Nightcore: Efficient and Scalable Serverless Computing for Latency-Sensitive, Interactive Microservices
Extended Abstract

Zhipeng Jia and Emmett Witchel

The University of Texas at Austin

1. Motivation

The microservice architecture is a popular software engineering approach for building large-scale online services. In microservice-based applications, loosely coupled microservices communicate with each other via pre-defined APIs, mostly using remote procedure calls (RPC) [14]. The dominant design pattern for microservices is that each microservice is an RPC server and they are deployed on top of a container orchestration platform such as Kubernetes [14, 7, 4]. Serverless functions, or function as a service (FaaS), provide a simple programming model of stateless functions which are a natural substrate for implementing the stateless RPC handlers of microservices, as an alternative to containerized RPC servers.

However, current FaaS systems have invocation latency overheads in the 1–10s of milliseconds range [1, 8, 19] (see Table 1), making them a poor choice for latency-sensitive interactive microservices, where RPC handlers only run for hundreds of microseconds to a few milliseconds [14, 27, 26, 18]. The microservice architecture also requires a high invocation rate for FaaS systems, as our experiments show that 100K RPCs are processed per second on five 8-vCPU VMs, when running social network microservices [14]. Therefore, for a FaaS system to efficiently support interactive microservices, it must achieve at least **two performance goals**, which are not accomplished by existing FaaS systems: (1) invocation latency overheads are well within 100μs; (2) the invocation rate must scale to 100K/s with low CPU usage.

2. Limitations of the State of the Art

Invocation latency overheads of FaaS systems are largely overlooked, as recent studies on serverless computing focus on data intensive workloads [23, 16, 13, 9, 12, 11, 25], where function execution times range from hundreds of milliseconds to a few seconds. There is no previous FaaS system that directly addresses the problem of efficient support for microsecond-scale microservices that also provides container-level isolation.

Faasm [25] achieves invocation latency overheads of hundreds of microseconds and invocation rates of 4K/s on a 8-core machine, despite targeting at machine learning workloads, not microservices. However, Faasm leverages software-based fault isolation (SFI) provided by WebAssembly, which provides less isolation than containers. We prefer container-based isolation because that is the standard set by containerized RPC servers and provided by popular FaaS systems such as Apache OpenWhisk [6] and OpenFaaS [5].

FaaS systems	50th	99th	99.9th
AWS Lambda	10.4 ms	25.8 ms	59.9 ms
OpenFaaS [5]	1.09 ms	3.66 ms	5.54 ms
Nightcore (external)	285 μs	536 μs	855 μs
Nightcore (internal)	39 μs	107 μs	154 μs

Table 1: Invocation latencies of a warm nop function.

3. Key Insights

Current hardware and software systems are not well tuned for microsecond-scale latencies [10]. To achieve our target performance for a FaaS runtime appropriate for microservices (invocation latencies of $\leq$100μs and invocation rates of $\geq$100K/s), our design must ruthlessly hunt these "*killer microseconds*". Our FaaS runtime (Nightcore) contains several innovations to achieve microsecond-scale overheads, including a fast path for internal function calls, low-latency message channels for IPC, efficient threading for I/O, and function executions with dynamically computed concurrency hints.

Existing FaaS systems like OpenFaaS [5] and Apache OpenWhisk [6] share a generic high-level design: all function requests are received by a *frontend* (mostly an API gateway), and then forwarded to independent *backends* where function code executes. The frontend and backends mostly execute on separate servers for fault tolerance, which requires invocation latencies that include at least one network round trip. Although datacenter networking performance is improving, round-trip times (RTTs) between two VMs in the same AWS region range from 101μs to 237μs [3]. Nightcore is motivated by noticing the prevalence of internal function calls made during function execution (see Figure 1 in the main paper). An *internal function call* is one that is generated by the execution of a microservice, not generated by a client (in which case it would be an external function call, received by the gateway). What we call internal function calls have been called "chained function calls" in previous work [25]. Nightcore schedules internal function calls on the same backend server that made the call, eliminating a trip to through the gateway and lowering latency (main paper § 3.2).

Nightcore's support for internal function calls makes most communication local, which means its inter-process communications (IPC) must be efficient. Popular, feature-rich RPC libraries like gRPC work for IPC (over Unix sockets), but gRPC's protocol adds overheads of $\sim 10\mu s$ [10], motivating Nightcore to design its own message channels for IPC (main paper § 3.1). Nightcore's message channels are built on top of OS pipes, and transmit fixed-size 1KB messages, because previous studies [18, 21] show that 1KB is sufficient for most microservice RPCs. Our measurements show Nightcore's message channels deliver messages in $3.4\mu s$, while gRPC over Unix sockets takes $13\mu s$ for sending 1KB RPC payloads.

Previous work has shown microsecond-scale latencies in Linux's thread scheduler [10, 26], leading dataplane OSes [22, 20] to build their own schedulers for lower latency. Nightcore relies on Linux's scheduler, because building an efficient, time-sharing scheduler for microsecond-scale tasks is an ongoing research topic [24, 17, 20]. To support an invocation rate of $\geq$100K/s, Nightcore's engine (main paper § 4.1) uses event-driven concurrency, allowing it to handle many concurrent I/O events with a small number of OS threads. Our measurements show that 4 OS threads can handle an invocation rate of 100K/s. Furthermore, I/O threads in Nightcore's engine can wake function worker threads (where function code are executed) via message channels, which ensures the engine's dispatch suffers only a single wake-up delay from Linux's scheduler.

Existing FaaS systems do not provide concurrency management to applications. However, stage-based microservices create internal load variations even under a stable external request rate [29, 15]. Previous studies [29, 28, 15] indicate overuse of concurrency for bursty loads can lead to worse overall performance. Nightcore, unlike existing FaaS systems, actively *manages* concurrency providing dynamically computed targets for concurrent function executions that adjust with input load (main paper § 3.3). Nightcore's managed concurrency flattens CPU utilization (see Figure 4 in the main paper) such that overall performance and efficiency are improved, as well as being robust under varying request rates (main paper § 5.2).

4. Main Artifacts

We implement the Nightcore prototype with 11,549 lines of code (mostly in C++). Our prototype currently supports serverless functions written in C/C++, Go, Node.js, and Python.

We evaluate the Nightcore prototype on four interactive microservices, each with a custom workload. Three are from DeathStarBench [14] and one is from Google Cloud [4]. These workloads are originally implemented in RPC servers, and we port them to Nightcore. We compare Nightcore with containerized RPC servers and OpenFaaS, and conducted our experiments using AWS EC2 VMs.

5. Key Results and Contributions

By carefully finding and eliminating microsecond-scale overheads, Nightcore overcomes one long-lasting limitation of existing FaaS systems – millisecond-scale invocation latencies.

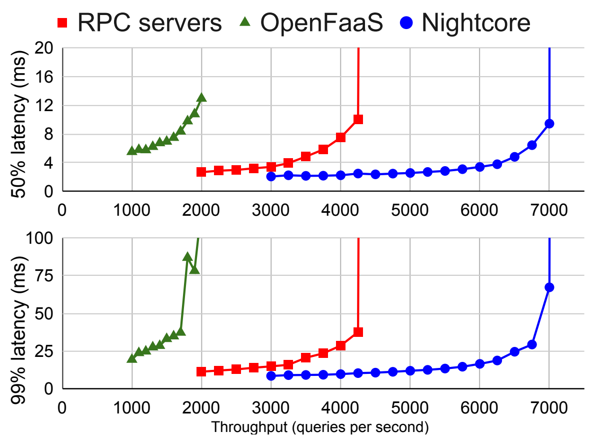

Figure 1: Comparison of Nightcore with other systems on hotel reservation microservices from DeathStarBench [14, 2].

Nightcore's performance enables the first practical serverless platform for latency-sensitive microservices.

Nightcore achieves invocation latencies of 10s of microseconds (see Table 1), while providing strong, container-based isolation between functions. Our evaluation of Nightcore not only shows Nightcore overwhelmingly surpassing OpenFaaS when running microservices, but also demonstrates serverless functions can be a more efficient choice for latency-sensitive microservices than RPC servers (Figure 1 gives one example, and refer to main paper § 5 for more experimental results).

This paper makes the following contributions.

- Nightcore is a FaaS runtime optimized for microsecond-scale microservices. It achieves invocation latency overheads under $100\mu s$ and efficiently supports invocation rates of 100K/s with low CPU usage.
- Nightcore's design uses diverse techniques to eliminate microsecond-scale overheads, including a fast path for internal function calls, low-latency message channels for IPC, efficient threading for I/O, and function executions with dynamically computed concurrency hints.
- With containerized RPC servers as the baseline, Nightcore achieves 1.36×–2.93× higher throughput and up to 69% reduction in tail latency, while OpenFaaS only achieves 29%–38% of baseline throughput and increases tail latency by up to 3.4× (main paper § 5).

Why ASPLOS? Nightcore is a networked software system, built to preserve the microsecond-scale latencies of hardware. ASPLOS provides the right audience to appreciate the crossover challenges of building such a serverless computing system.

Citation for Most Influential Paper Award. Nightcore showed that microsecond-scale serverless systems were possible, finally making them a viable substrate for latency-sensitive microservices. Follow-on work sped the convergence of system support for microservices on conventional and SmartNIC hardware.

References

[1] AWS Lambda FAQs. [Accessed Aug, 2020].

[2] delimitrou/DeathStarBench: Open-source benchmark suite for cloud microservices. [Accessed Aug, 2020].

[3] firecracker/network-performance.md at master · firecracker-microvm/firecracker. [Accessed Aug, 2020].

[4] GoogleCloudPlatform/microservices-demo. [Accessed Aug, 2020].

[5] OpenFaaS | Serverless Functions, Made Simple. [Accessed Aug, 2020].

[6] Uncovering the magic: How serverless platforms really work! [Accessed Aug, 2020].

[7] Why should you use microservices and containers? [Accessed Aug, 2020].

[8] Why so slow? - Binaris Blog. [Accessed Aug, 2020].

[9] Lixiang Ao, Liz Izhikevich, Geoffrey M. Voelker, and George Porter. Sprocket: A serverless video processing framework. In *Proceedings of the ACM Symposium on Cloud Computing, SoCC 2018, Carlsbad, CA, USA, October 11-13, 2018*, pages 263–274, 2018.

[10] Luiz Barroso, Mike Marty, David Patterson, and Parthasarathy Ranganathan. Attack of the killer microseconds. *Commun. ACM*, 60(4):48–54, March 2017.

[11] Joao Carreira, Pedro Fonseca, Alexey Tumanov, Andrew Zhang, and Randy Katz. Cirrus: A serverless framework for end-to-end ml workflows. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '19, page 13–24, New York, NY, USA, 2019. Association for Computing Machinery.

[12] Sadjad Fouladi, Francisco Romero, Dan Iter, Qian Li, Shuvo Chatterjee, Christos Kozyrakis, Matei Zaharia, and Keith Winstein. From laptop to lambda: Outsourcing everyday jobs to thousands of transient functional containers. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 475–488, Renton, WA, July 2019. USENIX Association.

[13] Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. Encoding, fast and slow: Low-latency video processing using thousands of tiny threads. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 363–376, Boston, MA, March 2017. USENIX Association.

[14] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayantara Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Yuan He, and Christina Delimitrou. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems. In *Proceedings of the Twenty Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, April 2019.

[15] Călin Iorgulescu, Reza Azimi, Youngjin Kwon, Sameh Elnikety, Manoj Syamala, Vivek Narasayya, Herodotos Herodotou, Paulo Tomita, Alex Chen, Jack Zhang, and Junhua Wang. Perfiso: Performance isolation for commercial latency-sensitive services. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '18, page 519–531, USA, 2018. USENIX Association.

[16] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. Occupy the cloud: Distributed computing for the 99%. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, pages 445–451, New York, NY, USA, 2017. ACM.

[17] Kostis Kaffes, Timothy Chong, Jack Tigar Humphries, Adam Belay, David Mazières, and Christos Kozyrakis. Shinjuku: Preemptive scheduling for μsecond-scale tail latency. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation*, NSDI'19, page 345–359, USA, 2019. USENIX Association.

[18] Nikita Lazarev, Neil Adit, Shaojie Xiang, Zhiru Zhang, and Christina Delimitrou. Dagger: Towards efficient rpcs in cloud microservices with near-memory reconfigurable nics. *arXiv preprint arXiv:2007.08622*, 2020.

[19] Collin Lee and John Ousterhout. Granular computing. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, HotOS '19, page 149–154, New York, NY, USA, 2019. Association for Computing Machinery.

[20] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. Shenango: Achieving high cpu efficiency for latency-sensitive datacenter workloads. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation*, NSDI'19, page 361–377, USA, 2019. USENIX Association.

[21] Arash Pourhabibi, Siddharth Gupta, Hussein Kassir, Mark Sutherland, Zilu Tian, Mario Paulo Drumond, Babak Falsafi, and Christoph Koch. Optimus prime: Accelerating data transformation in servers. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 1203–1216, New York, NY, USA, 2020. Association for Computing Machinery.

[22] George Prekas, Marios Kogias, and Edouard Bugnion. Zygos: Achieving low tail latency for microsecond-scale networked tasks. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 325–341, New York, NY, USA, 2017. Association for Computing Machinery.

[23] Qifan Pu, Shivaram Venkataraman, and Ion Stoica. Shuffling, fast and slow: Scalable analytics on serverless infrastructure. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 193–206, Boston, MA, 2019. USENIX Association.

[24] Henry Qin, Qian Li, Jacqueline Speiser, Peter Kraft, and John Ousterhout. Arachne: Core-aware thread management. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, page 145–160, USA, 2018. USENIX Association.

[25] Simon Shillaker and Peter Pietzuch. Faasm: Lightweight isolation for efficient stateful serverless computing. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 419–433. USENIX Association, July 2020.

[26] Akshitha Sriraman and Thomas F. Wenisch. μtune: Auto-tuned threading for oldi microservices. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, page 177–194, USA, 2018. USENIX Association.

[27] M. Sutherland, S. Gupta, B. Falsafi, V. Marathe, D. Pnevmatikatos, and A. Daglis. The nebula rpc-optimized architecture. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 199–212, 2020.

[28] Matt Welsh and David Culler. Adaptive overload control for busy internet servers. In *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, page 4, USA, 2003. USENIX Association.

[29] Matt Welsh, David Culler, and Eric Brewer. Seda: An architecture for well-conditioned, scalable internet services. *SIGOPS Oper. Syst. Rev.*, 35(5):230–243, October 2001.