Storage Systems

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Today’s Topics

- HDDs (Hard Disk Drives)
- Disk scheduling policies
- Linux I/O schedulers
Secondary Storage

- **Secondary storage usually**
  - is anything that is outside of “primary memory”.
  - does not permit direct execution of instructions or data retrieval via machine load/store instructions.

- **Characteristics**
  - It’s large: 100GB and more
  - It’s cheap: 1TB SATA2 disk costs ₩66,000.
  - It’s persistent: data survives power loss.
  - It’s slow: milliseconds to access.
HDDs (1)

Electromechanical
- Rotating disks
- Arm assembly

Electronics
- Disk controller
- Buffer
- Host interface
HDDs (2)

- **Seagate Barracuda ST31000528AS (1TB)**
  - 4 Heads, 2 Discs
  - Max. recording density: 1413K BPI (bits/inch)
  - Avg. track density: 236K TPI (tracks/inch)
  - Avg. areal density: 329 Gbits/sq.inch
  - Spindle speed: 7200rpm (8.3ms/rotation)
  - Average seek time: < 8.5ms (read), < 9.5ms (write)
  - Max. internal data transfer rate: 1695 Mbits/sec
  - Max. I/O data transfer rate: 300MB/sec (SATA-2)
  - Max. sustained data transfer rate: 125MB/sec
  - Internal cache buffer: 32MB
  - Max power-on to ready: < 10.0 sec
HDDs (3)

- Hard disk internals

- Our Boeing 747 will fly at the altitude of only a few mm at the speed of approximately 65mph periodically landing and taking off.
- And still the surface of the runway, which consists of a few mm-thick layers, will stay intact for years.
Managing Disks (1)

- Disks and the OS
  - Disks are messy physical devices:
    - Errors, bad blocks, missed seeks, etc.
  - The job of the OS is to hide this mess from higher-level software.
    - Low-level device drivers (initiate a disk read, etc)
    - Higher-level abstractions (files, databases, etc.)
  - The OS may provide different levels of disk access to different clients.
    - Physical disk block (surface, cylinder, sector)
    - Disk logical block (disk block #)
    - Logical file (filename, block or record or byte #)
### Interacting with disks

- Specifying disk requests requires a lot of info:
  - Cylinder #, surface #, track #, sector #, transfer size, etc.
- Older disks required the OS to specify all of this
  - The OS needs to know all disk parameters.
- Modern disks are more complicated.
  - Not all sectors are the same size, sectors are remapped, etc.
- Current disks provide a higher-level interface (e.g., SCSI)
  - The disks exports its data as a logical array of blocks [0..N-1]
  - Disk maps logical blocks to cylinder/surface/track/sector.
  - Only need to specify the logical block # to read/write.
  - As a result, physical parameters are hidden from OS.
Managing Disks (3)

▪ Disk performance
  • Performance depends on a number of steps
    – Seek: moving the disk arm to the correct cylinder
      → depends on how fast disk arm can move (increasing very slowly)
    – Rotation: waiting for the sector to rotate under head
      → depends on rotation rate of disk (increasing, but slowly)
    – Transfer: transferring data from surface into disk controller, sending it back to the host.
      → depends on density of bytes on disk (increasing, and very quickly)
  
▪ Disk scheduling:
  – Because seeks are so expensive, the OS tries to schedule disk requests that are queued waiting for the disk.
FCFS

- **FCFS (= do nothing)**
  - Reasonable when load is low.
  - Long waiting times for long request queues.

```
queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
```
SSTF

- Shortest seek time first
  - Minimizes arm movement (seek time)
  - Maximizes request rate
  - Unfairly favors middle blocks
  - May cause starvation of some requests

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
### Elevator algorithm

- Service requests in one direction until done, then reverse
- Skews wait times non-uniformly

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

Head starts at 53
**Circular SCAN**

- Like SCAN, but only go in one direction (e.g. typewriters)
- Uniform wait times

```
queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
```
LOOK / C-LOOK

- LOOK / C-LOOK
  - Similar to SCAN/C-SCAN, but the arm goes only as far as the final request in each direction.

<table>
<thead>
<tr>
<th>queue</th>
<th>98, 183, 37, 122, 14, 124, 65, 67</th>
</tr>
</thead>
<tbody>
<tr>
<td>head starts at 53</td>
<td></td>
</tr>
</tbody>
</table>

0 14 37 53 65 67 98 122 124 183 199

C-LOOK
Disk Scheduling (1)

- Selecting a disk scheduling algorithm
  - SSTF is common and has a natural appeal.
  - SCAN and C-SCAN perform better for systems that place a heavy load on the disk.
  - Either SSTF or LOOK is a reasonable choice for the default algorithm.
  - Performance depends on the number and types of requests.
  - Requests for disk service can be influenced by the file allocation method.
  - In general, unless there are request queues, disk scheduling does not have much impact.
    - Important for servers, less so for PCs
  - Modern disks often do the disk scheduling themselves.
    - Disks know their layout better than OS, can optimize better.
    - Ignores, undoes any scheduling done by OS.
Modern Disks

- **Intelligent controllers**
  - A small CPU + many kilobytes of memory.
  - They run a program written by the controller manufacturer to process I/O requests from the CPU and satisfy them.
  - Intelligent features:
    - Read-ahead: the current track
    - Caching: frequently-used blocks
    - Command queueing
    - Request reordering: for seek and/or rotational optimality
    - Request retry on hardware failure
    - Bad block/track identification
    - Bad block/track remapping: onto spare blocks and/or tracks
I/O Schedulers

- I/O scheduler’s job
  - Improve overall disk throughput
    - Merging requests to reduce the number of requests
    - Reordering and sorting requests to reduce disk seek time
  - Prevent starvation
    - Submit requests before deadline
    - Avoid read starvation by write
  - Provide fairness among different processes
  - Guarantee quality-of-service (QoS) requirement
Linux I/O Schedulers (1)

- Linus elevator scheduler
  - Merges adjacent I/O requests
  - Sorts I/O requests in ascending block order
  - Writes-starving-reads:
    - Stop insertion-sorting if there is a sufficiently old request in the queue
  - Trades fairness for improved global throughput
  - Not really an elevator: puts requests with a low sector number at the top of the queue regardless of the current head position
  - Real-time?
  - Was the default I/O scheduler in Linux 2.4
Linux I/O Schedulers (2)

- **Deadline scheduler**
  - Two standard Read/Write sorted queues (by LBA) + Two Read/Write FIFO queues (by submission time)
  - Each FIFO queue is assigned an expiration value.
    - Read: 500 msec
    - Write: 5 sec
  - Normally, send I/O requests from the head of the standard sorted queue.
  - If the request at the head of one of the FIFO queues expires, services the FIFO queue
  - Emphasizes average read request response time
  - No strict guarantees over request latency
Linux I/O Schedulers (3)

- **Anticipatory scheduler**
  - Considers “deceptive idleness”
    - Process A is about to issue next request, but scheduler hastily assumes that process A has no further requests!
  - Adds an anticipation heuristic:
    - Sometimes wait for process whose request was last serviced.
  - Was the default I/O scheduler in the 2.6 kernel
  - Dropped in the Linux Kernel 2.6.33 in favor of the CFQ scheduler
Linux I/O Schedulers (4)

- **Noop scheduler**
  - Minimal overhead I/O scheduler
  - Only performs merging

- For random-access devices such as RAM disks and solid state drives (SSDs)
- For storage with intelligent HBA or externally attached controller (RAID, TCQ drives)
Linux I/O Schedulers (5)

- **CFQ (Complete Fair Queuing) scheduler**
  - Assigns incoming I/O requests to specific queues based on the process originating the I/O request
    - Within each queue, requests are coalesced with adjacent requests and insertion sorted
    - Service the queues round robin, plucking a configurable number of requests (by default, four) from each queue before continuing on to the next
  - Fairness at a per-process level
  - Initially for multimedia applications, but works well across many workloads
  - Subsumes anticipatory scheduling
  - New default scheduler for Linux 2.6 (from 2.6.18)