How Google Uses eBPF to Tailor CPU Scheduling and Optimize Workload Performance

OVERVIEW

Google faced significant infrastructure challenges due to the performance limitations of general-purpose kernel CPU schedulers, which could not optimize for the infrastructure's highly specific application behaviors and hardware combinations. To resolve this, Google implemented application-tailored CPU scheduling policies using pluggable eBPF schedulers (originally ghOSt, and now via the sched_ext framework). This eBPF-driven architecture delivered a 5% queries per second (QPS) improvement in some of its core Remote Procedure Call (RPC) serving stacks, alongside double-digit throughput gains for memory-intensive workloads. By shifting scheduling policy execution to eBPF, Google successfully decoupled policy from core kernel mechanisms, enabling rapid iteration and execution optimization without modifying the base kernel or rebooting production hosts.

CHALLENGE

Google's global infrastructure operates at a scale where minor scheduling inefficiencies translate into major operational limitations. The organization faced critical constraints when managing its fleet workloads using standard, general-purpose kernel architectures:

- **Limitations of General-Purpose CPU Schedulers**: The built-in kernel scheduler is designed as a broad tool to handle most workloads adequately. However, it is unable to adapt dynamically to optimize for highly specialized application needs.
- **Inadequacy of Userspace Scheduling Daemons**: Traditional userspace daemons cannot easily enact scheduling priority adjustments at the microsecond level, and are severely limited in the types and granularity of hints they can provide the scheduler based on the available tuning knobs.
- **Kernel Modification Constraints**: Modifying the base kernel directly to accommodate specialized workloads was operationally restrictive, creating a distinct need for a safe, high-performance execution environment inside the kernel.

SOLUTION

To overcome these constraints, Google used eBPF to implement custom, workload-specialized scheduling policies directly within the kernel.

Custom CPU Scheduling
Google developed and deployed application-tailored CPU scheduling policies by running pluggable eBPF schedulers. This custom architecture outperforms the built-in kernel scheduler by employing three coordinated mechanisms:

- **Dynamic Priority Adjustments:** Google used eBPF maps to share task-state directly between userspace and the kernel. This implementation lets the system make rapid, dynamic thread priority and other heuristic adjustments within core RPC serving stacks. This is very low overhead and real-time, as new RPC requests come in and get mapped to worker threads.
- **Hardware Cache Domain Optimization**: The custom scheduling policy aggressively packs threads sharing an address space into specific hardware cache domains, such as AMD Core Complex (CCX) boundaries, to maximize cache locality. These assignments are flexible and can be configured with various parameters to adjust the tradeoff between memory locality and scheduling latency.
- **Coarser Runqueue Granularity**: Shifting runqueue management from a per-CPU basis to groups of CPUs significantly improved overall work conservation and scheduling fairness across multi-core systems.

RESULTS

By shifting scheduling policy to eBPF, Google achieved clear performance and operational enhancements:

- **~5% QPS Improvement**: Achieved in core RPC serving stacks through application-tailored CPU scheduling policies.
- **Double-Digit Throughput Gains**: Realized across memory-intensive workloads.
- **Safe Production Subsystem Iteration**: Provided developers with the ability to safely deploy, test, and iterate on core infrastructure execution policies on active production workloads without risking kernel panics or reboots.

FUTURE PLANS

Google is actively extending its eBPF infrastructure to support a wider array of compute workloads and core kernel subsystems:

- **Workload Expansion**: Google is migrating new and diverse workloads onto custom eBPF schedulers to deploy workload-specific execution policies across a broader footprint of compute clusters.
- **In-Kernel Memory Management (MM) Policies**: To replicate the architectural success of separating policy from mechanism within the CPU scheduler, Google is actively exploring eBPF-driven memory management policies. This includes an upstream patch series for an in-kernel OOM framework that lets eBPF proactively trigger OOM actions and execute alternative victim selection logic.
- **Page Cache Eviction Control**: Google is evaluating the upstream cache_ext proposal to dynamically manage page cache eviction policies per application, letting systems squeeze optimal performance out of existing hardware.

CONCLUSION

WHY eBPF?

Google selected eBPF because it uniquely satisfies the performance and safety requirements demanded by global-scale infrastructure:

- **Synchronous, In-Kernel Execution**: eBPF provides the microsecond-level execution speed necessary to adjust thread priorities in real time and maintain system responsiveness.
- **Decoupling Policy from Mechanism**: eBPF fundamentally separates policy execution from core kernel design. Engineers can load, attach, and unload scheduling behaviors dynamically in real time, letting teams iterate rapidly without patching base kernels or rebooting production hosts.
- **Safe Kernel Extensibility**: The eBPF verifier provides mathematical guarantees that developer-submitted scheduling policies cannot introduce memory safety errors, infinite loops, or system crashes, making wide-scale infrastructure innovation safe.