XX

Graduate Category

1 Motivation

Cloud providers like AWS, or containerization systems like Kubernetes, support reserving CPU for *latency critical* (LC) services. Developers choose reservations based on expected peak load [9, 14, 15], but load is variable, leading to a utilization problem. *Best effort* (BE) workloads do not have reservations, so providers run them opportunistically on the same resources as LCs reserved; the resource that this work focuses on is CPU. The challenge is to honor resource reservations for LCs while opportunistically running BEs.

Current popular scheduling systems fail to honor reservations. *Figure* 1 shows the latency of an endpoint in a web application on Kubernetes that gets a users feed, and the throughput of a BE image resize job on the same cores. The mean response latency of the server jumps from ~7ms to ~15ms after starting the BE.

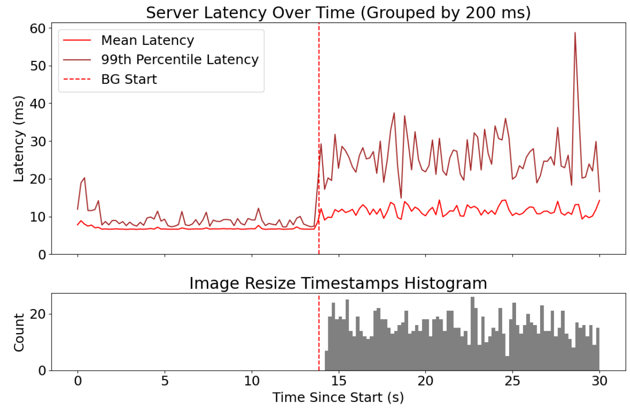

Figure 1: In Kubernetes, running a BE has latency impacts on an LC with reservations

2 Background & Problem

All Open Container Initiative compliant containers, including Kubernetes, enforce CPU reservations using `cgroups`' weight interface [3, 11, 12], as do VM frameworks, including Firecracker and libvirt [1, 4, 5].

A simplified experiment shows `cgroups` weights are unable to enforce reservations. We run a simple CPU-bound server sharing 4 cores with an image resize job, each in their own group. `cgroups` supports weights in the range [1,10000]. Figure 2 shows that running the server with different weights has no effect on the latency impact of the BE.

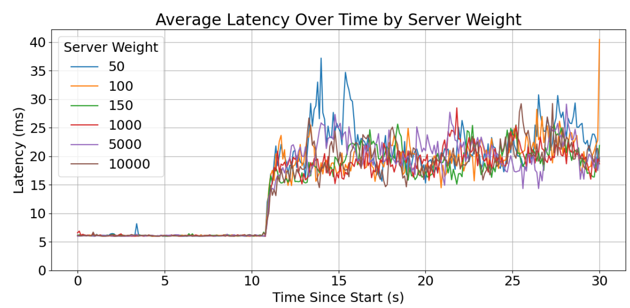

Figure 2: the weight of the server has little impact on how much the weight 1 BE task interferes

This latency impact happens because the BE occasionally runs uncontended on one core, while another has queued server threads. Figure 3 shows this happening in the trace: on core 6, the red BE thread runs for 10ms, while server threads, in shades of blue, are queued on the other cores.

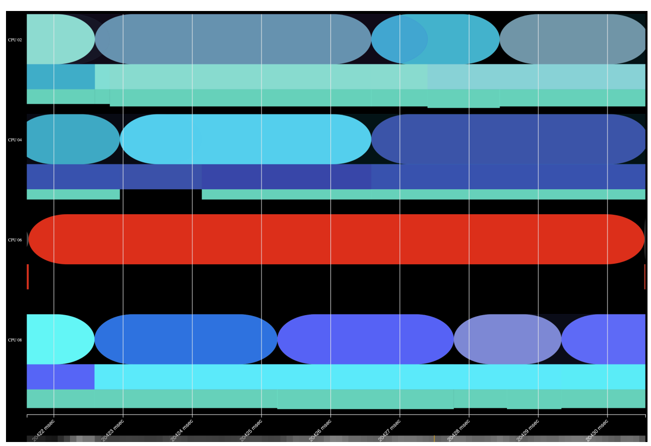

Figure 3: Core 6 runs an image resize process, unaware that the other cores have queued server threads

This happens because weights are only enforced within per-CPU runqueues. Load balancing eventually remedies imblances, but runs less frequently than scheduling does. For workloads with request processing times in the millisecond range, waiting for the load balancer influences final processing times.

In order to globally enforce weights, each scheduling decision would require a search of all runqueues for potentially higher-weight processes, which is too costly.

3 Related work

Some work focuses on isolating real-time applications in Linux [2, 13]. Wasted Cores [10] improves the idle behavior of Linux, whereas we change the way developers express what is LC and what is BE.

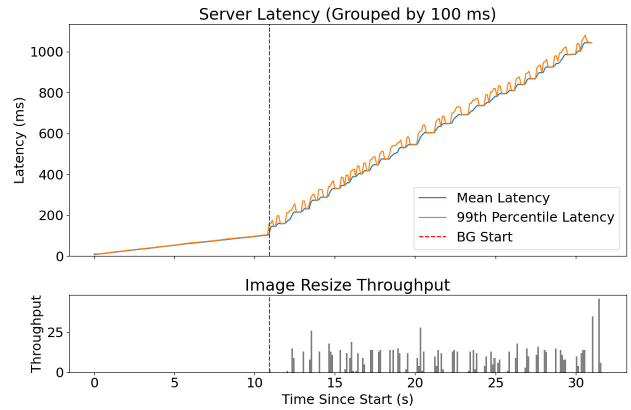

Figure 4: when running the server in real time, throttling degrades performance at high load

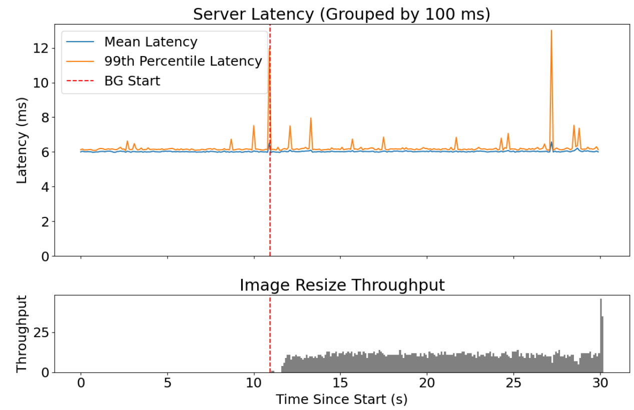

Figure 5: `BeClass` isolates the server's latencies from BE jobs

Linux's `sched_idle` policy addresses the observed weight inversion, but is still inadequate: running the microbenchmark using `sched_idle` leads to an increase in the mean latency from 6ms to 8ms.

Other schedulers run primarily in userspace in order to work around the kernel scheduler [6–8]. This paper identifies the core problem in the kernel scheduler and addresses it.

4 Approach & Uniqueness

In order to enforce reservations while running BE jobs opportunistically, our approach replaces weights with priority scheduling.

Enforcing priorities requires fewer global runqueue searches than weights, because searches only need to happen on *class boundary crossings*: on *exit*, when switching to running a low-class process, and on *entry*, when enqueueing a high-class process. These checks ensure that a core running a BE thread t knows that there are no queued LC threads anywhere: the *exit* check ensures there's none when starting to run t, and the *entry* check ensures if one wakes up while running t it will interrupt t.

Linux enforces priority across scheduling classes, but higher class schedulers are designed for real-time applications. Additionally, if these experience high load, Linux throttles them in order to not starve the default class. We see this happening in Figure 4, where throttling leads to spikes in the BE's throughput, and corresponding spikes in the server's latency.

We design a new scheduling class `BeClass` that sits at a lower priority than the default one, and enforces LCs' uncontended access to CPUs they reserved. To enforce reservations under high load, without throttling LCs or killing BEs, `BeClass` uses *parking*, wherein user-space code never runs, only kernel-level services.

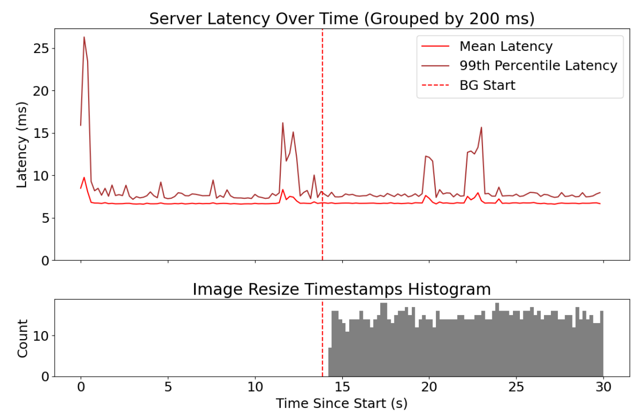

Figure 6: Kubernetes using `BeClass` honors rservations

5 Results

We run the microbenchmark experiment using `BeClass`; the results are in Figure 5. As desired, the latency of the server remains stable after the BE task starts, and the BE runs only in gaps where the core would otherwise be idle.

We also run the Kubernetes application using `BeClass`. Figure 6 shows that the baseline mean latency of the LC server stays around 6.5ms after starting the the BE image resizing.

Figure 7 shows parking enables the server to keep its reservation even under sustained 100% load. We run the same client load as Figure 4, with `BeClass` parking the BE. Notice that the BE does not make progress until the very end, when the server is done processing.

6 Contributions

We identify a serious limitation of `cgroups` weights: they are unable to honor the reservations of LC applications in the presence BE workloads. We show that this is because Linux uses per-core runqueues.

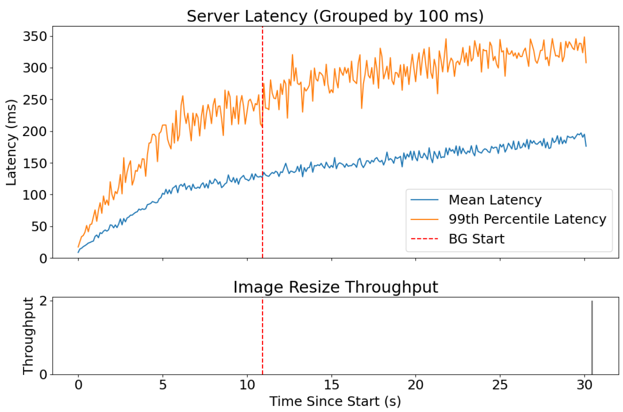

Figure 7: `BeClass` starves BE user-space threads, leaving all CPU-time to the LC

We propose an API that separates BE from LC by introducing `BeClass`. `BeClass` requires fewer cross-core interactions than a weight-based approach, and ensures that no BE is ever running when an LC is queued.

We implement this strict priority in Linux, and show that it allows `cgroups` and Kubernetes to ensure LCs' access to their reserved cores while running BE workloads opportunistically.

References

[1] Xiaohu Chai, Tianyu Zhou, Keyang Hu, Jianfeng Tan, Tiwei Bie, Anqi Shen, Dawei Shen, Qi Xing, Shun Song, Tongkai Yang, Le Gao, Feng Yu, Zhengyu He, Dong Du, Yubin Xia, Kang Chen, and Yu Chen. "Fork in the Road: Reflections and Optimizations for Cold Start Latency in Production Serverless Systems". In: *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, 2025. URL: https://www.usenix.org/system/files/osdi25-chai-xiaohu.pdf.

[2] Jonathan Corbet. *The state of realtime and embedded Linux*. online. 2024. URL: https://lwn.net/Articles/970555/.

[3] Docker. *dockerdocs: Resource constraints*. online. URL: https://docs.docker.com/engine/containers/resource_constraints/.

[4] libvirt Documentation. *CPU Allocation*. online. URL: https://libvirt.org/formatdomain.html#cpu-allocation.

[5] firecracker. *config-linux.md: Production Host Setup Recommendations*. github. URL: https://github.com/firecracker-microvm/firecracker/blob/main/docs/prod-host-setup.md.

[6] Joshua Fried, Zhenyuan Ruan, Amy Ousterhout, and Adam Belay. "Caladan: Mitigating Interference at Microsecond Timescales". In: *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pages 281–297. URL: https://www.usenix.org/conference/osdi20/presentation/fried.

[7] Calin 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: *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, July 2018, pages 519–532. URL: https://www.usenix.org/conference/atc18/presentation/iorgulescu.

[8] Yuekai Jia, Kaifu Tian, Yuyang You, Yu Chen, and Kang Chen. "Skyloft: A General High-Efficient Scheduling Framework in User Space". In: *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*. SOSP '24. Austin, TX, USA: Association for Computing Machinery, 2024, pages 265–279. URL: https://doi.org/10.1145/3694715.3695973.

[9] Will Kelly. *Overprovisioning in AWS? Cost-control tools and strategies can help*. online. 2013. URL: https://www.techtarget.com/searchaws/news/2240209793/Overprovisioning-in-AWS-Cost-control-tools-and-strategies-can-help.

[10] Jean-Pierre Lozi, Baptiste Lepers, Justin Funston, Fabien Gaud, Vivien Quéma, and Alexandra Fedorova. "The Linux scheduler: a decade of wasted cores". In: *Proceedings of the Eleventh European Conference on Computer Systems*. EuroSys '16. London, United Kingdom: Association for Computing Machinery, 2016. URL: https://doi.org/10.1145/2901318.2901326.

[11] Rory McClune. *How Do Containers Contain? Container Isolation Techniques*. online. 2021. URL: https://www.aquasec.com/blog/container-isolation-techniques/.

[12] opencontainers. *config-linux.md: Linux Container Configuration*. github. URL: https://github.com/opencontainers/runtime-spec/blob/main/config-linux.md.

[13] Kevin Purdy. *Real-time Linux is officially part of the kernel after decades of debate*. online. 2024. URL: https://arstechnica.com/gadgets/2024/09/real-time-linux-is-officially-part-of-the-kernel-after-decades-of-debate/.

[14] Zhenyuan Ruan, Seo Jin Park, Marcos K. Aguilera, Adam Belay, and Malte Schwarzkopf. "Nu: Achieving Microsecond-Scale Resource Fungibility with Logical Processes". In: *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pages 1409–1427. URL: https://www.usenix.org/conference/nsdi23/presentation/ruan.

[15] Muhammad Tirmazi, Adam Barker, Nan Deng, Md Ehtesam Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. "Borg: the Next Generation". In: *EuroSys'20*. Heraklion, Crete, 2020.