# Don't get SLAmmed: know your deadlines

Hannah Gross
MIT
Graduate Student

Frans Kaashoek
MIT

## 1 Introduction

Developers use Service Level Objectives (SLOs) between teams and Service Level Agreements (SLAs) towards customers, which represent guarantees about uptime and maximal latencies of the API that team/company is in charge of [2]. Monitoring systems watch performance and page on-call developers if these guarantees are not being met [1].

However, the interface to many scheduling frameworks, such as Kubernetes [10] and many research systems [6], offers developers two options: latency-critical (LC) applications have concrete reservations, and jobs with no immediate deadline can be run as best-effort (BE) tasks. The scheduler bin-packs LC work, and runs BE work opportunistically.

For a developer to translate their requirements into this interface requires two steps: pick a category (LC or BE), and, if the its the former, generate a concrete reservation. Both of these steps are difficult: some work might not be completely LC or BE, and reservations require the developer to make estimations about peak load, and in fact incentivizes them to overestimate [9] — making achieving high utilization hard.

Imagine a web developer, whose website has four different types of work it has to do:
(1) load static pages (eg the homepage) — shortest and very time critical; (2) load dynamic pages (eg a users profile page) — slightly longer and less time critical; (3) foreground data processing (eg processing a user uploaded file of image) — requires a fair amount of processing but still user-facing and thus latency sensitive; (4) background data processing (eg updating a data warehouse) — runs overnight and just needs to finish by morning.

The only candidate for BE is (4). The other three are user-facing and as such are LC and require reservations. But among 1-3 there are levels of criticality: it is essential that the homepage load time be low and constant, but processing a user upload can take more time during high load.

This submission presents *Cole*, a SLA-based scheduling system. Central to Cole is the observation that the priorities of utilization and latency don't have to be opposing. Cole changes the interface with which resource requirements are communicated: it makes *deadlines* and *maximum compute times* the central metrics provided, thereby side-stepping the LC/BE binary, and making the developer's requirements explicit to the scheduler.

In Cole, developers submit jobs, attached with the maximum execution time as well as a deadline. In the website example, rather than estimate load for each job, the resources required to run it, then add 20% padding, developers can simply give the scheduler the handler for each endpoint, and attach to each its deadline and an experiential maximum compute time.

## 2 Design

When a job needs to be run, Cole initially routes it to the relevant shard of the global scheduler, which then forwards the job to a chosen single machine.

### Distributed Global Scheduler

Each global scheduler shard has a set of machines that it can run jobs on. The goal of the global scheduler is to keep load as evenly distributed among these machines as possible, in the sense that each machine has a maximally heterogenous set of deadlines and computes. This allows processes with more *slack* (difference between deadline and max compute) to fill in gaps where shorter processes might have finished earlier than expected (similarly to how best effort work now opportunistically uses resources not currently used by latency critical tasks).

Depending on the amount of slack the global scheduler will either immediately place the job, or will probe a dynamic number of machines in order to find a good match.

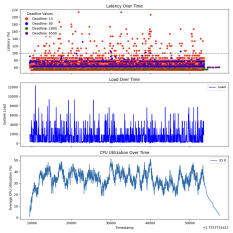
### Single machine

On the individual machines, the goal is to to meet all the deadlines of the work assigned to the machine. This has two parts: one is a dispatcher thread, which handles communication with its global scheduler shard and does admission control; and the other is a local scheduler that prioritizes work by deadline.

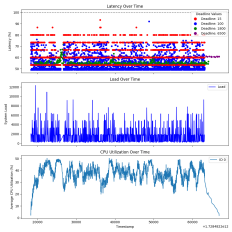
Cole's admission control is pessimistic, and assumes that every process will use the maximum compute time. The dispatcher looks at the amount of slack each process has (given the original slack and how much time it has already spent waiting), and judges whether with the new process it could still meet all the maximum compute guarantees.

The local scheduler approximates Earliest Deadline First (EDF) scheduling. Cole acheives EDF scheduling by using Earliest Eligible Virtual Deadline First (EEVDF) [13] — originally proposed in the 90s and recently integrated in linux [3].

In EEVDF, processes make requests for resources, and each request is assigned an eligible time and a deadline. At every scheduling point, of the eligible processes the one



**(a)** Latency and load in original linux



**(b)** Latency and load in modified linux

**Figure 1:** Results of running the same workload on (a) unmodified and (b) modified linux.

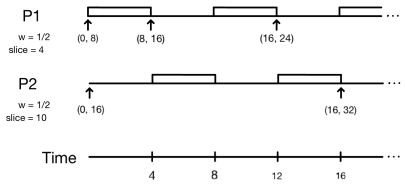



**Figure 2:** An example EEVDF timeline. Arrows represent requests, denoted with time eligible and deadline, and boxes show which process was chosen to run.

with the earliest deadline is chosen to run next. Eligible times and deadlines are in virtual time, which allows for oversubscription.

Processes need to be able to be ineligible in order to maintain fairness. If a process ran for the last tick and completed its latest request, its next eligible time will be set to be in the future (a weighted estimation of by when it should have gotten the time it just did, based on its weight as well the the total system load). This protects against starvation of processes with longer deadlines. Figure 2 shows a scheduling flow for two equally weighted processes.

If jobs make requests often and for small time increments, this policy is similar to Process Sharing (PS) — each process runs for a scheduling quanta before becoming ineligible. However, on the other extreme you get behavior closer to EDF — if jobs only ever make one request for all the resources that they think they will need, then the requests never reset and all processes are always elgibible. Since the EEVDF scheduler is by definition choosing the eligible process with the earliest deadline (and all processes are always eligible), it is effectively implementing EDF.

And creating this edge condiiton is what the dispatcher does — when it spawns a process for a new job, it sets the requested timeslice of that job to be the jobs maximum compute.

## 3 Current State

Implementing this design required changing linux's EEVDF implementation. Linux's EEVDF scheduler has a strong tenant of avoiding unfairness, and as a result implements eligibility solely as a function of how much time a process had gotten compared to other processes. This means that (assuming equally weighted processes), after a tick of running, a process becomes ineligible, because it got more time that it should have (in an ideal system the tick would have been shared equally among all processes).

In Cole, however, being able to be unfair for long periods of time is key: jobs with short deadlines need to get absolute priority over jobs that have later deadlines. To achieve this goal we had to modify Linux's EEVDF implementation, to include the notion of request-based eligibility.

We developed a prototype application that follows the scheme of the website: four different types of jobs, with deadlines ranging from 15ms to 6s, and maximum computes from 12ms to 4s. Figure 1 shows the results on a single machine, with and without the change to linux.

Overall the graph shows success: being temporarily unfair to longer processes that had a large amount of slack allowed the shorter processes with less slack to consistently complete on time.

## 4 Related Work

There is a large body of work that focusses on removing the need for reservations by creating elastic reservations that automatically scale up and down with load [10, 12]. These systems generally operate on a much larger timescale (control loops on the order of manys seconds [11]), and still work within the best effort - latency critical binary.

There are systems that focus on increasing utilization via much more fine-grained preemption behavior [6, 8]. These require building a kernel-bypass scheduler.

Other work does operate on the basis of deadlines, but take different appproach to realizing those deadlines. Some try to formally pre-compute compute time to make guarantees [5], while others use the deadline to generate a reservation [4]. Morpheus [7] targets jobs with invocation frequencies of around a day, and infers the deadline from historic data, automatically generating recurring reservations.

## References

[1] AWS. *Amazon Cloudwatch*. URL: https://aws.amazon.com/cloudwatch/ (visited on 02/18/2024).

[2] AWS. *AWS Service Level Agreements (SLAs)*. URL: https://aws.amazon.com/legal/service-level-agreements/?aws-sla-cards.sort-by=item.additionalFields.serviceNameLower&aws-sla-cards.sort-order=asc&awsf.tech-category-filter=*all (visited on 02/18/2024).

[3] Jonathan Corbet. *An EEVDF CPU scheduler for Linux*. URL: https://lwn.net/Articles/925371/.

[4] Carlo Curino, Djellel E. Difallah, Chris Douglas, Subru Krishnan, Raghu Ramakrishnan, and Sriram Rao. "Reservation-based Scheduling: If You're Late Don't Blame Us!" In: *Proceedings of the ACM Symposium on Cloud Computing*. SOCC '14. Seattle, WA, USA: Association for Computing Machinery, 2014, pages 1–14.

[5] Andrew D. Ferguson, Peter Bodik, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. "Jockey: guaranteed job latency in data parallel clusters". In: *Proceedings of the 7th ACM European Conference on Computer Systems*. EuroSys '12. Bern, Switzerland: Association for Computing Machinery, 2012, pages 99–112.

[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.

[7] Sangeetha Abdu Jyothi, Carlo Curino, Ishai Menache, Shravan Matthur Narayanamurthy, Alexey Tumanov, Jonathan Yaniv, Ruslan Mavlyutov, Inigo Goiri, Subru Krishnan, Janardhan Kulkarni, and Sriram Rao. "Morpheus: Towards Automated SLOs for Enterprise Clusters". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pages 117–134.

[8] Kostis Kaffes, Timothy Chong, Jack Tigar Humphries, Adam Belay, David Mazières, and Christos Kozyrakis. "Shinjuku: Preemptive Scheduling for mico-second-scale Tail Latency". In: *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. Boston, MA: USENIX Association, Feb. 2019, pages 345–360.

[9] Will Kelly. *Overprovisioning in AWS? Cost-control tools and strategies can help*. Nov. 22, 2013. URL: https://www.techtarget.com/searchaws/news/2240209793/Overprovisioning-in-AWS-Cost-control-tools-and-strategies-can-help.

[10] Kubernetes. *Autoscaling Workloads*. URL: https://kubernetes.io/docs/concepts/workloads/autoscaling/ (visited on 02/18/2024).

[11] Kubernetes. *Horizontal Pod Autoscaling*. URL: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/ (visited on 02/18/2024).

[12] Themis Melissaris, Kunal Nabar, Rares Radut, Samir Rehmtulla, Arthur Shi, Samartha Chandrashekar, and Ioannis Papapanagiotou. "Elastic cloud services: scaling snowflake's control plane". In: *Proceedings of the 13th Symposium on Cloud Computing*. SoCC '22. San Francisco, California: Association for Computing Machinery, 2022, pages 142–157.

[13] Ion Stoica and Hussein Abdel-Wahab. "Earliest Eligible Virtual Deadline First: A Flexible and Accurate Mechanism for Proportional Share Resource Allocation". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Nov. 1996.