Persistence: Fast File System

CS 537: Introduction to Operating Systems

Louis Oliphant

University of Wisconsin - Madison

Fall 2023

Administrivia

- Project 6 due Nov 22nd @ 11:59pm
- Exam 2 Grades posted, any issues email me
- Instructor office Hours tomorrow are cancelled
- One Discussion Section on Wednesday 11/22
  - Topic: P6 and Fuse Tutorial
  - Time: 10:00 AM
  - Zoom Meeting https://uwmadison.zoom.us/j/92013072844?pwd=b1hIY1dTK1dzR1FSYUZOTytGaWc0Zz09
  - Meeting ID: 920 1307 2844
  - Passcode: 681534
- Project 7 posted tomorrow, due Dec 8th & 11:59pm
- Midterm 3 scheduled for Dec 12th in-class
  - Alternate time Dec 18th @ 12:25pm (email me)

Review File System Implementation

- Data Structures
  - Superblock, inode and data bitmap, inode table, data blocks
- Access Methods
  - How does a call like `open()`, `read()`, or `write()` get mapped onto the data structures of the disk?

Quiz 16 Inodes & File Systems

https://tinyurl.com/cs537-fa23-q17

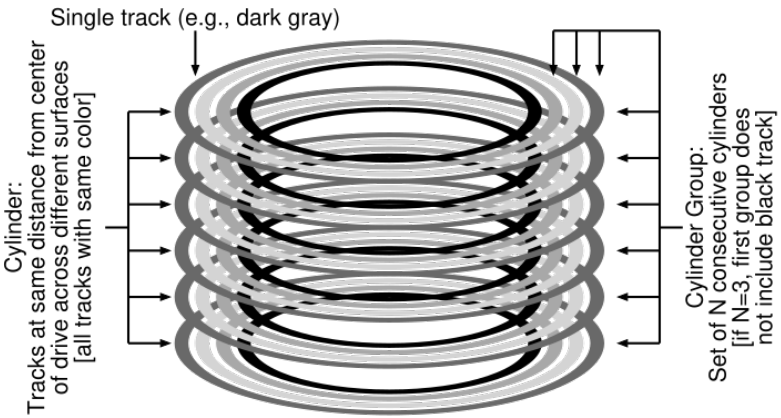
Locality and the Fast File System

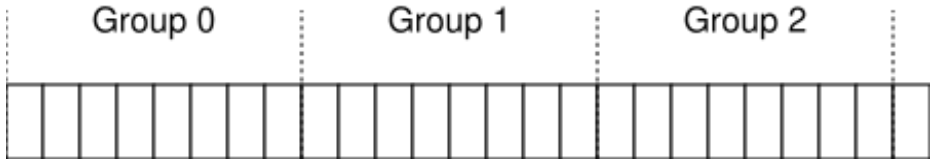
- Original Unix file system was slow, delivering only **2% of overall disk bandwidth**
  - Treated the disk like it was random-access memory
  - File system ended up getting **fragmented**

  - Original block size was too small, minimizing **internal fragmentation**, but bad for transfer as each block might require a positioning overhead
- Group at Berkeley built the **fast file system** designed to be **disk aware**

Fast File System Idea

- Organize file system structures and allocation policies to be **disk aware**
- Divided disk into collection of **cylinder groups**
- Modern file systems organize drive into similar **block groups** (consecutive portion of disk's address space)





- FFS includes all structures of a file system **within each group**

Per-Group Data Structures

- per-group **super-block** (needed to mount the file system, if one copy corrupt can us other copies)
- per-group **inode bitmap** and **data bitmap**
- per-group **data blocks**

- since all structures are per-group, they are close together on disk (less seek time)

Allocating Files and Directories

Keep related stuff together, keep unrelated stuff far apart.

Placement of Directories

- Find cylinder group with low number of allocated directories (to balance directories across groups) and high number of free inodes (to subsequently be able to allocate a bunch of files)
  - Put the directory data here
  - Put the directory inode here

Placement of Files

- Allocate a file's data blocks in same group as its inode
- Place all files in same directory in group with directory

Example Layout

Directories:

```
/
/a
/b
```

Files:

```
/a/c
/a/d
/a/e
/b/f
```

```
group   inodes      data
0       /---------  /---------
1       acde------  accddee---
2       bf--------  bff-------
3       ----------  ----------
4       ----------  ----------
5       ----------  ----------
6       ----------  ----------
7       ----------  ----------
...
```

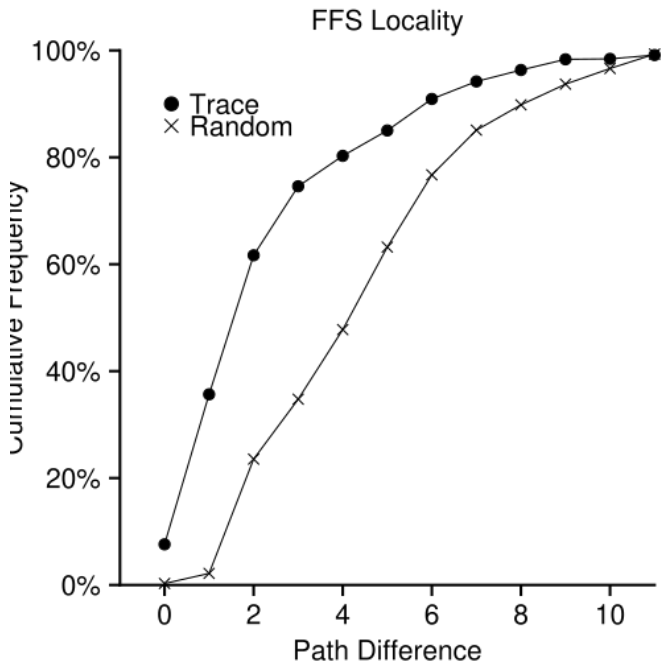
Common Sense suggests files in a directory are often accessed together
FFS will improve performance because (1) inode and data are together and (2) namespace-based locality

Measuring File Opening Locality

Analyzing the SEER workload trace of opening files:

Path Difference Metric measures how far up directory tree to find *common ancestor*:

- Same file – 0
- Another file, same directory – 1
- Another file, parent directory – 2
- Etc.



- Compared to randomly reordering file openings
- 7% were to same file
- 40% were to same directory

Large File Exception

A large file (e.g. 30 data blocks) would entirely fill most of the data blocks in a group, leaving little room for other files in the directory to be placed in the same group

```
group   inodes      data
0       /a-------  /aaaaaaaaa aaaaaaaaaa aaaaaaaaaa a---------
1       ----------  ---------- ---------- ---------- ----------
2       ----------  ---------- ---------- ---------- ----------
...
```

The large file exception (here set to 5 blocks) spreads the file across groups:

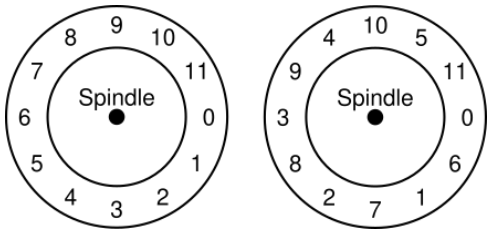
```
group   inodes      data
0       /a-------  /aaaaa---- ---------- ---------- ----------
1       ----------  aaaaa----- ---------- ---------- ----------
2       ----------  aaaaa----- ---------- ---------- ----------
3       ----------  aaaaa----- ---------- ---------- ----------
4       ----------  aaaaa----- ---------- ---------- ----------
5       ----------  aaaaa----- ---------- ---------- ----------
6       ----------  ---------- ---------- ---------- ----------
...
```

Large File Exception (cont.)

- Slows access to large files, but if chunk of a file in a group is large enough, this seeking will be **amortized**.
- FFS used 12 direct block pointers in inode (48KB) placed in group with inode
- Each indirect block pointer (4MB) pointed to block of pointers in different group, along with the data pointed to by those pointers.

Other FFS Innovations

- Introduction of **sub-blocks** (512-bytes) until file needs 4KB, then copy sub-blocks to a full block
  - Causes more I/O for each sub-block
  - Modified `libc` to buffer and do I/O in 4KB chunks
- Used skip-layout (called **parameterization**) so sequential I/O requests arrive before head rotates past them



  - Modern disks cache the entire track in an internal **track buffer**
- Added **long file names**
- Added **symbolic links**