File system trace papers

- **A Fast File System For UNIX. M.K. McKusick, W.N. Joy, S.J. Leffler, and R.S. Fabry. ACM TOCS. Vol. 2, No. 3 (August 1984), pp. 181-197.**

Disks when FFS was designed

- Slow (3600 RPM)
- Dumb (no disk caching, requests are issued sequentially – by the time that request was made to the next sector, the disk could've rotated away)
- CPU performs many block allocation and scheduling decisions + CPU was slow and was saturating

Old UNIX file system

- Disks split into partitions
  - Superblock (file system parameters)
  - Inodes
  - Files and directory blocks
  - 512 byte data blocks (disk sector size)
    - After a while, allocation becomes random
    - Inodes and data blocks randomly distributed throughout the disk
  - File names were small, no locking, no symbolic links etc.

FFS

- Block sizes are at least 4KB – different file systems on the same disk (but on different partitions) can have different block sizes
  - Use fragments to achieve smaller sizes
  - Are these still relevant? What problems are fragments solving? What is the cost?
- Use cylinder groups to distribute inodes around the disk. Keep related items on the same cylinder group
  - Space inefficient – reserve a certain amount of disk (10%)
  - Only root can use the last 10%. Performance severely degrades because FFS essentially becomes random allocation when there is no space

FFS

- Superblocks are replicated at predictable but different locations on different cylinder groups for better error recovery
- Cylinder groups: Keep related inodes and data together (directory, files).
  - Assign new directories to different cylinder groups. Challenge is to choose good cylinder groups because file sizes and number of files grow in the future while cylinder group allocation happens when a new directory is created.
  - Back then, since the CPU was involved with disk scheduling, free blocks had to be offset by certain number of blocks (rotational latency and issue latency)
    - Are they still a concern when disk controller is involved

Fragments

- Allocate new fragments at the end of file.
  - Fragments can belong to multiple files
  - As files grow, we would like to coalesce fragments of the same file into a single block
  - Challenge is to balance space usage with performance
  - Applications can help by finding out the block size and writing in block size worth of data
    - Easy if your program used stdio library
    - Cp program and other system utilities do this
    - Question: Are fragments still relevant? What is a good block size for modern files?

Block allocation

- Optimize for sequential access (why?)
  - Use rotational close blocks in the same cylinder
  - If not, use blocks from the same cylinder group
  - If not, use a quadratic hashing mechanisms (rare
  - Otherwise, random allocation
- What happens if one file in a directory uses all the blocks from a cylinder group. Every other file will have far pointers for data blocks.
  - Solution: Break large files across multiple cylinder groups. UNIX inode, good point to break are indirect pointers