Disk-based Storage

Oct. 20, 2009

Topics
- How disk storage fits in systems
- Performance effects of paging
- How disks work

Disk-based storage in computers

- Memory/storage hierarchy
  - Combining many technologies to balance costs/benefits
  - Recall the virtual memory lecture

An Example Memory Hierarchy

- Smaller, faster, and costlier (per byte) storage devices
- Larger, slower, and cheaper (per byte) storage devices

Memory/storage hierarchies

- Balancing performance with cost
  - Small memories are fast but expensive
  - Large memories are slow but cheap
- Exploit locality to get the best of both worlds
  - locality = re-use/nearness of accesses
  - allows most accesses to use small, fast memory

Reminder: Page Faults

A page fault is caused by a reference to a VM word that is not in physical (main) memory

- Example: An instruction references a word contained in VP 3, a miss that triggers a page fault exception
Performance and page faults

- First: how often do they happen?
  - depends! (on workloads and memory sizes)
  - in most systems, very rare
  - scenario: random access to 4GB of VM with 2GB real memory
    - 50% of memory access will generate page faults

Disk-based storage in computers

- Memory/storage hierarchy
  - Combining many technologies to balance costs/benefits
  - Recall the virtual memory lecture

- Persistence
  - Storing data for lengthy periods of time
  - DRAM/VRAM is “volatile”: contents lost if power lost
  - Disks are “non-volatile”: contents survive power outages
  - To be useful, it must also be possible to find it again later
    - this brings in many interesting data organization, consistency, and management issues
      - take 18-746/15-746 Storage Systems

What’s Inside A Disk Drive?

Disk Electronics

- Just like a small computer – processor, memory, network iface
  - Connect to disk
  - Control processor
  - Cache memory
  - Control ASIC
  - Connect to motor

Disk “Geometry”

- Disks contain platters, each with two surfaces
- Each surface organized in concentric rings called tracks
- Each track consists of sectors separated by gaps

Disk Geometry (Multiple-Platter View)

- Aligned tracks form a cylinder

Image courtesy of Seagate Technology
Disk Structure

- Read/Write Head
- Upper Surface
- Lower Surface
- Cylinder
- Track
- Sector
- Arm
- Actuator

Disk Operation (Multi-Platter View)

- Read/write heads move in unison from cylinder to cylinder
- Arm
- Spindle

Disk Structure - top view of single platter

- Surface organized into tracks
- Tracks divided into sectors

Disk Access

- Head in position above a track

Disk Access

- Rotation is counter-clockwise

Disk Access – Read

- About to read blue sector
Disk Access – Read

After reading blue sector

Disk Access – Read

Red request scheduled next

Disk Access – Seek

Seek to red's track

Disk Access – Rotational Latency

Wait for red sector to rotate around

Disk Access – Read

Complete read of red

Disk Access – Service Time Components

Seek
Rotational Latency
Data Transfer
Disk Access Time

Average time to access a specific sector approximated by:

- **Taccess** = **Tavg seek** + **Tavg rotation** + **Tavg transfer**

**Seek time (Tavg seek)**
- Time to position heads over cylinder containing target sector
- Typical **Tavg seek** = 3-5 ms

**Rotational latency (Tavg rotation)**
- Time waiting for first bit of target sector to pass under r/w head
- **Tavg rotation** = 1/2 x 1/RPMs x 60 sec/1 min
  - e.g., 3ms for 10,000 RPM disk

**Transfer time (Tavg transfer)**
- Time to read the bits in the target sector
- **Tavg transfer** = 1/RPM x 1/(avg # sectors/track) x 60 secs/1 min
  - e.g., 0.006ms for 10,000 RPM disk with 1,000 sectors/track
  - given 512-byte sectors, ~85 MB/s data transfer rate

### Disk Access Time Example

**Given:**
- Rotational rate = 7,200 RPM
- Average seek time = 5 ms
- Avg # sectors/track = 1000

**Derived average time to access random sector:**
- **Tavg rotation** = 1/2 x (60 secs/7200 RPM) x 1000 ms/sec = 4 ms
- **Tavg transfer** = 60/7200 RPM x 1/400 secs/track x 1000 ms/sec = 0.008 ms
  - **Taccess** = 5 ms + 4 ms + 0.008 ms = 9.008 ms

**Important points:**
- Access time dominated by seek time and rotational latency
- First bit in a sector is the most expensive, the rest are “free”
- SRAM access time is about 4 ns/doubleword, DRAM about 60 ns
  - ~100,000 times longer to access a word on disk than in DRAM

Performance and page faults

- **First:** how often do they happen?
  - depends! (on workloads and memory sizes)
  - in most systems, very rare
  - scenario: random access to 4GB of VM with 2GB real memory
    - 50% of memory access will generate page faults

- **Second:** how long do they take?
  - usually, one disk access
  - lets say 10ms, for our scenario

- **So, how fast does the program go in the scenario?**
  - 100 pageFaults/second * 2 memoryAccesses/pageFault
  - 200 memory accesses per second

Disk storage as array of blocks

OS’s view of storage device
(as exposed by SCSI or IDE/ATA protocols)

- Common “logical block” size: 512 bytes
- Number of blocks: device capacity / block size
- Common OS-to-storage requests defined by few fields
  - R/W, block #, # of blocks, memory source/dest

Reminder: Page Faults

A page fault is caused by a reference to a VM word that is not in physical (main) memory

- Example: An instruction that triggers a page fault
  - Virtual address (5)
  - Valid bit (1)
  - PTE 0
    - Valid bit (1)
    - PTE 1
      - Virtual memory (disk)
        - Memory resident page table (DRAM)
          - PTE 2
            - Physical memory (DRAM)
              - Physical page number
                - Disk Sector
                  (usually same size as block)
Mapping file offsets to disk LBNs

- Issue in question
  - need to keep track of which LBNs hold which file data
- Most trivial mapping: just remember start location
  - then keep entire file in contiguous LBNs
  - what happens when it grows?
  - alternately, include a “next pointer” in each “block”
  - how does one find location of a particular offset?
- Most common approach: block lists
  - an array with one LBN per block in the file
  - Note: file block size can exceed one logical (disk) block
  - so, groups of logical blocks get treated as a unit by file system
  - e.g., 8KB = 16 disk blocks (of 512 bytes each)

Reminder: How the Unix Kernel Represents Open Files

- Two descriptors referencing two distinct open disk files. Descriptor 1 (stdout) points to terminal, and descriptor 4 points to open disk file

Disk Capacity

- Capacity: maximum number of bits that can be stored
  - Vendors express capacity in units of gigabytes (GB), where 1 GB = 10^9 Bytes (Lawsuit pending! Claims deceptive advertising)

Capacity is determined by these technology factors:

- Recording density (bits/in): number of bits that can be squeezed into a 1 inch linear segment of a track
- Track density (tracks/in): number of tracks that can be squeezed into a 1 inch radial segment
- Areal density (bits/in^2): product of recording and track density
Computing Disk Capacity

Capacity = (# bytes/sector) x (avg. # sectors/track) x (# tracks/surface) x (# surfaces/platter) x (# platters/disk)

Example:
- 512 bytes/sector
- 1000 sectors/track (on average)
- 20,000 tracks/surface
- 2 surfaces/platter
- 5 platters/disk

Capacity = 512 x 1000 x 80000 x 2 x 5
= 409,600,000,000
= 409.6 GB

Looking back at the hardware

Connecting I/O devices: the I/O Bus

Reading from disk (1)

Reading from disk (2)

Reading from disk (3)