12/04 CS240 - Review

Announcements

- Final Exam: Tuesday, December 9, 3:30 PM - 6:30 PM in Gates B3
  - Format like Midterm: open papers, no electronic devices
  - Cumulative with strong emphasis on material not covered by the Midterm Exam
  - Less time pressure than the Midterm Exam

- Course Feedback Now Open until end of day 12-15
  - http://course-evaluations.stanford.edu

See you on Tuesday @ 3:30 PM in Gates B03

Fault-tolerance

- End-to-end
  - Relationship to hints
  - ARPAnet routing
  - Two problems? Checking, behavior under heavy load
- Log updates
  - CS240 Papers?
  - Approach: Update procedure name and arguments
- Make actions atomic or restartable
  - What are atomic and restartable? Idempotent?
  - Shadow pages?

Final Exam Review

Machine Learning and Systems

Salus: Fine-Grained GPU Sharing Primitives for Deep Learning Applications

- Motivating problem:
  - GPUs are coarse-grained resources (1 job ≈ 1 GPU)
  - Training/inference jobs waste capacity due to fluctuating memory/compute demand
  - Existing schedulers support only sequential multiplexing (fragmentation & low utilization).
- Key Idea: Timeshare GPU
  - Provide GPU sharing at operator granularity using:
    i. Lane – time-sliced, partitioned compute streams.
    ii. Memory Namespace – isolates DL model memory allocations.
  - Scheduler can interleave operators from multiple jobs without modifying DL kernels

Salus turns the GPU into a schedulable shared resource by interleaving operators

Efficient Memory Management for Large Language Model Serving with Paged Attention

- Motivating problem:
  - Serving LLMs needs dynamic attention KV caches per request.
  - Traditional allocator uses tensor-level contiguous allocations → severe fragmentation.
  - Leads to dropped queries, OOMs, and low throughput.
- Key Idea: Page KV cache
  - Represent KV cache as a virtual memory abstraction:
    i. Break cache into fixed-size blocks (pages).
    ii. Allow non-contiguous allocation + per-request logical address space.
    iii. Block table acts like a page table for attention memory.
  - Reuse virtual memory techniques like easy growth, copy-on-write, etc.

Paged Attention treats KV cache like paged virtual memory, eliminating fragmentation.

MLSys paper sample questions

What is the limiting factor in preemptive scheduling of deep learning training jobs?

Is there something analogous of Salus' **GPU Lanes** in vLLM? If so, what is it. If not, why was it not needed?

Salus, vLLM, and the Superpages system all address memory fragmentation but in different contexts and with different goals. Describe how each system handles fragmentation and explain why its approaches differ.

Rethinking OS Abstractions

Application Performance and Flexibility on Exokernel Systems

- Motivating problem:
  - Traditional OSes (monolithic, microkernel) provide fixed abstractions:
    - Hide hardware details users may want to optimize
    - Impose one-size-fits-all policies for resource management
    - Limit application-level innovation and specialization
- Key Idea: Exokernel
  - Expose hardware to applications securely; push abstractions into user space
  - Kernel securely multiplexes resources, but does not impose abstractions.
  - Applications (or libraries) build their own OS abstractions (e.g., pager, file system)

Exokernels separate protection (kernel) from management (apps), enabling specialization and higher performance

Dune: Safe User-level Access to Privileged CPU Features

- Motivating problem:
  - Modern CPUs (e.g., x86) have features critical for performance and isolation:
    - Page tables, TLB control, cache control
  - …but applications cannot access them directly because they require privileged mode
  - High OS overhead, poor performance for high-performance runtimes (JVM, DBs, GCs, JITs)
- Key idea: Use VMX to safely expose privileged features
  - Treat user processes as lightweight VMs by running thread in ring-0 in a sandbox
    - Nested page tables used to give sandbox access to Linux process memory
  - User-level memory management: JITs/GCs control page faults, TLB management, huge pages

Dune uses hardware virtualization to give apps safe, direct access to privileged CPU features

Enabling OS-bypass without losing isolation.

Sample exam questions

Exokernel vs Dune: Both reduce the abstraction gap between applications and hardware. Where do they differ in what they expose and how they protect it?

In Exokernel, why is revocation of resources (like physical pages or disk blocks) difficult to implement transparently?

List one operation that would be faster in a Dune process when compared to running under Linux. List one operation that would be slower.

Log-structured file systems

The Design and Implementation of a Log-Structured File System

- Motivating problem:
  - Disk performance is bottlenecked by small random writes.
  - Traditional FS (e.g., FFS) performs many seeks and updates metadata in place
  - Workloads with many small files suffer very low write throughput
- Key Idea: Log-Structured File System
  - Write everything (data + metadata) in a single append-only stream
  - All writes are batched and appended sequentially to reduce seeks
  - Metadata stored with data in log segments → faster crash recovery
  - Inode map, segment cleaner, checkpoint/roll forward recovery

LFS trades cleaning overhead for high write performance through sequential, log-like disk layout

F2FS: A New File System for Flash Storage

- Motivating problem
  - SSDs & flash storage behave differently from disks:
    - No seek penalty for random access
    - Erase-before-write constraints + limited write endurance
    - Existing LFS have issues: write amplification
- Key idea: Flash-Friendly File System
  - Design a log-structured file system specifically for flash devices, optimizing cleaning, allocation, and metadata layout for NAND characteristics
    - Segmented + Node/ Data logs
    - Multi-head, adaptive logging
    - Alignment with Flash Translation Layer (FTL)

F2FS brings LFS ideas into the flash era by optimizing placement, cleaning, and metadata for NAND

Rethink the sync

- Motivating problem
  - Synchronous I/O gives clean durability + ordering,
    - 2 orders of magnitude slower on disk-intensive workloads
  - Asynchronous I/O is fast but unsafe
    - Users can see outputs that depend on data that is later lost in a crash/power failure
- Key Idea: External Synchrony
  - Guarantee sync semantics to external observers, not to the calling process
  - Buffer all external output (screen, network, etc.) that causally depends on uncommitted writes until those writes are durable

External synchrony lets apps pretend everything is synchronous,
while the OS enforces durability only at the external interface
Getting synchronous semantics with near-asynchronous performance

Sample LFS questions

Why does LFS benefit more from large sequential writes than F2FS on SSDs?

What is the wandering tree problem in LFS? How did F2FS avoid it?

Why were indirect blocks considered cold in F2FS but not LFS?

LFS needs to find the disk location of inode 7 but imap[7] has no entry. Assuming this is not a bug: JimBob states LFS can traverse from the root directory down to find it; BobJim states it has to scan all segments to find the most recent copy. Who is correct (and why)?

Distributed file systems

Design and Implementation of the Sun Network Filesystem

- Motivating problem
  - Need a transparent shared file system across a distributed network
  - Must support existing UNIX applications without modification
  - Networks are unreliable; servers and clients crash independently
- Key idea: Stateless server, specify as a network protocol
  - Server keeps no client state, so it can crash and recover without coordination
  - Every RPC request contains full file identifiers and offsets (File handles)
  - Client caching with weak consistency

NFS trades strict consistency for portability and crash tolerance via server statelessness

Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency

- Motivating problem
  - Client caching improves performance but introduces consistency problems
  - Existing systems:
    - Callbacks (e.g., AFS) → break on failures unless state is persisted
    - Timeout validation (e.g., NFS) → weak semantics and excessive checking
- Key idea: Leases
  - A client obtains a lease to cache or modify a file block
  - Lease grants exclusive or shared access, but only for a fixed duration
  - If the server fails or client is unreachable, leases simply expire, avoiding stale state

Leases provide strong consistency like callbacks but remain safe under failures because expiration naturally cleans state

Sample questions

You build an NFS server as an ordinary process that can run on either xsyncfs or LFS. Your NFS server guarantees persistence by calling sync() before replying to clients. Your marketing wants better performance and deletes these sync() calls. What effect will this have for correctness or performance in normal usage for either file system? If there is a crash?

Random questions

You compare the system in the Lease's paper to two others: one with infinitely fast CPUs but the same network as in the leases paper and one with an infinitely fast network but the same CPUs as in the leases paper. Rank these three systems in terms of the relative benefit of leases for each and give the intuition behind your ordering.

Assume LFS has split a 100MB disk that has 50% utilization into 1MB. Draw the a distribution of segment utilization that will give a write cost of 1.

Comparison paper: Recall that guest OS page tables map VPNs to PPNs, and the real ("shadow") page tables in ESX map these same VPN to MPNs. In Section 2.4 of the Comparison paper, they state: "avoiding all use of traces causes ... a large number of hidden faults." When would these hidden page faults occur? How does their system differ from that approach?

Good luck on the exam