CSci555: Advanced Operating Systems
Lecture 8 - October 17, 2003
File System - Performance
(slides by Dr. Katia Obraczka)

Dr. Tatyana Ryutov
Dr. Dongho Kim
University of Southern California
Information Sciences Institute

Outline

- **Leases**
  - **Continuum of cache consistency mechanisms.**
- **Log Structured File System and RAID.**
  - **FS performance from the storage management point of view.**

Review

- **File Systems.**
- **File System Case Studies:**
  - **NFS.**
  - **Sprite.**
  - **Andrew.**
  - **Coda.**

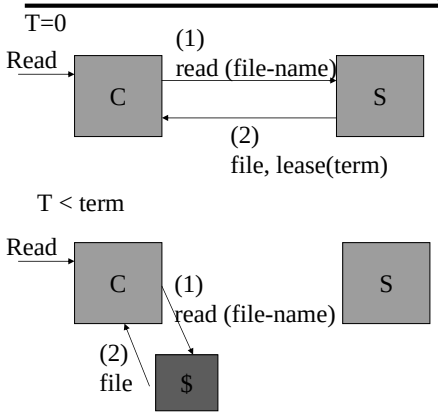
Caching

- **Improves performance in terms of response time, availability during disconnected operation, and fault tolerance.**
- **Price: consistency**
  - **Methods:**
    - **Timestamp-based invalidation**
      - **Check on use**
    - **Callbacks**

Leases

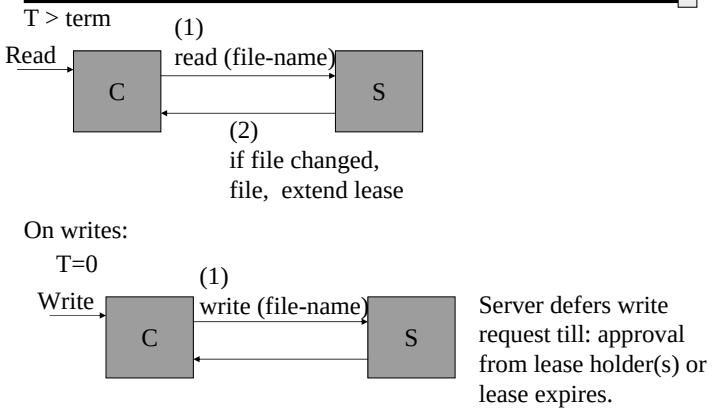
- **Time-based cache consistency protocol.**
- **Contract between client and server.**
  - **Lease grants holder control over writes to corresponding data item** during lease term**.**
  - **Server must obtain approval from holder of lease before modifying data.**
  - **When holder grants approval for write, it invalidates its local copy.**

Protocol Description 1



If file still in cache:
if lease is still valid, no need to go to server.

Protocol Description 2



Server defers write request till: approval from lease holder(s) or lease expires.

Considerations

- **Unreachable lease holder(s)?**
- **Leases and callbacks.**
  - **Consistency?**
  - **Lease term**

Lease Term

- **Short leases:**
  - **Minimize delays due to failures.**
  - **Minimize impact of false sharing.**
  - **Reduce storage requirements at server (expired leases reclaimed).**
- **Long leases:**
  - **More efficient for repeated access with little write sharing.**

Lease Management 1

- **Client requests lease extension before lease expires in anticipation of file being accessed.**
  - **Performance improvement?**

Lease Management 2

- **Multiple files per lease.**
  - **Performance improvement?**
  - **Example: one lease per directory.**
  - **System files: widely shared but infrequently written.**
  - **False sharing?**
  - **Multicast lease extensions periodically.**

Lease Management 3

- **Lease term based on file access characteristics.**
  - **Heavily write-shared file: lease term = 0.**
  - **Longer lease terms for distant clients.**

Clock Synchronization Issues

- **Servers and clients should be roughly synchronized.**
  - **If server clock advances too fast or client's clock too slow: inconsistencies.**

Next...

- **Papers on file system performance from storage management perspective.**
- **Issues:**
  - **Disk access time >>> memory access time.**
  - **Discrepancy between disk access time improvements and other components (e.g., CPU).**
- **Minimize impact of disk access time by:**
  - **Reducing # of disk accesses or**
  - **Reducing access time by performing parallel access.**

Log-Structured File System

- **Built as extension to Sprite FS (Sprite LFS).**
- **New disk storage technique that tries to use disks more efficiently.**
- **Assumes main memory cache for files.**
- **Larger memory makes cache more efficient in satisfying reads.**
  - **Most of the working set is cached.**
- **Thus, most disk access cost due to writes!**

Main Idea

- **Batch multiple writes in file cache.**
  - **Transform many small writes into 1 large one.**
  - **Close to disk's full bandwidth utilization.**
- **Write to disk in one write in a contiguous region of disk called *log*.**
  - **Eliminates seeks.**
  - **Improves crash recovery.**
    - **Sequential structure of log.**
    - **Only most recent portion of log needs to be examined.**

LSFS Structure

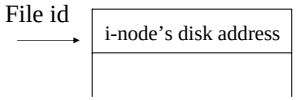
- **Two key functions:**
  - **How to retrieve information from log.**
  - **How to manage free disk space.**

File Location and Retrieval 1

- **Allows random access to information in the log.**
  - **Goal is to match or increase read performance.**
  - **Keeps indexing structures with log.**
- **Each file has i-node containing:**
  - **File attributes (type, owner, permissions).**
  - **Disk address of first 10 blocks.**
  - **Files > 10 blocks, i-node contains pointer to more data.**

File Location and Retrieval 2

- **In UNIX FS:**
  - **Fixed mapping between disk address and file i-node: disk address as function of file id.**
- **In LFS:**
  - **I-nodes written to log.**
  - **I-node map keeps current location of each i-node.**



  - **I-node maps usually fit in main memory cache.**

Free Space Management

- **Goal: maintain large, contiguous free chunks of disk space for writing data.**
- **Problem: fragmentation.**
- **Approaches:**
  - **Thread around used blocks.**
    - **Skip over active blocks and thread log through free extents.**
  - **Copying.**
    - **Active data copied in compacted form at head of log.**
    - **Generates contiguous free space.**
    - **But, expensive!**

Free Space Management in LFS

- **Divide disk into large, fixed-size segments.**
  - **Segment size is large enough so that transfer time (for read/write) >>> seek time.**
- **Hybrid approach.**
  - **Combination of threading and copying.**
  - **Copying: segment cleaning.**
  - **Threading between segments.**

Segment Cleaning

- **Process of copying "live" data out of segment before rewriting segment.**
- **Number of segments read into memory; identify live data; write live data back to smaller number of clean, contiguous segments.**
- **Segments read are marked as "clean".**
- **Some bookkeeping needed: update files' i-nodes to point to new block locations, etc.**

Crash Recovery

- **When crash occurs, last few disk operations may have left disk in inconsistent state.**
  - **E.g., new file written but directory entry not updated.**
- **At reboot time, OS must correct possible inconsistencies.**
- **Traditional UNIX FS: need to scan whole disk.**

Crash Recovery in Sprite LFS 1

- **Locations of last disk operations are at the end of the log.**
  - **Easy to perform crash recovery.**
- **2 recovery strategies:**
  - **Checkpoints and roll-forward.**
- **Checkpoints:**
  - **Positions in the log where everything is consistent.**

Crash Recovery in Sprite LFS 2

- **After crash, scan disk backward from end of log to checkpoint, then scan forward to recover as much information as possible: *roll forward*.**

More on LFS

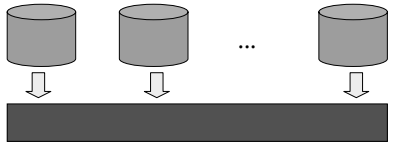
- **Paper talks about their experience implementing and using LFS.**
- **Performance evaluation using benchmarks.**
- **Cleaning overhead.**

Redundant Arrays of Inexpensive Disks (RAID)

- **Improve disk access time by using arrays of disks.**
- **Motivation:**
  - **Disks are getting inexpensive.**
  - **Lower cost disks:**
    - **Less capacity.**
    - **But cheaper, smaller, and lower power.**
- **Paper proposal: build I/O systems as arrays of inexpensive disks.**
  - **E.g., 75 inexpensive disks have 12 * I/O bandwidth of expensive disks with same capacity.**

RAID Organization 1

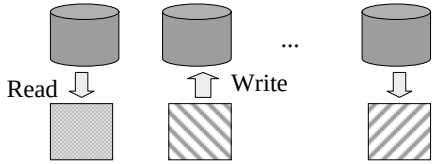
- **Interleaving disks.**
  - **Supercomputing applications.**
  - **Transfer of large blocks of data at high rates.**



Grouped read: single read spread over multiple disks

RAID Organization 2

- **Independent disks.**
  - **Transaction processing applications.**
  - **Database partitioned across disks.**
  - **Concurrent access to independent items.**



Problem: Reliability

- **Disk unreliability causes frequent backups.**
- **What happens with 100*number of disks?**
  - **MTTF becomes prohibitive**
  - **Fault tolerance otherwise disk arrays are too unreliable to be useful.**
- **RAID: use of extra disks containing redundant information.**
  - **Similar to redundant transmission of data.**

RAID Levels

- **Different levels provide different reliability, cost, and performance.**
- **MTTF as function of total number of disks, number of data disks in a group (G), number of check disks per group (C), and number of groups.**
- **C determined by RAID level.**

First RAID Level

- **Mirrors.**
  - **Most expensive approach.**
  - **All disks duplicated (G=1 and C=1).**
  - **Every write to data disk results in write to check disk.**
  - **Double cost and half capacity.**

Second RAID Level

- **Hamming code.**
- **Interleave data across disks in a group.**
- **Add enough check disks to detect/correct error.**
- **Single parity disk detects single error.**
- **Makes sense for large data transfers.**
- **Small transfers mean all disks must be accessed (to check if data is correct).**

Third RAID Level

- **Lower cost by reducing C to 1.**
  - **Single parity disk.**
- **Rationale:**
  - **Most check disks in RAID 2 used to detect which disks failed.**
  - **Disk controllers do that.**
  - **Data on failed disk can be reconstructed by computing the parity on remaining disks and comparing it with parity for full group.**

Fourth RAID Level

- **Try to improve performance of small transfers using parallelism.**
- **Transfer units stored in single sector.**
  - **Reads are independent, i.e., errors can be detected without having to use other disks (rely on controller).**
  - **Also, maximum disk rate.**
  - **Writes still need multiple disk access.**

Fifth RAID Level

- **Tries to achieve parallelism for writes as well.**
- **Distributes data as well as check information across all disks.**