Even More on File System

CS 256/456

Dept. of Computer Science, University of Rochester

Recap of File System

- File system interface
  - files/directories
  - access models and operations
- Space allocation for disk files
  - contiguous allocation, linked allocation, indexed allocation
  - compare on space efficiency and random access efficiency
- Free space management
  - bit map, linked list, grouping
- I/O buffer management
  - caching and prefetching

Recovery from Machine Crash

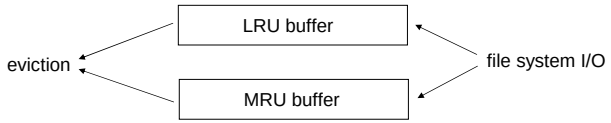
- File system operations are not atomic; a sudden machine crash may leave the file system in an inconsistent state
- Consistency checking and fix takes too long
- Journaling file system: Ext3, ReiserFS, NTFS
  - maintain a dedicated journal that logs all operations
  - the logging happens before the real operation
  - each logging is made to be atomic
  - after the completion of an operation, its entry is removed from the journal
  - at the recovery time, only journal entries need to be examined ⇒ fast recovery
  - similar to transactions in database systems
  - performance impact?

Log-Structured File Systems

- With CPUs faster, memory larger
  - buffer caches can also be larger
  - most of read requests can come from the memory cache
  - thus, most disk accesses will be writes
  - poor disk performance when most writes are small
- LFS Strategy [Rosenblum & Ousterhout SOSP1991]
  - structures entire disk as a log
  - always write to the end of the disk log
  - when updates are needed, simply add new copies with updated content; old copies of the blocks are still in the earlier portion of the log
  - periodically purge out useless blocks

I/O Buffer Management

- buffer cache replacement policy
  - non-repetitive sequential file access – MRU
  - potentially repetitive access with locality – LRU



- how to allocate space between the two buffers?
  - objective: minimize system-wide miss rate
  - assume continuous buffer size to miss rate function for each buffer, locate a point of equal marginal gain for both buffers

Speculative I/O Prefetching

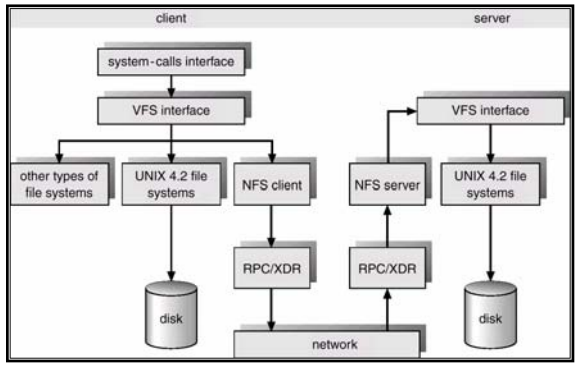
- Prefetching is effective to reduce wait time on I/O
  - that is, when you prefetch the right data
- Speculative I/O prefetching [Chang & Gibson OSDI1999]
  - have a speculative execution of the program running ahead of the actual execution
  - when the speculative execution reads, it issues the read request and then proceeds without waiting for the data to return
    - now you understand why it runs "ahead" of the actual execution
  - when the actual execution reaches the read operation, the data is likely already there – prefetching!
  - problems?
    - speculative execution may not run the same way as the actual execution if it depends on the returned I/O data
    - double changes of persistent system state

Distributed-File Systems

- Distributed file system (DFS) – unified file system managing distributed storage resources.
- Purpose of DFSs
  - larger file system
  - performance through parallel & concurrent I/O
  - availability/fault-tolerance through replication

Network File System (NFS)

NFS: a mechanism to transparently access remote files.



Distribution & Access Granularity

- Distribution granularity:
  - File-level distribution
  - Block/chunk-level distribution
    - tradeoff of large/small block/chunks
    - management overhead vs. storage flexibility
- Access granularity:
  - Data accesses are in block/chunks
    - tradeoff of large/small block/chunks
    - management overhead vs. unnecessary fetching and false sharing

Remote File Caching

- Reduce network traffic by retaining recently accessed disk blocks in a cache
  - repeated accesses to the same information can be handled locally.
  - if needed data not already cached, a copy of data is brought from the server to the user.
- Cached on disk or in main memory?
  - advantages of disk caches:
    - larger and can survive power down
  - memory cache:
    - faster
    - allow integrated management of local file system cache and distributed file system cache

Cache Consistency

- Cache consistency
  - keeping the cached copies consistent with the master copy
- Client-initiated approach
  - client initiate a validity check
  - server checks whether the local data is consistent with the master copy
- Server-initiated approach
  - server records, for each client, the files it caches
  - whenever there is a change, the server notifies clients
- Compare them
  - consistency
  - robustness

Stateful vs. Stateless Designs

- Stateful file management
  - Server maintains state (e.g., set of opened files, file access pointers, file access pattern) for each connection
- Stateless file management
  - Avoids state info by making each request self-contained
- Stateful management is more efficient, stateless management is more robust and less complex
- Soft-state
  - the server maintains per-client state
  - each client has to continuously send refresh messages to the server to keep them alive

File Replication

- File replication
  - Replicas of the same file reside on failure-independent machines.
  - Improves availability and throughput.
  - Directory mechanism maps a file name to the set of replicas.
- For read-only accesses
  - one copy needs to be accessed
- For updates
  - update must be reflected on all replicas
  - consistency?

Replication Consistency

- Primary-secondary replication
  - updates first go to one replica; then propagated to others
  - problem: low update availability
- Optimistic update anywhere
  - updates can first go anywhere and then gradually propagated to others
  - resolve inconsistencies when they occur
    - logging and undoing
    - commutative updates; total updates

Disclaimer

- Parts of the lecture slides contain original work of Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Andrew S. Tanenbaum, and Gary Nutt. The slides are intended for the sole purpose of instruction of operating systems at the University of Rochester. All copyrighted materials belong to their original owner(s).