A Linux CPU scheduler for interactive multi-threaded workloads

Al Amjad Tawfiq Isstaif

Research Fellow at Smart Sensing Lab, Nottingham Trent University, Research Visitor at the Systems Research Group, University of Cambridge

Scheduling interactive workloads alongside other workloads has long been studied in the literature, with techniques such as run-to-completion execution and prompt preemption. Despite this body of work, Linux only provides basic support for colocating interactive workloads with other workloads, which limits resource utilisation and reduces system flexibility. Examples of these techniques include soft real time scheduler (sched_rt) as well as CPU pinning (cpusets).

This work explores the design of an interactive scheduling policy for Linux, **Latency-Aware Group Scheduler (LAGS)**, which provides improved support for interactive multithreaded workloads in the mainline kernel. It builds on the standard Linux scheduler (**CFS/EEVDF**), including **group scheduling**, which is necessary to prevent starvation and support workload co-location. This proposal aims to make these benefits accessible to administrators and users by utilising the **cgroup interface**, which is already widely used by resource management frameworks such as Kubernetes. This requires a complex implementation within the highly concurrent and hierarchical data structures of **CFS/EEVDF**, as well as overcoming two key challenges:

1. **Fairness among competing interactive workloads**, ensuring that multiple latency-sensitive tasks can coexist without starvation.
2. **Multi-core scheduling support**, enabling workloads to utilise the full capacity of modern servers by exploiting idle CPU cores while prioritising latency-sensitive tasks.

Overall, LAGS aims to provide a flexible mechanism for managing interactive workloads without the drawbacks of static CPU reservations or soft real-time scheduling policies, while remaining compatible with existing CFS/EEVDF and cgroup-based resource management. The talk presents early results and discusses the technical challenges faced by modern scheduler implementations, such as the Linux scheduler, when handling modern containerized workloads.