Advanced Operating Systems Lecture notes

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

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

Dr. Clifford Neuman
Dr. Tatyana Ryutov
University of Southern California
Information Sciences Institute

Outline

- Leases (covered last class)
  - Time-based cache consistency protocol.
- Log Structured File System and RAID.
  - FS performance from the storage management point of view.
- Locus system

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

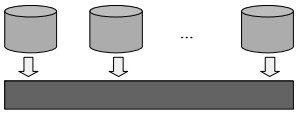
- Paper talks about their experience implementing and using LFS.
- Performance evaluation using benchmarks.
- Cleaning overhead.

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

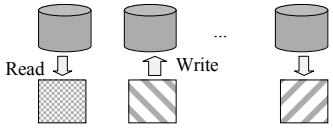
- Interleaving disks.
  - Supercomputing applications.
  - Transfer of large blocks of data at high rates.



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.

The LOCUS System

- Developed at UCLA in early 80's
  - Essentially a distributed Unix
- Major contribution was transparency
  - Transparency took many forms
- Environment:
  - VAX 750's and/or IBM PCs connected by an Ethernet
- UNIX compatible.

LOCUS

- Network/location transparency:
  - Network of machines appear as single machine to user.
  - Hide machine boundaries.
  - Local and remote resources look the same to user.

Transparency in Locus

- Network Transparency
  - Ability to hide boundaries
- Syntactic Transparency
  - Local and remote calls take same form
- Semantic Transparency
  - Independence from Operand Location
- Name Transparency
  - A name always refers to the same object
  - No need for closure, only one namespace

Transparency in Locus (cont)

- Location Transparency
  - Location can't be inferred from name
  - Makes it easier to move objects
- Syntactic Transparency
  - Local and remote calls take same form
- Performance Transparency
  - Programs with timing assumptions work
- Failure Transparency
  - Remote errors indistinguishable from local
- Execution Transparency
  - Results don't change with location

LOCUS Distributed File System

- Tree-structured file name space.
  - File name tree covers all file system objects in all machines.
  - Location transparency.
  - File groups (UNIX file systems) "glued" via mount.
- File replication.
  - Varying degrees of replication.
  - Locus responsible for consistency: propagate updates, serve from most up-to-date copy, and handle partitions.

Replication in LOCUS

- File group replicated at multiple servers.
- Replicas of a file group may contain different subsets of files belonging to that file group.
- All copies of file assigned same descriptor (i-node #).
  - File unique name: <file group#, i-node #).

Replica Consistency

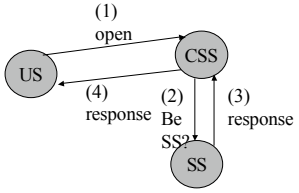
- Version vectors.
  - Version vector associated with each copy of a file.
  - Maintain update history information.
  - Used to ensure latest copies will be used and to help updating outdated copies.
  - Optimistic consistency.
    - Potential inconsistencies.

File System Operations 1

- Using site (US): client.
- Storage site (SS): server.
- Current synchronization site (CSS): synchronization site; chooses the SS for a file request.
  - Knowledge of which files replicated where.

File System Operations 2

- Open:



File Modification

- At US:
  - After each change, page sent to SS.
  - At file close, all modified pages flushed to SS.
- At SS: atomic commit.
  - Changes to a file handled atomically.
  - No changes are permanent until committed.
  - *Commit* and *abort* system calls.
  - At file close time, changes are committed.
  - Logging and shadow pages.

CSS

- Can implement variety of synchronization policies.
  - Enforce them upon file access.
  - E.g., if sharing policy allows only read-only sharing, CSS disallows concurrent accesses.