Towards Latency-Aware Linux Scheduling for Serverless Workloads

Al-Amjad Tawfiq Isstaif, Prof Richard Mortier
Systems Research Group,
Department of Computer Science & Technology,
University of Cambridge
Motivation

• Originally interested in local-first self-scaling unikernels

• Running on Kubernetes requires rapid adjustment of Linux CPU shares in response to load spikes

• However, we observed that
  • At low utilisation, a small CPU share doesn't matter because of work-conservation [OK! ]
  • At high utilisation, the system suffers performance degradation and adjusting CPU shares did not help [BAD! ]

• Problem? How the Linux Completely Fair Scheduler (CFS) treats cgroups
Completely Fair Scheduling with cgroups

• CFS ensures each runnable task receives a minimum timeslice (~4ms), growing the scheduling period as needed (i.e., to $4N$ ms for $N$ tasks)

• Consider high-density serverless workloads
  • $10^2$ – $10^3$ functions per node
  • Concurrent requests per function

• With group scheduling, each cgroup is scheduled as a single whole to prevent a cgroup gaming the system by creating many tasks

• Results in many context switches to achieve CFS fairness across task hierarchy
Experimental workload

- Azure Function Invocation trace
  - 119 functions over two weeks
  - K8s limit of 110 pods/nodes
  - Use top 100 functions, leaving 10 pods for admin functions

- Then, per 5 minute bucket,
  - Sort into 10 demand bands ordered by request arrival rate
    - *group-high* mixes highest intensity band for each bucket
    - *group-low* mixes lowest 9 intensity bands for each bucket
CFS—Least Loaded First (CFS–LLF)

• Serverless workloads are skewed:
  • most compute happens in a few functions, while
  • most functions are short-lived and mostly idle

• Reduce context switch overheads by allowing some of the tail functions to complete and get out of the way

• Adjust the CFS dynamic priority based the existing per-entity load tracking (PELT) mechanism rather than on minimum vruntime
  • Each task has a dynamic load credit estimated over a ~4 seconds (i.e., youngest tasks first)
  • Group tasks (equivalently, cgroups) into function sandboxes via addition of cpu.func_sandbox property
CFS–Least Loaded First (CFS–LLF)

- Serverless workloads are skewed:
  - most compute happens in a few functions, while
  - most functions are short-lived and mostly idle

- Reduce context switch overheads by allowing some of the tail functions to complete and get out of the way

- Adjust the CFS dynamic priority based the existing *per-entity load tracking* (PELT) mechanism rather than on minimum *vruntime*
  - Each task has a dynamic load credit estimated over a ~4 seconds (i.e., youngest tasks first)
  - Group tasks (equivalently, cgroups) into *function sandboxes* via addition of *cpu.func_sandbox* property
Result? CFS–LLF mitigates impact of overload

- Allows the tail of least-loaded functions to complete and get out of the way
- Allows the (relatively) small number of most-loaded functions to also make useful progress
Conclusions

• CFS can be “good enough” under low CPU utilisation
• CFS-LLF mitigates performance degradation as load increases in high density serverless setups
  • Relaxing fairness in favour of short-lived functions (useful, fair, & safe)
  • Reduces stress on CFS run queues
• CFS-LLF is compatible with K8s and portable to other frameworks
  • Requires no coordination with control planes, requires no workload training
• Next
  • Extending evaluation to diverse serverless benchmarks
  • Hotspot functions can be rapidly identified using the kernel load metric: can this allow them to be rapidly auto scaled to other cluster nodes?
Backup slides
Figure 10. Arrival rates of the top segment found in the 10 demand bands from Figure 3
Contestion under serverless workloads

As utilisation increases, fewer requests meet their latency target
Attainment of latency targets

(a) Group high

(b) Group low
Overload management

- **CFS-LLF**, a kernel scheduler that protects the performance of the long tail of least loaded functions with no prior training or knowledge of workload, and no coordination with Kubernetes control plane.

- **Contention-aware autoscaling**, a Linux host agent that detects when a cluster node is overloaded and scales the resources of hotspot functions without escalating the issue to other cluster nodes.
Evaluation

- CDFs of achieved latency per function under highest-contention scenario

![Diagram showing CDFs for latency distribution under different scenarios with legends for Static LLF (sched_rr) [oracle], Static LLF (cpu.shares), Dynamic LLF (CFS–LLF), and Vanilla CFS.]