**[System] A parallel framework for Linux cmd-line tools:** Many Linux cmd-line tools (e.g., awk, sed, grep) are efficient and powerful in text/data processing. Using them to process large amounts of files can be slow, however, simply because they are not inherently parallel. A parallel workflow framework is thus needed. You will design and implement such a framework, which can concatenate multiple cmd-line tools into a directed acyclic graph (DAG); in the generated DAG, each graph node is a fine-grained task based on a configured cmd-line tool; this DAG is used to parallelize processing on multiple CPU cores of your computer. How would you build a first-of-its-kind framework that parallelizes your favorite Linux tools?

***[Team: Doan, Martinez, Rajput] Friday***

**[System] Implementing priority scheduling on** [**ghOSt**](https://dl.acm.org/doi/10.1145/3477132.3483542)**:** Linux CFS is inefficient in handling short-job-dominant workloads. Emerging FaaS (Function-as-a-Service) workloads are short-job-dominant, and therefore, requiring a new suite of CPU scheduling policies that approximates the offline SRTF (shortest remaining time first) scheduling policy. How can you build a userspace priority scheduler that dynamically adjusts the time slices of short-lived CPU tasks to optimize the turnaround time for relatively short tasks? Note this project would require you to build a custom Linux kernel that supports the ghOSt kernel module.

**[Measurement] Measuring ghOSt overhead and killing** [**ghOSt**](https://dl.acm.org/doi/10.1145/3477132.3483542)**:** The ghOSt scheduling framework provides APIs for userspace applications to directly control CPU scheduling. ghOSt allows user space applications to communicate with the kernel scheduler using message queues and transactions. Message queues and transactions incur communication overhead between the user space and kernel space. The challenge is to study ghOSt at scale; how well does ghOSt handle many on-the-fly communications transferred between the userspace application and Linux kernel? Can you create adversarial workloads that hit the bottleneck and potentially break ghOSt? Note this project would require you to build a custom Linux kernel that supports the ghOSt kernel module.

***[Team: Sri Ram]***

**[Measurement] Measuring CFS:** CFS, while being called “Completely Fair Scheduler”, may not be fair as expected under a certain workload. In this project, you will build tools and techniques to measure and analyze Linux CFS in not only regular scheduler performance (throughput, turnaround time, etc.) but also new metrics that you define (e.g., how would you define a fairness metric). This will be a fun measurement project of a complex OS component.

***[Team: Srinivasan, Vishwakarma] Friday***

**[Measurement] Isolation and scheduling of multi-threaded cloud functions:** Serverless functions are becoming increasingly multi-threaded, and yet, they are sensitive to scheduling (as they are inherently short-lived). However, the Linux cgroups mechanism is buggy and cannot fairly and efficiently schedule multi-threaded programs as a user would expect. Read [this paper](https://cs.gmu.edu/~yuecheng/docs/socc21-rkube.pdf) for more insight. The challenge is to identify deficiencies of Linux CFS + cgroups in multi-threaded FaaS scheduling: How would aggressive functions interfere with less aggressive ones and therefore impact their performance? If you identify interesting observations, you could then take the next step and start to build a more efficient Linux solution.

**[Tool] Elastic container memory management:** Docker containers are a widely used technique to provide OS-level virtualization. Docker is backed using Linux cgroups and namespace mechanisms. Cgroups handle resource isolation for containerized applications. However, when a memory-intensive, containerized application is about to hit the memory cap specified by its cgroups configuration, Linux OOM (out-of-memory) killer will kick in to silently kill the application. A useful feature is to temporarily allocate more memory for the application that is about to cause OOM and notify users so that proper action is taken in a timely manner. How can you augment existing Linux cgroups mechanism to support this feature?

***[Team: Douglas] – security aspect***

**[Tool/system] Learned hash map or learned bloom filter:** Learned index is a non-traditional way to index large amounts of data. Unlike traditional indexing structure (e.g., B+tree, hash map), a learned index exploits statistical ML and overfitting to approximate the data distribution of a database. Read the [learned index paper](https://dl.acm.org/doi/10.1145/3183713.3196909), and in this project you will design and implement a new learned index structure, a learned hash map, or a learned bloom filter, and conduct experimental evaluation to compare your learned structure with existing data structure libraries in terms of query latency and memory consumption.

***[Team: Ramos] – learned hash table***

***[Team: Bhargavi, Ashwath] – learned bloom filter Friday***

**[Study] Tail latency and predictable learned index storage:** Tail latency has become a central focus in performance of large-scale systems. The question is not what the average response time is, but rather what the 99th percentile of requests will see. Clearly, the closer 99th percentile behavior is to average, the more predictable the behavior of your system is. In this project, you'll start by picking one or multiple learned index structures (the key building blocks of future distributed storage) and then measuring latencies of said structure to understand what kind of latency profiles are common. What functionality in the learned index structures can lead to different observed tail performance? (think about reading/writing, impact of CPU caching, lots of small and large objects, and other functionality that could affect performance) If you find some interesting problems, you could then take the next step and start to build a more predictable learned index; how can you make it such that the learned index is a highly predictable building block for next-gen, larger scale systems?

**[System/tool] Applying learned index to practical systems:** In this project, you will pick an available learned index library from GitHub and port it to a real-world system / tool that you like. The system / tool can be anything from a storage system, a cluster management system, to even the Linux kernel, whichever uses a traditional (sorted) data structure for indexing. You will then evaluate the ported learned index structure in a practical setup and compare its performance and memory efficiency with the baseline version.

***[Team: Dumouchelle] Friday***

**[Study] Cache simulation study over the Snowflake data warehousing workloads:** Snowflake published its data warehousing [workload trace datasets](https://github.com/resource-disaggregation/snowset), which record the execution of 70 million queries over a 14-day period. Read [this paper](https://www.usenix.org/conference/nsdi20/presentation/vuppalapati) for an overview of the Snowflake system designs; pay attention to Section 4.4: Persistent Data Caching. In this project, you will conduct a trace-driven simulation study to analyze the cache hit ratios of different caching policies, including LRU, LFU, FIFO, ARC, GDSF (Greedy-Dual-Size-Frequency), and offline Belady MIN. How do different policies perform? Can you come up with a new, ML-guided caching policy that outperforms the best policies simulated?

***[Team: Chowdhury]***

**[System] Learned flash caching:** Flash-based solid-state drives (SSDs) are known for providing better performance than magnetic disk drives, but they have limits on endurance, the number of times data can be erased and overwritten. When flash drives are used as a cache for a larger, disk-based storage system, the choice of a cache replacement algorithm can make a significant difference in both performance and endurance. In this project, you will explore learned flash caching to improve the flash endurance without impacting the cache performance. You will explore ML methods that learn an admission control policy to not insert data into cache that would not be accessed before being evicted.

***[Team: Yang, Lu]***

**[Study] Billion-file file systems:** OS local file systems form the core of many distributed storage systems. As local hard drives and SSDs scale, the number of objects stored in each node grows. The challenge here is to measure and study such local systems at scale; how well do file systems handle your billion-file challenge? How can we study such systems? Which file systems hold up well under such workloads, and don't? What techniques are needed to use said file systems effectively?

***[Team: Shaikh] Friday***

**[Study/system] Smarter caching for DL training:** Not all data samples are created equally: some samples contribute more to DL (deep learning) model training, while some contribute less. Importance sampling brings a new level of data locality—which does not exist in conventional uniform sampling data load methods—to the data sample access patterns during a training process. The goal of this project is to use Belady MIN—an optimal offline caching policy—as a guideline to guide informed sample caching to significantly improve the performance of data loading process. This will be a fun, Sys4ML project (hacking PyTorch and/or a storage system) that combines OS principles and emerging DL, if you like that sort of thing.

***[Team: Peng, Chen]***

**[System] Serverless caching for DL training:** The cloud-native movement imagines how we can build software systems assuming that the cloud exists. Here, we take one small step: how can we take[InfiniCache](https://mason-leap-lab.github.io/infinicache/) and make it more suitable for cloud-native DL (deep learning) training? How can we combine InfiniCache as a tier-1 cache and a storage system such as S3 as a backing store to provide data sample accesses?

***[Team: Newman, Limbu, Sultana]***

**[System] RL-guided scheduling:** In this project you will explore using RL (reinforcement learning) to guide the decision making of a picked scheduler. Not all workloads can be optimized using RL. You will need to pick: (1) workloads that exhibit a certain pattern (i.e., predictability) – microservices or FaaS (workload traces available online), and (2) pick an RL method, e.g., Q-learning. As a first step, you can design and implement an RL-guided scheduler simulator to study the feasibility. As a next step, you could incorporate your idea into a real system such as Linux or Kubernetes.

**[System] Scaling faster than Lambda:** A common trend in serverless analytics/SQL is to put the entire runtime (e.g., Spark executor) inside the container. This is too wasteful since you have to start the entire runtime (download files, JIT warmup, etc.) when scaling up. This project would explore sandboxing just the user-defined functions (UDFs) of a user query, keeping the runtime shared between many users. This project would prototype the approach in a system like Ray / Spark and explore the efficiency gains. This can be a challenging system implementation project that may take considerable engineering efforts, if you have strong hands-on skill.

**[Study] Linux eBPF application study:** eBPF is a revolutionary technology that allows developers to safely and efficiently extend Linux kernel capabilities without needing to change the kernel source code. In this study you will be collecting a large number of eBPF tools/applications from the Internet and analyzing some interesting aspects of theirs. GitHub is a great source of such applications. You will need to (1) define those aspects and (2) conduct qualitative/quantitative study. This interesting class of system is understudied (I am not aware of anyone who has done it before): only you can fill the gap through your efforts.

***[Team: Su, Liu]***

**[Tool] Developing a tool using eBPF:** eBPF is an exciting technique that offers great potential for expanding the capabilities of the Linux kernel. In this project, you will explore eBPF and use it to implement a practical tool that enables some new features that are not previously available. There are lots of possibilities in this project. One idea is to combine eBPF and learned index and build an eBPF-based learned index that accelerates some core functionality in the kernel. Another possibility is to extend Linux CFS using eBPF: read [this thread](https://lwn.net/ml/linux-kernel/20210916162451.709260-1-guro@fb.com/) that introduces a CFS eBPF extension. See if you can do something similar. Also OK if you want to further extend Facebook’s scheduler eBPF effort (from the above link).

**[System] Transparent Jupyter Notebook replicator:** If you have used managed Python notebook service, e.g., Google Colab, you probably have had the following experience: your assigned notebook container, once sitting idled for a period of time, would be destroyed and you lose the execution state that was generated from previous runs. In this project, you will design tools that transparently replicate a Jupyter Notebook container from one server to another; this way, when your container gets preempted by the global scheduler, the runtime state of your previous executions will still be preserved by a separate duplicate of the same container.

**[Study] Silent data corruption (SDC):** Silent data corruption (SDC) can have a bad impact on large-scale datacenter infrastructure services. SDC cannot be effectively captured by regular fault tolerance techniques and therefore are not traceable. But SDC can silently propagate across the whole system stack and eventually lead to catastrophic consequences. In this study, you will use Internet as a resource to: 1. Collect SDC bugs/examples that were listed on the Internet (e.g., hacker news, Linux community, blogs, GitHubs, articles, etc etc.); 2. Build a database using the collected datasets; and 3. Draw insights and observations from the datasets.

***[Team: Tran]***