1 Summary

It is hard to appreciate this paper without understanding problems with existing kernel schedulers in Linux: namely, kernel programming is tough especially in terms of customizing policies based on new workloads/hardware, upgrading production machines is time-consuming, and only allows you to make per-CPU policies. The ghOSt Linux kernel scheduling framework is the authors' solution to these issues. It builds upon previous research by providing the ability to: delegate decisions to userspace, run scheduling policies in userspace processes, and allow fast abstractions, support variety of scheduling policies, and upgrade production machines efficiently (seconds vs hours).

2 Strengths of the paper

Given how low level kernel programming is, I never considered that there could be an API, like what ghOSt provides, which lets users customize their scheduler based on system preferences directly from userspace. Offering users this level of granular control no doubt liberates them in a way that allows them to further tinker with their systems in a manner that was previously considered infeasible.
I liked the clear evaluation criteria consisting of three questions that the authors used to evaluate ghOSt. The questions served to convey the most reasonable metrics for possible adopters to consider using ghOSt: *a* overheads specific to ghOSt; *b* comparison to prior work, like Shinjuku; and *c* evaluating the viability of ghOSt for large-scale and low-latency workloads like Google Snap, Google Search, and virtual machines.
I was surprised to see Google Search being used as a test bed for ghOSt in that the authors replaced the CFS scheduler on machines with it. It is impressive that they managed to reduce tail latency by 40-45% while maintaining throughput when using ghOSt for CPU and memory-intensive queries, as well as queries that access the SSD serviced by a collection of short-lived workers.

3 Weakness of the paper

This was perhaps the most technically dense paper of the semester so far. While the authors outlined the issues prevalent in classic schedulers as motivation to design ghOSt, I wish there was a figure that visually portrayed the difference between ghOSt and its traditional counterpart. A visual figure detailing this comparison would be especially useful for a reader like me in §3: Design.

4 Future work opportunities

Integrating the CPU and I/O schedulers[1] will likely yield even more performance gains, but doing so is tough - my lack of in-depth scheduler knowledge prevents me from concisely explaining why, but for starters, if it wasn't hard I'm sure the authors would have done it already. . .

[1] SOSP 2021 QA: Session 13: Scheduling: GhOSt: Fast Flexible User-Space Delegation of Linux ..., 2021. https://www.youtube.com/watch?v=AOPdm4y080.