Lecture #26: Distributed File System

****

Review -- 1 min

****

Distributed file system
- general distributed OS ideas
- file system case study

4 ideas
- RPC
- Caching
- Cache consistency
- If time: two-phase commit

File systems
- RPC – Netware
- RPC + caching -- NFS

****

Outline - 1 min

****

File systems:
    Cache consistency
2 phase commit

****

Preview - 1 min

****

Next week: security, course wrap-up, course review

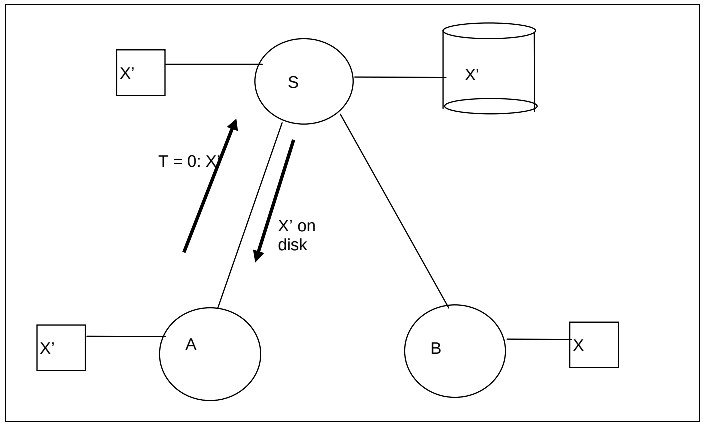
****

Lecture - 20 min

****

1. Cache consistency: problem

Consider updates in NFS (recall: cache files at clients)
Suppose A and B both read object X, then A updates X to X'. What will B read?

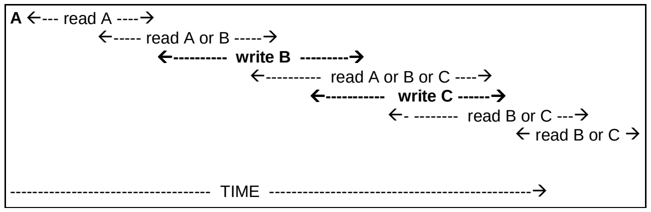


1.1 Sequential ordering constraints

Cache coherence – what should happen? What if one Cpu changes file and before it's done, another CPU reads file?

strict consistency – any read on a data item x returns a value corresponding to the most recent write on x

Seems simple, but in distributed system reads and writes take time: actual read could occur anytime between when system call is started, and when system call returns
    if read finishes before write starts, then get old copy
    if read starts after write finishes, then get new copy
    if reads and writes overlap, get ??? (either old or new)
    if writes overlap, get ??? (either old or new)


A ←--- read A ----→
←----- read A or B -----→
←---------- write B ---------→
←---------- read A or B or C ----→
←----------- write C ------→
←- -------- read B or C ---→
← read B or C →
------------------------------------ TIME ---→

- in above diagram, non-deterministic as to which write – C or B – ends up winning
- once I read B, all later reads must return B or C

1.2 NFS: weak cache consistency

What if multiple clients are sharing same files? Easy if they are both reading – each gets a copy of the file

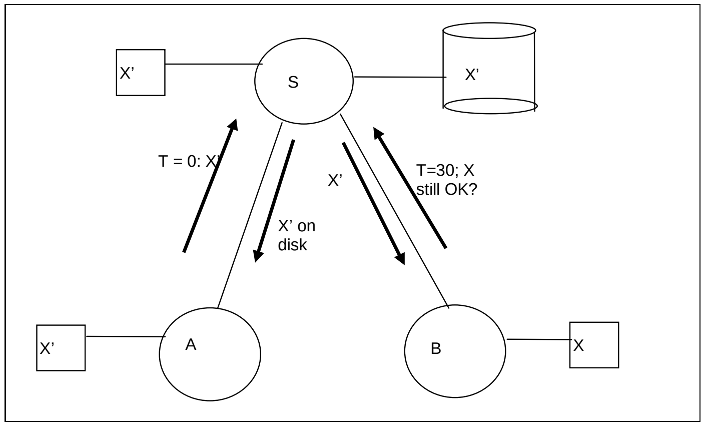
What if one writing? How do updates happen?

At writer – NFS has hybrid delayed write/write through policy
- write through within 30 seconds or immediately when file closed

How does other client find out about change (it has cached copy, so doesn't see any reason to talk to the server)

In NFS, client polls server periodically, to check if file has changed. Poll server if data hasn't been checked in last 3-30 seconds (exact timeout is tunable parameter)

Thus, when file is changed on one client, server is notified, but other clients use old version of file until timeout. They then check server, and get new version.



What if multiple clients write the same file? In NFS, can get either version (or parts of both). Completely arbitrary!

In NFS, if read starts more than 30 seconds after write finishes, get new copy. Othewise, who knows? Could get partial update.

1.3 NFS Summary

NFS pros & cons
+ simple
+ highly portable
- sometimes inconsistent
- doesn't scale up to large # of clients

Might think NFS is really stupid, but Netscape/WWW does something similar: cache recently seen pages, and refetch them if they are too old. Nothing in WWW to help with cache coherence

How to provide consistency across clients?

2. Andrew File System

AFS (CMU late 80's) → DCE DFS (commercial product)

1) files cached on local disk
NFS caches only in memory
→ reduce server load
2) more precise consistency model

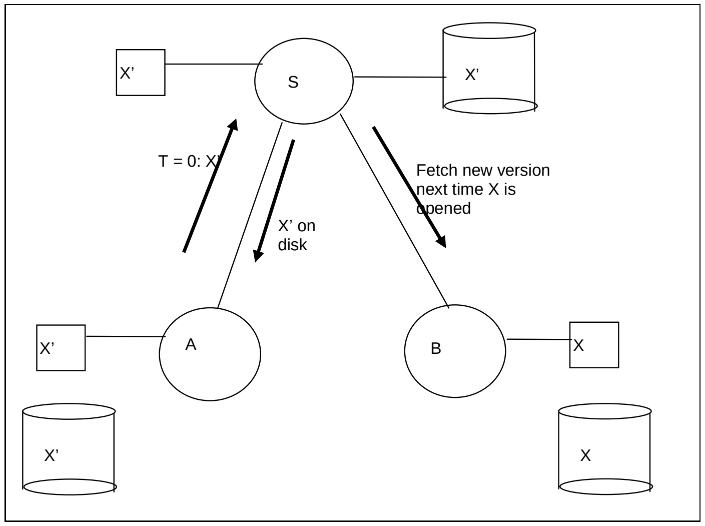
1) callbacks
   - server records who has copy of file
   - send "callback" on each update
2) write-through on close
   If file changes, server is updated (on close)
   Server then immediately tells those with old copy

3) session semantics – updates visible only on close
   In UNIX (single machine) updates visible immediately to other programs who have file open
   In AFS, everyone who has file open sees old version; anyone who opens file again will see new version

In AFS:

a) on open and cache miss – get file from server; set up callback

b) on write close: send copy to server; tells all clients with copies to fetch new version on next open



Challenge: improved caching + consistency increases failure handling complexity:
What if server crashes? Lose all callback state
Reconstruct callback information from clients – go ask everyone "who has which files cached?"

AFS pros & cons
Relative to NFS, less server load:
+ disk as cache → more files can be cached locally
+ callbacks → server not involved if file is read-only
■ on fast LANs, local disk much slower than remote memory

****
Admin - 3 min
****

****

Lecture - 23 min

****

3. Reliability

Want to be able to reliable update state on two different machines

e.g., atomically move directory from file server A to file server B
e.g., atomically move $100 from my account to Visa account

Challenge:

- messages can be lost
- machines can crash

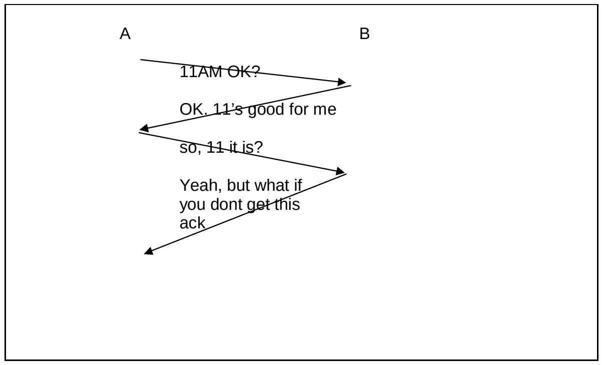
3.1 General's paradox

Can I use messages and retries over an unreliable network to synchronize two machines so that they are guaranteed to do same op at same time?

Remarkably, no. Even if all messages end up getting through

General's paradox: two generals on separate mountains. Can only communicate via messengers; the messengers can get lost or be captured

Need to coordinate the attack; if they attack at different times, then they all die. If they attack at same time, they win.



Even if all messages are delivered, can't coordinate (B/c a chance that the last message doesn't get through). Can't simultaneously get two machines to aggre to do something at same time

No solution to this – one of the few things in CS that is just impossible.
Proof: by induction

4. 2-phase commit

Since I cannot solve General's Paradox, let me solve a related problem

Abstraction – distributed transaction – two machines agree to do something or not do it, atomically
(but not necessarily at exactly the same time)

example: my account is at NationsBank, yours is at Wells Fargo. How to transfer $100 from you to me? (Need to guarantee that both banks agree on what happened).
Example: file system – move a file from directory A on server a to directory B on server b

Two-phase commit protocol does this. Use log on each machine to keep track of whether commit happened

Phase 1: coordinator requests

1. coordinator sends REQUEST to all participants

   e.g. C→S1 "delete foo from /", C→S2 "add foo to /"

2. participants recv request, execute transaction locally, write VOTE_COMMIT or VOTE_ABORT to local log, and send VOTE_COMMIT or VOTE_ABORT to coordinator

Failure case	*Success case*
S1 decides OK, writes "rm /foo; VOTE_COMMIT" to log, and sends VOTE_COMMIT *S2 decides no space on device and writes and sends VOTE_ABORT*	*S1 and S2 decide OK and write updates and VOTE_COMMIT to log, send VOTE_COMMIT*

Phase 2: coordinator decides

3. case 1: coordinator recv VOTE_ABORT or timeout
   → coordinator write GLOBAL_ABORT to log, and send GLOBAL_ABORT to participants

   case 2: coordinator recvs VOTE_COMMIT from all participants
   → coordinator write GLOBAL_COMMIT to log, and send GLOBAL_COMMIT to participants

4. participant receives decision; write GLOBAL_COMMIT or GLOBAL_ABORT to log

What if

- Participant crashes at 2? Wakes up, does nothing. Coordinator will timeout, abort transaction, retry
- Coordinator crashes at 3? Wakes up,
  - Case 1: no GLOBAL_* in log → Send message to participants "abort"

  - Case 2: GLOBAL_ABORT in log → send message to participants "abort"
  - Case 3: GLOBAL_COMMIT in log → send message to participants "commit"
- Participant crashes at 4? On recovery, ask coordinator what happened and commit or abort

This is another example of the idea of a basic atomic operation. In this case – commit needs to "happen" at one place

Limitation of 2PC – what if coordinator crashes during 3 and doesn't wake up? All nodes block forever
What if participants times out waiting in step 4 for coordinator to say what happened. It can make some progress by asking other participants

1. if any participant has heard "GLOBAL_COMMIT/ABORT", we can safely commit/abort
2. if any participant has said "VOTE_ABORT" or has made no vote, we can safely abort
3. if all participants have said "VOTE_COMMIT" but none have heard "GLOBAL_*", can we commit? A: no – coordinator might have written "GLOBAL_ABORT" to its disk (e.g., local error or timeout)
   Turns out – 2PC always has risk of indefinite blocking
   Solve with 3 phase commit (look it up if you ever need it…)

   In practice 2PC usually good enough – but be aware of the limits

If you come to a place where you need to do something across multiple machines, don't hack

- use 2PC (or 3PC)
- if 2PC, identify circumstances under which indefinite blocking can occur (and decide if acceptable engineering risk)

5. NETWORK OUTTAKES:

Not covered in class this year

In both AFS and NFS
Central server is a bottleneck
Performance bottleneck:
- all data written through to server
- all cache misses go to server

Availability bottleneck:
- server is single point of failure

Cost bottleneck
- server machiines high cost relative to workstation

6. xFS: serverless network file service

key idea – file system as parallel program; exploit opportunities provided by fast LANs

Four key ideas:
- cooperative caching
- software RAID
- distributed control

6.1 cooperative caching

use remote memory to avoid going to disk (manage client memory as global shared resource)

a) on cache miss, get file from someone else's cache instead of disk
b) on replacment, if last copy of file, send to idle client instead of discarding

+ better hit rate for read-shared data
+ active clients get to use memory of idle clients

6.2 software RAID

distribute data across all machines' disks → better bandwidth

but: we've made availability story a whole lot worse; now pieces of the file system are spread all over. If any machine fails, part of file system unavailable

xFS solution: stripe data redundanty over multiple disks, using SW RAID. Each client writes modifications to a log stored on redundant stripe of disks

<PICTURE>

On failure, others can reconstruct data from other disks in order to figure out missing data; logging makes reconstruction easy

A detail: need to be able to find things on disk; done as in LFS via an inode/ifile header map, containing locations of every inode on disk. This map is spread over all machines, kept by the last writer

Inode map is checkpointed to disk periodically. ON failure, read checkpoint from disk, then update from logs written after checkpoint

6.3 Distributed control

We've decentralized the cache, the disk, writes and reads, but there is still a central server to record who has which copies of data

xFS solution: spread manager over all machines; if anyone fails, poll clients to know who has what, and then shift its responsibilities to a new client

6.4 summary

xFS: build large ysstem out of large numbers of small unreliable components

Key: everything dynamic – data, metadata, control can all live anywhere on any machine, in any memory, on any location on disk. Also, this means easy to migrate to tape: anything can be located anywhere

Started w promise v. reality of distributed sytsems

xFS is example of how distributed systems will look in the future: higher performance, higher availability than any centralized system. Improves performance as you add more machines: more CPUs, more DRAM, more disks, ought to mean better performance and availability, not worse!

Also: automatic reconfiguration – machine goes down, everything continues to work. Machine gets added, start using its disk and CPU (in hardware called "hot swap" – key to high availability)

still some challenges – how do you upgrade software to new OS, new version of xFS, new version of disk, CPU, etc. while system continues to operate? Can we build systems that operate continuously for a decade?

Rest of lecture – abstractions for structuring distributed application

7. Cross-domain communication

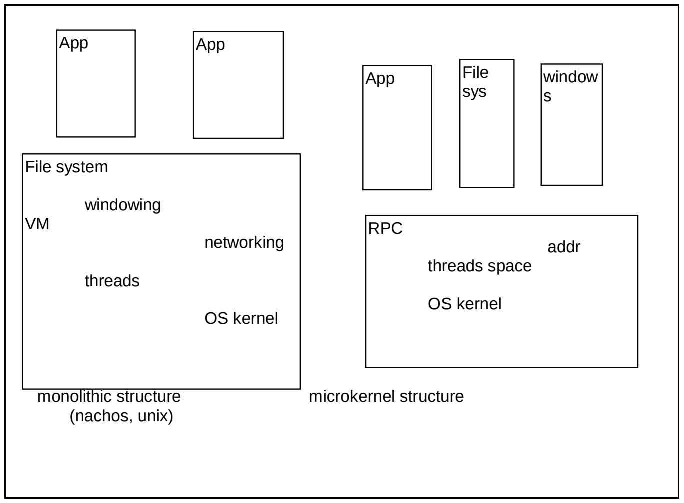
How do address spaces communicate with one another

- file system
- shared memory
- pipes (1-way communication)
- LRPC "Local Remote Procedure Call" (2-way communication)

RPCs can be used to communicate between address spaces on different machines or between address spaces on the same machine

7.1 Microkernel operating systems

Example: split kernel into application-level servers. File systems look like it is remote, even though on same machine



Why split OS into separate domains?

- fault isolation – bugs are more isolated
- enforces modularity - allows incremental upgrades f pieces
- location transparant – service can be local or remote

Example - X window system

8. Network performance

overhead – CPU time to put packet on wire
latency – how long to send 1 byte packet
throughput – maximum bytes per second

Latency – significant fraction of speed of light (1foot/ns) → <1us anywhere in taylor

	throughput	overhead	100 byte	4 KB	remote 4K read
TCP/IP Ethernet	10 Mbit/s	0.5-1ms	.5ms + .08ms	.5ms + 3ms	4 ms
TCP/IP ATM	155 Mbit/s	0.5-1ms	.5ms + .005ms	.5ms + .2ms	1.2ms
AM/Myrinet	1200 Mbit/s	.007ms	.007ms + .001ms	.007 + .03ms	.04ms

What is latency if go cross-country?
- 3000 miles * 5000 ft/mile → 15ms
- now 4KB read dominated by latency for all networks

Key to good performance
- in LAN – minimize overhead
- in WAN – keep pipeline full

****
Summary - 1 min
****