15-712:
Advanced Operating Systems & Distributed Systems

Scale and Performance in a Distributed File System

Prof. Phillip Gibbons

Spring 2023, Lecture 8

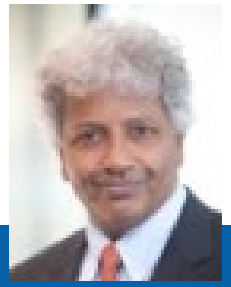
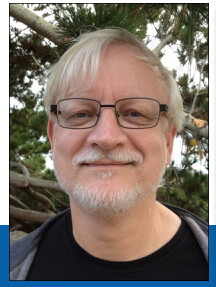
"Scale and Performance in a Distributed File System"

John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, Michael J. West 1988

- John Howard (Project Lead, retired from Sun)
- Michael Kazar (CTO Avere, IEEE Info Stor. Sys Award)
  Co-founded Transarc (commercial AFS, now IBM)
- Sherri Menees (community activist, Redmond WA)
- David Nichols (Microsoft, founded Live Meeting)
- M. Satyanarayanan (CMU, ACM & IEEE Fellow)
- Robert Sidebotham (Google)
  - Inventor of volumes
- Michael West (IBM, deceased)







Andrew File System Developers Receive 2016 ACM Software Sys...
Watch later
Share
John H. Howard
Michael L. Kazar
David A. Nichols
Sherri M. Nichols
Michael J. West
Mahadev
Robert N. Sidebotham
Alfred Z. Spector
MORE VIDEOS


Andrew File System Developers Receive 2016 ACM Software Sys...
Watch later
Share
John H. Howard
Michael L. Kazar
David A. Nichols
Sherri M. Nichols
Michael J. West
Mahadev
Robert N. Sidebotham
Alfred Z. Spector
MORE VIDEOS

“Scale and Performance in a Distributed File System”

John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, Michael J. West 1988

SigOps HoF citation (2008):

None given ☹

Andrew v.1

- **Morris, Satyanarayanan, Conner, Howard, Rosenthal, Smith "Andrew: A Distributed Personal Computing Environment" [CACM 1986]**
  - Project started in 1983
- **Prototype had 400 users, 100 workstations, 6 servers**
- **Cache whole file locally: client process (Venus) contacts server (Vice) only on file open/close**
- **Each directory had a single server site for updates**
- **File location: Navigate server directory with stub directories pointing to other servers**

Why Whole File Caching

vs. remote-open (e.g. NFS), partial caching

- **Workload locality makes caching attractive**
- **Servers only contacted on opens/closes**
- **Most files read in their entirety** [SOSP'85 study of 4.2BSD]
  - Can use efficient bulk data transfer protocols
- **Disk caches retain contents across (frequent) reboots**
- **Simplifies cache management**

Drawbacks?

- Workstations must have disks
- Files larger than local disk cannot be accessed at all
- Can't support 4.2BSD read/write semantics
- Ill-suited for a distributed DB (exhibits too much write-sharing)

Qualitative Observations on v.1

- **Commands were noticeably slower than w/local files**

"The performance was so much better than that of the heavily loaded timesharing systems…that our users willingly suffered!"

- **Performance anomaly**
  - Apps used the "stat" primitive to test for presence of files or to obtain status info before opening them – resulted in too many client-server interactions

- **Difficult to operate & maintain**
  - Use of dedicated process per client on each server: excessive context switches, page faults, resource exhaustion
  - Kernel RPC support: network-related resource exhaustion
  - Stub directories: difficult to migrate directories between servers

Andrew Benchmark

"Although we do not demonstrate any statistical similarity between these file references and those observed in real systems,..."

MakeDir	Constructs a target subtree that is identical in structure to the source subtree.
Copy	Copies every file from the source subtree to the target subtree.
ScanDir	Recursively traverses the target subtree and examines the status of every file in it. It does not actually read the contents of any file.
ReadAll	Scans every byte of every file in the target subtree once.
Make	Compiles and links all the files in the target subtree.

Time in secs

70 files, 200 KB total

Table I. Stand-alone Benchmark Performance

	Machine type		
Benchmark phase	Sun2	IBM RT/25	Sun3/50
Overall	1054 (5)	798 (20)	482 (8)
MakeDir	16 (1)	13 (1)	10 (0)
Copy	40 (1)	37 (2)	31 (2)
ScanDir	70 (4)	51 (9)	44 (5)
ReadAll	106 (2)	132 (8)	51 (0)
Make	822 (2)	566 (11)	346 (1)

Performance Observations on v.1

- **File Cache: Avg hit ratio 81%. File Status Cache: Avg hit ratio 82%**

Table II. Distribution of Vice Calls in Prototype

		Call distribution						
Server	Total calls	TestAuth (%)	GetFileStat (%)	Fetch (%)	Store (%)	SetFileStat (%)	ListDir (%)	All others (%)
cluster0	1,625,954	64.2	28.7	3.4	1.4	0.8	0.6	0.9
cluster1	564,981	64.5	22.7	3.1	3.5	2.8	1.3	2.1
cmu-0	281,482	50.7	33.5	6.6	1.9	1.5	3.6	2.2
cmu-1	1,527,960	61.1	29.6	3.8	1.1	1.4	1.8	1.2
cmu-2	318,610	68.2	19.7	3.3	2.7	2.3	1.6	2.2
Mean		61.7	26.8	4.0	2.1	1.8	1.8	1.7
		(6.7)	(5.6)	(1.5)	(1.0)	(0.8)	(1.1)	(0.6)

  - TestAuth: validates cache entries
  - GetFileStat: status info about files not in cache

Performance Observations on v.1

Table III. Prototype Benchmark Performance

	Overall benchmark time		Time per TestAuth call	
Load units	Absolute (s)	Relative (%)	Absolute (ms)	Relative (%)
1	1789 (3)	100	87 (0)	100
2	1894 (4)	106	118 (1)	136
5	2747 (48)	154	259 (16)	298
8	5129 (177)	287	670 (23)	770
10	7326 (69)	410	1050 (13)	1207

scaling poorly

Recall: Stand-alone (i.e., not distributed) time = 1054

Significant performance gains possible if:

- Reduce the frequency of cache validity checks (TestAuth)
- Reduce the number of server processes (reduces context switches)
- Require clients rather than servers to do pathname traversals
- Balance server usage by reassigning users

AFS v.2: Cache Management Changes

- **Cache contents of directories & symbolic links, not just files**
- **On open, assume cached entries are/stay valid**
  - Server does **Callback** to client cache before allowing others to update the file
- **Pros?**
  - Reduces load on servers
  - Enables pathnames resolved locally
- **Cons?**
  - Client caches & Servers must maintain callback state
  - Such state may become inconsistent

File Consistency

- **Writes to an open file by a client process are visible to all other local processes immediately but invisible non-locally**
- **Writes become visible on file close**
  - Changes visible to any new open, invisible to already open
- **All other file ops are visible everywhere on op completion**
  - E.g., protection changes
- **No implicit locking: apps have to do their own synchronization**

Discussion: Summary Question #1

- **State the 3 most important things the paper says.** These could be some combination of their motivations, observations, interesting parts of the design, or clever parts of their implementation.

Changes to Name Resolution & Low-Level Storage Representation

- **Each Vice (server) file or directory identified by Fid**
  - (32-bit volume #, 32-bit vnode #, 32-bit uniquifier)
  - Vnode info includes the file's BSD inode
  - Files accessed by inodes: Servers unaware of pathnames
- **Volume Location Directory replicated on each server**
- **Can migrate files between servers w/o invalidating clients' cached directory contents**

Communication & Server Process Structure

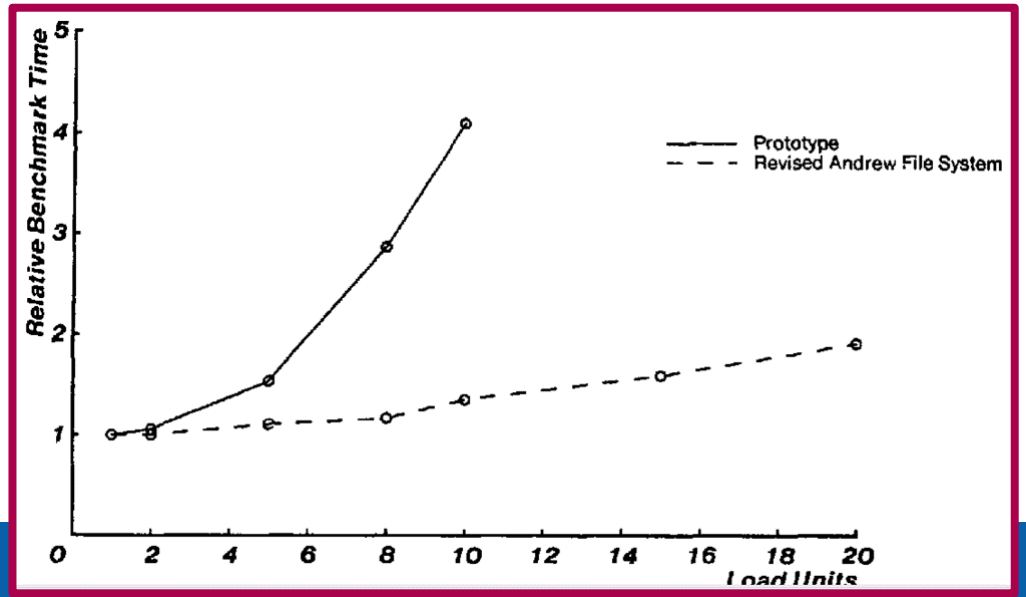
- **Single server process to service all its clients**
  - ~5 Lightweight processes (LWPs) within a process
  - LWP bound to a client only for duration of a single server op
  - Each client also uses LWPs
- **RPC code no longer in kernel**
  - But also argue that most Venus/server code SHOULD be in kernel

Scalability Improvements

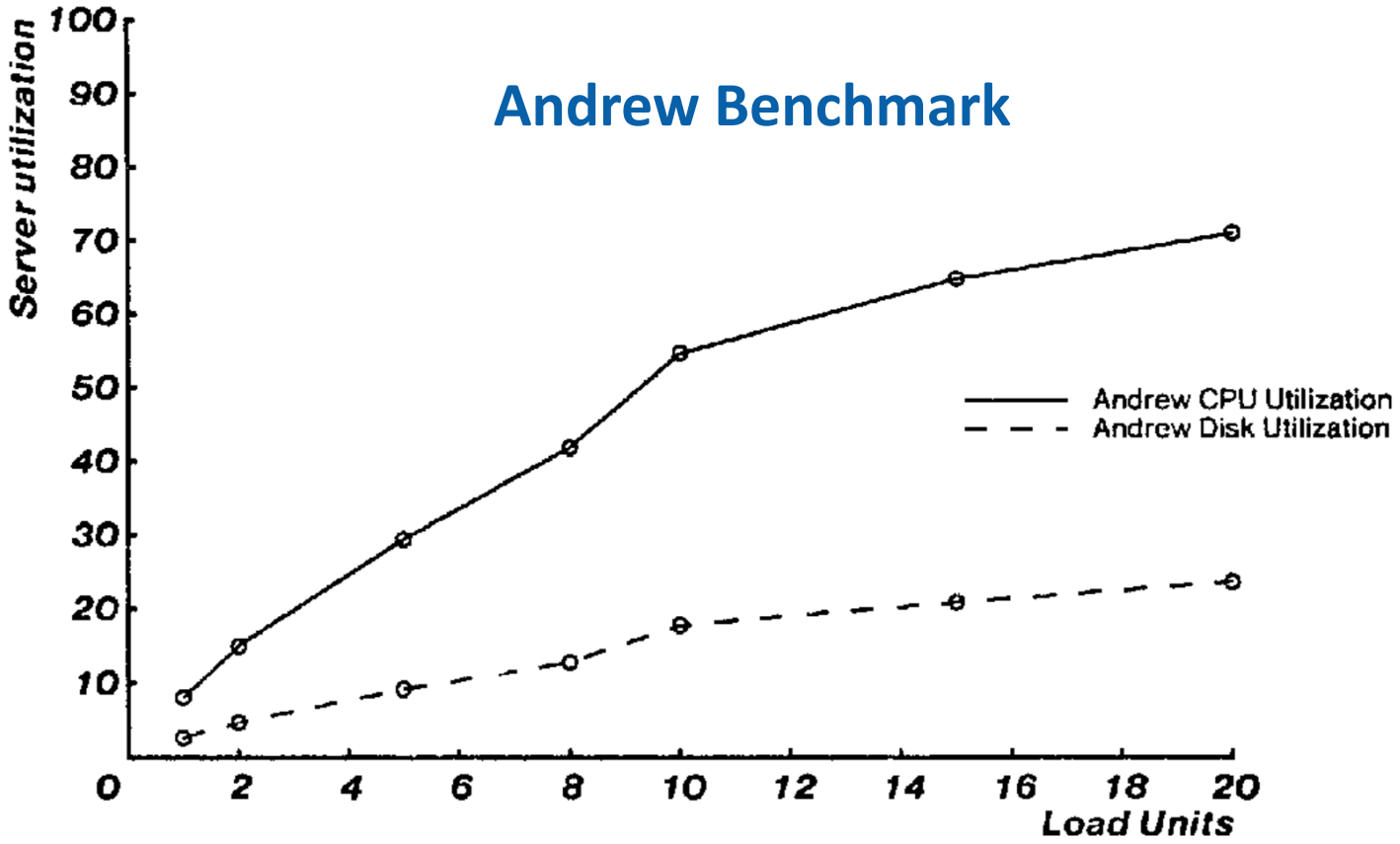
Table VI. Andrew Benchmark Times

Load units	Overall time (in seconds) Absolute	Relative (%)	Time for each phase (in seconds) MakeDir	Copy	ScanDir	ReadAll	Make
1	949 (33)	100	14 (1)	85 (28)	64 (3)	179 (14)	608 (16)
2	948 (35)	100	14 (1)	82 (16)	65 (9)	176 (13)	611 (14)
5	1050 (19)	111	17 (1)	125 (30)	86 (0)	186 (17)	637 (1)
8	1107 (5)	117	22 (1)	159 (1)	78 (2)	206 (4)	641 (6)
10	1293 (70)	136	34 (9)	209 (13)	76 (5)	200 (7)	775 (81)
15	1518 (28)	160	45 (3)	304 (5)	81 (4)	192 (7)	896 (12)
20	1823 (42)	192	58 (1)	433 (45)	77 (4)	192 (6)	1063 (64)

Stand-alone = 798 secs



Utilization Improvements

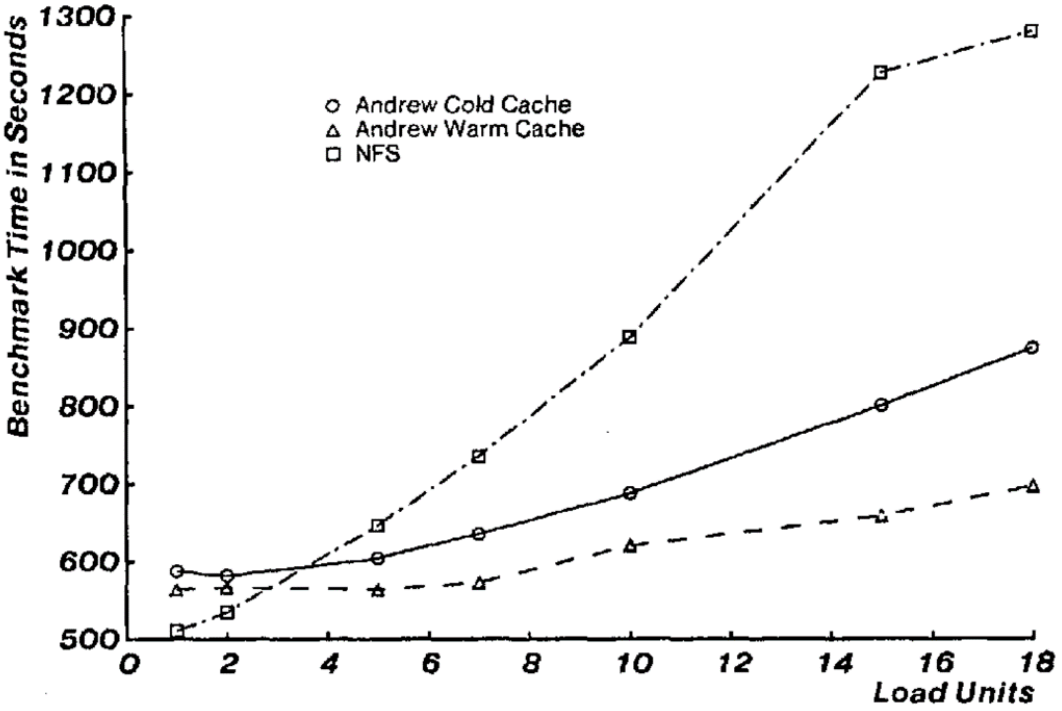


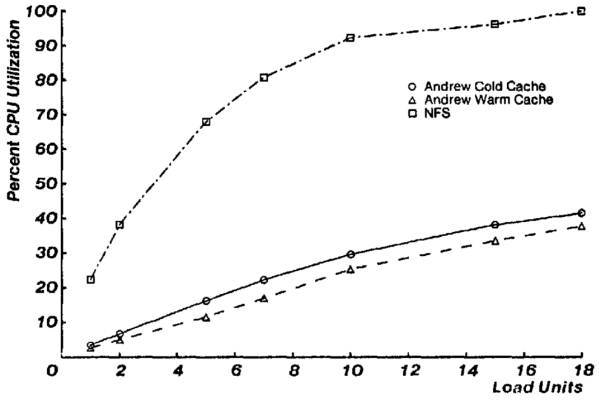
Also showed improved actual utilization during work hours

NFS

- **Once file open, remote site treated like local disk**
  - Return to server for each new page accessed (does prefetch)
  - Caches file pages locally in memory
- **No transparent file location facility; mounted individually**
- **Client & server components are in kernel**
- **Caches inodes locally in memory**
  - Performs validity check on file open
  - Suppressed for directory inodes if checked in last 30 seconds
- **File consistency is messier**

AFS vs. NFS





- **Single server experiments using Andrew Benchmark**
- **Some NFS clients terminated prematurely in final phase**
  - Lost RPC reply packets at high network load
- **ReadAll at load 15 & 18: NFS=211-215 vs. AFS=48**

AFS vs. NFS Latency

Table XIV. Latency of NFS and Andrew

File size (bytes)	Time (milliseconds)			
	Andrew cold	Andrew warm	NFS	Stand-alone
3	160.0 (34.6)	16.1 (0.5)	15.7 (0.1)	5.1 (0.1)
1,113	148.0 (17.9)			
4,334	202.9 (29.3)			
10,278	310.0 (53.5)			
24,576	515.0 (142.0)		15.9 (0.9)	

"Low latency is an obvious advantage of remote-open FS"

- **SMALL file sizes studied**
  - Benchmark: avg size is 3 KBs
- **Network Traffic: At load 1, NFS is 2.67x AFS**

Aside: AFS vs. NFS Latency by Workload

Workload	NFS	AFS	AFS/NFS
1. Small file, sequential read	$N_s \cdot L_{net}$	$N_s \cdot L_{net}$	1
2. Small file, sequential re-read	$N_s \cdot L_{mem}$	$N_s \cdot L_{mem}$	1
3. Medium file, sequential read	$N_m \cdot L_{net}$	$N_m \cdot L_{net}$	1
4. Medium file, sequential re-read	$N_m \cdot L_{mem}$	$N_m \cdot L_{mem}$	1
5. Large file, sequential read	$N_L \cdot L_{net}$	$N_L \cdot L_{net}$	1
6. Large file, sequential re-read	$N_L \cdot L_{net}$	$N_L \cdot L_{disk}$	$\frac{L_{disk}}{L_{net}}$
7. Large file, single read	L_{net}	$N_L \cdot L_{net}$	N_L
8. Small file, sequential write	$N_s \cdot L_{net}$	$N_s \cdot L_{net}$	1
9. Large file, sequential write	$N_L \cdot L_{net}$	$N_L \cdot L_{net}$	1
10. Large file, sequential overwrite	$N_L \cdot L_{net}$	$2 \cdot N_L \cdot L_{net}$	2
11. Large file, single write	L_{net}	$2 \cdot N_L \cdot L_{net}$	$2 \cdot N_L$

- **File sizes N_s, N_m, N_L ; N_L > Local memory**
- **Cold Cache; Latencies: $L_{net} > L_{disk} > L_{mem}$**

From [Operating Systems: Three Easy Pieces]

AFS vs. NSF: Performance Conclusions

- **Scales much better than NFS**
  - Claim: Scales to 500-700 clients

> "We are certain that further growth will stress our skill, patience, and ingenuity."

- **Small-scale performance is nearly on par**

- **NFS in kernel, AFS was not**

> "There is thus untapped potential for improved performance in Andrew, whereas we see no similar potential in NFS."

- **Supports well-defined consistency semantics, security, operability**

Discussion: Summary Question #2

- **Describe the paper's single most glaring deficiency.** Every paper has some fault. Perhaps an experiment was poorly designed or the main idea had a narrow scope or applicability.

Disadvantages / Room for Improvement

- Clients need disks. Can't access files larger than local disk
- Strict emulation of 4.2BSD semantics is not possible
- Can't build a distributed database using AFS
- Querying/updating authentication & network DBs
- Better fault tolerance
- Put in kernel & use a standard kernel intercept mechanism
- Allow users to define their own protection groups
- Provide replication for writable files
- Monitoring, fault isolation, diagnostics tools
- Decentralized administration & physical dispersal of servers

Volume

- **Collection of files forming a partial subtree of the name space**
- **Glued together at Mount Points (invisible to name space)**
- **Resides within a single disk partition on a server**
- **On-the-fly (atomic) migration:**
  - Create a Clone (copy-on-write snapshot)
  - Construct machine-independent rep of Clone
  - Regenerate at remote site
  - Any updates during migration patched using incremental clone
- **User assigned a volume; each volume has a quota**
- **Read-only volume replicas improve availability & efficiency**
  - Enable orderly release of SW updates

Advantages of Volumes

- **Provide a level of operational transparency not supported by other file systems**
  - From an operation standpoint, system is a flat space of named volumes
- **Quite valuable: Volume quotas & Ease of migration**
- **Backup mechanism is simple, efficient, non-disruptive**
  - Read-only clone transferred in background to staging machine
  - 24 hours of backup in read-only subtree in user's home directory

"Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency"

Cary G. Gray, David R. Cheriton 1989

- Cary Gray (Stanford PhD, Whitman College)

- David Cheriton (Stanford, emeritus)
  - SIGCOMM Lifetime Achievement Award
  - Angel investor in Google, VMware, many more. Net worth $9.8B!!

SigOps HoF citation (2009):

This paper pioneered through its analysis of the Leases mechanism, which has become one of the most widely-used mechanisms for managing distributed caches. The paper is particularly striking for its careful analysis of the semantics of leases, its detailed experiments, and its thoughtful discussion of fault-tolerance issues.

Leases

- **Time-based mechanism that provides efficient consistent access to cached data in distributed systems**
  - Only leaseholder(s) can read/write the data
  - To write the data, must first get approval from all leaseholders (or wait for their leases to expire)

- **Non-Byzantine failures affect performance, not correctness**
  - If can't communicate, wait for its leases to expire
  - On recovery, server honors leases granted before crash

- **Assumes write-through caches (but can extend to write-back)**

- **Leases of short duration (10s) provide good performance**
  - Longer term if accessed repeatedly with little write-sharing

Leases & AFS

- **AFS v.1 akin to lease term=0**

- **AFS v.2 akin to lease term=infinity**
  - Relies on server to notify client when cached data changes

But: Fails to guarantee consistency after communication failure

- If server can't reach client, updates still proceed
- Client doesn't learn of inconsistency until contacts server
- Polling every 10 minutes limits window of inconsistency

"Our work suggests that a short-term lease would be adequate for Andrew…to avoid inconsistency…and [provide] other benefits."

Why didn't AFS adopt leases?

What about: Lease vs. Time-to-live (TTL)?

Wednesday's Paper

File Systems and Disks (III)

"The Design and Implementation of a Log-Structured File System"

Mendel Rosenblum, John K. Ousterhout 1992

BACKUP SLIDES

Jim Morris Reflects on AFS at 25

Why WWW beat Global AFS

- Kernel mods were deadly.
  - Forgot Window Manager Lesson
- Consistency was overrated.
  - "Read-only" Web was useful.
  - File close is arbitrary check-point.
- URL was obvious, but crucial.
- HTTP & Browser blindsided us.
- WWW was a paradigm shift.
  AFS was incremental.

Carnegie Mellon
SILICON VALLEY