Optimizing FaaS Performance by Merging TLB Entries across Processes

Context

The modern cloud has introduced various new service paradigms such as Functions-as-a-Service (FaaS). In FaaS deployments, each function is hosted under its own isolated environment, often a language runtime process (Node.js, Python, etc.) running in a Linux container or virtual machine. Consequently, such a deployment will contain thousands of separate processes running the same binary. Such a high-density deployment causes various issues on the hosting environment: slow startup times [1]; communication inefficiencies [2]; security concerns [3]; etc.

Each function runtime process is given its own page tables for translating its virtual addresses to physical addresses. As a result, processes' translations will not be shared even when running the same program binary. Yet, modern runtimes are large programs containing lots of code and data. For example, Node v16 is a single binary 80 MB in size. As a consequence, duplicating translations puts pressure on the CPU's TLB, responsible for caching page table translations. Increased TLB miss rates has a damaging effect when switching between runtime processes, leading to poor performance and hosting density [4].

Project description

In this project, we aim to improve the performance of FaaS processes by sharing TLB entries across multiple processes. To elaborate, we plan to utilize the global bit of x86 page tables, where address translations marked with this bit will stay resident in TLB no matter the process [5]. By making sure that all processes of a binary use the same (global) mappings, we reduce that binary's code and data TLB footprint to a constant figure regardless of the number of runtime processes.

The main objectives of this project are as following:

- Implement a mechanism for managing global memory mappings of some binaries;
- Implement facilities for loading binaries and starting processes using these global memory mappings;
- Implement protection of mappings and binary contents shared across processes;
- Study the performance impact of address translation overhead with increasing function density, with and without our system.

Contact information

Boris Teabe, Tu Dinh Ngoc
SEPIA team, Institut de Recherche en Informatique de Toulouse (IRIT)
boris.teabedjomgwe@enseeiht.fr, dinhngoc.tu@irit.fr

References

[1] Dong Du, Tianyi Yu, Yubin Xia, Binyu Zang, Guanglu Yan, Chenggang Qin, Qixuan Wu, and Haibo Chen. 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, page 467–481, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371025. doi:10.1145/3373376.3378512.

[2] Zhipeng Jia and Emmett Witchel. Nightcore: Efficient and scalable serverless computing for latency-sensitive, interactive microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, page 152–166, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383172. doi:10.1145/3445814.3446701.

[3] Deepak Sirone Jegan, Liang Wang, Siddhant Bhagat, and Michael Swift. Guarding serverless applications with Kalium. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4087–4104, Anaheim, CA, August 2023. USENIX Association. ISBN 978-1-939133-37-3. URL `https://www.usenix.org/conference/usenixsecurity23/presentation/jegan`.

[4] Yufeng Zhou, Alan L. Cox, Sandhya Dwarkadas, and Xiaowan Dong. The impact of page size and microarchitecture on instruction address translation overhead. *ACM Trans. Archit. Code Optim.*, 20 (3), July 2023. ISSN 1544-3566. doi:10.1145/3600089.

[5] Intel Corp. *Intel® 64 and IA-32 Architectures Software Developer's Manual*, volume 3A, page 4-44. December 2022.