Advanced disk scheduling

“Freeblock scheduling”

Eno Thereska

(slide contributions by Chris Lumb and
Brandon Salmon)

PARALLEL DATA LABORATORY
Carnegie Mellon University

Outline

- Freeblock scheduling: some theory
- Freeblock scheduling: applied
- Some details
- Q & A

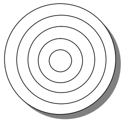
Some theory: preview

- Next few slides will review & show that:
  - disks are slow
    - mechanical delays (seek + rotational latencies)
  - there is nothing we can do during a seek
  - **there is a lot we can do during a rotation**
    - rotational latencies are very large
    - while rotation is happening go to nearby tracks and do useful work
  - “*freeblock scheduling*” = utilization of rotational latency gaps (+ any idle time)

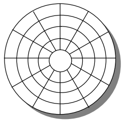
Are disks slow?

- Are the xfer speeds that slow?
  - no, xfer speeds of 200MB/s are pretty good
- So what is slow?
  - workload often not sequential
  - disk head has to move from place to place
  - seek (~ 4ms) + rotation (~ 3ms)
- Effective bandwidth can be very low
  - ~ 10-30MB/s
  - even when SPTF is used

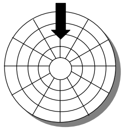
Surface organized into tracks



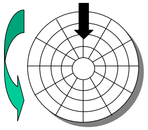
Tracks broken up into sectors



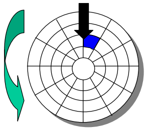
Disk head position



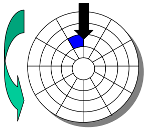
Rotation is counter-clockwise



About to read blue sector

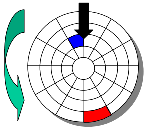


After reading blue sector

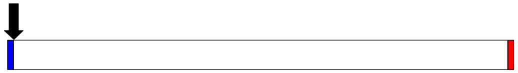


After BLUE read

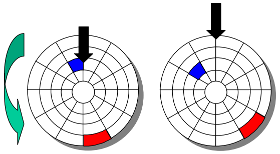
Red request scheduled next



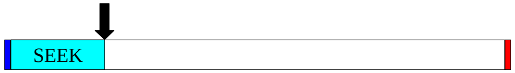
After BLUE read

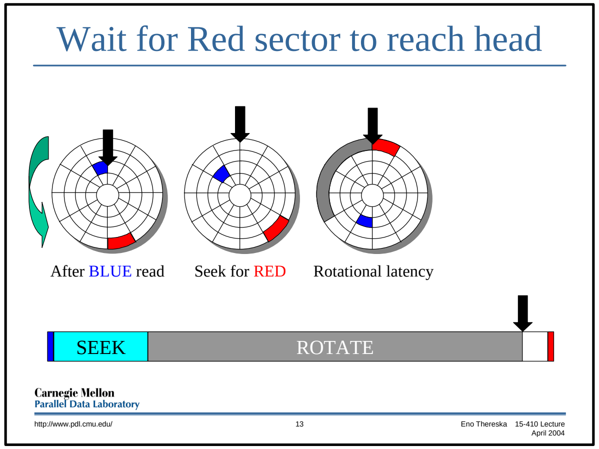


Seek to Red’s track



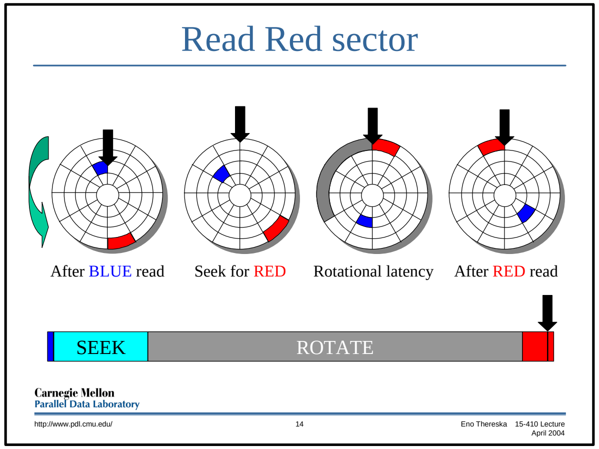
After BLUE read Seek for RED





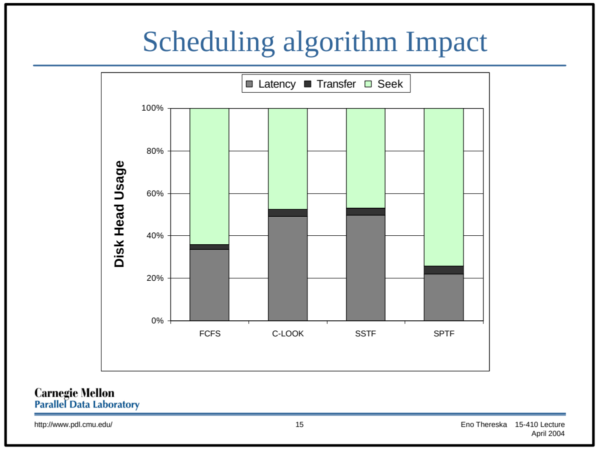
Wait for Red sector to reach head

Carnegie Mellon
Parallel Data Laboratory



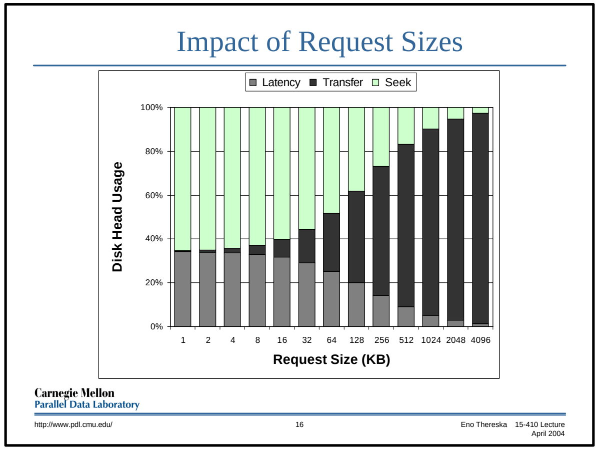
Read Red sector

Carnegie Mellon
Parallel Data Laboratory



Scheduling algorithm Impact

Carnegie Mellon
Parallel Data Laboratory



Impact of Request Sizes

Carnegie Mellon
Parallel Data Laboratory

What can we do?

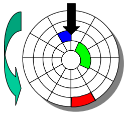
- Nothing we can do during a seek
  - disk head has to move to the right track
- Rotational latency is fully wasted
  - let's use this latency

- During a rotational latency
  - go to nearby tracks and do useful work
  - then, just-in-time, seek back to the original request

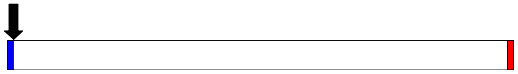
A quick glance ahead…

- What kind of "useful work" are we doing?
  - work that belongs to a "background" app
  - things like backup, defrag, virus scanning
- What do we really gain?
  - background apps don't interfere with fore. apps
  - background apps still complete
- What's in it for me?
  - can run defrag + virus scanner + backup in the background while working on your homework and you won't notice they are running ☺

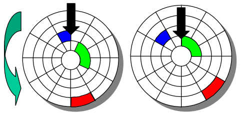
Rotational latency gap utilization



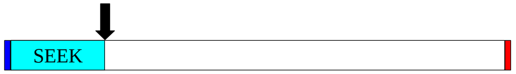
After BLUE read



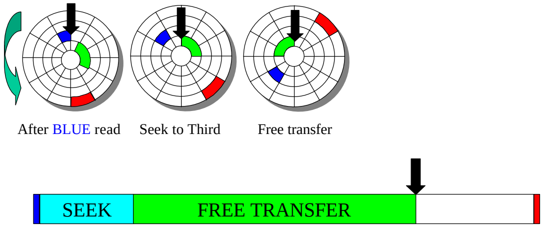
Seek to Third track



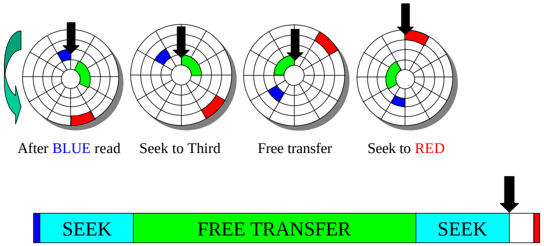
After BLUE read Seek to Third



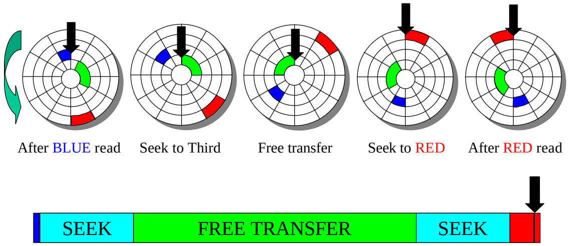
Free transfer



Seek to Red's track



Read Red sector

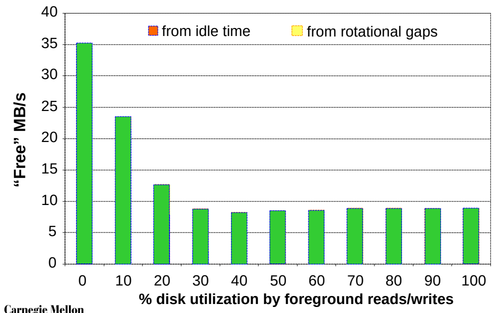


Final theory details

- Scheduler also uses disk idle time
  - high end servers have little idle time
- Idle time + rotational latency usage = "*freeblock scheduling*"

(it means we are getting things for free)

Steady background I/O progress



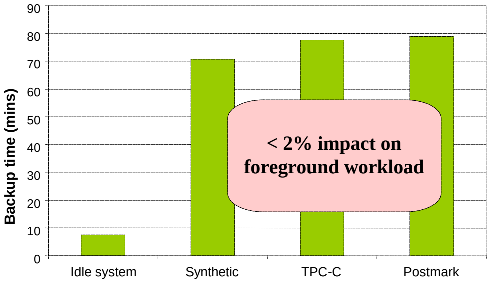
Applied freeblocks: preview

- Next few slides will show that:
  - we can build background apps
    - that do not interfere with foreground apps
    - that complete eventually
    - things like *backup*, *defrag*, *virus scanners*, etc
    - imagine the possibilities…

App 1: Backup

- Frequent backup improves data reliability and availability
  - companies take very frequent backups
  - a backup every 30 mins is not uncommon
- Our experiment:
  - disk used is 18GB
  - we want to back up 12GB of data
  - goal: back it up for free

Backup completed for free

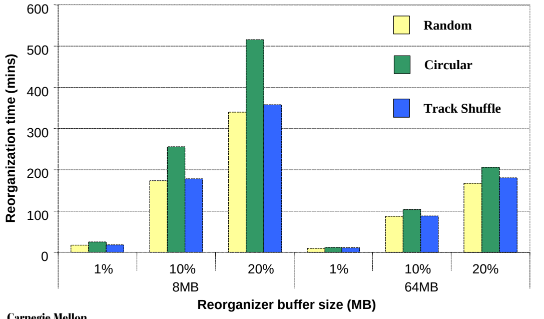


< 2% impact on foreground workload

App 2: Layout reorganization

- Layout reorganization improves access latencies
  - defragmentation is a type of reorganization
  - typical example of background activity
- Our experiment:
  - disk used is 18GB
  - we want to defrag up to 20% of it
  - goal: defrag for free

Disk Layout Reorganized for Free!



Other maintenance applications

- Virus scanner
- LFS cleaner
- Disk scrubber
- Data mining
- Data migration

Summary

- Disks are slow
  - but we can squeeze extra bw out of them
- Use freeblock scheduling to extract free bandwidth
- Utilize free bandwidth for background applications
  - they still complete eventually
  - with no impact on foreground workload

Details: preview (extra slides)

- Next few slides will show that:
  - it's hard to do fine grained scheduling at the device driver
  - background apps need new interfaces to express their desires to the background scheduler
  - what if background apps want to read/write to files (APIs talk in LBNs, remember)?
  - recommended reading material

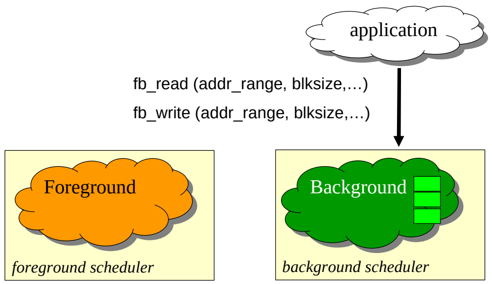
Ahh, the details…

- Hard to do at the device driver
  - need to know the position of the disk head
  - however, we have done it!
  - it's more efficient inside the disk drive
    - try to convince your disk vendor to put it in
- Efficient algorithms
  - SPTF for foreground (0.5% of 1GHz PIII)
  - Freeblock scheduling for background (<<8% of 1GHz PIII)
  - Small memory utilization

Application programming interface (API) goals

- Work exposed but done opportunistically
  - all disk accesses are asynchronous
- Minimized memory-induced constraints
  - late binding of memory buffers
  - late locking of memory buffers
- "Block size" can be application-specific
- Support for speculative tasks
- Support for rate control

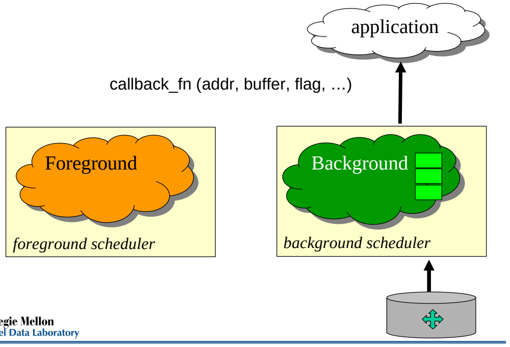
API description: task registration



fb_read (addr_range, blksize,…)

fb_write (addr_range, blksize,…)

API description: task completion



application

callback_fn (addr, buffer, flag, …)

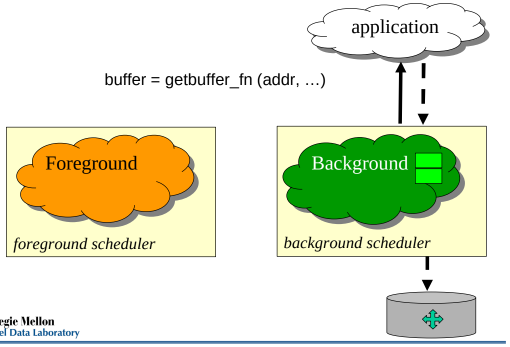
Foreground

foreground scheduler

Background

background scheduler

API description: late locking of buffers



application

buffer = getbuffer_fn (addr, …)

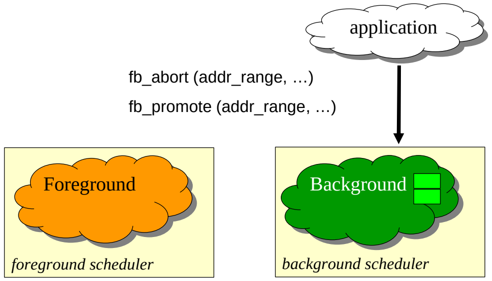
Foreground

foreground scheduler

Background

background scheduler

API description: aborting/promoting tasks



application

fb_abort (addr_range, …)

fb_promote (addr_range, …)

Foreground

foreground scheduler

Background

background scheduler

Complete API

Function Name	Arguments	Description
fb_open	*priority, callback_fn, getbuffer_fn*	Open a freeblock session (ret: *session_id*)
fb_close	*session_id*	Close a freeblock session
fb_read	*session_id, addr_range, blksize, callback_param*	Register a freeblock read task
fb_write	*session_id, addr_range, blksize, callback_param*	Register a freeblock write task (ret: *task_id*)
fb_abort	*session_id, addr_range*	Abort parts of registered task
fb_promote	*session_id, addr_range*	Promote parts of registered task
fb_suspend	*session_id*	Suspend scheduling of a session's tasks
fb_resume	*session_id*	Resume scheduling of a session's tasks
**(callback_fn)*	*session_id, addr, buffer, flags, callback_param*	Report that part of task completed
**(getbuffer_fn)*	*session_id, addr, callback_param*	Get memory address for selected write

Designing disk maintenance applications

- APIs talk in terms of logical blocks (LBNs)
- Some applications need structured version
  - as presented by file system or database
- Example consistency issues
  - application wants to read file “foo”
  - registers task for inode’s blocks
  - by time blocks read, file may not exist anymore!

Designing disk maintenance applications

- Application does not care about structure
  - scrubbing, data migration, array reconstruction
- Coordinate with file system/database
  - cache write-backs, LFS cleaner, index generation
- Utilize snapshots
  - backup, background *fsck*

Q & A

- See http://www.pdl.cmu.edu/Freeblock/ for more details on freeblocks
- Recommended reading :
  - Background fsck (describes snapshots)

    search for “M. K. McKusick. Running ’fsck’ in the background. BSDCon Conference, 2002”
  - IO-Lite (describes a unified buffer system)

    search for “V. S. Pai, P. Druschel, and W. Zwaenepoel. IO-Lite: a unified I/O buffering and caching system. Symposium on Operating Systems Design and Implementation 1998”