HPS

https://github.com/salaheldinsoliman

Salaheldin Sameh

Kernel Compilation and Configuration

Optimizing Linux for HPC Workloads

Outline

Introduction

- HPC applications demand predictable, low-latency execution.
- Linux kernel configurations (tickless kernel, scheduler choice, preemption model) directly affect performance.
- Goal: Identify optimal settings to minimize OS-induced noise and maximize throughput.

Overview & Project Structure

A single Github repository, a directory for each experiment:
https://github.com/salaheldinsoliman/kernel-experiments

1. Tickless Kernel Impact
2. Scheduler Benchmarking
3. Preemption Model Effects

Tickless Kernel: Overview

- **A tick** is a periodic timer interrupt generated by the system timer (typically at 100–1000 Hz, often 250 Hz by default). That means the CPU gets interrupted 250 times per second per core, just for timekeeping and system maintenance.
- **Tickless Kernel** Aims to reduce OS-induced noise and interruptions for deterministic HPC workloads.
- Applies CPU isolation and full tickless mode (`nohz_full`) to free cores from scheduler ticks.
- Detailed setup: https://github.com/salaheldinsoliman/kernel-experiments/tree/main/tickless-kernel

Tickless Kernel: Literature Expectations

- Tickless mode reduces timer interrupt overhead, lowering latency and jitter (Tsafrir et al., Linux RT documentation).
- Expected to improve mean execution time and stabilize runtimes for compute-bound tasks.
- Prior studies report up to 10–15% runtime improvement in real-time and HPC scenarios.

Tickless Kernel: Experiment Results

Setup:

- VirtualBox VM with 5 vCPUs running Linux Mint.
- OpenMP matrix multiplication (SIZE=1024) pinned to isolated cores 1 and 2.
- Kernel boot flags: `isolcpus=1,2` and `nohz_full=1,2`; compared against default.

Results (20 runs):

- **No tickless:** Mean = 0.8033s, Std Dev = 0.0235s
- **Tickless:** Mean = 1.8266s, Std Dev = 0.0673s

Interpretation: Results contradict literature; slowdown and jitter suggest misconfiguration, virtualization effects or wrong setup.

Tickless Kernel: Discussion & Debugging

- **Wrong Setup?**: Linux Mint(Desktop) comes with many background services—GUI compositors, power-management daemons, indexing services—that can sneak onto “isolated” cores and generate noise.
- **Missing Config?**: Should I set IRQ Affinity to non-isolated cores?
- **Virtualization Overhead?**: Does a run on bare metal differ from a VM?

Scheduler Benchmark: Overview

- Evaluate SCHED_OTHER, SCHED_FIFO, SCHED_RR policies.
- Workloads: compute-bound, memory-bound, I/O-bound, mixed.
- Goal: Identify policy with lowest runtime per workload.
- Detailed Setup: https://github.com/salaheldinsoliman/kernel-experiments/tree/main/different-schedulers

Scheduler Benchmark: Literature Expectations

- Real-time policies (FIFO, RR) expected to boost compute-bound determinism.
- CFS (OTHER) balances fairness; may unpredictably preempt HPC tasks.
- Prior work reports 5–10% speedup for compute-bound under FIFO vs CFS.

Scheduler Benchmark: Experiment Results

Compute-bound

- OTHER: 6.73 s
- FIFO: 7.03 s
- RR: 7.05 s

I/O-bound

- RR: 1.38 s
- FIFO: 1.45 s
- OTHER: 1.51 s

Memory-bound

- FIFO: 0.55 s
- RR: 0.55 s
- OTHER: 0.56 s

Mixed

- All: 0.16 s

Observation: I/O bound and memory bound benefited from RR/FIFO, compute bound did not. Compute bound did not benefit from FIFO contrary to literature

Scheduler Benchmark: Discussion

- **Results may differ on a dedicated system**: Desktop OS processes interfered, favoring CFS.
- **Policy selection is Workload-dependent**:
  Compute-bound $\rightarrow$ real-time policies should excel, if the environment is noise-free.
  Memory-bound $\rightarrow$ FIFO/RR give a slight edge by removing timeslice overhead.
  I/O-bound $\rightarrow$ RR often best, due to predictable timeslices allowing rapid wake-up.
- **Next steps**: Re-run on a dedicated system to reduce noise, compare the results

Preemption Model: Overview

- Preemption is the operating system's ability to interrupt a running kernel task—pausing its execution at safe points—so that a more critical or higher-priority task can take over the CPU.
- Compare kernel preemption models: `PREEMPT_NONE`, `PREEMPT_VOLUNTARY`, `PREEMPT`.
- Assess runtime performance of OpenMP workload of different types (interrupted and stable).
- Goal: Identify configuration with lowest latency and highest stability for HPC.

Preemption Model: Literature Expectations

- `PREEMPT_NONE` minimizes context-switch overhead, ideal for compute-bound tasks.
- `PREEMPT_VOLUNTARY` adds preemption points; may reduce latency but introduce spikes.
- `PREEMPT` enables full kernel preemption, improving responsiveness at some overhead.

Preemption Model: Experiment Results

PREEMPT_VOLUNTARY

- Stable workload: 15.015882 s
- Interrupt workload: 5.090901 s

PREEMPT_NONE

- Stable workload: 15.791380 s
- Interrupt workload: 5.396742 s

PREEMPT

- Stable workload: 15.298632 s
- Interrupt workload: 5.301512 s

Observation: PREEMPT_VOLUNTARY yielded the fastest stable workload time, while PREEMPT_NONE had the slowest. Under interrupt load, PREEMPT_VOLUNTARY again performed best, followed by PREEMPT and then PREEMPT_NONE. PREEMPT_NONE offered neither performance nor responsiveness advantages in this environment.

Preemption Model: Discussion

- **Workload type determines ideal model** The performance impact of preemption depends entirely on whether the workload is compute-bound, I/O-bound, or latency-sensitive. There's no "best" model universally.
- **Real-world benchmarks matter more than theory**: Since performance depends on load type and its environment, benchmarks are critical to determine the best preemption model.

Summary

- Performed three experiments (Tickless kernel, Scheduler choice, Preemption mode), none of which yielded the expected results
- Spent quite a bit of time to debug the results, using common tools: top, htop, free, ps, vmstat, and perf
- Dr.Giorgi mentioned there might be something wrong with the setup(distro choice), how I measure time or a virtualization effect.

Next Steps

- Re-run experiments on a server distro or bare metal
- Examine the Lustre distributed FS (detailed in next slide)
- Prepare detailed report with tables and charts.

Lustre FS

- **Problem:** Distributed filesystems like Lustre handle all metadata operations using simple FIFO queues, without considering workflow context.
- AI/ML workloads generate burst-heavy, metadata-intensive patterns (e.g. millions of small file opens), which are handled identically to generic HPC workloads.
- This leads to performance degradation for latency-sensitive AI/ML jobs, especially under filesystem contention.
- **Task:** Examine how to append Lustre with mechanisms to prioritize more critical I/O requests

References I

- Dan Tsafrir, Yonatan Etsion, Dror G. Feitelson. *System Noise, OS Clock Ticks, and Fine-Grained Parallel Applications*. In: Proceedings of the 21st Annual International Conference on Supercomputing (ICS '07), ACM, 2007, pp. 303–312. doi:10.1145/1274971.1275013
- The Linux Foundation. *Linux Kernel Documentation: Real-Time System Design*. kernel.org, 2023. https://www.kernel.org/doc/html/latest/real-time/index.html
- Vijay Tam, Saeid Abtahi, Ibrahim Kharbutli, Dhabaleswar K. Panda. *The Impact of OS Schedulers on Multithreaded Applications*. International Journal of Parallel Programming, vol. 43, no. 1, 2015, pp. 163–188. doi:10.1007/s10766-014-0321-2
- Thomas Gleixner, Mickaël Desnoyers. *Preemption Model in the Linux Kernel*. Linux Plumbers Conference, 2013. https://events.linuxfoundation.org/sites/events/files/slides/Preemption_Model_in_Linux_Kernel.pdf