Isolating data structures in the Linux Kernel for stronger performance isolation guarantees

Ross Cannon, Eliot Wearden*, Yuvraj Patel

Shared environments such as datacenters account for the largest growth and value in computing as they provide highly efficient enterprise computing power, reducing costs and CO_2 emissions. However, datacenters have an Achilles' heel. While we want to idealize shared environments as being perfectly isolated, in reality, applications and services run atop concurrent shared infrastructure, where multiple tenants share physical hardware. As tenants with varied requirements run concurrently, strong performance isolation must be guaranteed to prevent one tenant's workload harming another tenant's performance, breaking higher-level goals defined in service-level agreements.

Despite multiple solutions controlling the usage of the four main resources – CPU, disk, memory, and network [2, 4, 7, 12, 5, 11], guaranteeing strong performance isolation is hard. Several reports demonstrate performance interference [6, 3, 1, 10]. Recently, it has been shown how unfair usage of synchronization primitives can lead to fairness [9] and security issues [8], degrading performance. By considering synchronization primitives as resources, one can improve isolation guarantees by providing lock usage fairness. However, we hypothesize there are other aspects within a system to consider which will strengthen isolation. For this purpose, we propose looking to shared data structures within operating systems.

Concurrent data structures are used to implement various applications such as key-value stores, hypervisors, and operating systems. Given their concurrent nature, multiple processes interact with these data structures. In a shared environment such as the operating system, these processes may or may not belong to the same tenants. Moreover, each tenant may have performance requirements that need to be guaranteed by the operating system. Therefore, it is imperative to ensure strong performance isolation in these environments.

Consider a linked list within an operating system which supports a functionality that tenants may access via syscalls. Consider two tenants making accesses, where the first inserts millions of kernel objects while the second only inserts a handful. The second tenant, while accessing its entries, may have to traverse the entries of the other tenant, wasting time unnecessarily through no fault of their own, degrading performance. As such, it makes sense to isolate the list so one tenant's actions do not impact the performance of another, thereby strengthening performance isolation.

Based on the above, we analyze various data structures, such as the inode cache and futex table, within the Linux kernel. Recent work [8] has targeted these structures to create performance interference and denial-of-service attacks. We aim to isolate the cost of other users' work from each other while still sharing data structures. However, there exists a conundrum where isolation is antithetical to sharing.

We present a strategy to reconcile sharing with isolation concerning Linux kernel data structures. In general, we create per-user substructures within shared structures, letting users experience predictable performance affected only by their workload, not others. In the futex table, a two-level hash isolates users into private buckets within the shared table, eliminating interference. For the inode cache, per-user lists paired with Bloom filters minimize global state checks while preventing collisions. Early results in futex testing show promising results in eliminating attack-induced latency under synthetic workloads.

References

[1] Another reason why your Docker containers may be slow. `https://devpress.csdn.net/linux/62f5542f7e6682346618a6ed.html`.

[2] Sudipto Das, Vivek R. Narasayya, Feng Li, and Manoj Syamala. Cpu sharing techniques for performance isolation in multi-tenant relational database-as-a-service. *Proc. VLDB Endow.*, 7(1):37–48, September 2013.

[3] Gianluca Borello. Container isolation gone wrong. `https://sysdig.com/blog/container-isolation-gone-wrong/`.

*Both authors have made equal contributions.

[4] Diwaker Gupta, Ludmila Cherkasova, Rob Gardner, and Amin Vahdat. Enforcing performance isolation across virtual machines in xen. Middleware '06, pages 342–362.

[5] Lanyue Lu, Yupu Zhang, Thanh Do, Samer Al-Kiswany, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Physical disentanglement in a container-based file system. OSDI'14, pages 81–96.

[6] Maxim Leonovich. Another reason why your Docker containers may be slow. https://medium.com/hackernoon/another-reason-why-your-docker-containers-may-be-slow-d37207dec27f.

[7] Paul Menage. CGroup documentation. https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt, 2018.

[8] Yuvraj Patel, Ye Chenhao, Akshat Sinha, Abigail Matthews, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Michael M. Swift. Using Trāṭr to tame adversarial synchronization. In Submission.

[9] Yuvraj Patel, Leon Yang, Leo Arulraj, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Michael M. Swift. Avoiding scheduler subversion using scheduler-cooperative locks. EuroSys '20.

[10] Pavlos Parissis. Troubleshooting: A journey into the unknown. https://medium.com/booking-com-infrastructure/troubleshooting-a-journey-into-the-unknown-e31b524fa86.

[11] David Shue, Michael J. Freedman, and Anees Shaikh. Performance isolation and fairness for multi-tenant cloud storage. OSDI'12, pages 349–362.

[12] Carl A. Waldspurger. Memory Resource Management in VMware ESX Server. OSDI '02.