LECTURE 3.2
DISKS

COP4600

Dr. Matthew Gerber

4/11/2016

DISKS

Disks

- Secondary storage devices represent both the effective bottom of the memory hierarchy (at least, for on-line storage purposes) and the most obvious I/O devices
- For practical purposes, secondary storage devices can now be thought of as disks
  - Tape devices are still in operation but fading quickly
- Again for practical purposes, we can divide disks into three types
  - Magnetic disks
  - Optical disks
  - Solid state disks
- Optical disks have the same overall characteristics as magnetic disks, but are simpler and slower
- Solid state disks are their own thing entirely

Magnetic Disks (10.1.1)

- Magnetic fixed disks are made up of a series of *platters*
  - The platters are divided into circular *tracks*
  - Tracks are in turn divided into *sectors*
  - The collection of tracks at the same distance from the center is a *cylinder*
- Whenever the disk is running, a drive motor spins it
  - For modern disks, this is typically somewhere between 5,400 and 15,000 RPM
- We used to make *sector addresses* out of all this, but…
  - Drive electronics are advanced enough that it doesn't really work any more
  - So we just think of the drive as one big array of bytes and let the drive's own controller worry about it
- Disks have *read-write heads* that float just above the surface of every platter, attached to a *disk arm* that moves all the heads at once
  - A cylinder can also be seen as all the tracks at a given arm position
- What do you think it's called if the read-write head malfunctions and contacts the platter?

Magnetic Disks (10.1.1)

- There are two important speed metrics for magnetic disks
- The *transfer rate* is the **effective** rate of data flow between the disk and the computer
  - Many disks include highly theoretical transfer rates on their packaging; **don't believe them**
  - High-performance consumer magnetic disks can top 150MB/s for fixed contiguous transfers!
- The *random access time* is equal to the *seek time* plus the *rotational latency*
  - The *seek time* is the time it takes to move the disk arm to the right cylinder
    - Real-world seek times have been 8-12ms for about the past 20 years
  - The *rotational latency* is the time it takes the right sector to rotate to the disk head
    - What is the average rotational latency for a 7200RPM drive?

Solid State Disks (10.1.2)

- Solid state disks use memory controllers rather than
- For SSDs, the random access time is obviously *far* faster than for magnetic disks
  - Typically on the order of 100 microseconds rather than on the order of 10 milliseconds
- PCI Express SSDs can reach transfer speeds measured in gigabytes per second
  - Note that this is still *much* less of a performance gain than for random access
- **Rough order-of-magnitude rules**
  - SSDs *seek* 100 to 200 times as fast as similar-class magnetic disks
  - SSDs *transfer* five to ten times as fast as similar-class magnetic disks
  - SSDs *cost* slightly less than 10 times as much as magnetic disks of similar class and size

I/O Buses and Attachment (10.3.1-3)

- Most disks in the personal computing context are *host attached*. These days, that usually means…
  - ATA in its various forms for magnetic disks and low-end SSDs
  - PCI Express for high-end SSDs
  - USB for low-end external drives
  - Thunderbolt for high-end external drives
- *Network attached storage* uses standard networking protocols to provide storage over the network.
  - This typically means NFS/CIFS network devices or shares
- *Storage Area Networks* act topologically like networks but are specialized for storage over general networking. Some examples are…
  - Fibre Channel
  - InfiniBand
- iSCSI bridges the NAS and SAN concepts, allowing SAN-like interconnects over commodity networks

Disk Scheduling (10.4)

- It's not uncommon for more than one I/O request to be pending
- What order should we service them in?
- It seems obvious, *but…*
- Recall that magnetic disks have **seek time**
- Think of a request as specifying at least:
  - Input versus output
  - Disk location of the transfer
  - Memory location of the transfer
  - Number of sectors to be transferred

Disk Scheduling: FCFS (10.4.1)

- Consider a disk queue with requests that map to cylinders:

  98, 183, 37, 122, 14, 124, 65, 67

- (No, we don't really use cylinders directly any more, but we can still make guesses based on the linear location of the data)
- In pure first-come, first-served scheduling, what is performance going to be like as we bounce around?
- Seek distances from 53:

  45→98, 85→183, 146→37, 85→122,
  108→14, 110→124, 59→65, 2→67

- Total: 640
- We're going to waste a lot of extra time seeking

Disk Scheduling: SSTF (10.4.2)

- In *shortest seek time first* scheduling, we move the shortest possible distance first
- Consider a disk queue with requests that map to cylinders:

  98, 183, 37, 122, 14, 124, 65, 67

- Seek distances from 53:

  12→65, 2→67, 30→37, 23→14,
  84→98, 24→122, 2→124, 59→183

- Total: 236
- *Much* better – but prone to starvation!

Disk Scheduling: Scanning (10.4.3)

- In the *SCAN* algorithm we act like an elevator. The head always moves from one end of the disk to the other, then back.
- Consider a disk queue with requests that map to cylinders:

  98, 183, 37, 122, 14, 124, 65, 67

- Seek distances from 53, moving down:

  16→37, 23→14, (14→0), 65→65, 2→67,
  31→98, 24→122, 2→124, 59→183

- Total: 236
- Same class performance as SSTF but no starvation

Disk Scheduling: Scanning Variants

- C-SCAN acts like SCAN but always starts over from 0 when it reaches the last cylinder
  - Adds one extra full seek to each cycle, but provides fairer wait times to processes unlucky enough to read from right “behind” the head’s current location
- LOOK acts like SCAN but checks the queue and doesn’t read past the last request in either direction
  - Would save 28 cylinders’ worth of seeking versus our SCAN example
- C-LOOK acts like LOOK but always starts over from the lowest-cylinder-numbered request in the queue when it reaches the highest-cylinder-numbered request

Disk Scheduling: SSDs

- None of this matters for SSDs, *except*:
- We combine adjacent reads and writes
- Lowers pure transactional latency

Disk Management, In Brief (10.5.1-10.5.2)

- *Low Level Formatting* is the process of writing the storage structures that order the disk at the sector level
- Usually, it is done once and rarely needs to be redone
- Partition tables and subsequently partitions are then laid out on to the disk, and *logically formatted*
- A partition at a specific location is typically designated as the *boot partition*, code from which is loaded by the chain of bootstrap programs

RAID ARRAYS

RAID (10.7)

- A *redundant array of independent disks* is a common method of chaining together disks for better performance, reliability, or both
- RAID arrays increase the *reliability* of data by storing it *redundantly* at some level
  - Can allow for up to 1-*n* drives to fail and retain the data, where *n* is the number of drives in the array
- RAID arrays increase the *performance* of data retrieval and manipulation by storing it *across multiple drives*
  - Can allow transfer speeds of up to approximately *n* times the speed of a single drive, where *n* is the number of drives in the array
- Reliability and performance are in tension here

RAID 0 and 1

- RAID 0 stripes data block-by-block across multiple disks
- With four disks, block 0 will be stored on disk 0, 1 on 1, 2 on 2, 3 on 3, 4 on 0, and so on
  - Write performance theoretically equal to the combined disks
  - Read performance theoretically equal to the combined disks
  - Severe *decrease* in reliability
  - Disk space of all disks combined
- RAID 1 mirrors data across multiple disks, writing identical data to each disk in the array
  - Write performance equal to a single disk
  - Read performance equal to RAID 0
  - All but 1 disk must fail for data to be lost
  - Disk space of a single disk

RAID 5 and 6

- Given *n* disks, RAID 5 stripes data across *n-1* disks and uses the final disk to store a parity block
- Which disk is the parity disk shifts block by block, to avoid bottlenecking parity writes to a single disk
  - Read performance theoretically equal to the combined disks, minus one
  - Write performance higher than a single disk, but bottlenecked compared to pure striping
  - Any *one* disk in the array can fail without data loss
  - Disk space of all disks combined, minus one
- RAID 6 does the same thing, but uses *two* disks for parity instead of one

NEXT TIME: FILES