradigm supported by major cloud providers, including AWS Lambda [10], Azure Functions [24], Google Serverless [16], Alibaba Serverless Application Engine [11] and Huawei Cloud Functions [19]. Serverless promises *auto-scaling*—users only provide serverless functions, and serverless platforms automatically allocate computing resources (e.g., containers[1]) to execute them. Auto-scaling makes serverless computing economical: the platforms will reclaim the containers after functions finish, so they only bill when functions are executed (no charge for idle time).

However, *coldstart* (i.e., launching a container from scratch for each function) is a key challenge for fast auto-scaling, as the start time (over 100 ms) can be orders of magnitude higher than the execution time for ephemeral serverless functions [14, 25, 35]. Unfortunately, scaling functions to multiple machines is common because a single machine has a limited function capacity to handle the timely load spikes. Consider the two functions sampled from real-world traces of Azure Functions [28]. The request frequency of function `9a3e4e` can surge to over 150 K calls per minute, increased by 33,000× within one minute (see the top part of Figure 1).

Even worse, container segregates the address spaces of functions. Thus, dependent functions can only transfer states via message passing or cloud storage, which add data serialization/de-serialization, memory copy and accessing external storage overheads to functions that require interactions. Recent reports have shown that these may count up to 95% of the function execution time [23, 17]. Unfortunately, transferring states between functions is common in serverless workflows—a mechanism to compose functions [2, 1].

2 Limitations of the State of the Art

Function startup. Accelerating coldstart has become a hot topic in both academia and industry [15, 36, 25, 8, 28, 14, 9]. Most of them resort to a form of 'warmstart' by *provisioned concurrency*, e.g., launching a container from a cached one. However, 'warmstart' fundamentally faces a tradeoff between container startup time and provisioned concurrency (i.e., cached instances).

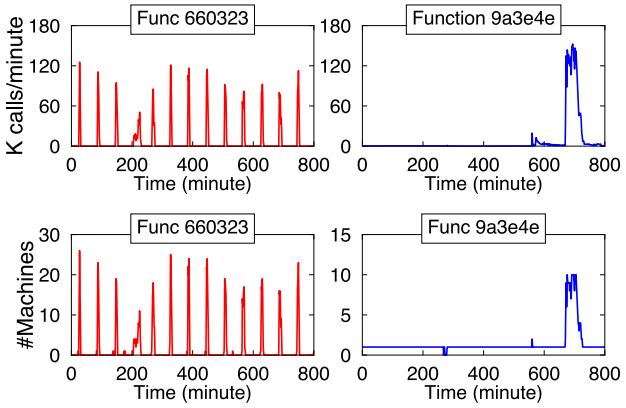

Figure 1. *The timelines of call frequency (top) and sufficient resource provisioning (bottom) for two serverless functions in a real-world trace from Azure Functions [28].*

Specifically, *Caching* [20, 21, 37, 15, 36, 25, 8, 28] caches the finished containers in memory such that later functions can reuse cached ones without startup. Though being optimal in startup performance, the platform must provision $O(n)$ cached containers to prevent stalling n concurrent function invocations.

Fork [14, 8, 13] optimizes caching by using `OS fork` to create multiple function containers from one cached instance on-the-fly. It reduces the per-machine container provisioned to $O(1)$, but still requires $O(m)$ cached instances running functions on m machines (in the case of Figure 1).

Checkpoint/Restore (C/R) [34, 14, 32] starts containers from pre-checkpointed files to reduce the startup cost. It is optimal in resource usage: one file ($O(1)$) is sufficient to start any subsequent functions, because we can transfer the file on demand across machines. However, C/R is orders of magnitude slower in startup compared with Caching or Fork, especially when the file is not locally stored at the function execution machine.

Table 1 summarizes different techniques for function startup. The foundational limitation of 'warmstart' is that they require *provisioned concurrency*: the platform or the user must specify the number of cached instances to prevent

*This work will be presented at OSDI'23.

[1]We focus on executing serverless functions with containers, which is widely adopted [36, 37, 18, 21].

Table 1: *A comparison of startup techniques for autoscaling n concurrent invocations of one function to m machines.* Local *means the resources for the startup are provisioned on the function execution machine. The function is a simple python program that prints 'hello world'.*

	Coldstart [4, 33]	Caching [20, 37, 25, 28, 36]	Fork [14, 8, 13]	Checkpoint/Restore [34, 14, 32, 9]	Remote fork MITOSIS
Local startup performance	Very slow ($100\,ms$)	Very fast ($< 1\,ms$)	Fast ($1\,ms$)	Medium ($5\,ms$)	Fast ($1\,ms$)
Remote startup performance	Very slow ($1,000\,ms$)	N/A	N/A	Slow ($24\,ms$)	Fast ($3\,ms$)
Overall resource provisioning	$O(1)$	$O(n)$	$O(m)$	$O(1)$	$O(1)$

the coldstart. There is "no free lunch" for such resources: vendors require users to reserve and pay for them, e.g., AWS Lambda Provisioned Concurrency [5].

State sharing. To accelerate the state transfer between functions, the state-of-the-art approaches propose serverless-optimized messaging primitives [8] or specialized storage systems [30, 22, 26]. However, none of them completely eliminated the overhead of memory copy and data serialization/deserialization [23]. On a single node, Faastlane [23] co-locates functions in the same container with *thread*s so that it can bypass these overheads with shared memory accesses. SPRIGHT [27] achieves a similar effect by retrofitting eBPF. However, they don't support efficient data sharing across nodes.

3 Key Insights: Remote Fork

We argue that *remote fork* is an efficient primitive for both function startup and state sharing:

- When generalizing FORK to a remote setting, a single *parent* container is sufficient to launch any subsequent containers. Therefore, FORK realizes *no provisioned concurrency*: the users only need to specify whether they require a cached container to accelerate function startup, not how many. Further, FORK has been proved efficient in starting functions for a single machine [14].
- FORK bridges the address spaces of parent and child containers. The transferred states are pre-materialized in the parent memory, so the child can seamlessly access them with shared memory abstraction with no data serialization, zero-copy (for read-only accesses[2]) and cloud storage costs.

Challenge: remote fork efficiency. To the best of our knowledge, existing containers can only achieve a conservative remote fork with a C/R-based approach [29, 12]. To fork a child, the parent first *checkpoints* its states by copying them to a file, and then *transfers* the file to the child—either using a remote file copy or a distributed file system. After receiving the file, the child *restores* the parent's execution by loading the container states from the checkpointed file. Note that C/R may load some states (i.e., memory pages) on-demand for better performance [34].

[2]In the case of the traditional fork. MITOSIS further optimizes with one-sided RDMA, allowing zero-copy even for read-write accesses.

Our detailed analyses on CRIU [3]—the state-of-the-art C/R technique on Linux reveal that it is inefficient for realizing remote fork for serverless computing: it can even be 2.7X slower than coldstart for serverless functions. We attribute the performance issues to three folds: First, checkpointing the memory is costly, whose overhead is proportional to the function's working set. Note that checkpointing is unnecessary for function startup but is crucial when considering state sharing. Second, transferring the file between machines is slow and is usually unnecessary because serverless functions typically touch a subset of the container memory [34]. Finally, accessing the remote file pays the software overheads of distributed file systems.

4 RDMA-OS co-designed fast remote fork

To this end, we present MITOSIS: an OS-codesigned ultra-fast remote fork primitive for serverless computing. Our key observation is that the OS can leverage RDMA—a widely adopted datacenter networking feature to efficiently realize remote fork by imitating local fork. Specifically, OS can directly access the remote physical memory with RDMA bypassing remote OS and CPU [31], which is extremely efficient. Therefore, the parent container no longer needs to checkpoint its memory to the file. The child container can leverage RDMA to directly access the parent's memory, bypassing the software overheads introduced by copying the file and accessing the distributed file system.

We implemented MITOSIS on Linux with its core functionalities written in Rust as a loadable kernel module. It can remote-fork $10,000$ containers on 5 machines within 0.86 second. MITOSIS is fully compatible with mainstream containers (e.g., runC [6]), making integration with existing container-based serverless platforms seamlessly. Though being efficient, MITOSIS preserves the security model of containers, i.e., the OS and hardware (RDMA-capable NIC) are trustworthy while malicious containers (functions) may exist.

To demonstrate the efficiency and efficacy, we integrated MITOSIS with Fn [37], a popular open-source serverless platform. Under load spikes in real-world serverless workloads, MITOSIS reduces the 99^{th} percentile latency of the spiked function by 89% with orders of magnitude lower memory usage. For a real-world serverless workflow (FINRA [7]), MITOSIS reduces its execution time by 86%.

5 Key Results and Contributions

We highlight our contributions as follows:

- **Problem**: We analyze the performance-resource provision trade-off of existing container startup methods, and the costs of state transfer between functions.
- **MITOSIS**: An RDMA-co-designed OS remote fork that quickly launches containers on remote machines without provisioned concurrency and enables efficient function state transfer. Specifically, MITOSIS achieves fast startup with no provisioned concurrency ($O(1)$), and supports transparent state sharing between serverless functions.
- **Demonstration**: We implemented MITOSIS on Linux and integrated it to Fn. Evaluations on on both microbenchmarks and real-world serverless applications demonstrate the efficacy of MITOSIS. The preprint of MITOSIS can be found at https://arxiv.org/abs/2203.10225. The source code is also publically available at https://github.com/ProjectMitosisOS.

References

[1] Apache OpenWhisk Composer. https://github.com/apache/openwhisk-composer, 2022.

[2] AWS Step Functions. https://aws.amazon.com/step-functions/, 2022.

[3] CRIU Website. https://www.criu.org/Main_Page, 2022.

[4] Docker Website. https://www.docker.com/, 2022.

[5] Provisioned concurrency for lambda functions. https://aws.amazon.com/cn/blogs/aws/new-provisioned-concurrency-for-lambda-functions/, 2022.

[6] runc. https://github.com/opencontainers/runc, 2022.

[7] United States Financial Industry Regulatory Authority. https://aws.amazon.com/cn/solutions/case-studies/finra-data-validation/, 2022.

[8] AKKUS, I. E., CHEN, R., RIMAC, I., STEIN, M., SATZKE, K., BECK, A., ADITYA, P., AND HILT, V. SAND: towards high-performance serverless computing. In *2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018* (2018), H. S. Gunawi and B. Reed, Eds., USENIX Association, pp. 923–935.

[9] AO, L., PORTER, G., AND VOELKER, G. M. Faasnap: Faas made fast using snapshot-based vms. In *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022* (2022), Y. Bromberg, A. Kermarrec, and C. Kozyrakis, Eds., ACM, pp. 730–746.

[10] AWS. Aws lambda. https://aws.amazon.com/lambda, 2022.

[11] CLOUD, A. Alibaba serverless application engine. https://www.aliyun.com/product/aliware/sae, 2022.

[12] CRIU. CRIU Usage scenarios. https://criu.org/Usage_scenarios, 2022.

[13] DU, D., LIU, Q., JIANG, X., XIA, Y., ZANG, B., AND CHEN, H. Serverless computing on heterogeneous computers. In *ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022* (2022), B. Falsafi, M. Ferdman, S. Lu, and T. F. Wenisch, Eds., ACM, pp. 797–813.

[14] DU, D., YU, T., XIA, Y., ZANG, B., YAN, G., QIN, C., WU, Q., AND CHEN, H. Catalyzer: Sub-millisecond startup for serverless computing with initialization-less booting. In *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020* (2020), J. R. Larus, L. Ceze, and K. Strauss, Eds., ACM, pp. 467–481.

[15] FOR AWS LAMBDA CONTAINER REUSE, B. P. https://medium.com/capital-one-tech/best-practices-for-aws-lambda-container-reuse-6ec45c74b67e, 2022.

[16] GOOGLE. Google serverless computing. https://cloud.google.com/serverless, 2022.

[17] HELLERSTEIN, J. M., FALEIRO, J. M., GONZALEZ, J., SCHLEIER-SMITH, J., SREEKANTI, V., TUMANOV, A., AND WU, C. Serverless computing: One step forward, two steps back. In *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings* (2019), www.cidrdb.org.

[18] HENDRICKSON, S., STURDEVANT, S., HARTER, T., VENKATARAMANI, V., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. Serverless computation with openlambda. In *8th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud 2016, Denver, CO, USA, June 20-21, 2016* (2016), A. Clements and T. Condie, Eds., USENIX Association.

[19] HUAWEI. Huawei clound functions. https://developer.huawei.com/consumer/en/agconnect/cloud-function/, 2022.

[20] JIA, Z., AND WITCHEL, E. Boki: Stateful serverless computing with shared logs. In *SOSP '21: ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event / Koblenz, Germany, October 26-29, 2021* (2021), R. van Renesse and N. Zeldovich, Eds., ACM, pp. 691–707.

[21] JIA, Z., AND WITCHEL, E. Nightcore: efficient and scalable serverless computing for latency-sensitive, interactive microservices. In *ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, USA, April 19-23, 2021* (2021), T. Sherwood, E. D. Berger, and C. Kozyrakis, Eds., ACM, pp. 152–166.

[22] KLIMOVIC, A., WANG, Y., STUEDI, P., TRIVEDI, A., PFEFFERLE, J., AND KOZYRAKIS, C. Pocket: Elastic ephemeral storage for serverless analytics. *login Usenix Mag. 44*, 1 (2019).

[23] KOTNI, S., NAYAK, A., GANAPATHY, V., AND BASU, A. Faastlane: Accelerating function-as-a-service workflows. In *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021* (2021), I. Calciu and G. Kuenning, Eds., USENIX Association, pp. 805–820.

[24] MICROSOFT. Azure functions. https://azure.microsoft.com/en-us/services/functions/, 2022.

[25] OAKES, E., YANG, L., ZHOU, D., HOUCK, K., HARTER, T., ARPACI-DUSSEAU, A., AND ARPACI-DUSSEAU, R. SOCK: Rapid task provisioning with serverless-optimized containers. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)* (Boston, MA, July 2018), USENIX Association, pp. 57–70.

[26] PU, Q., VENKATARAMAN, S., AND STOICA, I. Shuffling, fast and slow: Scalable analytics on serverless infrastructure. In *16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, Boston, MA, February 26-28, 2019* (2019), J. R. Lorch and M. Yu, Eds., USENIX Association, pp. 193–206.

[27] QI, S., MONIS, L., ZENG, Z., WANG, I., AND RAMAKRISHNAN, K. K. SPRIGHT: extracting the server from serverless computing! high-performance ebpf-based event-driven, shared-memory processing. In *SIGCOMM '22: ACM SIGCOMM 2022 Conference, Amsterdam, The Netherlands, August 22 - 26, 2022* (2022), F. Kuipers and A. Orda, Eds., ACM, pp. 780–794.

[28] SHAHRAD, M., FONSECA, R., GOIRI, I., CHAUDHRY, G., BATUM, P., COOKE, J., LAUREANO, E., TRESNESS, C., RUSSINOVICH, M., AND BIANCHINI, R. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020* (2020), A. Gavrilovska and E. Zadok, Eds., USENIX Association, pp. 205–218.

[29] SMITH, J. M., AND IOANNIDIS, J. *Implementing remote fork () with checkpoint/restart*. Department of Computer Science, Columbia Univ., 1987.

[30] SREEKANTI, V., WU, C., LIN, X. C., SCHLEIER-SMITH, J., GONZALEZ, J. E., HELLERSTEIN, J. M., AND TUMANOV, A. Cloudburst: Stateful functions-as-a-service. *Proc. VLDB Endow. 13*, 12 (jul 2020), 2438–2452.

[31] TSAI, S.-Y., AND ZHANG, Y. Lite kernel rdma support for datacenter applications. In *Proceedings of the 26th Symposium on Operating Systems Principles* (New York, NY, USA, 2017), SOSP '17, ACM, pp. 306–324.

[32] USTIUGOV, D., PETROV, P., KOGIAS, M., BUGNION, E., AND GROT, B. Benchmarking, analysis, and optimization of serverless function snapshots. In *ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, USA, April 19-23, 2021* (2021), T. Sherwood, E. D. Berger, and C. Kozyrakis, Eds., ACM, pp. 559–572.

[33] WANG, A., CHANG, S., TIAN, H., WANG, H., YANG, H., LI, H., DU, R., AND CHENG, Y. Faasnet: Scalable and fast provisioning of custom serverless container runtimes at alibaba cloud function compute. In *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021* (2021), I. Calciu and G. Kuenning, Eds., USENIX Association, pp. 443–457.

[34] WANG, K. A., HO, R., AND WU, P. Replayable execution optimized for page sharing for a managed runtime environment. In *Proceedings of the Fourteenth EuroSys Conference 2019, Dresden, Germany, March 25-28, 2019* (2019), G. Candea, R. van Renesse, and C. Fetzer, Eds., ACM, pp. 39:1–39:16.

[35] WANG, L., LI, M., ZHANG, Y., RISTENPART, T., AND SWIFT, M. Peeking behind the curtains of serverless platforms. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)* (Boston, MA, July 2018), USENIX Association, pp. 133–146.

[36] WEBSITE, A. O. https://openwhisk.apache.org, 2022.

[37] WEBSITE, F. P. https://fnproject.io, 2021.