File Systems

Broken into three sections:
1.) Disk properties
2.) Interface of file and devices; transparency
3.) Metadata and fault tolerance

Disk Properties

General trends:

1.) Disks are becoming relatively slower compared to CPUs.
2.) Disks rely on mechanical properties so not likely to get faster.

What do we do?

1.) Reduce arm motion
2.) Reduce useless rotation

How do we do this?

1.) Reorder accesses to take into account current position and locality.
2.) Take rotation times (disk properties) into placement strategy on disk.
    - because of sequential access, allocate blocks in large extents
    - leave enough in between space to deal with rotation time

Cedar:

- file name table towards middle cylinders
- log for metadata
- makes writes sequential
    - buffer metadata to create a larger sequential extent (group commit)
    - strategically place pages based on disk (hw) properties

Unix:

- inode data segregated from data (no locality)
- files in same directory not allocated in consecutive slots of inode region
- free list becomes random because of small block size => decreased ability to have sequential access
- after free list creation, ignores all disk (hw) properties

FFS:

- increase block size (but with fragments)
- to improve extent finding ability (sequential access) use bitmap
- bit map covers only a cylinder group (locality)
- rotationally optimal blocking within cylinder groups
- inodes located near data
- try to keep directory and data within cylinder group (locality)

LFS:

- assumption: more writes than reads going to disk

- want to perform writes sequentially

What they do:

- log both metadata and data
- log is only data structure on disk
- log is broken into segments
- writes to log are buffered
- exploit temporal locality instead of logical locality
- wants bimodal distribution of segments (hot and cold pages)
- cleaning, segment threading
- only bad for sequential reads after random writes

<u>Interface to files/devices (transparency)</u>

Multics:

- stream abstraction allows you to write program independent of devices
    - allows change of hw and program still runs
    - used for both devices and segments
    - all devices addressed by stream name
- segments
- synonym used to efficiently bind to files/devices

Unix:

- follows suit
- everything addressed as files
- devices are special files
- device dependent components shoved into os
- no locking
- advisory locks in FFS

Vnodes:

- abstract file system (another level of indirection)
- gives you ability to access many different types of fs uniformly
- locking on lower level
- devices manipulated through inode interface
- versioning allows you to have multiple fs implementations
- stacking allows you to have different implementations for functions for different fs versions. (paragraph below fig. 6 pg. 111: invokes op at the top of stack. It may execute the code itself or forward it down)

VAXclusters:

- filename includes: disk device name, directory name, filename
- each cluster disk has unique name
    - ⇨ loss of transparency, abstraction

- interface in terms of datagrams and messages (handshake between nodes)
- messages used for device control commands
  - ⇨ messages (vs. control registers) used to activate strong devices, meaning:
  1.) sharing of storage between hosts simplified
  2.) addition of new storage doesn't require changes to host software because everything occurs through communication interface (not device specifics)
  3.) controller can queue and reorder requests from multiple hosts

<u>Metadata and Fault Tolerance</u>

Need to keep fs consistent
If you lose metadata, you also lose data.
All of the papers use buffering to reduce disk I/O.

Unix:

- superblock located on specific contiguous region
- inodes located on specific contiguous region (separate from data)
  1.) lack of locality between metadata and data
  2.) corruption of one sector of superblock or inode forever loses info (no backup)
- any corruption causes worse case recovery

FFS:

- improves locality by moving inodes to cylinder groups and trying to keep directories and data together
- cylinder groups start at offsets
  - bit maps moved here
  - redundant copy of superblock (# blocks, max # files)
- superblocks contain vector of rotational layout tables

Cedar:

- all changes to the metadata are logged (improved locality)
- 2 copies of the log and 2 copies of file name table
- replicated copies are not placed on consecutive sectors
- buffer metadata writes to
  1.) decrease frequency of I/O
  2.) decrease amount of I/O (decrease # of writes by coalescing)
- log cleaning updates file name table
- bitmap/freelist stored in memory and can be completely reconstructed from disk file name table

LFS:

- metadata and data are in log (improves locality for writes and reads in same order as writes)
- inode map cached in memory (decrease disk accesses) and stored in fixed checkpoint region on log
- there are no data structures other than log
- checkpoint the log
- recovery goes to checkpoint and then roll-forward (fast recovery)

Bitmaps:

- easy way for fs to become inconsistent is to lose bitmap or write to bitmap
- Cedar doesn't really have one because VAM can be reconstructed quickly
- LFS doesn't either

VAXclusters:

- nodes can go up/down without causing major disruptions
- use quorum to establish live nodes
- when a node is considered to have gone down, complete new allocation of locks and lock managers
    - any lock held by node that failed are assumed to be freed
- dual paths from node to star cluster for fault tolerance
- lock management distributed