Week6: TMO: Transparent Memory Offloading in Data Centers

- When using SSDs as a memory tier, how does TMO prevent them from being worn out when used for offloading memory?
- When considering offloading, what are the aspects that TMO has considered and what aspects TMO can do better? Considering the following aspects: the time to offload, the amount of data to offload, what data to offload.
- How does TMO perform memory offloading to compressed memory versus SSD, and under what conditions does it choose between the two?
- What are the disadvantages of implementing Senpai in userspace? Can you think of any scenarios in which a kernel space implementation of Senpai would be more performant?

Week5: Practical, transparent operating system support for superpages

- It is known that x86 has a limited number of superpage TLB entries in the per-core TLB structure. Suppose superpages were used pervasively throughout an application, what would be the downside of such an approach, and for what kind of applications? Think in terms of reduced entries available in the TLB if only superpages are used, and how the working set of an application interacts with TLB usage.

- Consider the scenario in which a superpage is read from disk and only a small fraction of it is modified. Propose a system that would be able to only write the necessary base pages to disk without having to demote the superpage.

- Consider a case where the system dram is not enough and you have to swap. What is the performance impact of superpages on swap? And the impact on space efficiency? How would you mitigate these issues caused by superpages?

Week4: ghOSt: Fast & Flexible User-Space Delegation of Linux Scheduling

- List some disadvantages of ghOSt's delegated scheduling model (i.e., offloading decisions to the user space ghOSt agent). Explain how these limitations can be addressed by implementing all of the scheduling logic within BPF programs instead.
- How does ghOSt work help minimize system downtime during scheduler updates and rollbacks? How does ghOSt ensure fault tolerance when an agent fails or a policy malfucntions?
- Why is eBPF not sufficient to implement a flexible scheduling policy here? How does ghOSt overcome the limitations while still leveraging the eBPF? In redesigning or modifying eBPF, what changes will you perform to offload the whole ghOSt logic into the kernel and what can be the tradeoffs?
- How is isolation achieved between different applications using ghOSt to write scheduling policies?

Week 3: The Linux Scheduler: A Decade of Wasted Cores

- Issue: Choosing a process that is asleep for a long time
  Example: A is a CPU-bound process, B is a IO-bound process and is asleep for 10 seconds. When B wakes up, its vruntime is 10 seconds behind A's vruntime
  B may end up monopolizing the CPU for the next 10 seconds while it catches up, effectively starving A. Q. Will this situation occur?

- Comment on whether CFS is considered a work-conserving scheduler. Explain an example scenario of thread scheduling where the behavior of a work-conserving and non-work-conserving scheduler will differ.

- **Suppose there is a scheduler which enabled full preemption like Linux and another scheduler with partial preemption only (user threads cannot preempt each other). Is the preemptable kernel always better than the non-preemptable one? If not, explain a case where it is actually not (The workload consist of a load injector and workers that handle requests. Load injector got preempted)**

- Explain one of the bugs described in the paper and why it affects the overall performance of the Linux scheduler. Can you think of any potential issues with the proposed fix for the bug? If not, explain why the solution is optimal.

Week 2: Light-Weight Contexts: An OS Abstraction for Safety and Performance

- What is the difference between lwC and process? Why is creating lwC much cheaper than forking a process? How would you compare the cost of creating lwC against creating other light weight asynchronous execution units, e.g., C++ coroutines?
- Why does switching between lwCs faster than switching between kernel threads?
- How do snapshots and rollbacks work for lwC? Is it possible to reuse the original lwC instead of recreating the snapshot every time to reduce the overhead?