Edsger W. Dijkstra
The Structure of the "THE" Multiprogramming System
Communications of the ACM 11(5), May 1968.

(1)

1) a model for multiprogramming OS
2) to prove delaying process does not hurt.
3) provable system.
~~4) To provide~~

2) (1) Hierarchical abstractions ~~which~~
(2) semaphors ~~a~~ as synchronizing primitives & for mutual exclusion

(3) Hierarchy

- Level 0 - abstract the processor with ~~a~~ process scheduler
- Level 1 - segment controller, abstracts secondary storage into "segments"
- Level 2 - message interpreter, support conversation b/t user and their high-level processes
- Level 3 - buffering I/O
- Level 4 - user programs
- Level 5 - operator (not implemented by THE)

4.) Virtual address space - separates data from where it is stored. Can map virtual address to physical address.
Advantage is that [virtual address space can be much bigger than physical address space]? Also don't need contiguous memory, meaning ~~the~~ more efficient memory usage.

Disadvantage is takes extra space (page table) and lookups take extra time.

5 Semaphore for mutual exclusion

Process 1	Process 2	Time
P(mutex)		↓
Critical Section	P(mutex) Wait	
V(mutex)	↓ Resume	
	Critical Section	
	V(mutex)	

✓

Semaphore for Scheduling

Consumer	Producer
P(mutex)	P(mutex)
If element on queue	Add element to queue
V(private)	V(mutex)
V(mutex)	
P(private)	☆

Per Brinch Hansen
The Nucleus of a Multiprogramming System
Communications of the ACM 13(4), April 1970

Motivation:
1. Create an extensible OS, which lets the applications decide their own policies and even behave as OS themselves. They wanted to do this because current OS don't allow enough flexibility to applications.

2. High level point
The nucleus is used to implement the simulation of processes, comm. b/w them, creation, deletion & control of the processes.

3. Inter-process communication:
Get away with semaphore driven synchronization for better efficiency & security. Use message passing through buffers. This technique is an asynchronous way of communication & scales well. Basic message calls:
send, wait, } message & send, wait } response

Awesome Writing!

4. Process Heirarchy
The memory space originally belongs to the initiating process, which can allocate a subset of its memory to a process it creates. This results in a parent-child protection scheme for memory access. As an extensible / generic process management scheme, this also allows arbitrary OS's to exist/run without interference, and needing no specialized management.

5. 2 kinds of processes: internal, external
internal ⇓ processes.
external ↳ I/O devices

- treat internal and external processes uniformly to create extensible system. The difference between them is merely a matter of processing capability.

P.T.O

Ext

Creation: assignment of name to a peripheral device.

Internal

assignment of name to a contiguous storage area selected by the parent

~~Real time~~ synch

Review:

1. Introduced process names.
2. ~~Pro~~ Document is also a process (external)

Ritchie, D.M. and Thompson, K.
The UNIX Time-Sharing System
Communications of the ACM, Vol. 17, No. 7, July 1974, pp. 365-375.

1) Wanted to make a general multi-user operating system that was simple, easy to use. (Authors thought filesystem was the most important role of an OS)?? + so they focused mostly on that. Also wanted an easy to use shell and command execution ✓✓

2) Conceptual contributions - I/O equivalent to file system access ☆
- fork/exec/pipe/wait/exit - style process management

Practical contributions - the Unix system, including the file system & shell
- particularly that a multiprocessing system can work on relatively inexpensive hardware ✓

3) → File system with directory a tree, inodes links
→ Special files. for. I/O devices - /dev
→ Linking of files → protection bits → r/w/x
→ setuid.

adv
→ for general purpose use
→ not for any predefined objectives

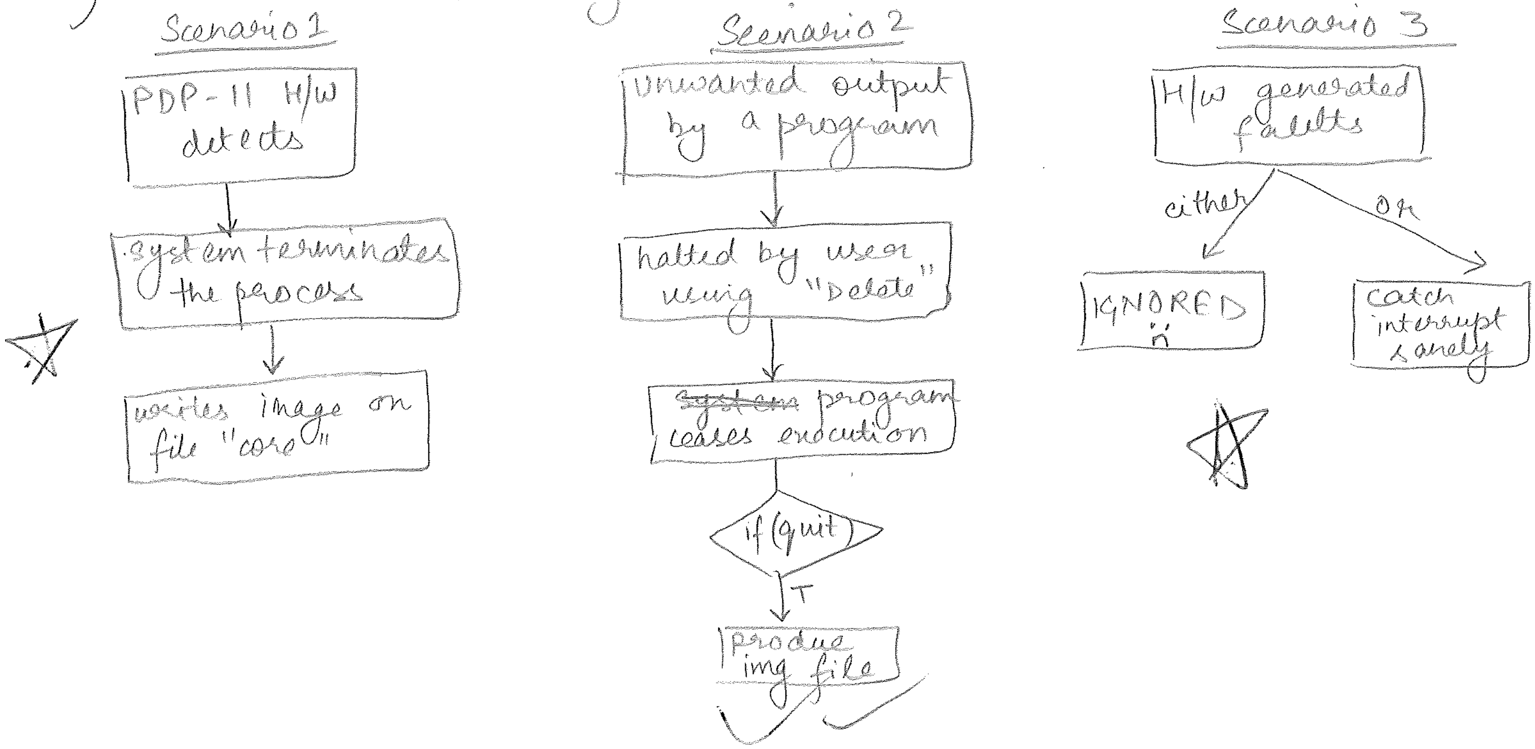
Dis adv
- Separate inode from the data/file → lot of disk seeks
- small block size → wastes the disk b/w for large files

4) Process - execution of an image (code, registers, open files, current directory, etc.)
- Exists in memory
- Created by fork() in parent process: copy of memory, shared files, etc.
- IPC with pipes
- Execute new program with exec(ute)()

Adv.
fork() + exec() clean & easy

P.T.O.

5) TRAPS → How they work?



Chris Ruemmler and John Wilkes
An introduction to disk drive modeling
IEEE Computer 27(3):17–29, March 1994.

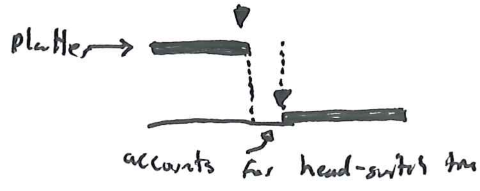
1) The motivation of this paper is that researchers use disk models to improve overall I/O performance in disks by running simulations on these models. However the problem is that little work has been done to create accurate disk drive models. This severely limits the useful information gathered, because simulations are run on models that do not reflect reality. Therefore, the author aims to build an extremely accurate disk model. ✓✓✓ ✓

2. The authors discuss the physical characteristics of disk drives and explain how they affect r/w behavior. They present an accurate model of the disk drive. They created an simulator based on ~~their~~ this understanding which can accurately ~~predict disk~~ emulate disk I/O behavior. ✓✓✓ ✓

3. The primary technique was to model specific physical elements of the disk's operation. The simulator tried a model with linear seek times, which was not that accurate, and then a model with seek times that accurately reflected the accelleration, max speed movement, decelleration, and head settling time of a seek, resulting in much more accurate modelling. ✓✓ ✓

4. Another major component they modeled was the caching behavior of typical disks. They were able to show that a model that does not provide a cache simulation is widely inaccurate since a large percentage of I/O is handled at the cache. After implementation their model was much more inline with a real disk. ✓ ✓

5) Track Skewing: offset block data on each track so that ~~worst co~~ data can be read continuously with having to seek back on the disk. when head or track switches occur. ✓



McKusick, M.K., Joy, W.N., Leffler, S.J., and Fabry, R.S.
A Fast File System for UNIX
ACM Transactions on Computer Systems, Vol. 2, No. 3, August 1984, pp. 181-197.

1) the problem was that the original Unix FS was incredibly slow, getting a blazing 2% of max disk transfer. ✓ ✓

there were 3 problems:

1. small block sizes - more transfers, w/ fixed cost per transfer
2. poor free list organization - consecutive blocks not close together?
3. No locality - inodes far from data blocks

2) Contribution.

- Better locality is proposed - using cylinder group. ✓
- larger block size. ✓
- fragments for small files ✓

3) - Organise data which is accessed together in close spatial locality. Like inodes and data blocks.
- Use 4KB blocks to get better bandwidth, but also support smaller fragments to ~~save~~ reduce wastage ✓ ✓
- Place file data in rotationally optimal blocks to increase sequential r/w performance.

4) - Layout policy: Files in a <u>folder</u> are put in the same cylinder group if they fit and are smaller than 48kb.? Advantage: Less seek time when accessing files in same directory. Seek time is less of overall cost for large files. More files in folder fit together.
Disadvantage: Large files are split across disk at 1 MB intervals, which increases read time. ✓

5) if the CPU doesn't have I/O channel, the data ~~the~~ has to be put in ~~a~~ "rotationally optimal" location, so the ~~B~~ disk head will be on the right position after the CPU interrupt.

? ✓

Levy, H. and Lipman, P.,
Virtual Memory Management in the VAX/VMS Operating System
IEEE Computer, Vol. 15, No. 3, March 1982, pp. 35-41.
Operating System Structure

1.) Goal was to support a variety of physical mem sizes (250K + 8M) and demands of timeshared systems, realtime and batch processes. ✓✓

2.) Predictable performance (?) can be given by ~~maximizing~~ giving more space in memory (using techniques like swapping and paging out page tables) and reducing I/O (using clustering). ✓ ?? ?

3) Their implementation of the swapper was a solution to the problem of the high cost associated with paging in pages after a process is able to execute again. Without the swapper keeping a process resident set to load in pages on a process waking or during process start each page would have been faulted in. ✓☆ ✓☆

4) VAX also implemented a pager service designed to efficiently manage paging in and out. In particular, when a page is removed from a process' resident set, the pager puts it on a free list (if it is unmodified) or a modified list, which serve dual duty as caches for pages that might be read in again later and as sources of free pages for when new pages are read into memory. They could also be used to batch I/O of writing modified pages back out to disk. ☆ ☆ ☆

5) VAX's virtual addresses were as follows:

		VPN 21	Offset 9

512B pages

first 2 bits mapped to different segments
- 00 – user (P0)
- 01 – user (P1)
- 10 – system
- 11 – reserved

To ~~convert~~ translate an address into a physical memory location, VAX would:
1. Look at the first 2 bits, and go to the corresponding segment's base register.
2. Jump to that segment's page table, and translate VPN into PFN.
3. Jump to that physical page, and apply the offset

✓✓

R. Rashid and A. Tevanian and M. Young and D. Golub and R. Baron and D. Black and W. Bolosky and J. Chew,
Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures
Proceedings of the 2nd International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS), 1987.

1. UNIX's virtual memory system relied on limited functionality for portability. Mach aimed to create an easily-portable VM system that provided more functionality than UNIX and minimized hardware-dependent code.

2. The high level contribution of this paper is the abstraction they provide between the machine dependent/independent code. Before this work virtual memory management was almost always machine dependent. However this work demonstrated/implemented the construction of a memory management system that was independent of hardware. This allowed the system to run on any hardware configuration; something commonplace now. Pmap is machine dependent part.

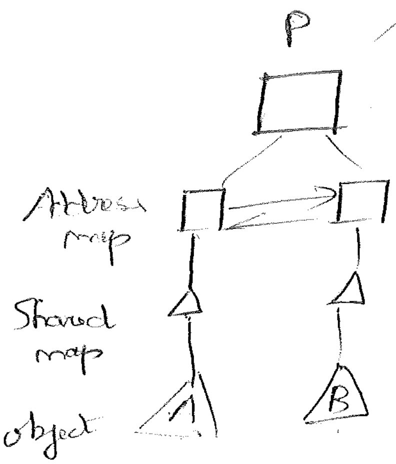
3. Shadow objects allow multiple processes to use the same memory for the same objects. Instead of allocating memory twice, the second process contains a shadow object which points to the original memory. This uses less memory for shared objects. A disadvantage is that when memory is written to an shadow object, many new smaller memory objects may be created and this can be complicated to manage.

4. Inter-procedure Communication (IPC) was ~~central~~ an important part of their design. Each task had "ports", which other tasks were able to use to send/receive messages. Messages could be passed back and forth through the copy-on-write mechanism, which significantly reduced the IPC overhead, relative to Nucleus.

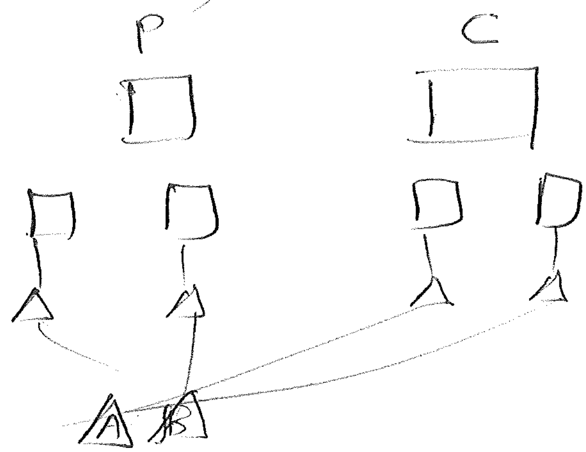
Pmap acts as a cache of virtual-physical mapping, but kernel mappings need always be kept. Other mappings can be reconstructed by the ~~MMU~~ machine-independent part.

5. Example of Copy on Write

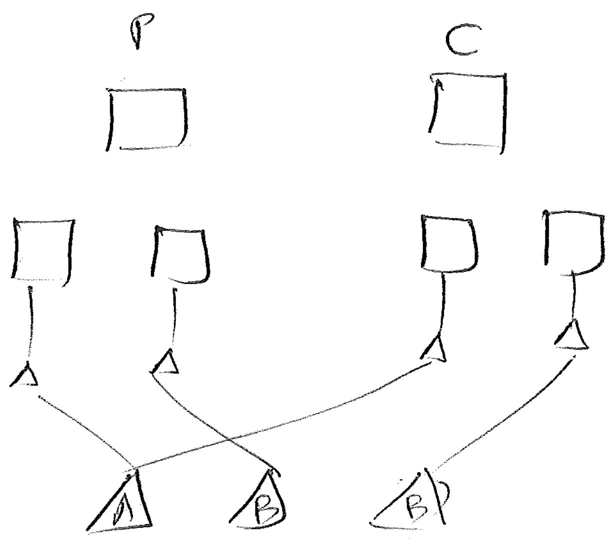
i) Initial



ii) Fork



iii) C Write B to B'



Edouard Bugnion, Scott Devine, Mendel Rosenblum.
Disco: Running Commodity Operating Systems on Scalable Multiprocessors
Proceedings of The Sixteenth Symposium on Operating Systems Principles (October 1997)

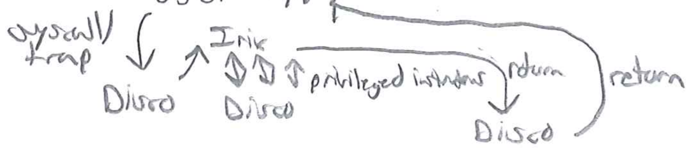
① They wanted to enable the running of commodity OS's on ccNUMA machines. At the time, new scalable computers with 10s or 100s of processors were being released, but the complexity of implementing software for those machines was too great, and thus, software was lagging behind hardware. ✓

② Disco allowed multiple OS's to run mostly unmodified on one system. It manages resources between all OS's without requiring OS to worry about non-uniform memory. Commodity OS's can be used. ✓ ✓

3) Introduced new layer between hardware and software. New layer supports virtual CPUs, virtual Physical Memory, virtual I/O devices, memory management, & Virtual Network Interface. ✓

4) Disco took advantage of existing virtual memory hardware to virtualize guests' "physical" memory. This allows guests to be moved from processor to processor, share memory, and share disk reads. However, it causes TLB misses to cause traps to Disco, which then has to emulate hardware for guest kernels, increasing the cost of TLB misses. ✓

5) In Disco, system calls to a guest OS look like this:



Nooks

Michael M. Swift, Brian N. Bershad, and Henry M. Levy
Improving the Reliability of Commodity Operating Systems
Symposium on Operating System Principles, SOSP'03, October 19–22, 2003, Bolton Landing, New York, USA

1) The prevalence of operating system extensions — code executing in fully privileged mode yet potentially written by relatively inexperienced programmers with little testing — leads to many system crashes. ✓ ✓ ✓ ✓ ✓ "nice."

2) Introduced mechanisms (2 wrappers - kernel & extension) to isolate driver code from the kernel by using wrappers and XPC. ✓ ✓ ✓ "agreed."

3). Basically, Nooks introduces indirection (→ XPC handles extensions, isolating them in the kernel) to implement isolation, interposition, object tracking and recovery.
By using XPC, they put function, arguements, domain info. And they do deferred calls, so ~~they~~ that they can do batch calls & avoid the overhead of call XPC frequently. ✓ ✓ "mmhmm"

4) Nooks Isolation Manager (NIM) is inserted between the extension and OS kernel. Provides lightweight kernel protection domain by limiting write access of extension to kernel address space. Tracks kernel resources used by extensions & cleans during recovery upon extension crash/misbehaviour ✓ "well written!"

5) - Isolating Kernel from extension failures provides 99% recovery from system crashes. ✓

→ high reliability

- Disadv.
  - large overhead introduced: ✓
    e.g. Web server: 60% ↓ in thruput.

point: Was not good at recovering the modules, just good at recovering the system.

RecoveryManager - releases resources used by extension
cleans the system state

W. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack
HYDRA: The Kernel of a Multiprocessor Operating System
Communications of the ACM 17(6), June 1974, pp. 337-344.

① Want to support multiprocessor environment and provide sufficient primitives for protection of resources (not security!) ✓✓ Also wanted to avoid a hierarchical system structure, which can be limiting. ✓

② - Provided a way to separate mechanisms from policies.
- Abstraction of resources as objects ✓✓✓

③ i) Caller dependant capabilities & caller independent capabilities

ii) Type dependant access rights (for custom objs) & Kernel access rights for common operations (like copying) for all objects.

iii) Kernel can provide protection without "understanding" the rights bits.

iv) Path-names to avoid explosion in no. of capabilities passed between procedure calls. ✓✓

4) i) Capabilities cant be forged. Ensured by H/W or OS support? User process.
Every word of memory is tagged.
Tag: ↳ 1: cap
0: other.
cant write cap words.
Only kernel can modify capabilities → put in segmented memory

ii) procedures → name, data (code & static data)
capability list.

5) Flexibility to obtain extension from both representation of object as both data & capabilities ?

Dawson R. Engler, M. Frans Kaashoek, and James O'Toole Jr.
Exokernel: An Operating System Architecture for Application-level Resource Management
In the Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP '95), Copper Mountain Resort, Colorado, December 1995, pages 251-266.

separate protection from management

1. The motivation of exokernel is that the lower the level of a primitive, the more efficiently it can be implemented and more flexibility to implementors of higher level abstractions. High level abstractions hurt application performance, hide information & limit its functionality and hence the need for library OS.

? ★

2) 1. securely expose hardware → run privileged instructions, access resource

2. expose allocations → request physical resources

3. expose physical names ???

4. expose revocations

Multiplex resources, don't abstract.

3). i) Secure Bindings → to securely provide applications the low level resources of the systems

- Can be implemented in hardware or software

(ii) Downloading code into kernel to extend the capabilities of kernel.

- Packet filtering by downloading code which by dynamically generated.

(4) Visible resource revocation - eg. CPU is explicitly revoked from libos. Libos saves the processor state.

Abort Protocol - If a libOS misbehaves with a resource / is disobedient then Exokernel takes that resource away from libOS forcibly.

(5) ASHs - Application Specific safe handler - An implementation of downloading code into kernel. They do message processing for packets arriving over network. It runs on packet receptions and it can also initiate a message to be sent. This avoids the application to be scheduled to ~~[illegible]~~ send a response to the received n/w packet. ✓

Tyler Harter, Chris Dragga, Michael Vaughn, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau
A file is not a file: understanding the I/O behavior of Apple desktop applications
SOSP '11 Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles Pages 71-83

MOTIVATION:
→ Not enough analysis done in the past about home ~~workloads~~ applications. ✓ ✓
→ To understand the I/O behavior and how users perceive system latency and app. performance ✓ ✓ ✓ ✓

Contribution:

In depth case study and analysis of the I/O behaviour of modern home-user applications. The analysis has several interesting revealations. (file is not a file, seq access is not seq, writes are forced, multiple threads perform I/O, frameworks influence I/O etc). ✓ ✓ + renaming is popular + auxillary files dominate

Solution / Technique advan./disadv. ← copied from abstract
The results have a strong ramifications for the design of next generation local and cloud-based storage systems. Constructing the iBench task suite two-fold: 1. representative of tasks performs by home users. ??

2. Sol/Tech:
4) The authors study four categories of the behavior of the ibench suite,
1) Nature of files.
2) Access patterns ✓
3) Transactional properties
4) Multi-threading behavior

5) iTunes attempted to block efforts to trace it, but the authors found a way to circumvent this by using gdb to force a system call into a NO-OP. ☆

Rosenblum, M. and Ousterhout, J.
The Design and Implementation of a Log-Structured File System
ACM Transactions on Computer Systems, Vol. 10, No. 1, February 1992, pp. 26-52.

1. The motivation for LFS is to match the write performance (speed) of the disk with that of the CPU. ✓✓

✓✓ 2. The use of log to club small random writes ito large sequential writes to achieve ~~large~~ very good utilization of the disk bandwidth.

3. ✓✓ They combined ~~Interleaving~~ ~~threading~~ threading and compaction into a hybrid approach where they grouped blocks into segments and threaded the segments. They periodically ran a cleaner to find underutilized segments and write their data at the front of the log.

✓✓ 4. i) Identifying ~~to~~ live blocks using Segment summary block which contains for every datablock the file inode and offset.
- Some more optimizations like maintaining versions in inode map to avoid costly identification of live blocks (Eg. truncation of files). ✓

ii) Using 2 timestamps for each of two CR regions and writing to alternate CRs.
Adv - No inconsistent CR
DisAdv - The CR commit interval is 30s which affects freshness of data.

5. ~~As write cost = 2/(1-u), ~~writing~~ cleaning under utilizated segments ~~blocks~~ would give high performance. ~~The Assumptions of the above~~ ~~As~~ As ~~clean~~ having more under utilized blocks, increases cost per byte storage, approach to have segments into ~~nearly~~.~~ Performance impact of cleaning.

✓✓ 9 A cost benefit segment cleaning policy is proposed. ✗ The cost benefit policy involves bimodal distribution of segments,

$$\frac{\text{benefit}}{\text{cost}} = \frac{(1-u) \times age}{(1+u)}$$

✓ To use cost-benefit policy, a data structure which records #live bytes and recent modified time called segment us~~age~~ble table. is used!

Segment Usage Table

✓

Vijayan Prabhakaran, Lakshmi N. Bairavasundaram, Nitin Agrawal, Haryadi S. Gunawi, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau
IRON File Systems
Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP '05), Brighton, United Kingdom, October, 2005.

Motivation:
Present disk failure model for commodity system is faulty — It assume that disk either work or fail.
✓ In reality, disks exhibit complex disk failure
* ☆ ✓ pattern.
~~Present paper does have following contributions~~

2. The paper presents a more advanced failure model for disks that provides partial failure support and develops a framework to determine how a file system handles failures.
Analyzing commodity FSes show that their handling of failures is buggy and inconsistent.
✓✓ Finally, a more robust file system, "ixt3", is demonstrated to show that error-checking techniques such as checksumming provide substantial reliability improvements at a low cost.

3. To determine how the FS handled disk failures, the authors
✓ defined the IRON Taxonomy for various levels of detection
✓ and tabulated the behaviour for various workloads.

Adv: The tabulation (complex picture) gives a very detailed picture of the behaviour ✓

4. Build a prototype version of an IRON filesystem (ext3)
✓ that is robust to various kinds of partial disk failures. Evaluation of ext3. ✓
+ checksumming, metadata replication, parity

5. Works! - Transactional checksums for ordering journal data. This improves the performance of Journal data I/O.
✓
(checksum the transaction blocks + write the checksum with the transaction commit)

Yupu Zhang, Abhishek Rajimwale, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau,
End-to-end Data Integrity for File Systems: A ZFS Case Study
Proceedings of the 8th Conference on File and Storage Technologies (FAST '10)

1. Motivation:
To understand modern file systems' (ZFS as a representative FS) reliability mechanisms for dealing with disk & memory corruptions. ✓ ☆

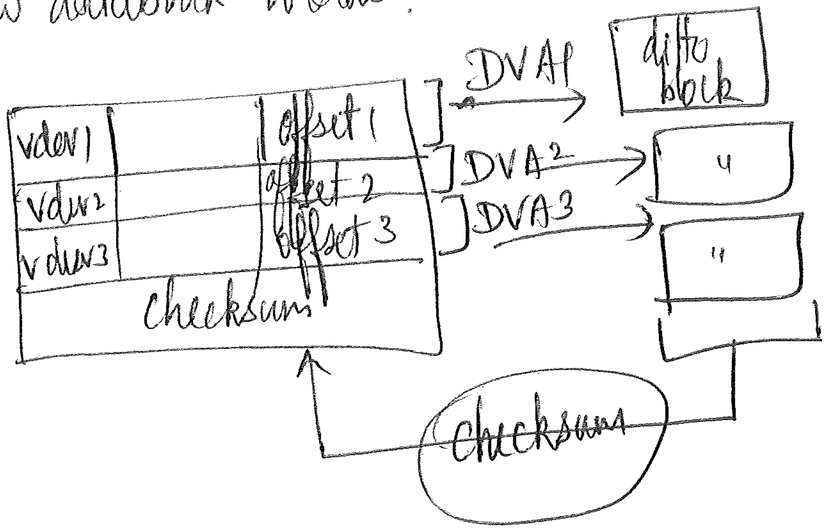
2.) ZFS while does checksum verification for on disk blocks, it doesn't use those same checksums in memory to protect against in memory corruption. ✓✓

3. Solution/technique: ZFS uses checksums for all blocks on disk. This checksum is stored in the parent block for a given block; by doing this rather than storing it alongside the checksummed block, it becomes possible to detect errors such as misplaced writes. Maintaining and checking these checksums does cause some overhead, and updating a checksum necessitates propagating new checksums up the file system, though the atomicity of this is guaranteed through copy-on-write semantics. ✓ ✓

4 a) ZFS in addition to checksum, use replication to maintain reliability ☆
→ metadata → 2 ditto blocks
global metadata → 3 ditto blocks

b) ZFS ~~uses~~ tries to ~~simulate~~ provide software based RAID system by having ~~virtual~~ storage pools → zpool
zpool → ~~vi~~ consists of virtual devices (physical and logical). ✓
→ RAID Z (similar to RAID 5)
diff → variable stripes.

5. How datablock works?



Vijay Chidambaram, Thanumalayan Sanakaranarayana Pillai, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau
Optimistic Crash Consistency
Symposium on Operating System Principles, SOSP 2013

Motivation - Crashes are rare but possible in FS. Pessimistic approaches anticipate crashes and do the careful logging to the journal before writing to ~~FS~~ disk. These ordering points make the FS perf slow. Main motive is to avoid ~~orderi~~ flushing as much as possible and also decouple ordering from durability ✓ ☆

High Level Point: Probability of inconsistency because of crash is high for workloads with a high number of random write I/Os. for probablistic crash inconsistency ☆
Maintain consistency to same extent as pessimistic but nearly same performance with probablistic consistency

Solution:
1. Add data checksums to avoid write ordering
2. Use asynchronous durability notification ☆ to delay checkpointing until durable commitment has happened.
3. Ordered Journaling. (transactions)

Advantage 1. disk itself determines when to flush. don't need to listen to higher-level FS. ☆

disadvantage: - Recovery takes a long time X (not much longer than normal journaling)
- greater code / memory overheads, + stale data

Essential operations of osync:
When OptFS calls on osync, it writes, ~~in order~~ ? the data blocks, journal meta-data, and journal commit block. Checksums are added to the commit block so these can become durable in any order. Then, only after receiving notification (or a simulation of this) that the blocks are durable, will it 'checkpoint'; ~~or order to~~ write the metadata to disk, which occurs in background.

D. Ongaro, S. Rumble, R. Stutsman, J. Ousterhout and M. Rosenblum
Fast Crash Recovery in RAMCloud
Proc. 23rd ACM Symposium on Operating Systems Principles, October 2011, pp. 29-41.

1. ~~For~~ DRAM → more imp, need full performance of DRAM → ✓
Fast Recovery on crash ← durable + available ✓
↳ Make use of scale + try to provide ☆ low latency → ½ ✓
↳ impact: enable new class of ☆ applications ☆

2) .

3) Techni~~que~~ ~~single recovery master~~

2. i) Log structured storage (both in back-up + in-memory) ✓ ✓
ii) Extremely low-latency storage ✓ ✓ ✓
iii) Randomization techniques to make decisions in scalable fashion ✓ ✓
+ Pick best of randomly chosen ✓

3) Disadv.
~~cannot at~~
- Very hard to achieve such short switching times and very hard to, optimizations like ? kernel bypass & faster NICs. ✓✓ but badly worded

Adv
- ~~Us~~ Harness the power of several ~~b~~ nodes to retrieve data & perform recovery. ✓ ✓

4) • Used DRAM ~~+~~ to support low latency ✓ ✓
dis-adv. • Volatile so backup efforts made to avoid dataloss ✓ ✓
• Can potentially be used for massive r.t. computations ✓ ✓

5.) Recovery Process

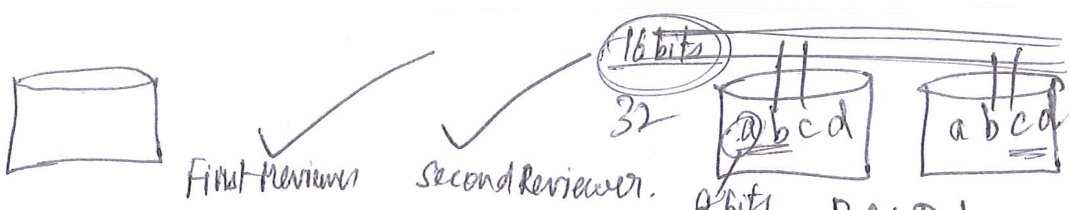
1.) When DRAM crashes, controller asks all nodes what pieces of the missing data they have ✓✓

2). assigns a subset of recovery masters, tells them which segments to incorporate into their own memory ✓✓

→ recovery is spread out among many disks ✓✓

Done, serve requests from restored data ✓✓

+ servers keep 'wills' indicating how data should be grouped ✓



Patterson, D., Gibson, G., and Katz, R.,
A Case for Redundant Arrays of Inexpensive Disks (RAID)
Proceedings of the 1988 ACM SIGMOD Conference on Management of Data, Chicago IL, June 1988.

CAN WRITES BE PARALLEL in RAID1?

1) * Single disk become bottleneck (because seek latency & bandwith are not improving ~~at~~ a good rate compared to processor speed). Thus, how can we use ~~multiple~~ multiple ~~disks or~~/array of inexpensive disks & collectively use them to get greater bandwidth for I/Os

2). a). ~~categorize~~ model & categorize ~~a~~ different techniques
b) take advantage of parallelism
c) 12 times higher bandwidth than state-of-art device at that time. ?
d) good reliability ~~a~~ evalution of each case.

3.) low reliability can be dealt with using replication. The technique is simple - buy another disk, and copy the first disk over entirely. An expansion of this is data redundancy, used in all levels. *
The advantage is it can improve reads and reliability, while the downside is it requires extra disk space (whether replicating or checksumming, etc.)

4.) Parallel access improves aggregate bandwidth when ? data is replicated or when partitioned (RAID-0). RAID-0 lacks redundancy, however RAID 1 can benefit from parallel reads. Write performance of RAID-1 ~~however~~ is no worse than a single disk assuming no bus contention.

5) From RAID-0 to RAID-5 the models progress by improving disk utilization (although the preformance for small writes decrease as we go through the levels) and reliability through parity disks. RAID-5,

improves upon RAID 3 & 4 by distributing the parity.

John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan
The HP AutoRAID Hierarchical Storage System
ACM Transactions on Computer Systems, Vol. 14, No, 1, February 1996, Pages 108-136

Automatically manages storage with RAID 0 & RAID 5 by abstracting from file system.

RAID10, RAID 01

1. X? Motivation: a two-level storage hierarchy implemented inside a single disk-array controller → RAID 0 vs RAID 5 performance vs cost ~~Hence a RAID lev~~

? HP AutoRAID: fully redundant storage system ?

2. ~~Hig~~ Contribution: ~~A Two~~ ~~A RAID~~ A storage hierarchy is developed ~~using~~ which distinguishes between write-active (RAID 0) and write inactive (RAID 5);

3. Active / freq. accessed data is kept in RAID 0 (mirroring)
Inactive / less " " " RAID 5 ξ.
Transfer of data between the diff. RAID levels according to the need of the workloads.
Data stored as: RBs, segments, PEXes, PEGs, LUNs.

4. Advantages:
- Automates the management of RAID groups.
- Good job at balancing performance vs disk utilization (active vs inactive) move data between groups)
- Scalable: possible to add new disks ~~at~~ ~~at time~~ on the fly.
- Upgrade: Replace disks without disrupting operations.
- extremely easy to use

Disadvantage.
- Has variable response times. Not suitable for some workloads.