

CSE 120 Discussion 7

Chengmo Yang

11/28/07

Topic of Today

- We will simply go through the FFS paper by McKusick, Joy, Leffler, and Fabry in 1984:
 - A Fast File System for UNIX

Original UNIX FS

- A FS described by its *superblock* that contains
 - The # of data blocks
 - The max # of files
 - A pointer to the free list
 - A linked list of all the free blocks
- Every file has an *inode* that stores metadata info
 - Ownership and access types
 - Time stamps of last modification
 - Access times
 - 15 indices pointing the data blocks
 - 12 direct blocks + single, double, and triple indirect

Basic Performance Improvement

- The original UNIX FS cannot provide high bandwidth
 - No alignment constraints on data transfers
 - All operations made to appear synchronous
 - Block size is only 0.5KB
- Increase block size to 1KB enables more than 2 times performance improvement, why?
 - Each disk transfer accessed twice as much data
 - Most files can be described without need to access indirect blocks
- But, still only 4% of the disk bandwidth is used

Performance Problems

- *Inode* info separated from the data blocks
 - File accesses incur a long seek, going back and forth from inode to data blocks
- Files in a single directory not allocated consecutive slots of *inodes*
 - Many nonconsecutive blocks of *inodes* to be accessed
- Data blocks allocated randomly in aging file systems
 - As FS “ages” and fills, blocks of deleted files need to be freed up for new files
 - Blocks of new files being scattered across the disk

Changes Made in the New FS

- Disk partitioned into multiple *cylinder groups*
 - Data blocks in same file allocated in same cylinder
 - Files in same directory allocated in same cylinder
 - Inodes for files allocated in same cylinder as file data blocks
- Each *cylinder group* records
 - A redundant copy of *superblock*
 - A bit map of free blocks in the group (replace the old free list)
 - Summary of usage of data blocks
 - A fixed # of inodes allocated at FS creation
- Bookkeeping info stored not at the beginning of each cylinder group, but at a varying offset from the beginning, *why?*

Optimizing Storage Utilization

- Increase minimal block size to 4KB
 - Block size = $2^n \times 4K$, recorded in the *superblock*
 - A higher disk transfer throughput achieved
 - Large files only need two levels of indirection for 2^{32}
- But, a large block size wastes space since most files in UNIX are small
- Solution: allowing the division of a single block into multiple *fragments*
 - Fragment size specified at FS creation
 - The block map of each cylinder group records the available space at the *fragment level*

Space Utilization Comparison

- Roughly same amount of disk utilization
 - The new FS needs to use more space for recording available free blocks
 - The new FS uses less space for indexing info of large files and the same amount of space for small space
- But, a FS cannot be kept full
 - To be able to allocate according to cylinder groups, the disk must have free space scattered across cylinders
 - Throughput cut in half if no free blocks is available
 - A *free space reserve* parameter constrains the min % of free blocks
 - 10% of the disk needs to be reserved

FS layout policies

- General performance improvement strategies
 - Increase the locality of reference to minimize seek latency
 - Improve the layout of data to make larger transfers possible
- The new FS has two levels of layout policies
 - Local allocation routines to layout data blocks
 - System-wide global policies for the placement of new *inodes* and data blocks
- Global policies
 - Call local allocation routines with request for specific blocks
 - Spread unrelated data among different cylinder groups
 - New directory placed in a cylinder group that has more than average # of free inodes. Why?
 - Inodes of files in the same directory are frequently accessed together

Performance Results

Table IIa. Reading Rates of the Old and New UNIX File Systems

Type of file system	Processor and bus measured	Speed (Kbytes/s)	Read bandwidth %	% CPU
Old 1024	750/UNIBUS	29	29/983 3	11
New 4096/1024	750/UNIBUS	221	221/983 22	43
New 8192/1024	750/UNIBUS	233	233/983 24	29
New 4096/1024	750/MASSBUS	466	466/983 47	73
New 8192/1024	750/MASSBUS	466	466/983 47	54

Table IIb. Writing Rates of the Old and New UNIX File Systems

Type of file system	Processor and bus measured	Speed (Kbytes/s)	Write bandwidth %	% CPU
Old 1024	750/UNIBUS	48	48/983 5	29
New 4096/1024	750/UNIBUS	142	142/983 14	43
New 8192/1024	750/UNIBUS	215	215/983 22	46
New 4096/1024	750/MASSBUS	323	323/983 33	94
New 8192/1024	750/MASSBUS	466	466/983 47	95

Performance Results (cont'd)

For the new FS

- Throughput rate tied more strongly to the amount of free space
- Both reads and writes are faster
 - Mainly because of larger block size
- Cost per byte allocated is about the same
 - Fewer blocks need to be allocated
 - A larger overhead in block allocation
 - Mainly because of the more complicated allocation policies
- Old FS has faster file writing than reading
 - Asynchronous write but synchronous read
- New FS has faster file reading than writing
 - Blocks are more ordered, while kernel needs to allocate blocks for writing