Storage Hierarchy III: I/O System

- often boring, but still quite important
  - ostensibly about general I/O, mainly about disks
- performance: latency & throughput
- disks
  - parameters
  - extensions
  - redundancy and RAID
- buses
- I/O system architecture
  - DMA and I/O processors
- current research in I/O systems

I/O (Disk) Performance

- who cares? you do
  - remember Amdahl’s Law
  - want fast disk access (fast swap, fast file reads)
- I/O performance metrics
  - bandwidth of requests: I/Os per second (IOPS)
  - raw data bandwidth: bytes per second
  - latency: response time
- is I/O (disk) latency important? why not just context-switch?
  - context-switching isn’t fast (although faster than disk access)
  - context-switching requires jobs to context-switch to
  - context-switching annoys users (productivity = f(1/response time))

Readings

H+P
- chapter 7 (note that we’ve temporarily skipped chapter 6)

Readings in Computer Architecture
- [optional!] Smotherman: “A Sequencing-Based Taxonomy of I/O Systems and Review of Historical Machines”
- Patterson, Gibson, and Katz: “A Case for Redundant Arrays of Inexpensive Disks (RAID)”

Recent Research Paper
- Buonadonna and Culler: “QPIP”

I/O Device Characteristics

- type
  - input: read only
  - output: write only
  - storage: both
- partner
  - human
  - machine
- data rate
  - peak transfer rate

<table>
<thead>
<tr>
<th>device</th>
<th>type</th>
<th>partner</th>
<th>data rate KB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>mouse</td>
<td>I</td>
<td>human</td>
<td>0.01</td>
</tr>
<tr>
<td>CRT</td>
<td>O</td>
<td>human</td>
<td>60,000</td>
</tr>
<tr>
<td>modem</td>
<td>I/O</td>
<td>machine</td>
<td>2-8</td>
</tr>
<tr>
<td>LAN</td>
<td>I/O</td>
<td>machine</td>
<td>500-6000</td>
</tr>
<tr>
<td>tape</td>
<td>storage</td>
<td>machine</td>
<td>2000</td>
</tr>
<tr>
<td>disk</td>
<td>storage</td>
<td>machine</td>
<td>2000-10,000</td>
</tr>
</tbody>
</table>
Disk Parameters

- 1–20 platters (data on both sides)
  - magnetic iron-oxide coating
  - 1 read/write head per side
- 500–2500 tracks per platter
- 32–128 sectors per track
  - sometimes fewer on inside tracks
- 512–2048 bytes per sector
  - usually fixed number of bytes/sector
  - data + ECC (parity) + gap
- 4–24GB total
- 3000–10000 RPM

Disk Performance

\[ t_{\text{disk}} = t_{\text{seek}} + t_{\text{rotation}} + t_{\text{transfer}} + t_{\text{controller}} + t_{\text{queuing}} \]

- \( t_{\text{seek}} \) (seek time): move head to track
- \( t_{\text{rotation}} \) (rotational latency): wait for sector to come around
  - average \( t_{\text{rotation}} = \frac{0.5}{\text{RPS}} \) // (RPS = RPM / 60)
- \( t_{\text{transfer}} \) (transfer time): read disk
  - \( \text{rate}_{\text{transfer}} = \frac{\text{bytes/sector} \times \text{sector/track} \times \text{RPS}}{} \)
  - \( t_{\text{transfer}} = \frac{\text{bytes transferred}}{\text{rate}_{\text{transfer}}} \)
- \( t_{\text{controller}} \) (controller delay): wait for controller to do its thing
- \( t_{\text{queuing}} \) (queueing delay): wait for older requests to finish
  - not a fixed latency - depends on older requests

Disk Performance Example

- parameters
  - 3600 RPM \( \Rightarrow 60 \text{ RPS} \) (may help to think in units of tracks/sec)
  - avg seek time: 9ms
  - 100 sectors per track, 512 bytes per sector
  - controller + queuing delays: 1ms
- Q: average time to read 1 sector (512 bytes)?
  - \( \text{rate}_{\text{transfer}} = 100 \text{ sectors/track} \times 512 \text{ B/sector} \times 60 \text{ RPS} = 2.4 \text{ MB/s} \)
  - \( t_{\text{transfer}} = \frac{512 \text{ B}}{2.4 \text{ MB/s}} = 0.2\text{ms} \)
  - \( t_{\text{rotation}} = \frac{0.5}{60 \text{ RPS}} = 8.3\text{ms} \)
  - \( t_{\text{disk}} = 9\text{ms (seek)} + 8.3\text{ms (rotation)} + 0.2\text{ms (xfer)} + 1\text{ms} = 18.5\text{ms} \)
  - \( t_{\text{transfer}} \) is only a small component! counter-intuitive?
  - end of story? no! \( t_{\text{queuing}} \) not fixed (gets longer with more requests)

Disk Performance: Queuing Theory

- I/O is a queuing system
  - in equilibrium: \( \text{rate}_{\text{arrival}} = \text{rate}_{\text{departure}} \)
  - total time \( t_{\text{system}} = t_{\text{queue}} + t_{\text{server}} \)
  - Little’s Law: \( \text{rate}_{\text{arrival}} \times t_{\text{system}} = \text{QueueLength}_{\text{system}} \)
  - LL corollary: \( \text{rate}_{\text{arrival}} \times t_{\text{server}} = \text{utilization}_{\text{server}} \)
- the important result (derivation in H+P)
  - \( t_{\text{queue}} = \frac{t_{\text{server}} \times \text{utilization}_{\text{server}}}{(1 - \text{utilization}_{\text{server}})} \)
  - \( t_{\text{system}} = \frac{t_{\text{server}}}{(1 - \text{utilization}_{\text{server}})} \)
  - if server highly utilized, \( t_{\text{system}} \) gets VERY HIGH
  - lesson: keep utilization low (below 75%)
Disk Usage Models

- data mining + supercomputing
  - large files, sequential reads
  - raw data transfer rate ($rate_{transfer}$) is most important
- transaction processing
  - large files, but random access, many small requests
  - IOPS is most important
- time sharing filesystems
  - small files, sequential accesses, potential for file caching
  - IOPS is most important

Must design disk (I/O) system based on target workload
- use disk benchmarks (they exist)

Disk Alternatives

- solid state disk (SSD)
  - DRAM + battery backup with standard disk interface
    + fast: no seek time, no rotation time, fast transfer rate
      - expensive
  - FLASH memory
    + fast: no seek time, no rotation time, fast transfer rate
    + non-volatile
      - slow
      - “wears” out over time
  - optical disks (CDs)
    - cheap if write-once, expensive if write-multiple
      - slow

Extensions to Conventional Disks

- increasing density: more sensitive heads, finer control
  - increases cost
- fixed head: head per track
  + seek time eliminated
  - low track density
- parallel transfer: simultaneous read from multiple platters
  - difficulty in looking onto different tracks on multiple surfaces
  - lower cost alternatives possible (disk arrays)

More Extensions to Conventional Disks

- disk caches: disk-controller RAM buffers data
  + fast writes: RAM acts as a write buffer
  + better utilization of host-to-device path
    - high miss rate increases request latency
- disk scheduling: schedule requests to reduce latency
  - e.g., schedule request with shortest seek time
  - e.g., “elevator” algorithm for seeks (head sweeps back and forth)
  - works best for unlikely cases (long queues)
Disk Arrays

- collection of individual disks ($D = \# \text{ disks}$)
  - distribute data across disks
  - access in parallel for higher b/w (IOPS)
  - issue: data distribution $\Rightarrow$ load balancing
  - e.g., 3 disks, 3 files (A, B, C): each 2 sectors long (e.g., A0 & A1)

```
A0  B0  C0
A1  B1  C1
```

undistributed                 coarse-grain striping                fine-grain striping

Disk Arrays: Stripe Width

- fine-grain striping
  - $D \times \text{stripe width}$ evenly divides smallest accessible data (sector)
  - only one request served at a time (why?)
    - perfect load balance
    - effective transfer rate approx $D$ times better than single disk
  - access time can go up, unless disks synchronized (disk skew)
  - coarse-grain striping
    - data transfer parallelism for large requests
    - concurrency for small requests (several small requests at once)
    - “statistical” load balance

must consider workload to determine stripe width

Disk Redundancy and RAIDs

- disk failures are a significant fraction of all hardware failures
  - electrical failures are rare, but mechanical failures more common
- striping increases number of files touched by failure
- fix with replication and/or parity protection
- **RAID**: redundant array of inexpensive disks [Patterson+87]
  - arrays of cheap disks provide high performance + reliability
  - $D = \# \text{ data disks}, C = \# \text{ check disks}$
- 6 levels of RAID depend on redundancy/concurrency
  - level 1: full mirroring ($D := C$)
  - level 3: bit-interleaved parity (e.g., $D=8, C=1$)
  - level 6: two-dimensional error bits (e.g., $D=8, C=2$)

I/O System Architecture

- buses
  - memory bus
  - I/O bus
- I/O processing
  - program controlled
  - DMA
  - I/O processors (IOPs)
Bus Issues (Memory & I/O Buses)

- **clocking**: is bus clocked?
  - synchronous: clocked, short bus ⇒ fast
  - asynchronous: no clock, use “handshaking” instead ⇒ slow
- **switching**: when is control of bus acquired and released?
  - atomic: bus held until request complete ⇒ slow
  - split-transaction (pipelined): bus free between request & reply ⇒ fast
- **arbitration**: how do we decide who gets the bus next?
  - overlap arbitration for next master with current transfer
  - daisy chain: closer devices have priority ⇒ slow
  - distributed: wired-OR, low-priority back-off ⇒ medium
- some other issues
  - split data/address lines, width, burst transfer

I/O and Memory Buses

<table>
<thead>
<tr>
<th>memory buses</th>
<th>bits</th>
<th>MHz</th>
<th>peak MB/s</th>
<th>special features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summit</td>
<td>128</td>
<td>60</td>
<td>960</td>
<td></td>
</tr>
<tr>
<td>Challenge</td>
<td>256</td>
<td>48</td>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>XDBus</td>
<td>144</td>
<td>66</td>
<td>1056</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I/O buses</th>
<th>bits</th>
<th>MHz</th>
<th>peak MB/s</th>
<th>special features</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISA</td>
<td>16</td>
<td>8</td>
<td>16</td>
<td>original PC bus</td>
</tr>
<tr>
<td>IDE</td>
<td>16</td>
<td>8</td>
<td>16</td>
<td>tape, CD-ROM</td>
</tr>
<tr>
<td>PCI</td>
<td>32(64)</td>
<td>33(66)</td>
<td>133(266)</td>
<td>“plug+play”</td>
</tr>
<tr>
<td>SCSI/2</td>
<td>8/16</td>
<td>5/10</td>
<td>10/20</td>
<td>high-level interface</td>
</tr>
<tr>
<td>PCMCIA</td>
<td>8/16</td>
<td>8</td>
<td>16</td>
<td>modem, “hot-swap”</td>
</tr>
<tr>
<td>USB</td>
<td>serial</td>
<td>isoch.</td>
<td>1.5</td>
<td>power line, packetized</td>
</tr>
<tr>
<td>FireWire</td>
<td>serial</td>
<td>isoch.</td>
<td>100</td>
<td>fast USB</td>
</tr>
</tbody>
</table>

Who Does I/O?

- **main CPU**
  - explicitly executes all I/O operations
  - high overhead, potential cache pollution problem
  + no cache coherence problems
- **I/O Processor (IOP or channel processor)**
  - (special or general) processor dedicated to I/O operations
  + fast
  - may be overkill, cache coherence problems
- **DMAC (direct memory access controller)**
  - can transfer data to/from memory given start address (but that’s all)
  + fast, usually simple
  - still may be coherence problems, must be on memory bus

Communicating with DMAC/IOP

- not an issue if main CPU performs I/O by itself
- **I/O control**: how to initialize DMAC/IOP?
  - memory mapped: ld/st to preset, VM-protected addresses
  + privileged I/O instructions
- **I/O completion**: how does CPU know DMAC/IOP is finished?
  - polling: periodically check status bit ⇒ slow
  - interrupt: I/O completion interrupts CPU ⇒ fast
- Q: do DMAC/IOP use physical or virtual addresses?
  - physical: simpler, but can only transfer 1 page at a time (why?)
  + virtual: more powerful, but DMAC/IOP needs TLB
I/O System Example

- given
  - 500 MIPS CPU
  - 16B wide, 100 ns memory system
  - 10000 instrs per I/O
  - 16KB per I/O
  - 200 MB/s I/O bus, with room for 20 SCSI-2 controllers
  - SCSI-2 strings–20MB/s with 15 disks per bus
  - SCSI-2 1ms overhead per I/O
  - 7200 RPM (120 RPS), 8ms avg seek, 6MB/s transfer disks
  - 200GB total storage

- Q: choose 2GB or 8GB disks for maximum IOPS?
  - how to arrange disks and controllers?

I/O System Example (cont’d)

- step 1: calculate CPU, memory, I/O bus peak IOPS
  - CPU: 500 MIPS/ (10000 instructions/I/O) = 50000 IOPS
  - memory bus: (16-bytes / 100ns) / 16KB = 10000 IOPS
  - I/O bus: (200MB/s) / 16KB = 12500 IOPS

- memory bus (10000 IOPS) is the bottleneck!

- step 2: calculate disk IOPS
  - \( t_{\text{disk}} = 8\text{ms} + 0.5 / 120 \text{ RPS} + 16\text{BK} / (6\text{MB/s}) = 15\text{ms} \)
  - disk: 1 / 15ms = 67 IOPS
  - 8GB disks \( \Rightarrow \) need 25 \( \Rightarrow \) 25 * 67 IOPS = 1675 IOPS
  - 2GB disks \( \Rightarrow \) need 100 \( \Rightarrow \) 100 * 67 IOPS = 6700 IOPS
  - 100 2GB disks (6700 IOPS) disks are a new bottleneck!

- answer.I: 100 2GB disks!

I/O System Example (cont’d)

- step 3: calculate SCSI-2 controller peak IOPS
  - \( t_{\text{SCSI-2}} = 1\text{ms} + 16\text{KB} / (20\text{MB/s}) = 1.8\text{ms} \)
  - SCSI-2: 1 / 1.8ms = 556 IOPS

- step 4: how many disks per controller?
  - 556 IOPS / 67 IOPS = 8 disks per controller

- step 5: how many controllers?
  - 100 disks / 8 disks/controller = 13 controllers

- answer.II: 13 controllers, 8-disks each

New: Integrating I/O into Unified SAN

- I/O bottleneck is often the OS
  - how can we keep the OS involvement to a minimum?
  - user-level DMA (also called remote DMA or RDMA)

- VIA: Virtual Interface Architecture
  - describes system area network (SAN)
  - abstract model: processor has queues of requests/responses
  - OS only involved to set up queues

- Infiniband
  - another SAN specification for user-level RDMA
  - like VIA, might be DOA

- QPIP (read the paper!) tries to solve VIA/Infiniband problems
Summary

- disks
  - parameters
  - performance ($t_{queue}$ gets worse as utilization increases)
  - RAID
- buses
  - I/O vs. memory
- I/O system architecture
  - CPU vs. DMAC vs. IOP
- current research: SANs with user-level DMA

next up: multithreading and multiprocessing