Distributed File Service

In distributed system, files generally reside on different nodes.
Want distributed file system (no explicit files transfer). Can have:

- single, central file server,
- several file servers,
- completely distributed file service

Client computers may be

- diskless
- diskfull, but only caching
- diskfull, part of distributed file service

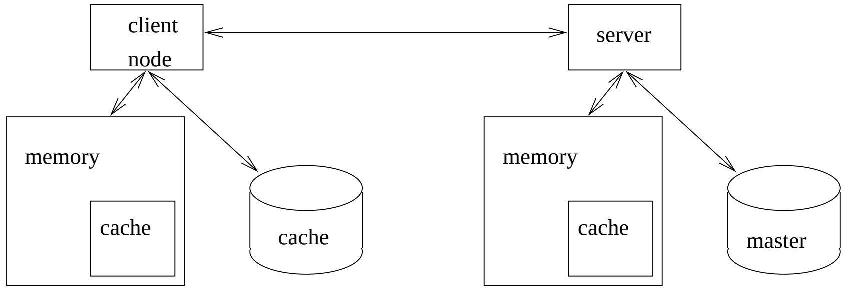
File Service Design Goals

diversity of application: support all applications
ease of use
matching requirements: don't pay for what you don't use
consistent naming: same name for all clients
location transparence: moving file shouldn't change its name
access transparent: no difference between local and remote files
data consistency: provide locking, transactions
information protection: authorised access only
user mobility: same access from anywhere (incl. performance!)
performance: not worse than centralised system
reliability/fault tolerance: crashes rare, recovery possible
hardware independence: accommodate different media
scalability: accommodate growth of system

Problems

- hard to satisfy all of these at the same time
- where to put files?
- performance $\Rightarrow$ caching

Client-server configuration



Caching strategies

Can cache

- whole files (original Andrew, Coda)
  - simple
  - limits file size
- pages/blocks (most systems)

Caching strategies...

Can cache

- in memory
- on disk
  - bigger
  - stable

Read ahead

Studies show that files are mostly

- small
- processed sequentially
- read in entirety
- reads dominate writes
- write sharing rare

Advantage: *bulk transfer*
transferring large packets or chunks reduces overheads

Cache coherency strategies

- passive: do nothing
- server *maintained*
  - server keeps list of cached blocks/files
  - clients inform server of updates, server invalidates caches
- server *monitored*
  - only cache whole files
  - server ensures that at most one client has writable copy
  - only way to guarantee strict coherence
  - efficiency ok as long as write sharing is low

File location strategies

Can use

- up-to-date file map (how keep it up-to-date?)
- location hints with broadcast on failure (potentially expensive)
- forwarding (potentially expensive)
- synchronisation site (single node knowing about file's whereabouts)
  node name may be encoded in name/UID (Locus)

File service issues

Mutable vs. immutable files

- never modify file, make new versions
- trivially guarantees consistency **if** only caching whole files
- still need to ensure changes aren't lost
- significantly increased storage requirements

File service issues...

Stateless vs. statefull servers

- state kept by clients (e.g. file positions)
- advantages
  - no loss of state if server crashes
  - no orphan state if client crashes
  - greatly simplifies design
- disadvantages
  - file migration leads to obsolete state
  - concurrent updates — how maintain consistency?
  - need transactions (necessarily have some (internal) state)

Concurrency control strategies

- locking
  - server locks file/page/record
  - lock labelled with transaction ID
- optimistic concurrency control
  - check for conflicts on *commit*, abort if necessary
- timestamps
  - server maintains timestamps on data (last written, last read)
  - on commit compare timestamp with client's version, abort if necessary

Replication

- improves performance
- improves fault tolerance
- increases coherency problems

Replication strategies

Master/slave

- centralised master copy services all update requests
- migrate changes back to slaves
- slaves consistent but not necessarily current
- ok if changes infrequent (Sun YP, CS&E UDB)

Distributed update

- maintain version numbers or timestamps
- check version on each read
- usual cache coherency problem

