Virtual Log Based File Systems for a Programmable Disk

Randy Wang
Tom Anderson
Dave Patterson

Princeton University
University of Washington, Seattle
University of California, Berkeley

February, 1999

Question

What's the minimal latency of writing a random 4K block to a disk surface?

Naive Answers

- Back of envelope calculation
  - *~20* µ*s* per sector
  - Scales with ***product*** of density and rotation
- Conventional wisdom
  - ***x ms***
  - Limited by ***mechanical*** delays and overhead

Traditional Approaches

- Small writes on traditional file systems
  - UFS: suffers from head movement
  - LFS: batches small writes and unsafe
  - NVRAM: costly & lacks sustained bandwidth
- Traditional storage systems
  - Host FS & disk controller guessing each other
  - Limited vocabulary between the two
  - Poor scaling of I/O bus

Intelligent Disks

- Technology opportunities
  - Fast CPU's driven by Moore's Law
  - Large disk bandwidth growth
- IDISK
  - Programmable embedded CPU with memory
  - Connected by fast switched machine area network
- Why run file systems in disks?
  - Intimate knowledge of disk & FS semantics
  - Bypass I/O bus bottleneck
- What can we do with it?
  - ***Low latency writes***
  - ***Self-organizing disk***
  - Much more....

Related Works

- Database machines (WADS)
- Write anywhere disks (parity logging, floating parity, Loge, Mime)
- Write anywhere file systems (WAFL)
- LFS and its applications (such as AutoRAID)
- Logical Disks

Contributions

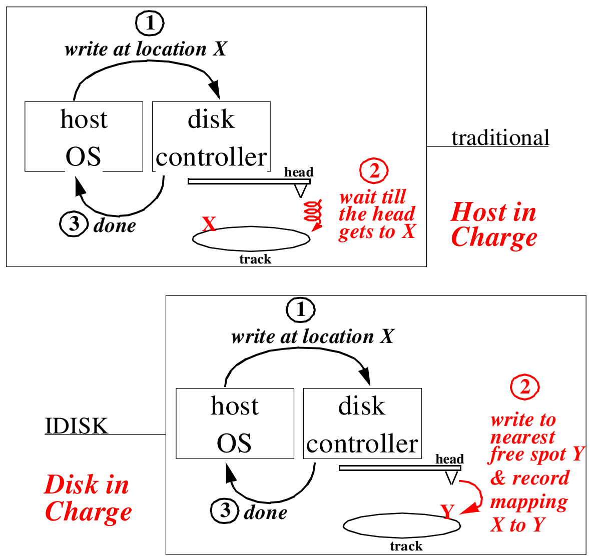
- Developed models to understand theoretical limit
- Designed "virtual log", the integration of LFS with a programmable disk:
  - No reliance on NVRAM
  - No reliance on altering disk format
  - Fast recovery
  - Primitives to support transactions
- Quantified differences among update-in-place, logging, and virtual logging

Outline

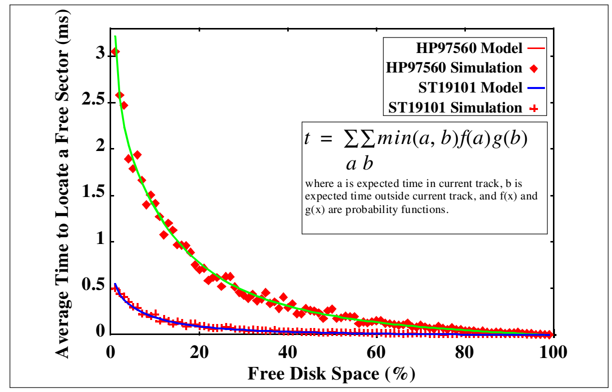
- Introduction
- ***Low latency write models***
- Virtual logging
- Experimental results
- Conclusion

Eager Writing

(Russian Roulette)



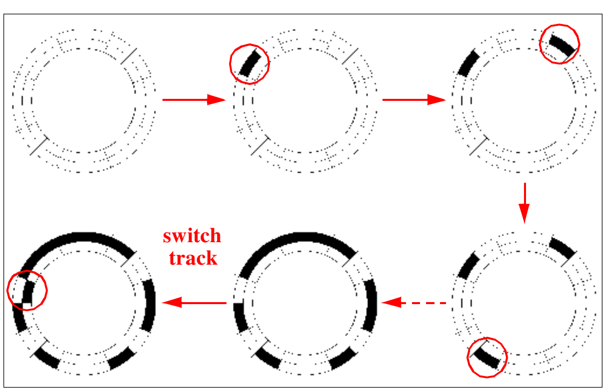
Low Latency? How Low?



- Model of average time to locate a free sector
- Time increases for higher utilization

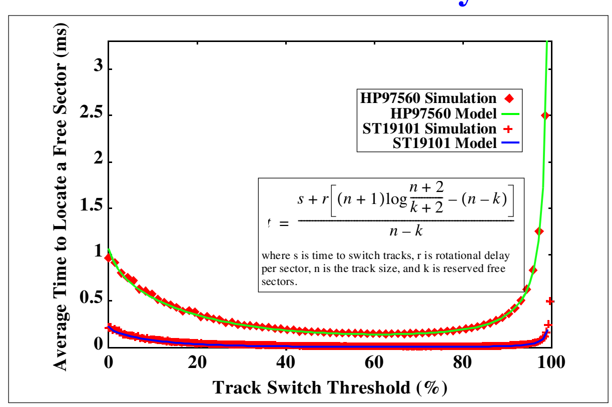
Eager Writing with Compactor

(Russian Roulette II)



- Cleaner adaptively generates empty tracks
- Start writing to an empty track
- Fill it to a certain utilization
- Switch to next track

Even Lower Latency



- (Note different x-axis from previous slide)
- Right hand side: switching tracks too often
- Left hand side: switching tracks not often enough
- Possible to have ***very*** low latency

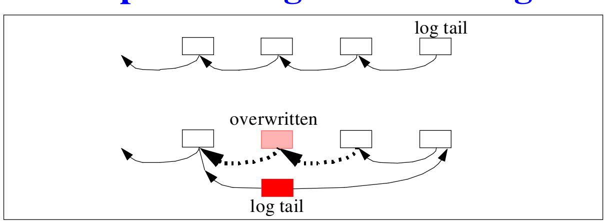
Outline

- Introduction
- Low latency write models
- ***Virtual logging***
- Experimental results
- Conclusion

Hard Questions

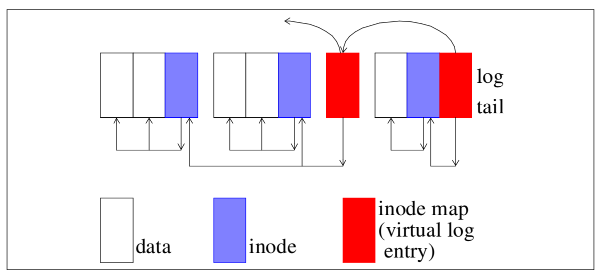
- How to map logical to physical addresses?
  Maintain a table.
- How to keep it persistent?
  Write pieces of it to disk (write anywhere).
- How to recover the table after crash?
  Options:
  - Scan disk
  - Scan part of disk after checkpoint
  - Log table entries - a ***virtual log***, a log with physically noncontiguous entries

Implementing a Virtual Log



- How to chain forward if future unknown?
  Reverse chain of map blocks.
- How to avoid garbage collecting map blocks?
  Implement chain with a tree.
- How to recover the map?
  Traverse from persistent root.
- How to keep root persistent?
  Write root to fixed location during power-down
- How to avoid doubling the number of I/O's?
  Add a level of indirection!

LFS on the Virtual Log (VLFS)

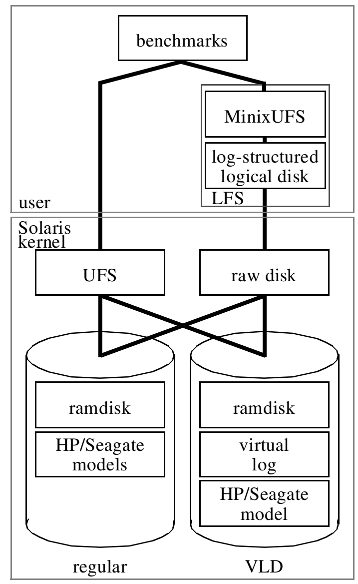


- How?
  - Put "real" addresses in blue and red nodes
  - Red nodes are the only entries of the virtual log
  - The virtual log can fit in memory
- Why? (VLFS advantages)
  - Better small synchronous write performance
  - Compactor less demanding than cleaner
  - Better accommodation of reads

Outline

- Introduction
- Low latency write models
- Virtual logging
- ***Experimental results***
- Conclusion

Software Architecture



Hardware

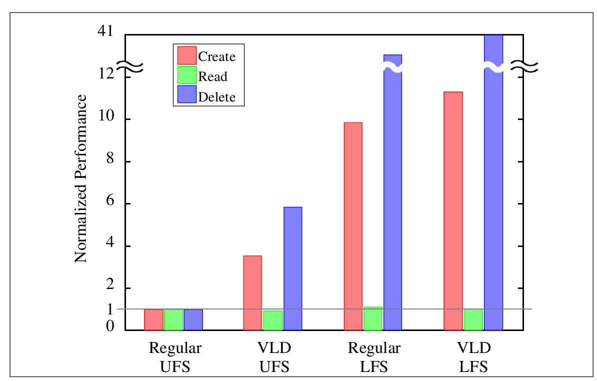
(Real or Simulated)

Disks	HP97560	ST19101
SCSI Overhead	2.3 ms	0.1ms
RPM	4004	10000
Track Size	72 sectors	256 sectors
Head Switch	2.5 ms	0.5 ms
Minimum Seek	3.6 ms	0.5 ms

Hosts	SPARC10	Ultra 170
OS	Solaris 2.6	Solaris 2.6
CPU Speed	50 Mhz	167 Mhz

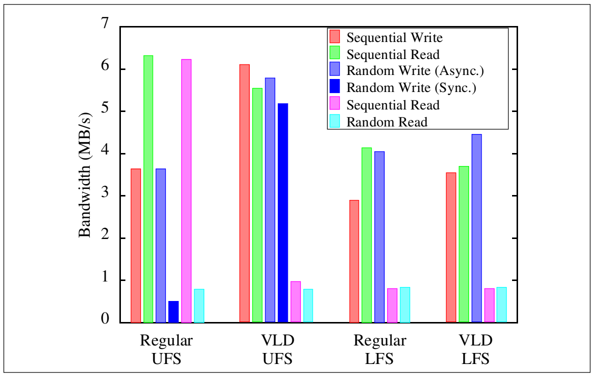
Base configuration: SPARC10 on ST19101

Small File Performance



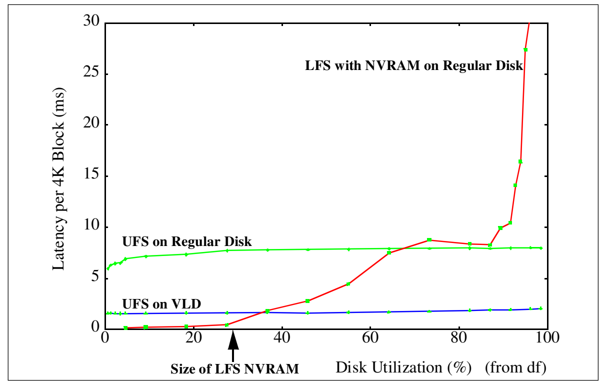
- Create 1500 1K files, read them back from disk, and delete them
- Speculation: VLFS combines best of UFS and LFS

Large File Performance



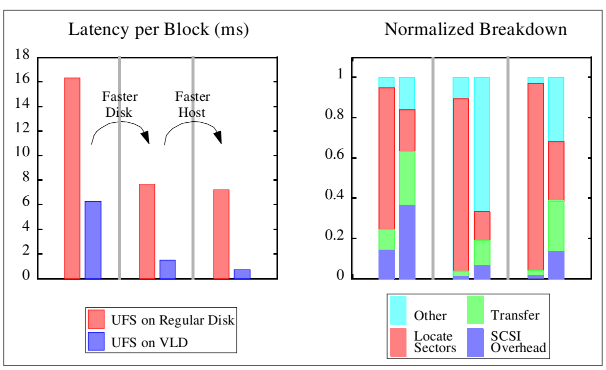
- Sequentially write 10MB, sequentially read, randomly write, sequentially read, and randomly read
- LFS and VLD destroy spatial locality, need data reorganization
- VLD always improves write performance

Effect of Disk Utilization



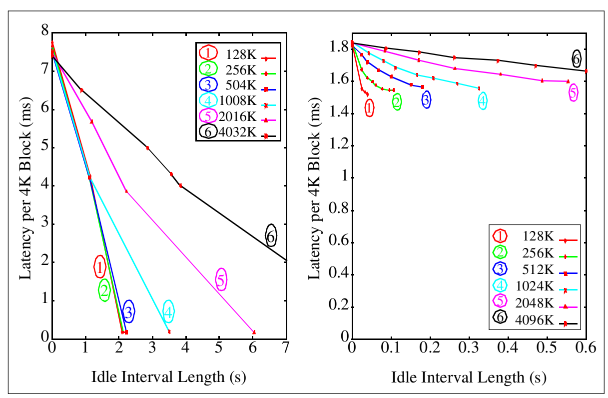
- Overwrites random blocks synchronously
- UFS on regular disk suffers from head movement
- LFS degrades as it runs out of NVRAM, and worse, when cleaning dominates
- VLD: no problem!

Effect of Technology Trends



- First use a faster disk, then move to a faster host
- Mechanical delay increasingly dominates update-in-place performance
- Importance of overhead reduction for VLD
- VLD achieves an order of magnitude improvement on newest platform!

Effects of Available Idle time



- Bursts of random small updates interleaved with idle periods
- LFS on a regular disk
  - Cleaning is a necessity
  - Needs large idle intervals to mask cleaning cost.
- VLD
  - Compacting is only an optimization
  - Can benefit from small idle intervals

Conclusions

- A modern alchemy:
  - Have: high network bandwidth
    Want: low message latency
  - Have: high disk bandwidth
    Want: low latency and sustainable high bandwidth
- How?
  - Pay attention to overhead
  - Exploit knowledge of disk mechanism
  - Exploit parallelism to build balanced systems
  - Explore new applications for fast disks