25 Years of Storage Research and Education: A Retrospective

Remzi H. Arpaci-Dusseau
Grace Wahba Professor and Department Chair
Computer Sciences Department
School of Computer, Data & Information Sciences
University of Wisconsin-Madison

Outline

Part I: From AI to storage to storage impact

- A bit autobiographical - apologies

Part II: Technology-driven research

- A bit technical - further apologies

Along the way: Lessons learned

Part I: From AI to Storage

Undergrad at Michigan: AI? Or not AI?

Like everyone, AI was my first idea

Got a job working at a UM robotics lab

Why I stopped working on AI

- **The story of two robots**

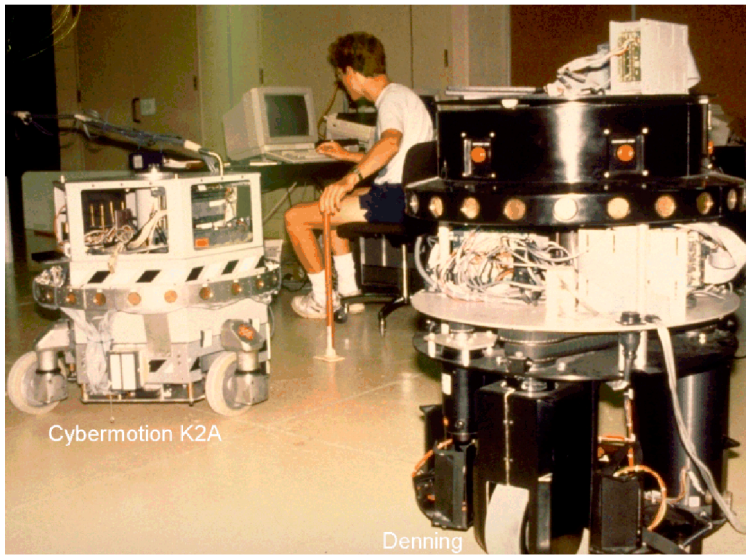


Figure 1.11: The Denning *Sentry* (foreground) incorporates a three-point *synchro-drive*

Lesson

- **When working on something, ask: how can I be the best? What skills and know-how are required?**

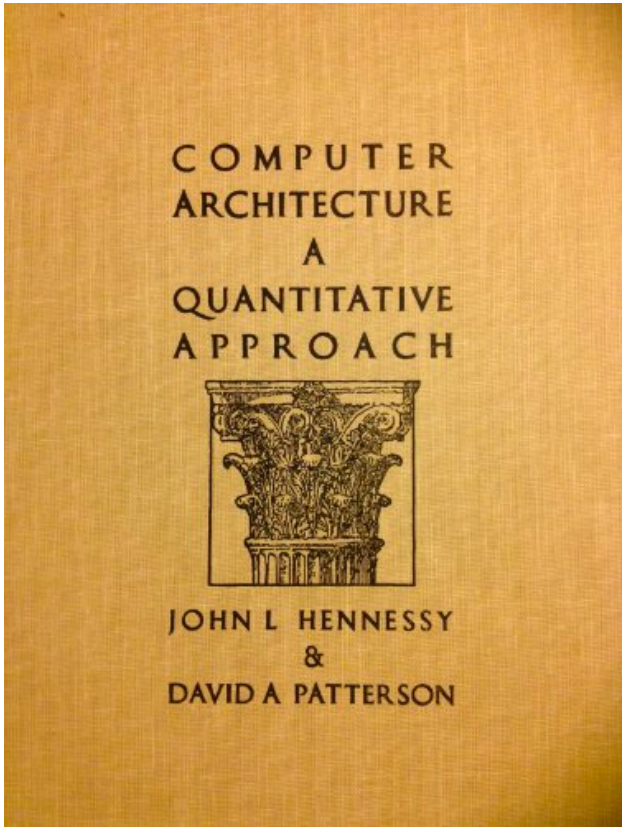
My next choice: Computer Architecture

Why architecture?

- A chance visit to the bookstore

Lesson

- **Never underestimate the power textbooks can have**



From Architecture to Systems

New project at UC Berkeley: **Network of Workstations** (**NOW**) [http://now.cs.berkeley.edu/]

Basic premise

- Supercomputers of the future would be made from commodity PCs
- Modern networks were a key enabler (“3 orders of magnitude”)

Thus, most of the work lay in **systems software**

- Distributed operating systems, file systems, high-speed networking, etc.

Thus, systems it was

First Few Projects

"Empirical Evaluation of the CRAY-T3D: A Compiler Perspective"
Remzi H. Arpaci, David E. Culler, Arvind Krishnamurthy, Steve G. Steinberg, Katherine A. Yelick. ISCA 1995

"The Interaction of Parallel and Sequential Workloads on a Network of Workstations" Remzi H. Arpaci, Andrea C. Dusseau, Amin Vahdat, Lok T. Liu, Thomas E. Anderson, David A. Patterson. SIGMETRICS 1995

"Effective Distributed Scheduling of Parallel Workloads"
Andrea C. Dusseau, Remzi H. Arpaci, David E. Culler. SIGMETRICS 1996

Mostly performance analysis and scheduling… so, what next?

First Few Projects

"Empirical Evaluation of the CRAY-T3D: A Compiler Perspective"
Remzi H. Arpaci, David E. Culler, Arvind Krishnamurthy, Steve G. Steinberg, Katherine A. Yelick. ISCA 1995

"The Interaction of Parallel and Sequential Workloads on a Network of Workstations" Remzi H. **Arpaci**, Andrea C. **Dusseau**, Amin Vahdat, Lok T. Liu, Thomas E. Anderson, David A. Patterson. SIGMETRICS 1995

"Effective Distributed Scheduling of Parallel Workloads"
Andrea C. **Dusseau**, Remzi H. **Arpaci**, David E. Culler. SIGMETRICS 1996

Mostly performance analysis and scheduling… so, what next?

A Class Project in Databases

Took graduate database course

- Not my main interest …

Had to pick a final project… but what?

- Happened to read a paper
- So much to like!
  (world records, cache performance)

Lesson

- **Use papers you like as inspirations**

VLDB Journal, **4**, 603-627 (1995), Stanley Y.W. Su, Editor

AlphaSort: A Cache-Sensitive Parallel External Sort

Chris Nyberg, Tom Barclay, Zarka Cvetanovic, Jim Gray, and Dave Lomet

Received September 8, 1994; revised version received, March 28, 1995; accepted March 28, 1995.

Abstract

A new sort algorithm, called AlphaSort, demonstrates that commodity processors and disks can handle commercial batch workloads. Using commodity processors, memory, and arrays of SCSI disks, AlphaSort runs the industry-standard sort benchmark in seven seconds. This beats the best published record on a 32-CPU 32-disk Hypercube by 8:1. On another benchmark, AlphaSort sorted more than a gigabyte in one minute. AlphaSort is a cache-sensitive, memory-intensive sort algorithm. We argue that modern architectures require algorithm designers to re-examine their use of the memory hierarchy. AlphaSort uses clustered data structures to get good cache locality, file striping to get high disk bandwidth, QuickSort to generate runs, and replacement-selection to merge the runs. It uses shared memory multiprocessors to break the sort into subsort chores. Because startup times are becoming a significant part of the total time, we propose two new benchmarks: (1) MinuteSort: how much can you sort in one minute, and (2) PennySort: how much can you sort for one penny.

Project Proposal

Build external (disk-to-disk) parallel sort on cluster

- Use it to break world record!

Feedback from professor

- Not an interesting proposal; it's unlikely you will beat professionals

Our decision: **Do the project anyhow**

Lesson

- **Sometimes, you have to ignore advice (even from smart people)**

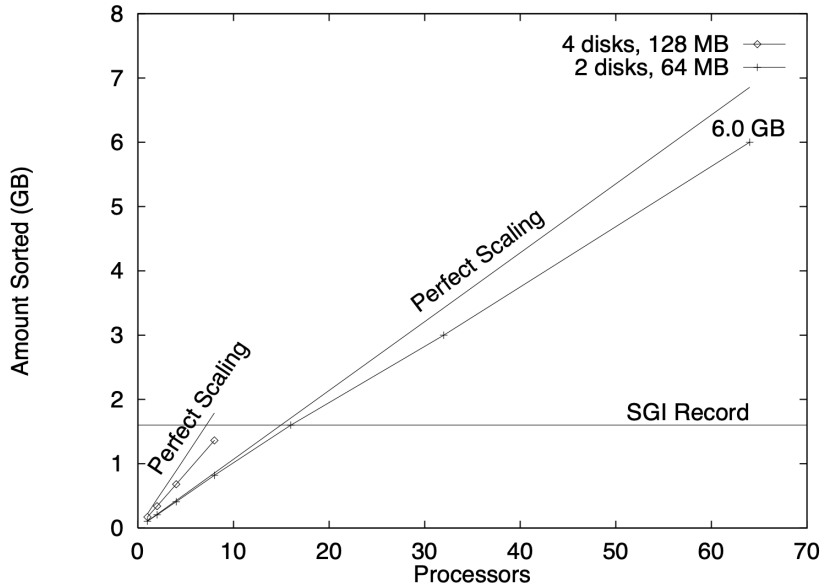
Result: NOW-Sort

Optimized every aspect of parallel sort

- Network communication and overlap
- CPU algorithm
- **Disk access methods**

Result

- Scalable, high performance
- Got the most out of available machines



Paper

- “High-Performance Sorting on Networks of Workstations”
  Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, David E. Culler, Joseph M. Hellerstein, David A. Patterson. SIGMOD 1997

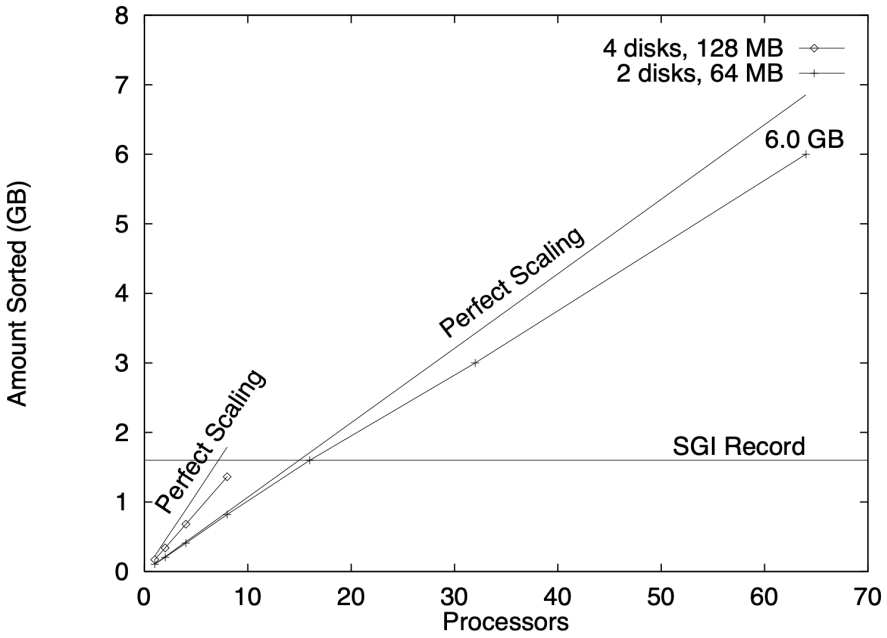
Result: NOW-Sort

Optimized every aspect of parallel sort

- Network communication and overlap
- CPU algorithm
- **Disk access methods**

Result

- Scalable, high performance
- Got the most out of available machines



Paper

- “High-Performance Sorting on Networks of Workstations”
  Andrea C. **Arpaci-Dusseau**, Remzi H. **Arpaci-Dusseau**, David E. Culler, Joseph M. Hellerstein, David A. Patterson. SIGMOD 1997

Main Lessons from Sorting

Storage is important

- All applications do I/O
- Many interesting applications do a lot of I/O

Storage is complex

- Complexity can be a source of challenges and opportunities

Storage is fun to optimize

- Because when you do, the thing you optimized goes a lot faster!

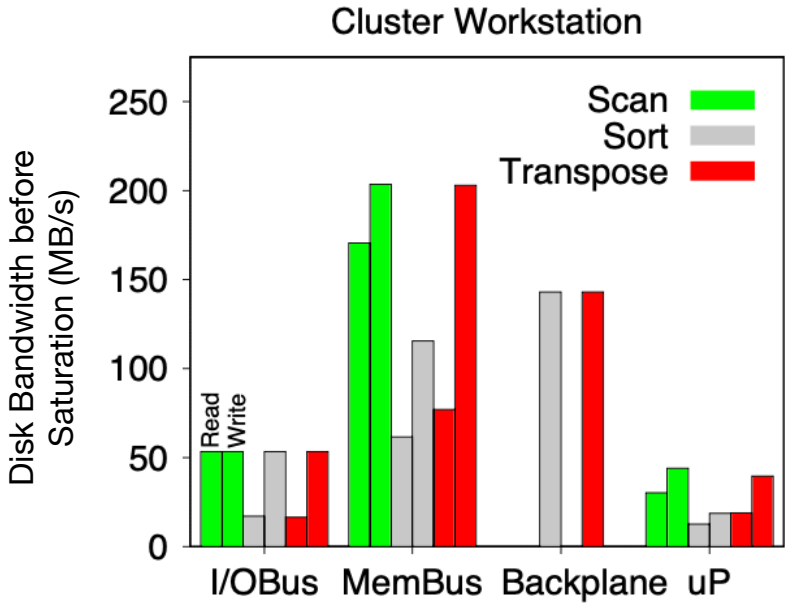
Next Work: Search for Balance

From a used computer architecture book:
Amdahl/Case "Rule of Thumb": A balanced computer system needs 1 MB of main memory capacity and 1 Mbit per second of I/O bandwidth per MIPS of CPU performance

After sorting work, asked

- What is a balanced cluster of workstations?
- How does that compare to other types of systems?

Result: "The Architectural Costs of Streaming I/O: A Comparison of Workstations, Clusters, and SMPs"
Remzi H. Arpaci-Dusseau, Andrea C. Arpaci-Dusseau, David E. Culler, Joseph M. Hellerstein, and David A. Patterson. HPCA '98



Lesson

- **Read widely and take notes**

Graduation Dilemma

Discussion with Patterson (advisor)

- Dave: “You should graduate”
- Me: “I want to do one more thing”
- Dave: “OK”

Why?

- My reasoning: a “systems” student should build a system

But what to work on?

A Tiny Observation

An interesting thing about NOW-Sort

- Not just **how** we did it but **when**

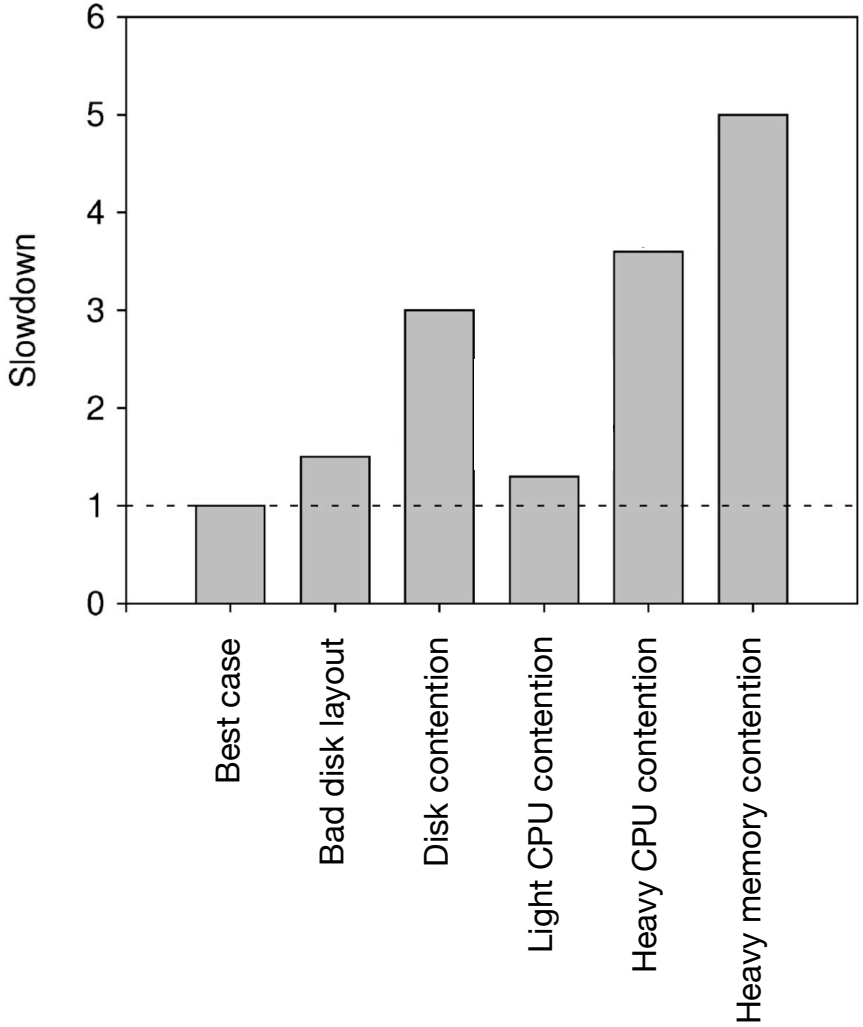
Record-breaking runs were always done at **4am**

Why?

- No one else was using the machines then
- Sorting on 100 machines was sensitive to the performance of the slowest 1 machine

Lesson

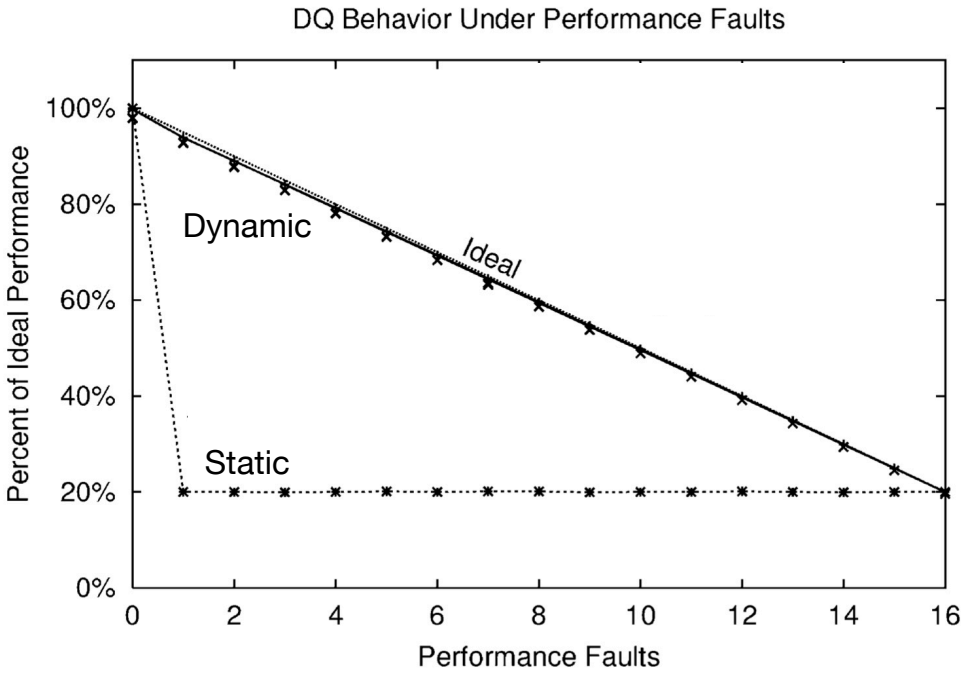
- **Keep your eyes open when doing research; doing so may unveil your next line of work**



River

How to make sorting run fast, even if one machine isn't?

- i.e., how to make sorting run well during the **day** and **night**?



"Cluster I/O with River: Making the Fast Case Common"

Remzi H. Arpaci-Dusseau, Eric Anderson, Noah Treuhaft, David E. Culler, Joseph M. Hellerstein, David A. Patterson, Katherine A. Yelick. IOPADS 1999

Pre-cursor to large-scale data processing environments

- i.e., MapReduce (1/6th of the papers cited by MapReduce paper are in this talk)

Assistant Profs @ Wisconsin

So, what to work on?

- Wanted to get away from dissertation work
- Who thinks large-scale clusters doing big data processing is important, anyhow? (oops)

But wanted to grow strength in storage, I/O

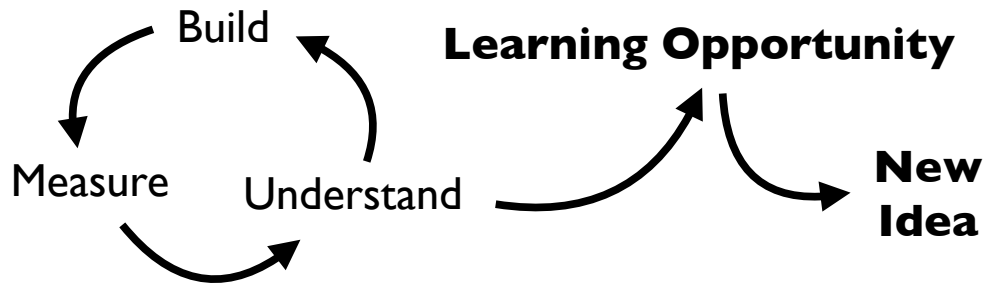
- Why? As before, interesting, complex, important
- But also, **opportunity**: A growing storage industry

But, need to be specific

Key: Measure Then Build

Previously at USENIX ’19: “Measure, Then Build”

- Main idea: use measurement to **learn** and to **find real problems**



Remember: Research is a **learning** exercise

Lesson

- **Always think about what can be measured, and how to learn from it**

Attack a Classic Problem: Costs of Layering

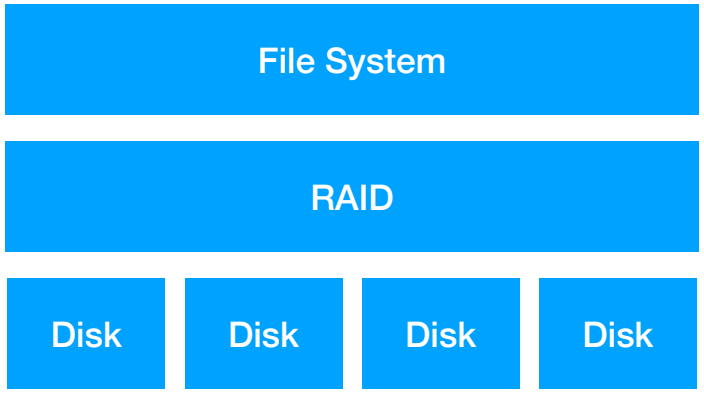
Systems are built in **layers**

- Reduces complexity
- Allows independent groups to build, optimize pieces

But inherently problematic

- **Information** loss
- **Control** loss

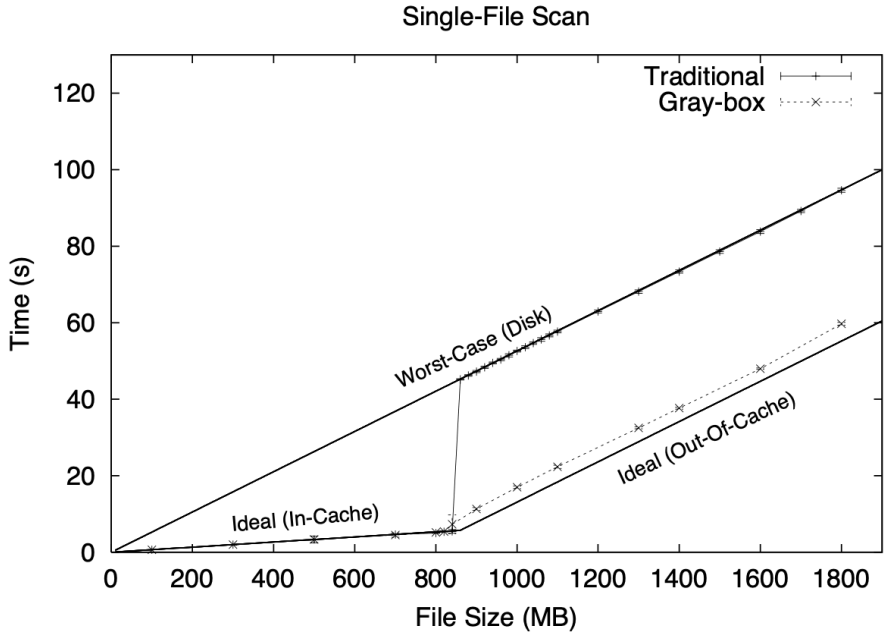
Can we use **measurement** to help?



An Idea: Gray Box Systems

Andrea called me, said "**gray box systems**"

- Idea: Use **measurement** in system itself to unveil properties of layers
- "Information and Control in Gray-Box Systems"
  Andrea Arpaci-Dusseau and Remzi Arpaci-Dusseau
  SOSP '01

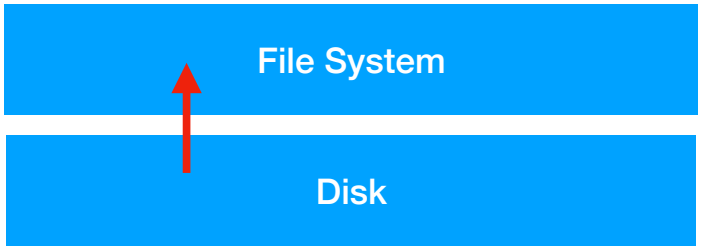


Example: Determine contents of **file cache**

- Probe (access a page and time it)
- Deduce (conclude which files must be in the cache)
- Benefit (schedule file accesses differently, to use in-cache data first)

Refining Gray Boxes

Original view: Looking down the stack



New idea: What if the system below could figure out things about the system above it?

Called **Semantically-Smart Disk Systems** (SSDs)

- Bad acronym (oops!) but interesting idea
- "Semantically-Smart Disk Systems" M. Sivathanu, V. Prabhakaran, I. Popovici, T. Denehy, A. Arpaci-Dusseau, R. Arpaci-Dusseau. FAST '03

Lesson

- **Ideas can be "close to right"; once you have one, keep thinking and refining**

Examples

“Improving storage system availability with D-GRAID”
M. Sivathanu, V. Prabhakaran, A. Arpaci-Dusseau, R. Arpaci-Dusseau. FAST ‘04

- A RAID system that understood which blocks belonged to which files
- Thus, could place some files within failure boundaries, to preserve in case of excess failure

“Life or Death at Block-Level”
M. Sivathanu, V. Prabhakaran, A. Arpaci-Dusseau, R. Arpaci-Dusseau. OSDI ’04

- A disk that could determine if blocks were live, and scrub those that were not

And a few others

- About caching (ISCA ’04), databases (FAST ’05), and even theory (FAST ’05)

Impact of **semantically-smart disks**

- ~750 citations across the body of work, and a few patents
- Many systems in the real world use block-level introspection

Part II: Technology-Driven Research

Beyond Measurement

Measurement-based approach works well

- e.g., we have written ~10 papers just analyzing the reliability of various storage systems(!)

But there are other methods to generate research

Another general method: **Technology-driven Research**

- Fundamental technologies are always being altered
- What is the impact on software systems?

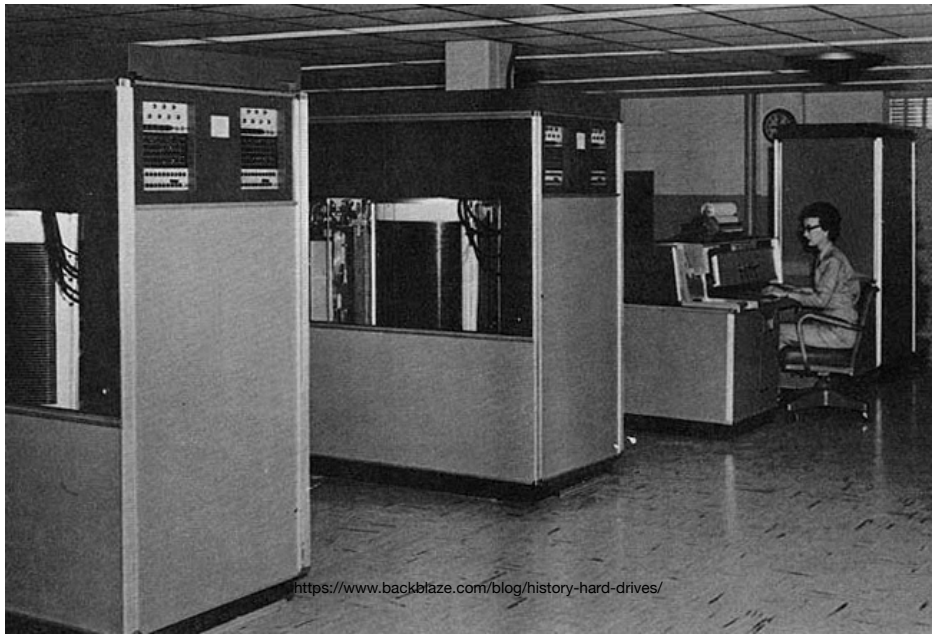
In The Beginning

The hard drive

- Invented in 1956 (IBM 305 RAMAC)

Specs

- 50 24-inch platters
- Stored about 5 MB
- Cost about $30k/month to use(!)



Smaller, Faster, Cheaper?

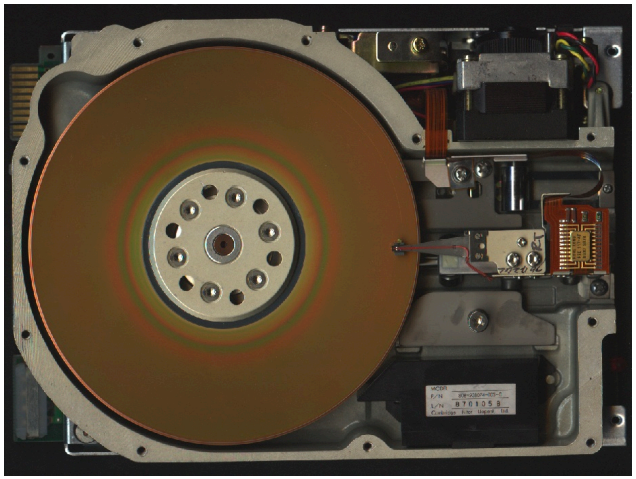
Next generation: IBM 1311

- 14-inch platters in packs
  (each about 2 MB)
- Only the size of a washing machine!

Personal Drives

Next leap: Into PCs

- 1980
- Shugart Tech 5MB drive
- 5.25 inch platter
- $1500 (about $5k today)



Company eventually changed name to **Seagate**…

Disruption: Solid-State

Solid-state: No more moving parts

- Flash-based Solid-State Drives (SSDs)
- 1988 invention (Fujio Masuoka)
- But not really a disk competitor until mid 2000s

Much different characteristics

- Disks: 10s of **milliseconds**
- SSDs: 10s-100s of **microseconds**

But cost is still much higher
than hard drives

- At least, for now…

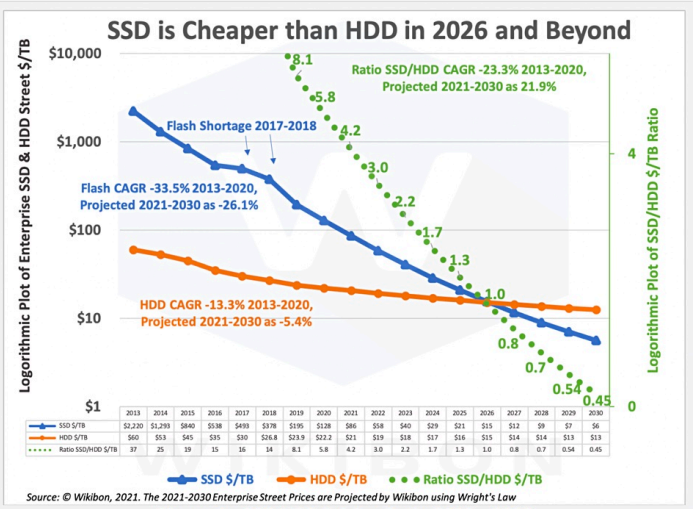


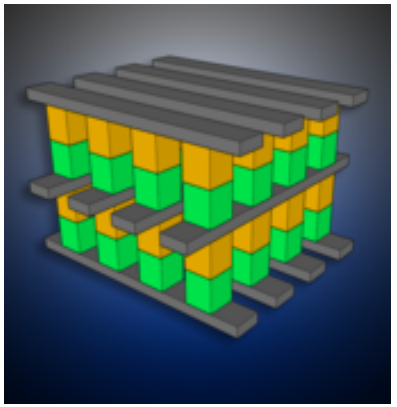
Figure 4 - SSD/HDD Pricing Ratio 2013 - 2030

Source: © Wikibon, 2021.

And The Disruptions Continue

Intel/Micro **X-Point** technology (“Optane”)

- In “Disk” and “Memory” form factors



Very expensive (now)

Promises even better performance

- Low latency operations (<10 microsecs)
- Bandwidth similar to flash-based SSDs

Remainder of Talk

Impact of SSDs on log-structured merge trees

- **WiscKey**

Impact of Optane on caching

- **Orthus**

Impact of Optane on file system structure

- **uFS**

WiscKey: LSMs Meet SSDs

Key-value stores are important

- Used in many important applications and services

Often implemented as **log-structured merge trees** (LSMs)

- Optimized for write-intensive workloads
- Widely deployed (BigTable, LevelDB, HBase, Cassandra, RocksDB…)

But, **technology has changed**

- Designed in the era of hard drives
- Do LSMs work well on SSDs?

LSM Background

Writes buffered in memory, then sorted and written into a file

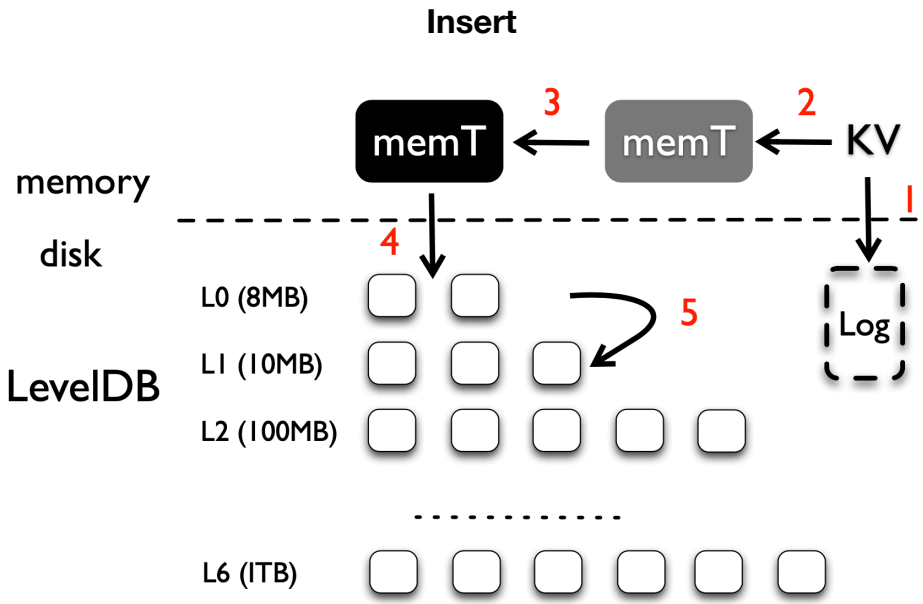
Over time, many such files collect

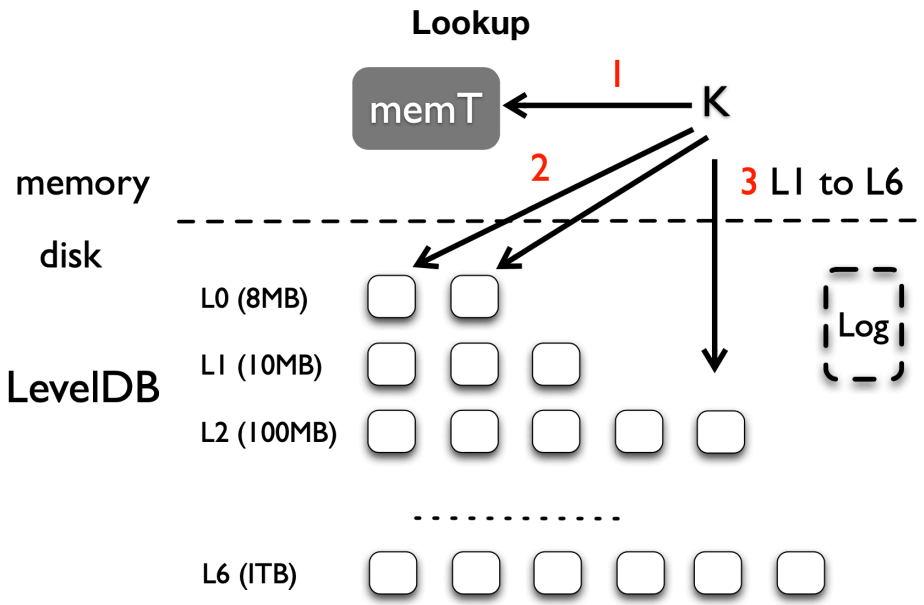
- Periodic compaction needed

Good for optimizing writes to disk

- All sequential

LSM Insert and Lookup





Inserts: May cause many compactions
Lookups: May traverse many levels of the tree

Problem: I/O Amplification

Random load: 100GB database

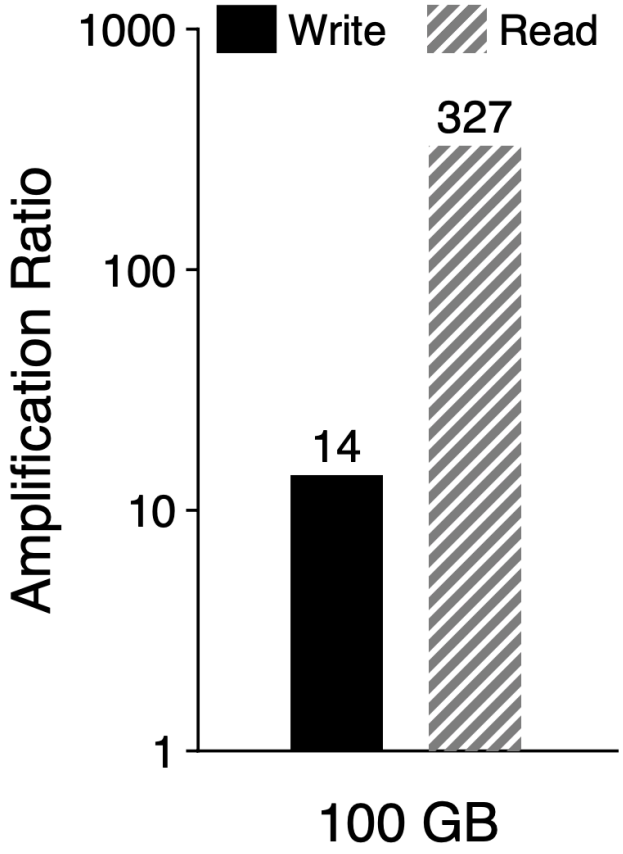
Random lookup: 100,000 lookups

Massive amplification

- Reads
- Writes

Made more sense for hard drive

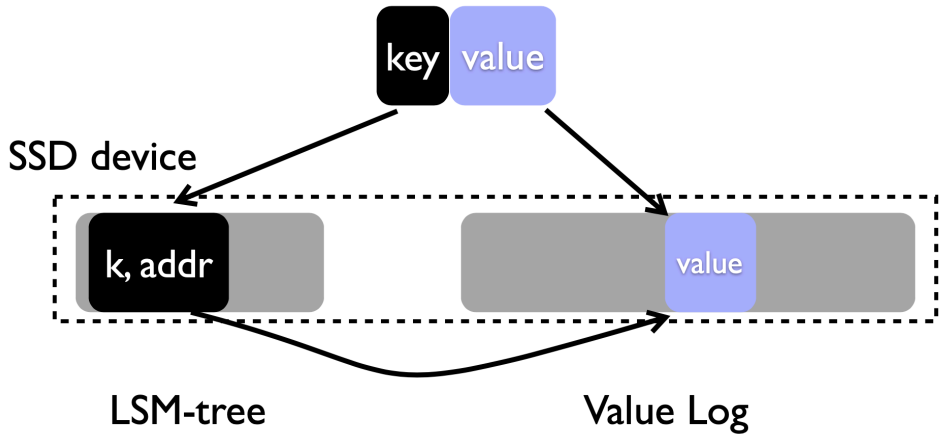
- Doing more I/O, but sequential
- But not for SSD…



Solution: WiscKey

Main idea

- **Separate keys from values**
- Keep keys in LSM and values in log



Results

- Significant reduction in I/O amplification
- Sometimes 100x faster than the state of the a

"WiscKey: Separating Keys from Values in SSD-conscious Storage"
Lanyue Lu, Thanumalayan Sankaranarayana Pillai, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. FAST '16

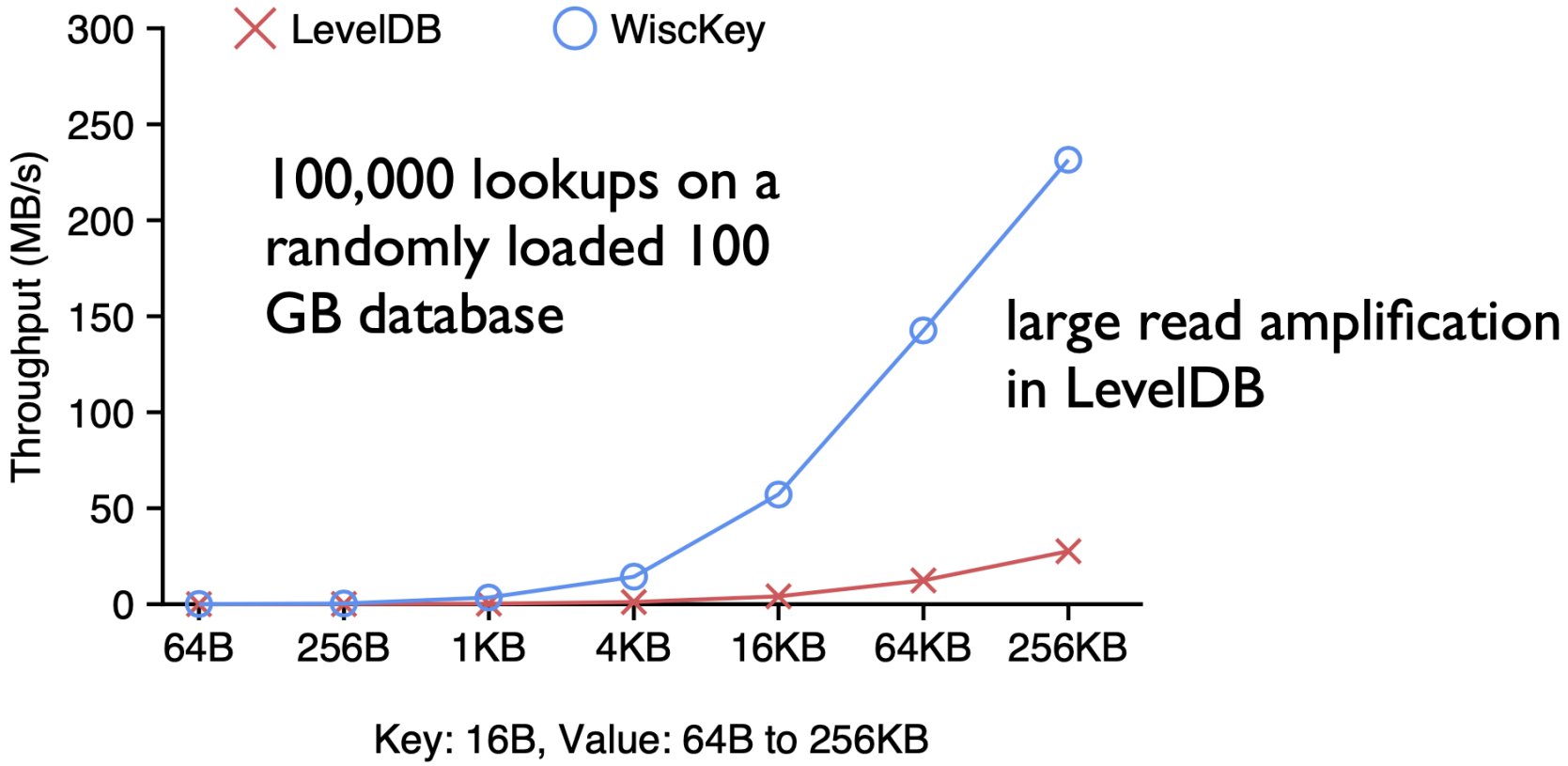
Lesson

- **Use a good idea again (in different context)**

Performance: Load

Key: 16B, Value: 64B to 256KB

Performance: Lookups



WiscKey Summary

WiscKey: an LSM-tree based key-value store

- Decouple sorting and garbage collection by separating keys from values
- SSD-conscious design (many other details)
- Significant performance gains across range of workloads
  (often 10x, sometimes 100x!)

A good example of technology-driven research

- And graduate student persistence!

Outline

Impact of SSDs on log-structured merge trees

- *WiscKey*

Impact of Optane on caching

- **Orthus**

Impact of Optane on file system structure

- uFS

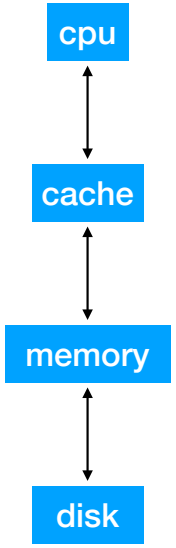
Classic Caching

Classic **Caching**

- “Hot” data moves towards CPU,
  “Cold” data moves away

Central **assumption**

- Performance further up the hierarchy
  is notably higher than lower down
- Data movement based on this assumption
  (e.g., hundreds of papers on replacement algs)



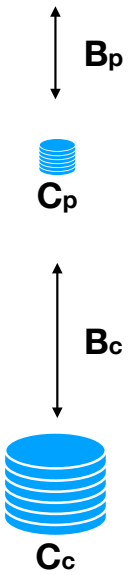
Question: Can We Do Better Than Caching?

Assumptions

- **Performance** device delivers B_p bandwidth, has C_p capacity
- **Capacity** device delivers B_c bandwidth, has C_c capacity

Normally, $B_p >> B_c$, and $C_c >> C_p$

- Thus, caching tries to deliver ~$\mathbf{B_p}$ performance while seeming to have $\mathbf{C_c}$ capacity - “an ideal device”



“Ideally, one would desire an indefinitely large memory capacity such that any particular binary digit number would be immediately available. It does not seem possible to achieve such a capacity. We are therefore forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible” -JvN (1946)

But What If B_p not >> B_c?

Assume

- H = Hit ratio

What is T, the time to access data, given H?

- $T = H * T_{hit} + (1 - H) * T_{miss}$

where…

- $T_{hit} = 1 / B_p$
- $T_{miss} = 1 / B_p + 1 / B_c = (B_p + B_c) / (B_p * B_c)$

Solving for bandwidth (inverse of T):

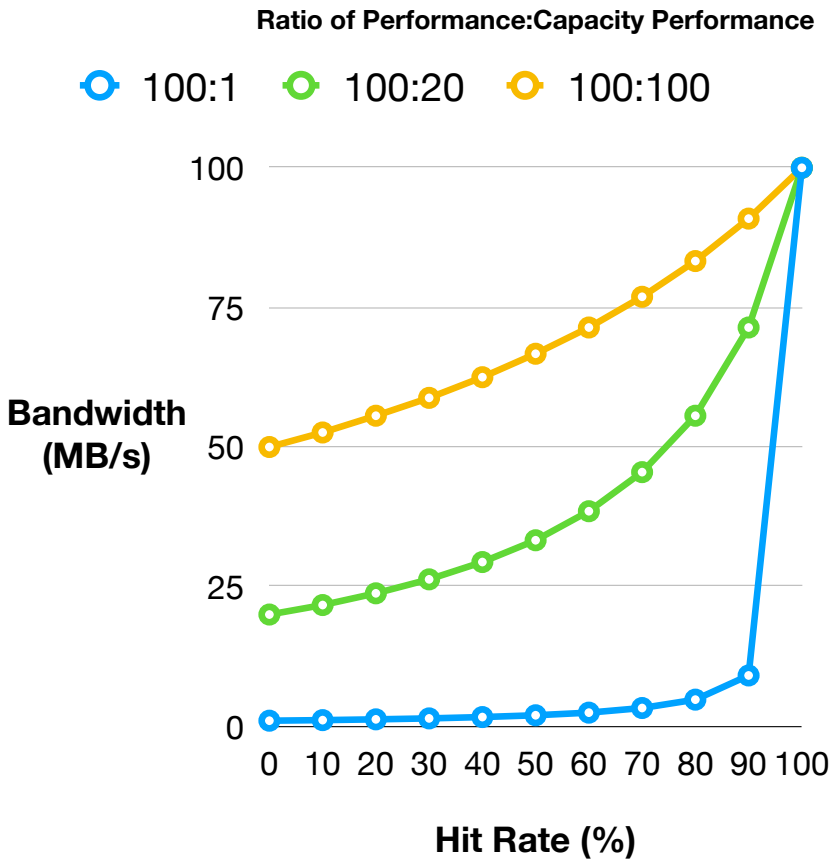
- $B_{overall} = (B_p * B_c) / (H * B_c + (1 - H) * (B_p + B_c))$

(Assumes just one request at a time)

Model: Results

Caching results

- Higher hit rate, approach max perf of performance device
- With higher bandwidth from “capacity” device, low miss rates are more tolerable



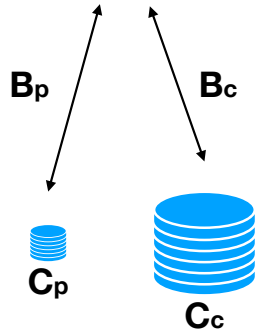
A Different Approach: Splitting (Offloading)

Same as before

- Performance and Capacity devices

Splitting (Offloading): Alternative to caching

- Directs some traffic to one device, other traffic to another



Splitting Model

Assume

- S = Split ratio

Solve for T, time to service request split across devices

- T = max(S * Tp, (1 - S) * Tc)

where

- Tp = 1/Bp
- Tc = 1/Bc

Solve for throughput

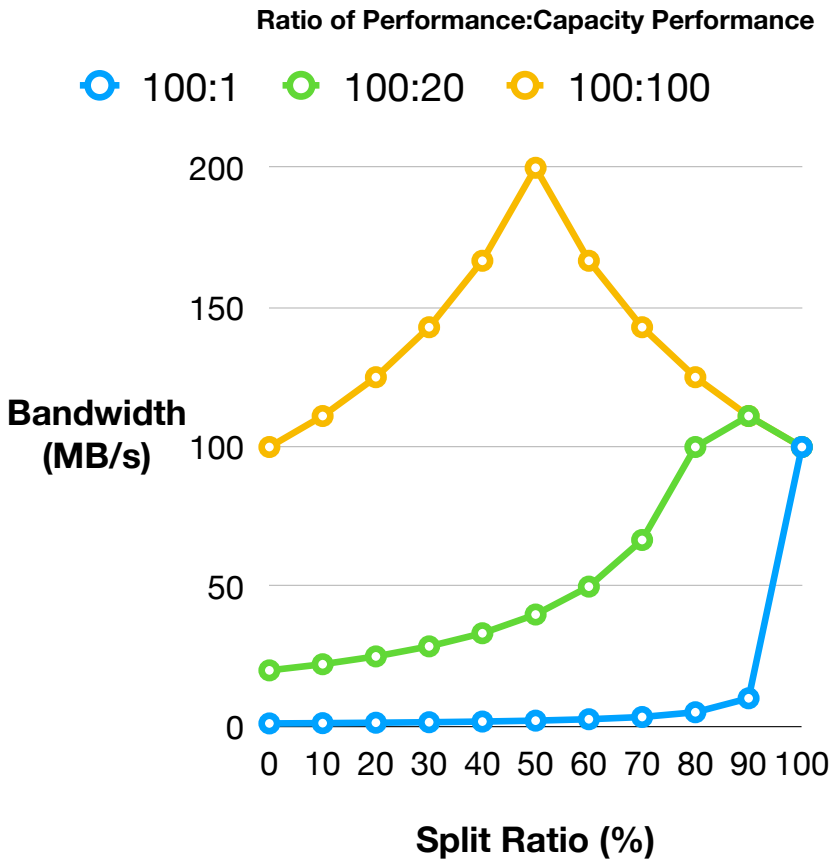
- Boverall = 1 / max(S/Bp, (1-S)/Bc)

(Assumes many outstanding requests)

Splitting: Results

Splitting results

- Getting “right” split delivers sum of performance of both devices
- Again, larger sensitivity to correct split when gap in performance is large (100:1 vs 100:100)



Caching vs. Splitting

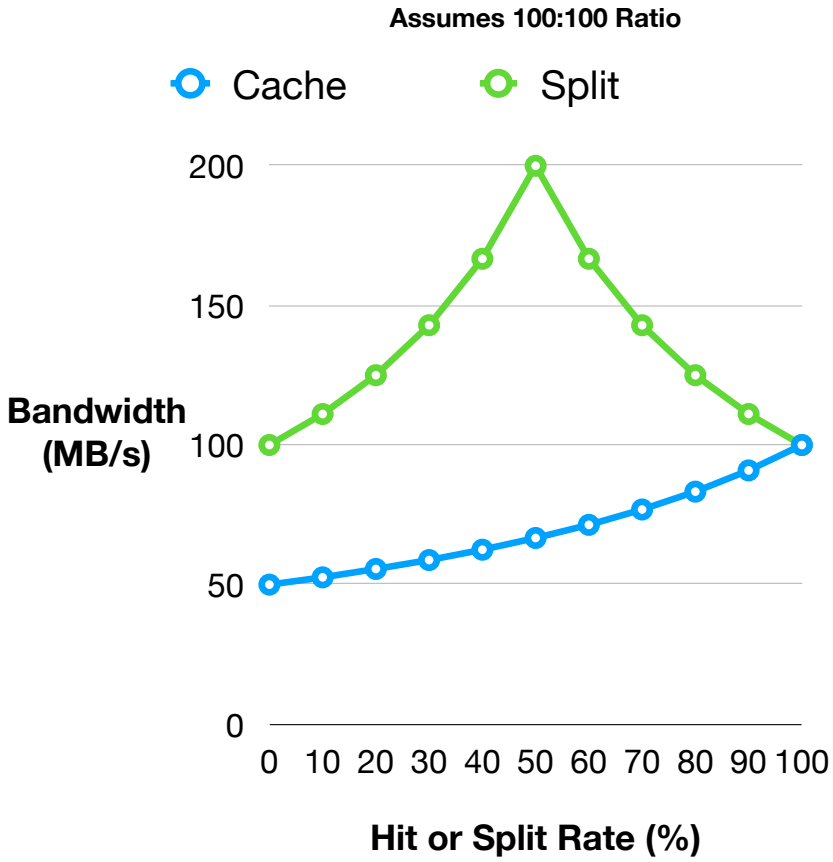
When B_c nears B_p, caching gives away a lot of performance

(in this example, 2x!)

Result: Capacity devices can actually deliver performance

Lesson

- **Modeling can be useful**



Solution

How to take advantage of multiple devices in hierarchy?

- Example: **Flash-based SSDs and Optane**

Approach: **Non-hierarchical caching**

- Use caching as base approach
- Add **read offloading** (directing reads to capacity device) as needed to maximize bandwidth of both devices

"The Storage Hierarchy is Not a Hierarchy: Optimizing Caching on Modern Storage Devices with Orthus" Kan Wu, Zhihan Guo, Guanzhou Hu, Kaiwei Tu, Ramnatthan Alagappan, Rathijit Sen, Kwanghyun Park, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau. FAST 2021

Classic Caching

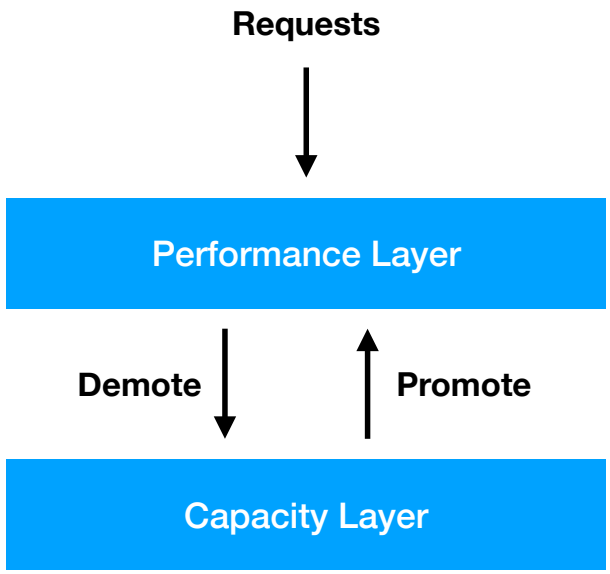
Always admit to performance layer on misses

- Even if cache bandwidth is fully saturated

Send all requests to performance tier

- Even if bandwidth is available in capacity tier

Decisions are **static** in nature



Non-Hierarchical Caching

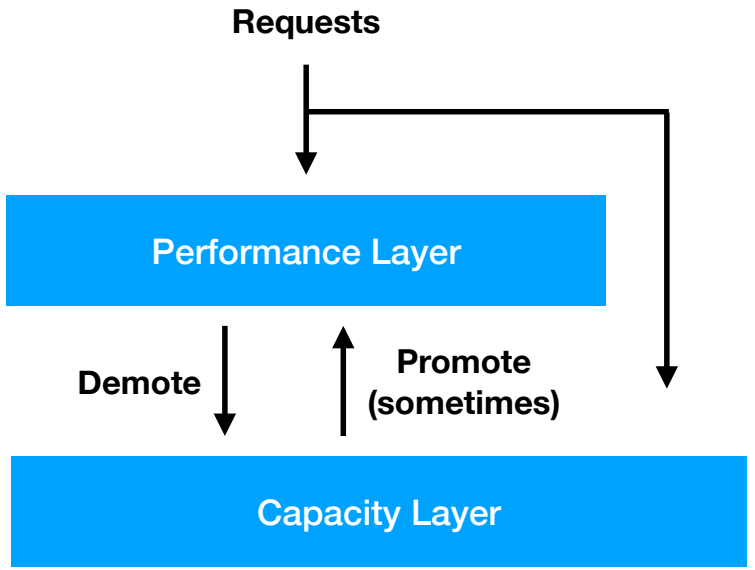
Decides whether to admit data into upper tier

- Promote sometimes (not always) on misses

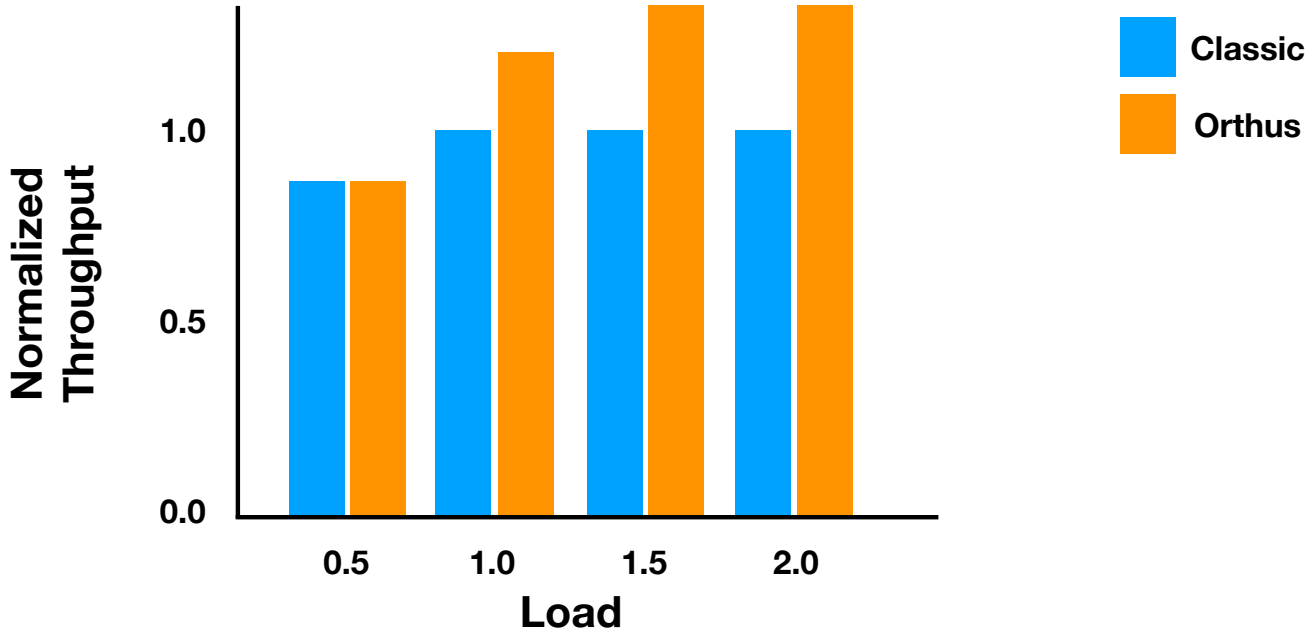
Decides to direct (some) reads to lower tier

Decisions made dynamically

- Change over time



Results



Classic caching: Bounded by single-device performance

Orthus: Utilize full device bandwidth from both

Outline

Impact of SSDs on log-structured merge trees

- *WiscKey*

Impact of Optane on caching

- *Orthus*

Impact of Optane on file system structure

- **uFS**

Problem: Devices Fast, Kernel Less So

System call handling: A few microseconds

In hard-drive era

- OK, because disk took a few milliseconds

In modern era

- Not OK, because device can take a few microseconds

Alternate Architectures

Semi-microkernel

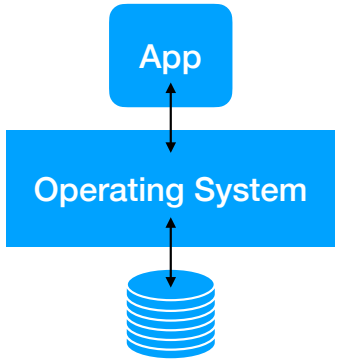
- Based on old “microkernel” approach
- But, not a “full” microkernel - instead, just single subsystem is hoisted into user space
- Subsystem built in user space, can directly control device
- Networking world (e.g., Google’s Snap) has been exploring this approach

Filesystem semi-microkernel

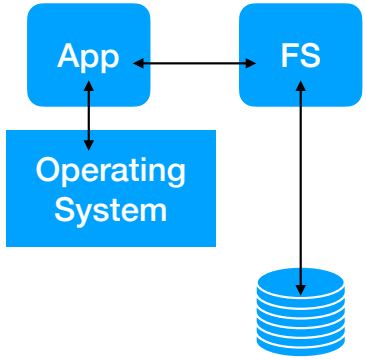
- What we investigate here

Lesson

- **Explore ideas from other fields**



Classic OS



FS Semi-microkernel

Advantages

Developer Velocity

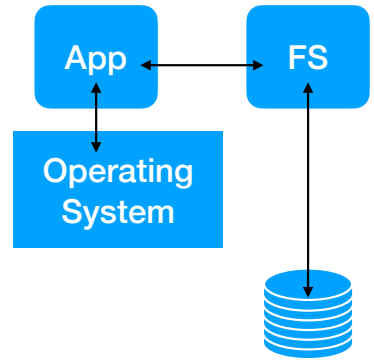
- Tools and libraries for “application” code
- Rapidly adopt new hardware and tailor to apps

Performance

- Optimize for device access (avoid OS overhead)
- Scale filesystem independently from apps

Simplify sharing and permission

- Untrusted aps cannot access the device



FS Semi-microkernel

uFS: A Filesystem Semi-Microkernel

"Scale and Performance in a Filesystem Semi-Microkernel" Jing Liu, Anthony Rebello, Yifan Dai, Chenhao Ye, Sudarsun Kannan, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau. SOSP 21

Build for performance and scalability from scratch

- Fully functional with crash consistency
- Employs lock-free access for main data structures
- Dynamically partitions inodes to filesystem threads
- Adapt # of uFS cores according to filesystem demands
- Implemented by C++ (~35K LoC)

uFS offers good base performance and excellent scalability

- 1.2x-4.6x throughput compared to ext4 when running 10 LevelDB instances

Lessons Summary

When working on something, ask: how can I be the best?

Never underestimate the power of textbooks

Use papers you like as inspirations

Sometimes ignore advice

Keep eyes open when doing research

Read widely and take notes

Ideas can be “close to right”; keep thinking and refining

Always think about what can be measured, and how to learn from it

Ask how new technologies change how we build systems

Use good ideas again (in different contexts)

Modeling can be useful

Explore ideas from other (sub)fields

Last Lesson: Thank People

Co-conspirator: **Andrea Arpaci-Dusseau**

Ph.D. Students

- **Muthian Sivathanu**
- **John Bent**
- **Vijayan Prabhakaran**
- **Nathan Burnett**
- **Tim Denehy**
- **Todd Jones**
- **Ina Popovici**
- **Lakshmi Bairavasundaram**
- **Nitin Agrawal**
- **Haryadi Gunawi**
- **Joe Meehean**
- **Swami Sundararaman**
- **Sriram Subramanian**
- **Yiying Zhang**
- **Yupu Zhang**
- **Thanh Do**
- **Vijay Chidambaram**
- **Lanyue Lu**
- **Tyler Caraza-Harter**
- **Thanu Pillai**
- **Suli Yang**
- **Leo Arulraj**
- **Zev Weiss**
- **Jun He**
- **Ram Alagappan**
- **Aishwarya Ganesan**
- **Yuvraj Patel**
- Jing Liu
- Kan Wu
- Anthony Rebello
- Yifan Dai
- Chenhao Ye
- Guanzhou Hu
- Kaiwei Tu
- Vinay Banakar
- Surabhi Gupta

And many more

- Post-docs!
- Masters and undergrads!