I/O Scheduling

Mechanisms

- Reordering
  - don't (necessarily) issue the I/Os in arrival order
  - SATF, SSTF, SCAN, CSCAN, etc.
- Merging
  \[ \{ s_n, s_{n+1} \} \]
- Delaying
  - non-work-conserving schedulers

Anticipation

- synchronous: wait for completion before next issue
- sequential: contiguous sector addresses
- thinktime: time from completion → issue

Anticipation problem: deceptive idleness
Anticipation

• good performance if:
  \[ t_{\text{antic}} < t_{\text{seek}} \text{ (average)} \]

• huge performance hit if we get it wrong!

• Iyer and Druschel (SOSP'01)
  *Anticipatory scheduling: A disk scheduling framework to
    overcome deceptive idleness in synchronous I/O*

  - Apache: +29% and +71%
  - AFS: +8%
  - TPC-B: +2–60%

Plugging

• Merging: combine sector-adjacent requests

• What happens if the queue is empty?

  \[ s_n \quad s_{n+1} \]

  missed merge opportunity!

• Plugging: delay new I/O streams

Fair Queueing

• Goal: distribute disk resources “fairly” to users and/or processes

• Proportional bandwidth
  - a disk is not a fixed-bandwidth device!
  - available bandwidth depends on access pattern
  - result: poor utilisation

• Proportional time
  - allocate disk in time-slices (c.f. CPU scheduling)
  - e.g. Linux’s CFQ

CFQ

• Completely “Fair” Queueing
  - Jens Axboe, 2003
  - currently at v3

• one queue per thread
  - rb-tree sorted by sector
  - service each queue round-robin
  - 100ms default time-slice
**Reads vs. Writes**

- **Reads:**
  - Usually synchronous
  - Sometimes waiting for the result
  - Latency sensitive!

- **Writes:**
  - Usually asynchronous
  - Coming from the buffer-cache

- Prefer reads

**Solution:**

- Handle reads and writes separately
- Issue `x` reads for every write

**Problem:**

- Lost spatial locality!

**Solution:**

- Batching

- As: issue reads for 500ms, writes for 125ms
- Deadline: 2 read batches for each write batch
- 16 requests per batch

- CFQ: Lump all writes in a single queue

**I/O Priorities**

- Some threads are more equal than others
- Scale time-slice length and scheduling frequency

- **CFQ**
  - Real-time
  - Best-effort
  - Idle

- Deceptive idleness strikes again!!!

**Solution:**

- Priority "grace period"
- Idle `x` ms before switching to lower priority

- CFQ:
  - Wait 100ms before servicing idle class
  - Anticipate even for non-sequential I/O

- Idle class: You must be root – why?
Hardware

Write-back Buffering
- reduce perceived latency
- overlap disk access with bus traffic
- disk can schedule
- lose your data with well-timed power loss

- can issue many buffered-writes per time-slice
- writes can “escape” the time-slice
- solution: don’t use write-back!
- CFQ: limit #writes per slice
- buffer flush

- no completion events
  - when can we try again?
- solution: disable write-back!
- “adversary effect”

TCQ: Tagged Command Queueing
- overlap disk and bus access
- disk can schedule
- completion events!
  - no adversary effect
  - explicit flush unnecessary
- problem: can issue many I/Os in a short period

RAID
- is I/O scheduling useful?
- properties:
  - black box
  - huge queues
- seek-limiting: no gain
- merging: very important!
- fairness
- anticipation: tiny arrays only
- key tradeoff: queue depth
Solid State Drives
... or, why nobody cares about I/O scheduling

- huge IOPS
- locality (mostly) irrelevant

scheduling SSDs:
- ???