EDGELESS

UNIVERSITY OF CAMBRIDGE

https://arxiv.org/abs/2508.15703

Unlocking extra cluster capacity with enhanced Linux cgroup scheduling

FOSDEM 2026, Kernel Track

Al-Amjad Tawfiq Isstaif, Evangelia Kalyviannaki, **Richard Mortier**

Systems Research Group,
Department of Computer Science & Technology,
University of Cambridge, UK

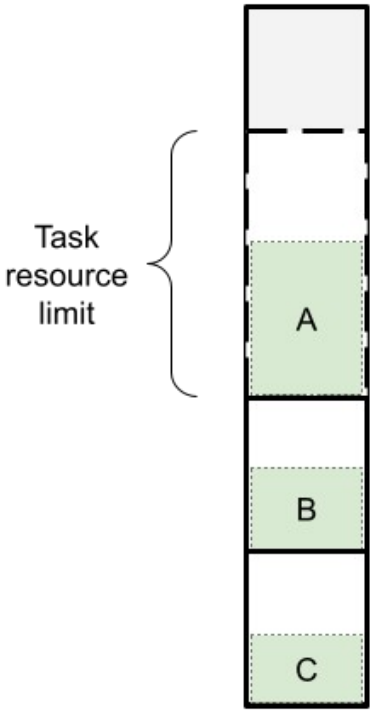
alamjad.isstaif@cl.cam.ac.uk, richard.mortier@cl.cam.ac.uk

UNIVERSITY OF CAMBRIDGE

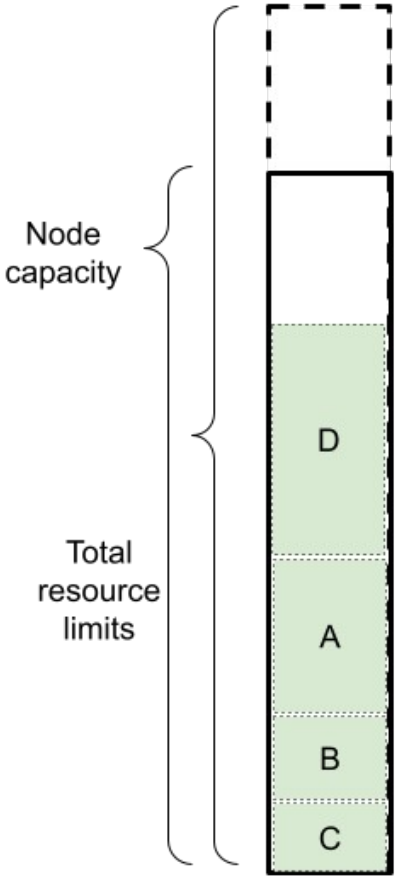
https://arxiv.org/abs/2508.15703

Motivation

- Originally interested in densely packing unikernels onto servers, aiming to improve on Kubernetes and serverless performance
- Measurements of serverless workloads made no sense: CPU utilisation was too high, throughput was too low – **performance was degraded**
- **Why?** With increasing workload density (10s or 100s of containers), **context-switch overhead takes up to 25% CPU time** due to how the scheduler manages cgroups and allocation across cores
- **How to fix it?** We developed an alternative scheduler that mitigates this problem, allowing the same performance on a **28% smaller cluster**



a) Container packing based on CPU resource reservation



b) Container packing based on CPU multiplexing

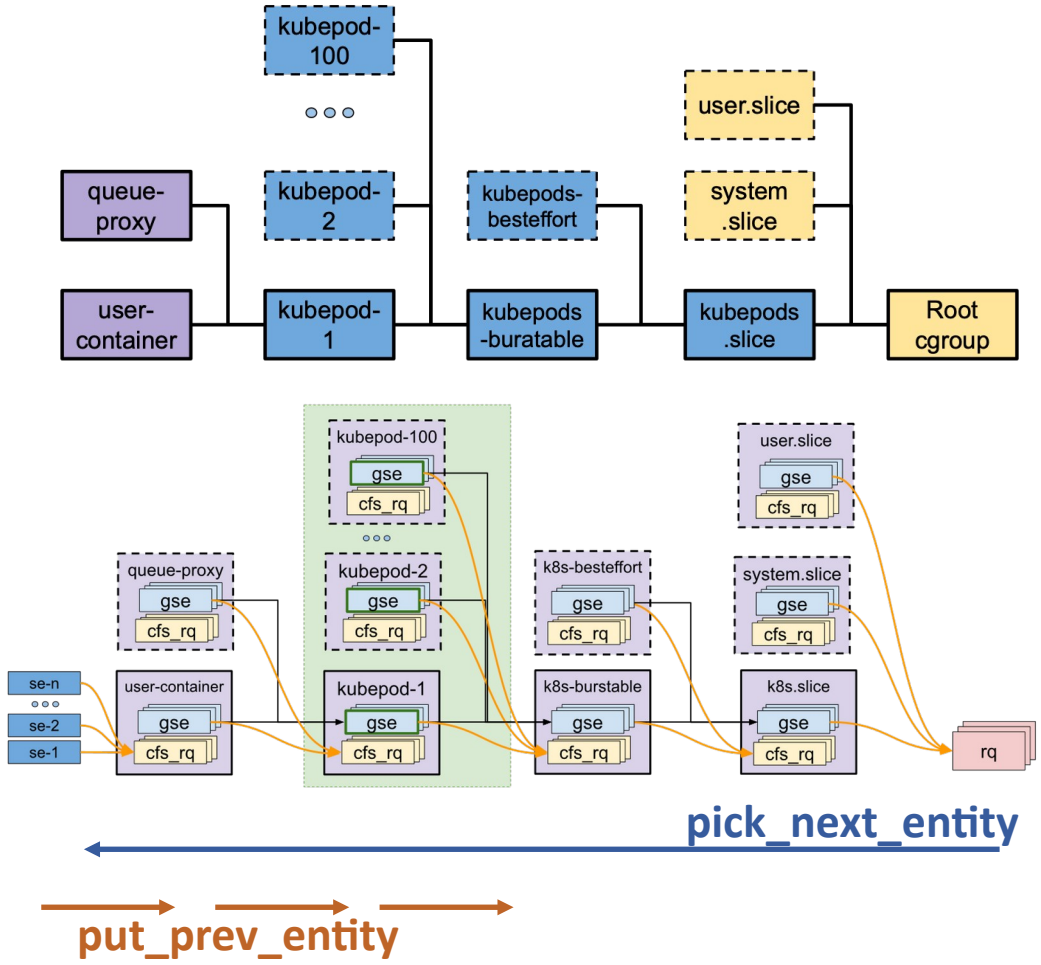
alamjad.isstaif@cl.cam.ac.uk, richard.mortier@cl.cam.ac.uk

UNIVERSITY OF CAMBRIDGE

https://arxiv.org/abs/2508.15703

Group scheduling with cgroups

- For *N* tasks, CFS gives each runnable task a minimum timeslice (~4ms) and grows the **scheduling period** to 4*N* ms
- Tasks are **grouped** by cgroup and scheduled as a whole using **per-cgroup** and **per-core** run queue structures to prevent gaming the scheduler
- Locating the next task is optimised, but **re-inserting preempted tasks increases cost** of pick_next_task_fair by several microseconds
- Higher per-context-switch cost and rate of context-switches **combine multiplicatively**



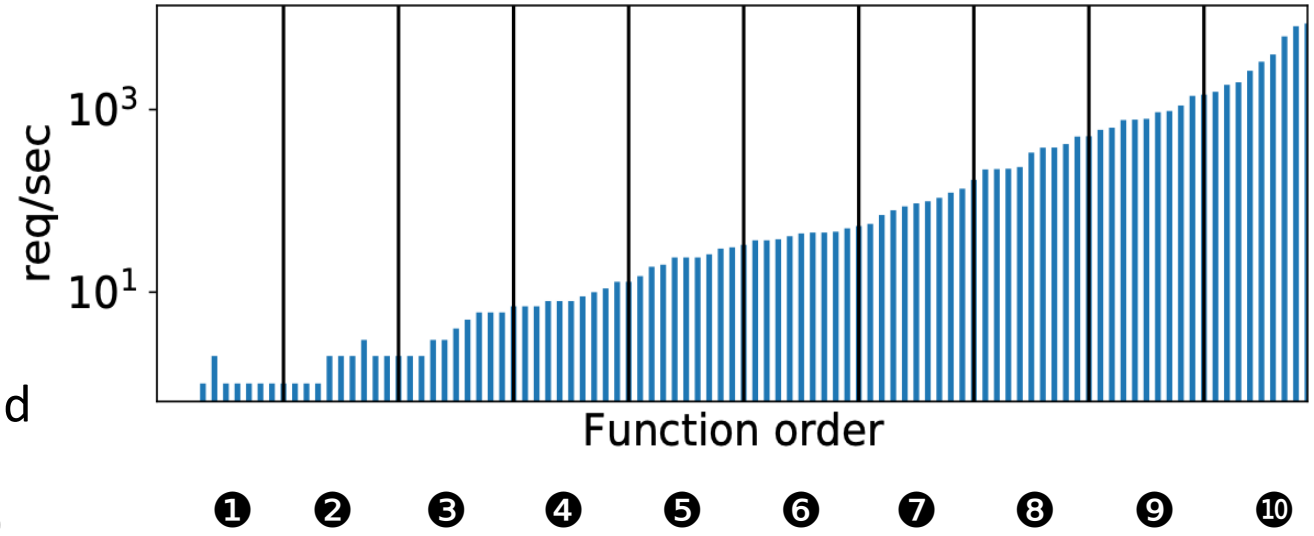
alamjad.isstaif@cl.cam.ac.uk, richard.mortier@cl.cam.ac.uk

UNIVERSITY OF CAMBRIDGE

https://arxiv.org/abs/2508.15703

Microbenchmark for function colocation

- Modify Meta's *resource control framework* to assess the impact of increasing the number of **concurrent cgroups**
- Investigated given a very simple cgroup structure (i.e. `faac.slice/func-{0-x}.service`)
- Compare the standard closed-loop workload (**resctl**) to one representing vertical-first scaling open-loop concurrency (**azure2021**)
- Overhead measured with `ftrace` by instrumenting the total time spent during the core scheduler logic `__schedule()`



alamjad.isstaif@cl.cam.ac.uk, richard.mortier@cl.cam.ac.uk

UNIVERSITY OF CAMBRIDGE

https://arxiv.org/abs/2508.15703

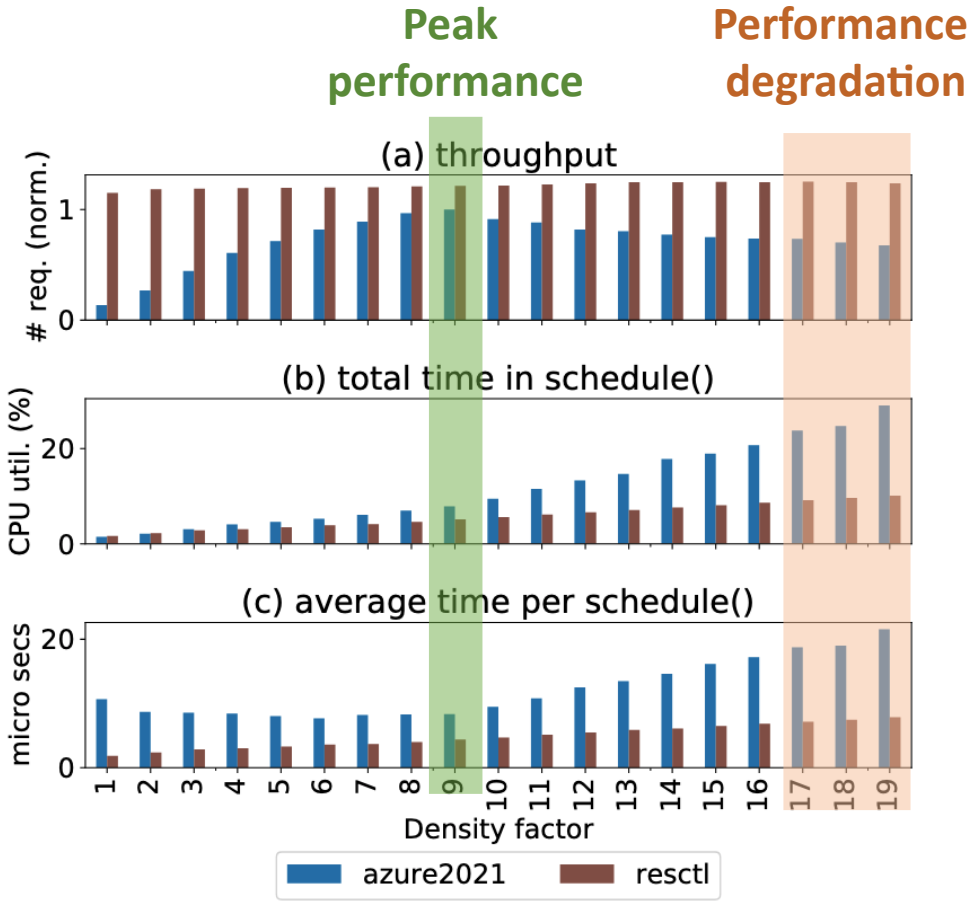
Quantifying CFS scheduling overheads

Examine (a) **throughput** as (b) **time spent context switching** and (c) **time of an individual context switch** increase for the vertical scaling workload, **azure2021**

(a) Increasing colocation beyond node capacity **degrades throughput up to 35%**

(b) Degradation can be attributed to **the growth of scheduling overhead to 5—20% of CPU time at peak load**

(c) Significant factor is the increase in **the average cost of a single context switch to 10—20 microseconds**



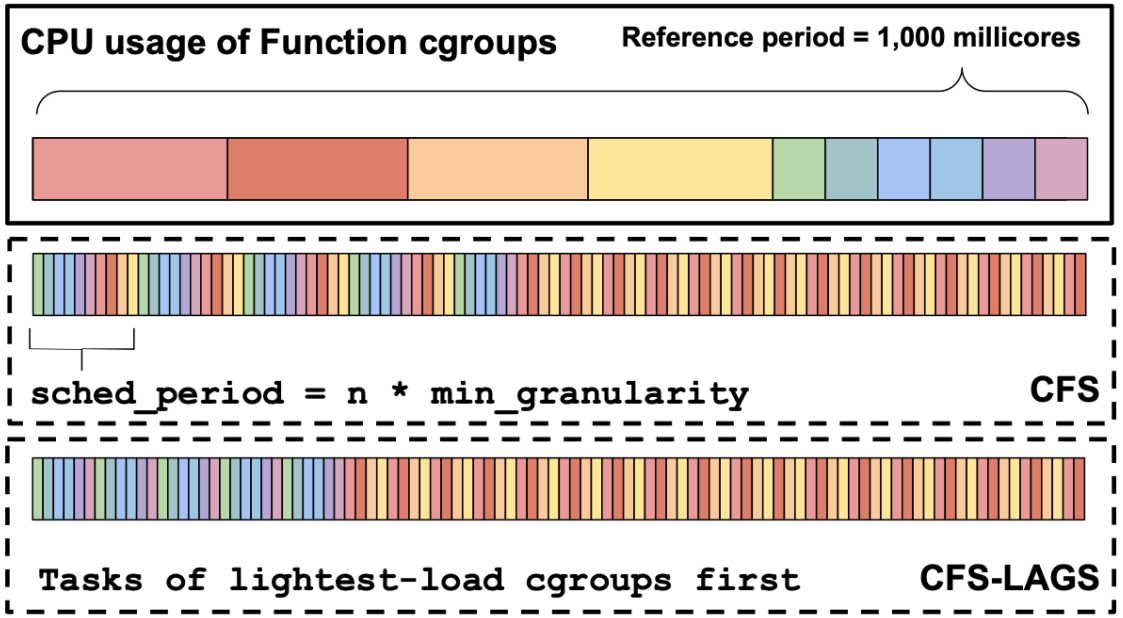
alamjad.isstaif@cl.cam.ac.uk, richard.mortier@cl.cam.ac.uk

UNIVERSITY OF CAMBRIDGE

https://arxiv.org/abs/2508.15703

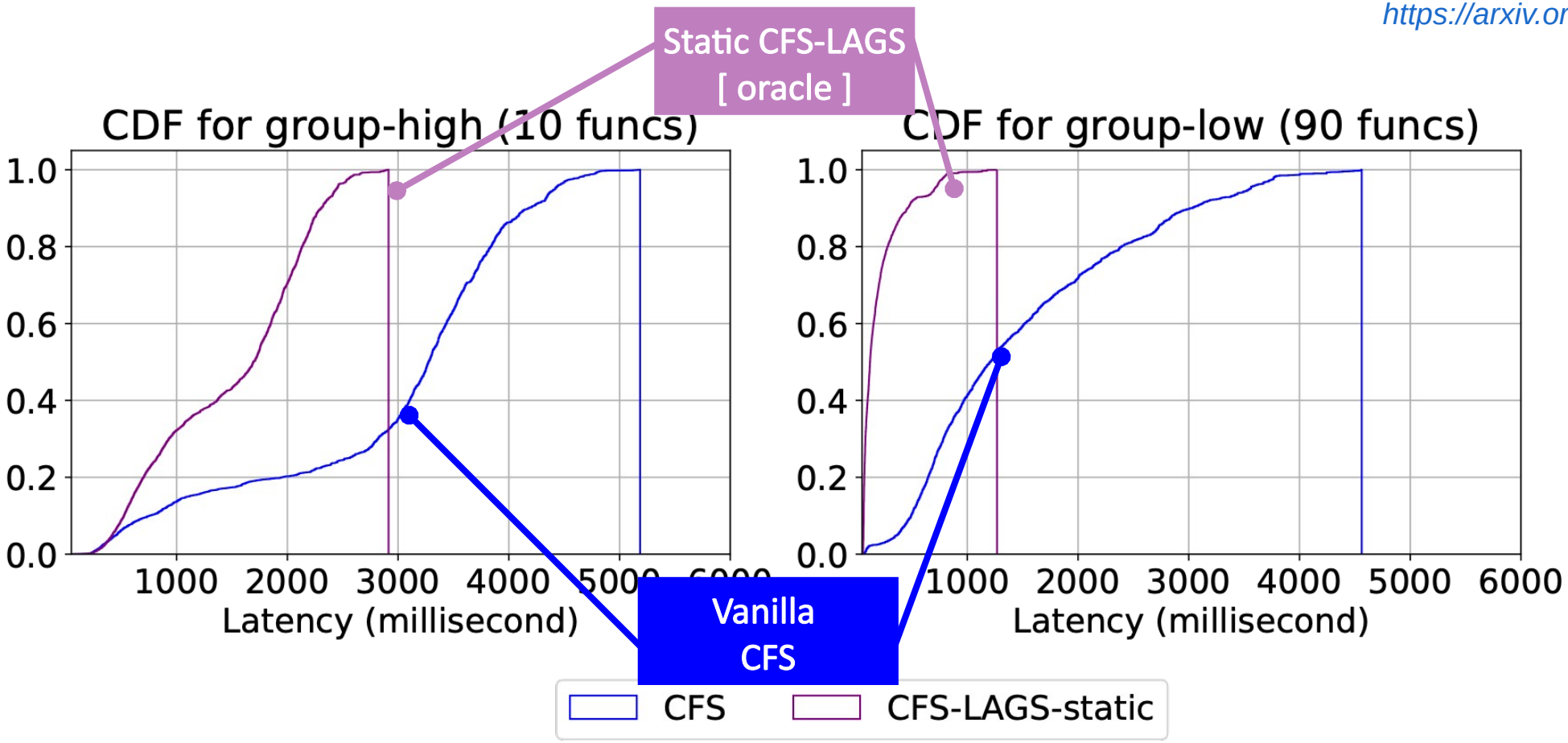
Latency-Aware Group Scheduling

- The key idea behind LAGS is to maximise **task completion rate within cgroups** beyond the scheduler's short scheduling interval
  - Use a Shortest-Remaining Time First approximation to **prioritise the lightest cgroups**
  - This improves latency and reduces overhead as **cgroups can exit the system earlier making runqueues shorter**
- Approximation via **Cgroup Load Credit** metric that tracks recent CPU usage for all threads within a cgroup



UNIVERSITY OF CAMBRIDGE

https://arxiv.org/abs/2508.15703



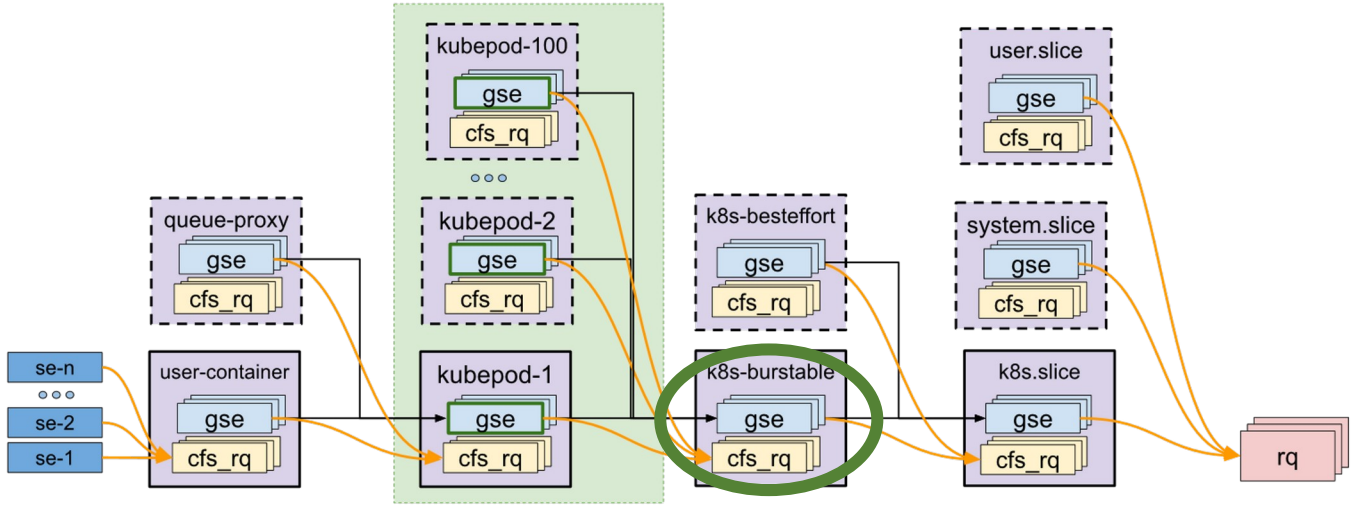
- Allows the tail of lightest functions to complete and get out of the way
- **And** so the (small) number of heaviest functions also make better progress

UNIVERSITY OF CAMBRIDGE

https://arxiv.org/abs/2508.15703

Realising CFS–LAGS

- CFS-LAGS is a **sub-scheduling architecture** applying custom policies to specific cgroups
- The **Load Credit metric** becomes the scheduling priority for serverless function cgroups
- The default CFS policy remains for all other cgroups



- **Load Credit implementation** averages the default *per-entity load tracking* (PELT) over a ~4s window *vs* default 32ms in PELT, preferring youngest tasks first
- **Function cgroup identification** identifies target group tasks (that is, cgroups) via a user space `cpu.latency_awareness` cgroup property

UNIVERSITY OF CAMBRIDGE

https://arxiv.org/abs/2508.15703

CFS-LAGS mitigates overhead

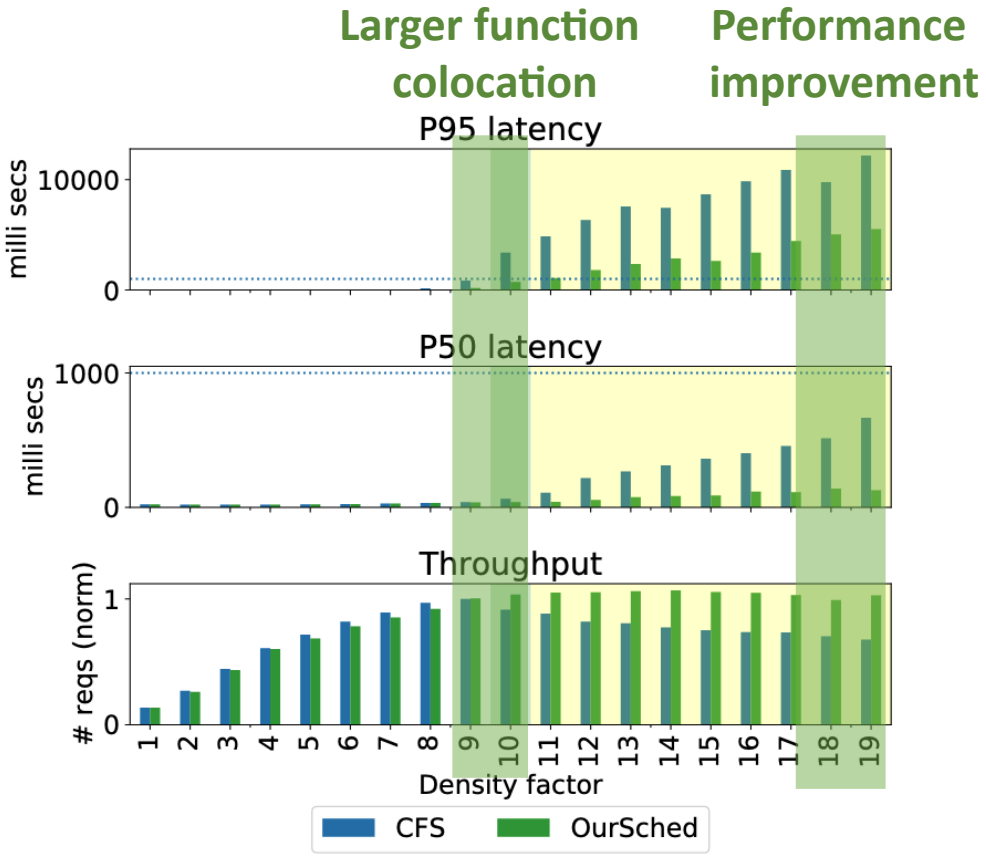


Figure 9. Performance given increasing colocated functions under realistic arrivals (azure2021).

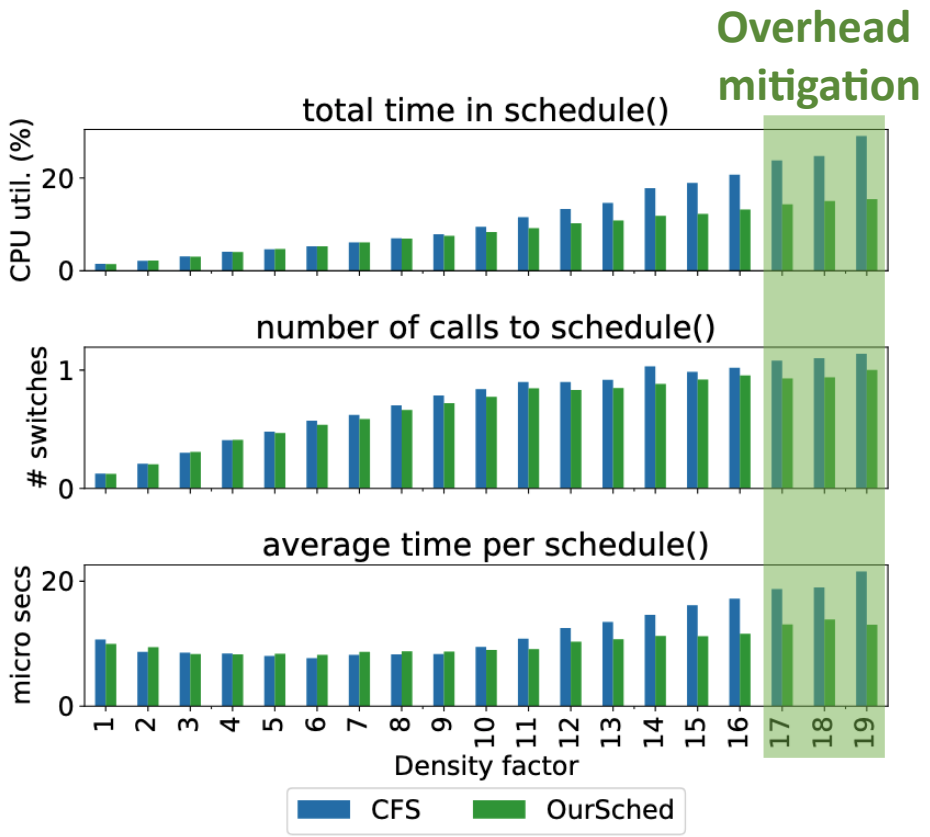


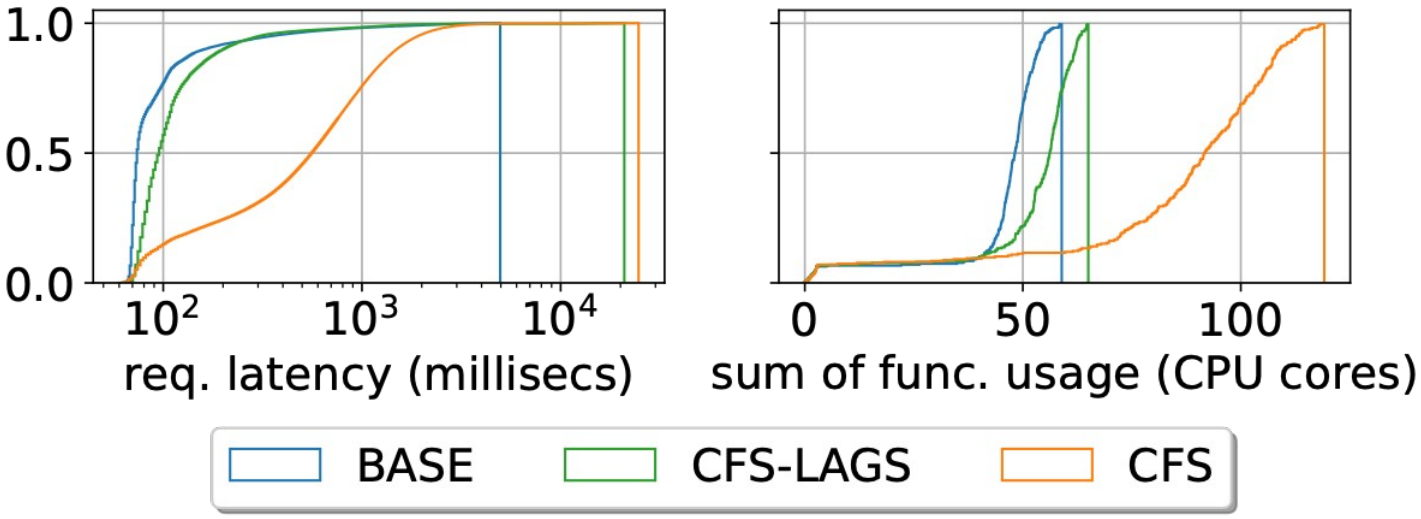
Figure 10. Scheduling overhead given increasing colocated functions under realistic arrivals (azure2021).

UNIVERSITY OF CAMBRIDGE

https://arxiv.org/abs/2508.15703

CFS-LAGS for tighter container packing

- **BASE** baseline of 14 nodes based on requested peak CPU load. No CPU resource sharing, low CPU utilisation **(~30%)** given ~800 containers from **azure2021** (**~60 cores**)
- **CFS** used to statistically multiplex containers onto 12 nodes. Max density achievable given requirement to **keep CPU utilisation below ~45%**
- **CFS-LAGS** achieves the same performance with 1 0 nodes and **higher CPU utilisation (~55%)**



CFS-LAGS cluster (10 nodes)

Cluster-wide latency
Given the same density under CFS, the scheduling overheads translates into ~6x increase in latency!

Cluster-wide CPU utilisation
CFS creates a significant gap between **effective** and **perceived** utilisation:
+100%, ~120 cores for CFS
vs +10%, ~65 cores for CFS-LAGS

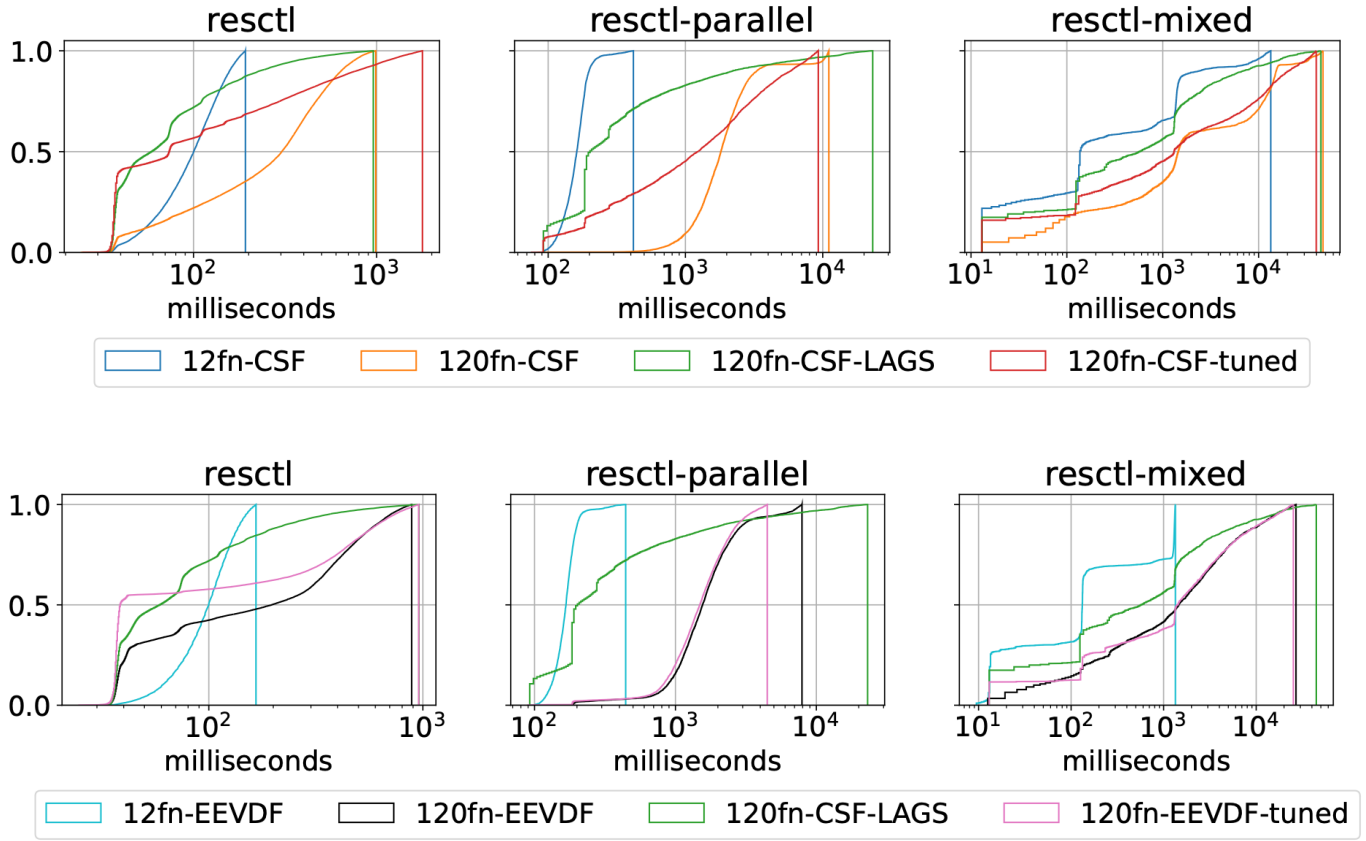
alamjad.isstaif@cl.cam.ac.uk, richard.mortier@cl.cam.ac.uk

UNIVERSITY OF CAMBRIDGE

https://arxiv.org/abs/2508.15703

Comparing CFS-LAGS to alternatives

- Simply increasing timeslice to 100ms increases cgroup-level task completion and improves latency but not for multi-threaded workloads (`resctl-parallel` and `resctl-mixed`)
- EEVDF is difficult to tune under high load (`120fn-EEVDF` and `120fun-EEVDF-tuned`) due to the virtual deadline scheduling used to enforce thread-level fairness



UNIVERSITY OF CAMBRIDGE

https://arxiv.org/abs/2508.15703

Conclusions

- Cgroups are crucial to manage workloads in production but
  - ...as load increases, reinsertion into the nested red-black tree structure increases context-switch times
  - ...which in turn increases the context-switch rate as CFS tries to achieve fairness
  - ...and these two effects combine multiplicatively
- **This increases latency variation dramatically while decreasing effective capacity leading to poor bin-packing decisions by the K8s scheduler**
  - Appears to be mitigated currently by expensive, wasteful over-provisioning
  - Even worse for serverless where desire to avoid cold-starts leads to artificially increasing multiplexing by keeping idle tasks around
- CFS-LAGS unlocks 10—20% capacity by builds on CFS/EEVDF while using cgroups as user-space control interface to encourage short tasks to complete early, keeping runqueues shorter, reducing context switch overhead, and improving binpacking

https://edgeless-project.eu/
Funded in part by EU Horizon Europe under Grant Agreement No 101092950

UNIVERSITY OF CAMBRIDGE

https://arxiv.org/abs/2508.15703

Status & Questions

- Currently
  - Porting CFS-LAGS to latest 6.x kernels
  - Extending to manage QoS for interactive, cgroup-managed, multi-threaded workloads.
  - Patch and benchmarks available via https://github.com/isstaif/CFS-LLF_main
  - Paper available via https://arxiv.org/pdf/2508.15703
- Several recent attempts to customise group scheduling
  - E.g., *sched_ext* cgroup sub-scheduling in v6.18 and ScyllaDB's user-space scheduler
  - Exploring how far we can get using *sched_ext* (on ARM…)

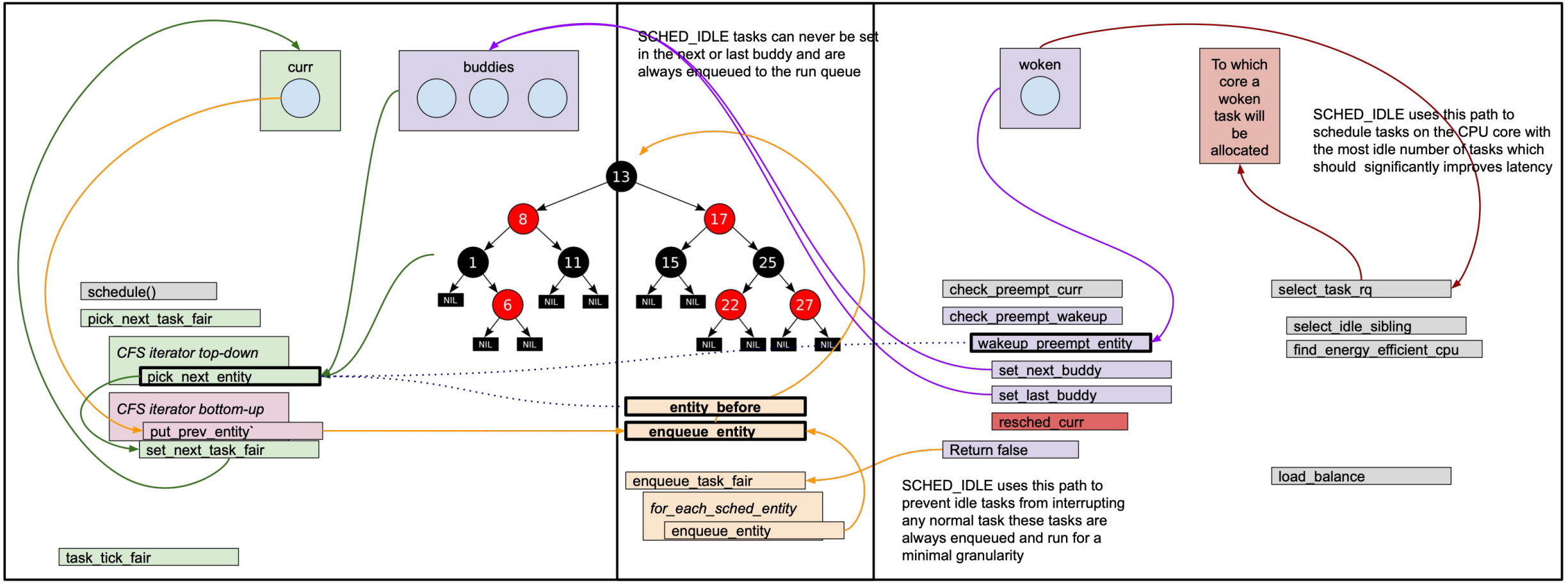
alamjad.isstaif@cl.cam.ac.uk, richard.mortier@cl.cam.ac.uk

Backup slides

UNIVERSITY OF CAMBRIDGE

https://arxiv.org/abs/2508.15703

CFS code paths



CFS group scheduling data structures

UNIVERSITY OF CAMBRIDGE

https://arxiv.org/abs/2508.15703