CS 194: Distributed Systems
Other Distributed File Systems

Scott Shenker and Ion Stoica
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720-1776

1

Four Other Distributed File Systems

Focus on goals, not details

- Plan 9: turn everything into a file
- xFS: turn every machine into a file server
- SFS: don't trust anything but the file server
- SUNDR: don't even trust the file server

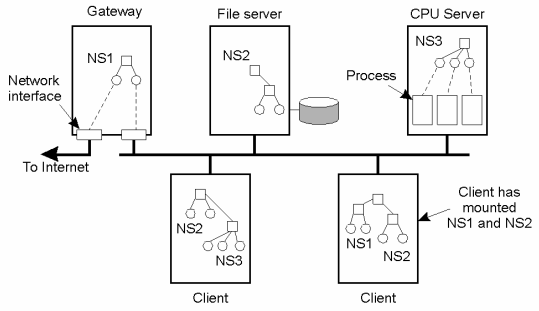
2

Plan 9

- Developed at Bell Labs by the UNIX group
- Not a new distributed file system…
- …but a new file-based distributed system
- Every resource looks like a file
  - Like UNIX, but more consistently
- Clients can locally mount a name space offered by a server
  - NFS style, not AFS style

3

Plan 9 Organization



4

Plan 9 Communications

- Uses custom protocol 9P
- Network interfaces represented by collection of special files
- TCP connection represented by subdirectory with:
  - Ctl: write protocol-specific control commands
  - Data: read and write data
  - Listen: accept incoming connection setup requests
  - Remote: information about other side of connection
  - Status: diagnostic information on current status

5

Plan 9 Example

- Open a telnet connection to 192.31.231.42 using port 23
  - Write "connect 192.31.231.42!23" to file ctl
- To accept incoming telnet connections
  - Write "announce 23" to ctl
- Window system offers files:
  - /dev/mouse: mouse position
  - /dev/cons: keyboard input

6

Plan 9 Naming

- Each process has own private namespace constructed by mounting remote name spaces
- File identified system-wide by four-tuple:
  - path: unique file number (relative to server)
  - version
  - device number (identifies its server)
  - type

7

Plan 9 Synchronization

- UNIX file sharing semantics:
  - All changes sent to server
- Caching is write-through

8

Plan 9 Rationale

- Distributed systems are hard
- Distributed file systems are a solved problem
- Build a distributed system that looks like a file system
- Impact: not clear....

9

xFS

- Developed at Berkeley for the NOW project ~1995
  - NOW = Network of workstations
- All machines can be a server, no server is “special”
- Like modern P2P systems in their symmetric and scalable design
- Unlike modern P2P systems in their assumptions about trust, churn, and bandwidth
  - Trusted, stable machines connected by high-bandwidth LAN

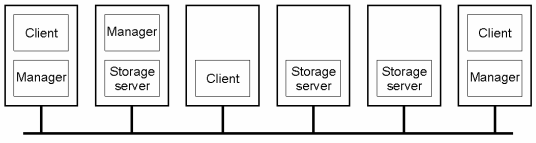
10

xFS Processes

- Storage server: stores parts of files on local node
- Metadata manager: keeping track of where blocks of files are stored
- Client: accepts user commands

11

Overall xFS Architecture



12

xFS Built on Three Ideas

- RAID
- Log-based File System
- Cooperative caching

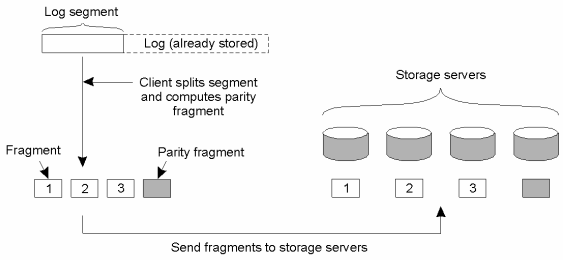
13

RAID

- Partitions data into k blocks and a parity block
- Stores blocks on different disks
- High bandwidth: simultaneous reading/writing
- Fault tolerance: can withstand failures

14

RAID on xFS



15

Log-Structured File System (LFS)

- Developed at Berkeley (~1992)
- Updates written to a log
- Updates in logs asynchronously written to file system
- Once written to file system, updates removed from log
- Advantages:
  - better write performance (sequential)
  - failure recovery

16

RAID + LFS = Zebra

- Large writes in LFS makes writes in the RAID efficient
- Implements RAID in software
- Log-based striping

17

Locating Files

- Key challenge: how do you locate a file in this completely distributed system?
- Manager map allows clients to determine which manager to contact
- Manager keeps track of where file is

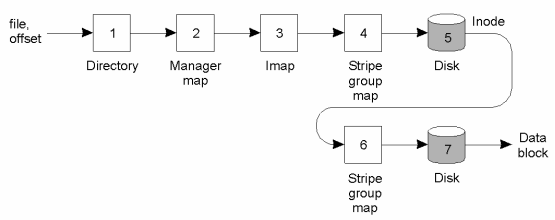
18

xFS Data Structures

Data structure	Description
Manager map	Maps file ID to manager
Imap	Maps file ID to log address of file's inode
Inode	Maps block number (i.e., offset) to log address of block
File identifier	Reference used to index into manager map
File directory	Maps a file name to a file identifier
Log addresses	Triplet of stripe group, ID, segment ID, and segment offset
Stripe group map	Maps stripe group ID to list of storage servers

19

Reading a File in xFS



20

Caching in xFS

- Managers keep track of who has cached copy of file
- Manager can direct request to peer cache
- To modify block, client must get ownership from manager
  - Manager invalidates all cached copies
  - Gives write permission (ownership) to client

21

xFS: Performance

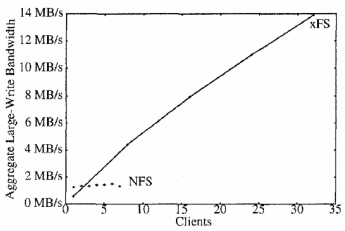


Figure 9. Aggregate disk write bandwidth. The x axis indicates the number of clients simultaneously writing private 10 MB files, and the y axis indicates the total throughput across all of the active clients. xFS used four groups of eight storage servers and 32 managers. NFS's peak throughput is 1.5 MB/s with 6 clients; xFS's is 13.9 MB/s with 32 clients.

22

xFS: Performance

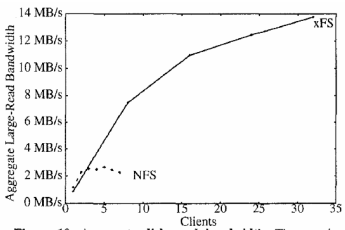


Figure 10. Aggregate disk read bandwidth. The x axis indicates the number of clients simultaneously reading private 10 MB files and the y axis indicates the total throughput across all active clients. xFS used four groups of eight storage servers and 32 managers. NFS's peak throughput is 2.7 MB/s with 5 clients; xFS's is 13.8 MB/s with 32 clients.

23

xFS Rationale

- Technology trends:
  - Fast LANs
  - Cheap PCs
  - Faster hardware expensive
- To get higher performance, don't build new hardware
- Just build networks of workstations, and tie them together with a file system
- RAID rationale very similar

24

Secure File System (SFS)

- Developed by David Mazieres while at MIT (now NYU)
- Key question: how do I know I'm accessing the server I think I'm accessing?
- All the fancy distributed systems performance work is irrelevant if I'm not getting the data I wanted
  - Getting the wrong data faster is not an improvement
- Several current stories about why I believe I'm accessing the server I want to access

25

Trust DNS and Network

- Someone I trust hands me server name: www.foo.com
- Verisign runs root servers for .com, directs me to DNS server for foo.com
- I trust that packets sent to/from DNS and to/from server are indeed going to the intended destinations

26

Trust Certificate Authority

- Server produces certificate (from, for example, Verisign) that attests that the server is who it says it is.
- Disadvantages:
  - Verisign can screw up (which it has)
  - Hard for some sites to get meaningful Verisign certificate

27

Use Public Keys

- Can demand proof that server has private key associated with public key
- But how can I know that public key is associated with the server I want?

28

Secure File System (SFS)

- Basic problem in normal operation is that the pathname (given to me by someone I trust) is disconnected from the public key (which will prove that I'm talking to the owner of the key).
- In SFS, tie the two together. The pathname given to me automatically certifies the public key!

29

Self-Certifying Path Name

/sfs/sfs.vu.sc.nl:ag62hty4wior450hdh63u623i4f0kqere/home/steen/mbox

- LOC: DNS or IP address of server, which has public key K
- HID: Hash(LOC,K)
- Pathname: local pathname on server

30

SFS Key Point

- Whatever directed me to the server initially also provided me with enough information to verify their key
- This design separates the issue of who I trust (my decision) from how I act on that trust (the SFS design)
- Can still use Verisign or other trusted parties to hand out pathnames, or could get them from any other source

31

SUNDR

- Developed by David Mazieres
- SFS allows you to trust nothing but your server
  - But what happens if you don't even trust that?
  - Why is this a problem?
- P2P designs: my files on someone else's machine
- Corrupted servers: sourceforge hacked
  - Apache, Debian,Gnome, etc.

32

Traditional File System Model

- Client send read and write requests to server
- Server responds to those requests
- Client/Server channel is secure, so attackers can't modify requests/responses
- But no way for clients to know if server is returning correct data
- What if server isn't trustworthy?

33

Byzantine Fault Tolerance

- Replicate server
- Check for consistency among responses
- Can only protect against a limited number of corrupt servers

34

SUNDR Model V1

- Clients send digitally signed requests to server
- Server returns log of these requests
  - Server doesn't compute anything
  - Server doesn't know any keys
- Problem: server can drop some updates from log, or reorder them

35

SUNDR Model V2

- Have clients sign log, not just their own request
- Only bad thing a server can do is a fork attack:
  - Keep two separate copies, and only show one to client 1 and the other to client 2
- This is hopelessly inefficient, but various tricks can solve the efficiency problem

36

Summary

- Plan 9: turn everything into a file
- xFS: turn every machine into a file server
- SFS: don't trust anything but the file server
- SUNDR: don't even trust the file server

37