File Systems Part 5
Benefits of Multiple Disks

- They hold more data than one disk does
- Data can be stored redundantly so that if one disk fails, they can be found on another
- Data can be spread across multiple drives, allowing parallel access
Logical Volume Manager

- Spanning
  - two real disks appear to file system as one large disk
- Mirroring
  - file system writes redundantly to both disks
  - reads from one
Disk striping: each stripe is written across all disks at once. The size of a “unit” may be anywhere from a bit to multiple tracks. If it’s less than a sector in size, then multiple stripes are transferred at once so that the amount of data per transfer per disk is an integer multiple of a sector.
Concurrency Factor

- How many requests are available to be executed at once?
  - one request in queue at a time
    - concurrency factor = 1
    - e.g., one single-threaded application placing one request at a time
  - many requests in queue
    - concurrency factor > 1
    - e.g., multiple threads placing file-system requests
If we have only one waiting disk request at a time (a *concurrency factor* of 1), then a smaller striping unit is generally better than a larger one, since we can spread one request across lots of disks. But with more than one waiting disk request (a larger *concurrency factor*), it begins to make sense to have larger striping units, as the slide illustrates.

The top half of the slide shows four disks with a four-sector striping unit. Suppose we have requests for the four data areas shown, each four sectors in length. The four requests can be handled in roughly the time it takes to handle one, since the positioning for each of the requests can be done simultaneously as can the data transfer. The bottom half of the slide shows four disks with a one-sector striping unit. We have the same four requests, but in this case each is spread across all four disks, one sector per disk. Handling the requests requires first positioning the heads on all four disks for the first, then positioning the heads on all four disks for the second, and so forth. Thus the total positioning delays are four times that of the top half of the figure which has a larger striping unit.
Striping: The Effective Disk

- Improved effective transfer speed
  - parallelism
- No improvement in seek and rotational delays
  - sometimes worse
- A system depending on $N$ disks is much more likely to fail than one depending on one disk
  - if probability of one disk’s failing is $f$
  - probability of $N$-disk system’s failing is $(1-(1-f)^N)$
  - (assumes failures are IID, which is probably wrong …)

IID = independent and identically distributed.
RAID to the Rescue

• Redundant Array of Inexpensive Disks
  – (as opposed to Single Large Expensive Disk: SLED)
  – combine striping with mirroring
  – 5 different variations originally defined
    - RAID level 1 through RAID level 5
      • RAID level 0: pure striping
        – numbering extended later
      • RAID level 1: pure mirroring
RAID level 1: mirroring.
RAID level 2: bit interleaving with an error-correcting code.
RAID level 3: bit interleaving with parity bits.
RAID level 4: block interleaving with parity blocks.
RAID level 5: block interleaving with parity blocks. Rather than dedicating one disk to hold all the parity blocks, the parity blocks are distributed among all the disks. For stripe 1, the parity block might be on disk 1; for stripe 2 it would be on disk 2, and so forth. If we have eleven disks, then for stripe 11 the parity block would be back on disk 1.
RAID 4 vs. RAID 5

- Lots of small writes
  - RAID 5 is best
- Mostly large writes
  - multiples of stripes
  - either is fine
- Expansion
  - add an additional disk or two
  - RAID 4: add them and recompute parity
  - RAID 5: add them, recompute parity, shuffle data blocks among all disks to reestablish check-block pattern
Beyond RAID 5

- RAID 6
  - like RAID 5, but additional parity
  - handles two failures
- Cascaded RAID
  - RAID 1+0 (RAID 10)
    - striping across mirrored drives
  - RAID 0+1
    - two striped sets, mirroring each other
### Beyond Disks: Flash

<table>
<thead>
<tr>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Flash block ≈ file-system block</td>
<td>• Limited lifetime</td>
</tr>
<tr>
<td>• Random access</td>
<td>• Writes can be expensive</td>
</tr>
<tr>
<td>• Low power</td>
<td>• Cost more than disks</td>
</tr>
<tr>
<td>• Vibration-resistant</td>
<td>- 1TB SSD: $164.93</td>
</tr>
<tr>
<td></td>
<td>- x4.9 roughly five years ago</td>
</tr>
<tr>
<td></td>
<td>- 3TB disk: $87.99</td>
</tr>
<tr>
<td></td>
<td>- x1.45 roughly five years ago</td>
</tr>
</tbody>
</table>

Prices are from Amazon.com; current prices as of 3/7/2017. 2-day shipping included for Amazon Prime members.
It’s the nand technology that’s used for SSDs.
Coping

- Wear leveling
  - spread writes (erasures) across entire drive
- Flash translation layer (FTL)
  - specification from 1994
  - provides disk-like block interface
  - maps disk blocks to flash blocks
    - mapping changed dynamically to effect wear-leveling
Flash with FTL

• Which file system?
  – FAT32 (sort of like S5FS, but from Microsoft)
  – NTFS
  – FFS
  – Ext3

• All were designed to exploit disks
  – much of what they do is irrelevant for flash
Here we have an SSD block whose pages are initially allocated. Two pages are then freed, then a page is modified.
What actually must happen, as implemented by the SSD firmware, is the following. Since the block must first be erased before any page can be modified, a copy of the entire block must be made. Then the block is erased. Then the original contents along with the modifications to the page are copied back.

Note that the fact that two of the pages are free, and thus their contents are not important and don’t need to be copied, is not known to the SSD – the notion of free pages is known only at the file-system level.
To get around the problem of the previous slide, SSD provides a “trim” command by which the file system can tell it that certain pages are free and thus their contents are not important. Thus the two freed pages don’t need to be twice copied as in the previous slide.
Flash without FTL

- Known as memory technology device (MTD)
  - software wear-leveling
  - perhaps other tricks
JFFS and JFFS2

- Journaling flash file system
  - log-based: no journal!
    - each log entry contains inode info and some data
    - garbage collection copies info out of partially obsoleted blocks, allowing block to be erased
    - complete index of inodes kept in RAM
  - entire file system must be read when mounted

UBI/UBIFS

- UBI (unsorted block images)
  - supports multiple logical volumes on one flash device
  - performs wear-leveling across entire device
  - handles bad blocks
- UBIFS
  - file system layered on UBI
  - it really has a journal (originally called JFFS3)
  - file map kept in flash as B+ tree
  - no need to scan entire file system when mounted
Flash as Part of the Hierarchy

- Flash as log device
  - aggregate write throughput sufficient, but latency is bad
  - augment with DRAM and a “super-capacitor”
- Flash as cache
  - large level-2 cache
    - integrated into ZFS
    - can use cheaper (slower) disks with no loss of performance
    - reduced power consumption
Apple Fusion Drives (FD)

- SSD used along with hard drive
- Implemented in the LVM
  - total capacity is sum of disk and SSD capacities
    - works with both HFS+ and ZFS
  - SSD used to buffer all incoming writes
  - data is moved from disk to SSD if used sufficiently often
    - migration happens in background

The information in this slide is not confirmed by Apple (who considers the technology proprietary) but comes from https://en.wikipedia.org/wiki/Fusion_Drive.
FD Observations

• Implementation is in the LVM, i.e., below the file system
  – all decisions based on block access
• 4GB available on SSD for writes
  – all writes go to SSD while there’s space
  – otherwise go to HDD
• Frequent reads trigger promotion to SSD
• Data transferred between SSD and HDD in units of 128KB
FD Write

if !onSSD(block) {
    if SSDfreeSpace > 0 {
        remap(block)
        writeOnSSD(block)
    } else {
        writeOnHDD(block)
    }
} else {
    writeOnSSD(block)
}
FD Read

Update_Usage(block)
if onSSD(block)
    readFromSSD(block)
else
    readFromHDD(block)
FD Background Activities

    when (accessThresholdReached(block)) {
        if SSDfreeSpace > 0 {
            remap(block)
            writeOnSSD(block)
        }  
    }  

    when(SSDFreeSpace < 4GB) {
        for each infrequent block {
            remap(block)
            writeOnHDD(block)
        }  
    }
FD Block Map

- Vital data structure
- Kept up to date on SSD
- Perhaps backed up on HDD