Storage services

Consider various computing environments and scenarios

professional, academic, commercial, home - *based on traditional wired and wireless networks*

mobile users with computing devices: internet-connected
and/or using wireless/ad hoc networks - *new - wired and wireless networks*

pervasive/active environments - sensor networks' logs/databases - *new - wired and wireless*

Some scenarios (consider domain architecture, naming, location,
security (authentication, authorisation, communication))

1. single domain behind firewall - local files served by network-based file service
& accessing remote files and services

2. Open, internet-based file services: commercial services, cooperative P2P file sharing

3. e-science/GRID: storage for compute-service environments, database services

4. digital libraries, copyright, professional societies, publishers: scientific archive

(high-level issues: persistence of data through technology change
persistence of scientific archive - who **guarantees** *persistence?)*

Examples of requirements - 1

Traditional environments

* program/document storage, development
  application/system program load and run-time data access

* application-level services (local and remote)
  databases, CAD, email, naming directories,
  photo editing, newsgroups, digital libraries

Different media types and file structure

* integration of various media within a service, as opposed to dedicated servers e.g. VoD
  continuous media, audio/video, work best with QoS guarantees

* composite documents with components of different media types
  linking related information across files (copying - vs - dangling references)
  issue: persistence of material linked to
  structure helps cooperative work and synchronisation of updates

Should a storage service provide support for structure representation, indexing and retrieval ?

* vast amount of material is accumulated:
  collections of images (e.g., support for memory loss patients - "my day" images)
  audits of professional caring activities of NHS and SS
  logs of sensor data - traffic, pollution, building projects such as tunnels

Examples of requirements - 2

(consider naming, location, security(authentication, authorisation, communication))

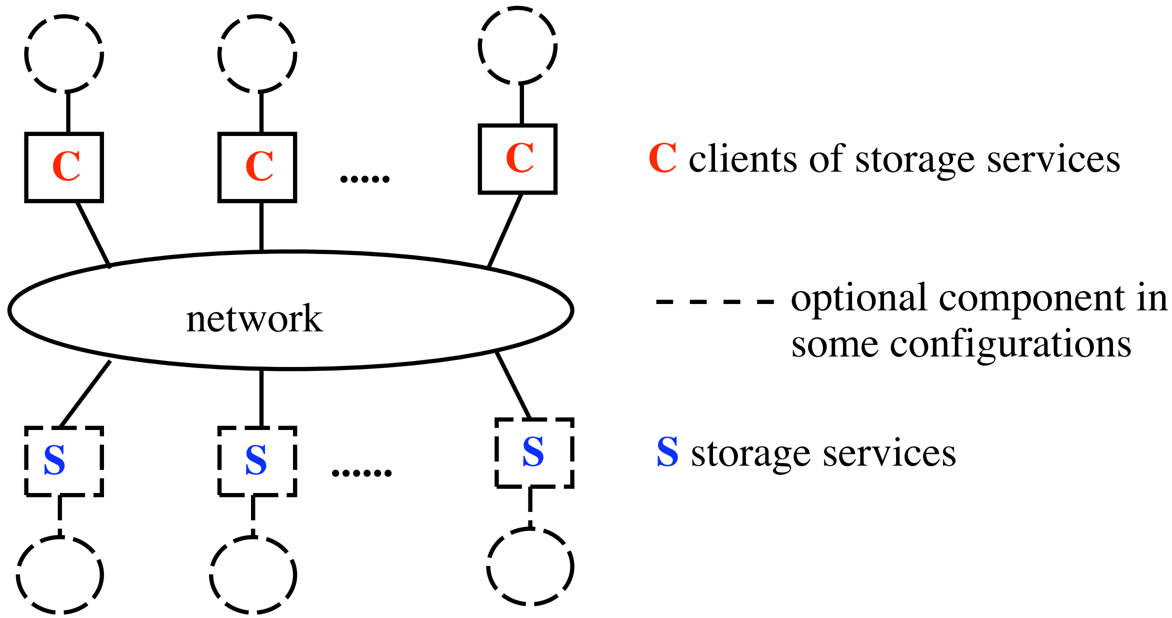
* applications for download into home/other systems e.g. into thin clients

* mobile users - access to files from remote locations
  - secure connection to home domain?
  - or use a commercial, internet-based file service?
    (to place wanted files close to where they will be used)
  - support for detached operation: copy, disconnect, work on local copy of file, reconnect, synchronise

* peer-to-peer (P2P)
  using spare capacity across the Internet for file storage, backup/archive
  issues of privacy, integrity, persistence, trust
  cooperative rather than commercial model (e.g. "sharing music with friends")
  what scarce resource are you saving? (particularly important in the new world of big, cheap discs)

* grid services
  e.g. GRID-accessible petabytes of astronomical or genomic data
  e.g. storage to support e-science computations
  e.g. data shared by "virtual organisations" - controlled access, non-repudiation
  e.g. data provenance
  e.g. public data such as EHRs (security/trust is crucial)

Storage in a single-domain distributed system



clients have no local discs, system provides shared storage servers
- early design - V system at Stanford
- network computers

clients have local discs, no dedicated storage services
- part of shared filing system - Unix mount

clients have local discs, system provides shared storage servers
- use of clients discs:
  - for private system (local desktop separate from shared servers - Xerox, Windows?)
  - part of shared filing system - Unix mount
  - system files for bootstrapping
  - cached files: first-class copy is in shared service
  - temporary files - not backed up by sys-admin

Storage service functionality

*** open or closed?**

is it bound into a single OS file system model
e.g. single pathname format?

*** functionality - how to distribute?**

- storage and retrieval of data (whole files or parts)
- name resolution (directory service)
- access control
- existence control (garbage collection)
- concurrency control

*** level of interface**

- remote blocks (some early systems e.g. RVD remote virtual disc; SANs do it now)
  client system may do block layout - minimal overhead at server
  or server does block layout - interface in terms of blockID
- remote, UID-named files (interactions may involve whole files or parts)
  server does block layout - more overhead at server
- remote path-named files (NAS)
  bound into a single style of naming

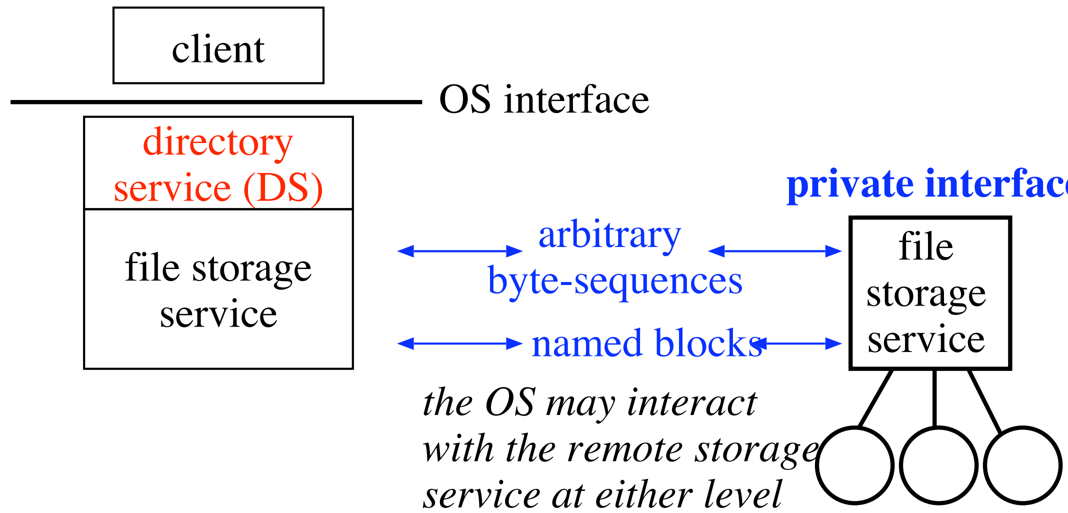
*** caching and replication**

is the service responsible for managing, or assisting with:
- multiple cached copies of a file
- replicas of a file (replicated on servers for reliability)

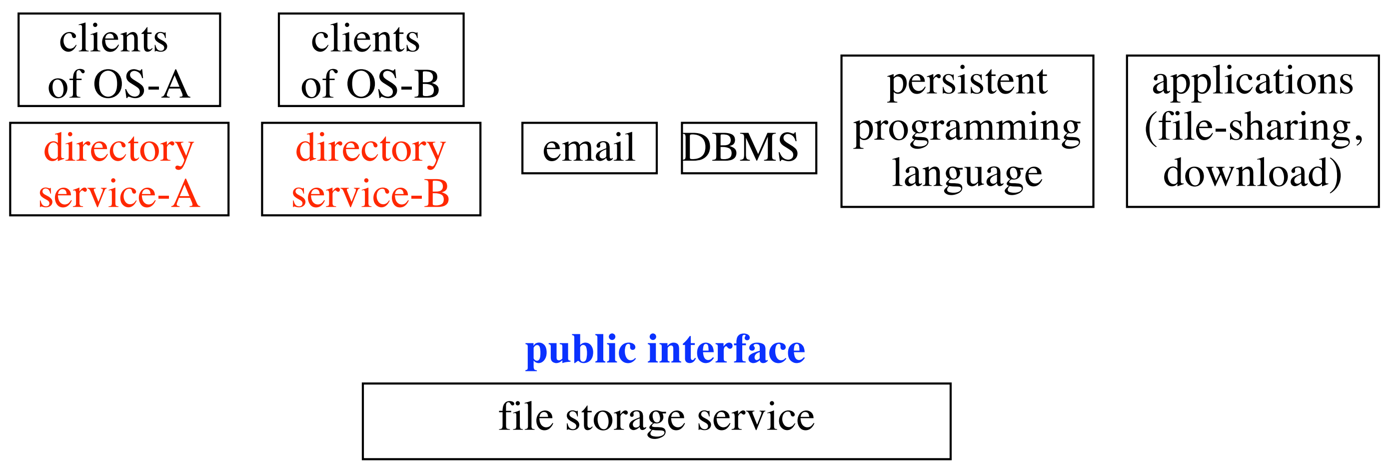
or are these application-level concerns?

Storage service architectures

a) closed storage architecture (single OS accesses SS)



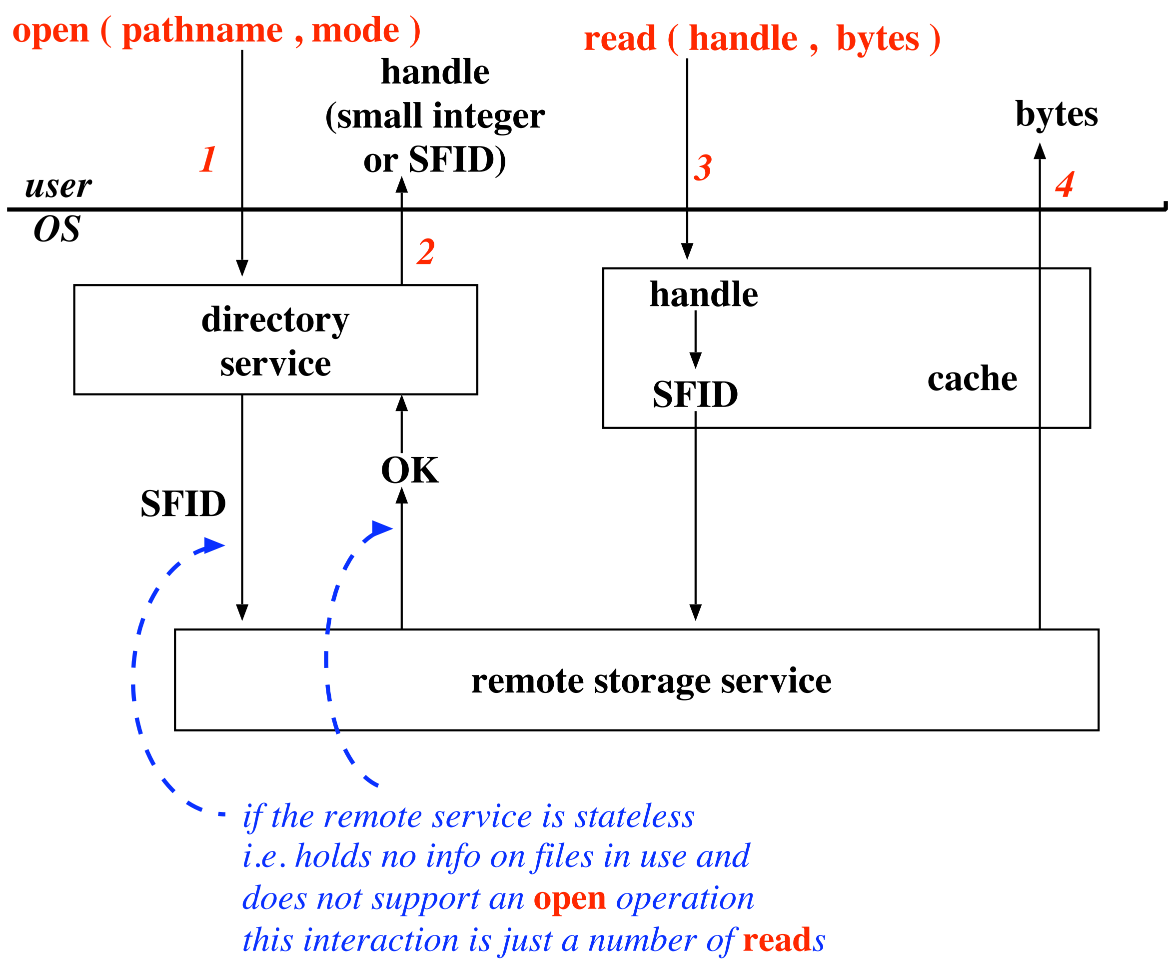
b) open storage architecture



remote interface at file storage level - example

assumes interactions can involve byte sequences rather than only whole-files

SFID = system file-identifier



operations in remote storage service interface

SFID ←→ **create**
read (SFID, byte-range)
write (SFID, byte-range)
delete (SFID) ?
lock (SFID) ?
open (SFID) ?
close (SFID) ?

assumes interaction at byte sequence level, as in **S-7,** *rather than whole-file*

? *depend on design decisions*

Does the server hold state?

* NO specified as stateless e.g. NFS

- simple crash recovery
- can't help with concurrency control
- can't help with cache management (clients have to ask for time-last-modified)

* YES
- supports open/close
  holds who has files open and access mode
- crash recovery - need to interact with clients to rebuild state
- concurrency control
  exclusive or shared locks better than write or read locks
- cache management
  can notify holders of copies when a new version is written

Existence control - garbage collection

a file should stay in existence for as long as it is reachable from the root of the directory naming graph

* storage service at file level can’t help (doesn’t see naming graphs)

* a directory service (multiple instances?) can do existence control for its own objects, ref S-6 b)
  OK for a closed architecture and for a single naming scheme within an open architecture
  provided sharing is restricted to that scheme’s files.

* what about
  - objects shared by different systems? (e.g. video clip in document)
  - objects not stored in directories?

* lost object problem SFID ←→ create (...)
  server allocates metadata in persistent store
  either: - **server crash** -
  or: reaches client’s main memory only,
  - **client crash** -
  on server or client restart, client repeats create (...)

* Consider a "touch" operation provided by the storage service. All clients (i.e. services, not users) must touch all their files periodically. Untouched files are deleted (archived)

e.g. A Birrell and R Needham "A Universal File Server" IEEE Trans SE 6(5), pp 450-453, May 1980
Cambridge File Server: - open architecture - many OS clients
- minimal support for structure without enforcing path-naming
- some composite operations with transactional semantics

CFS - the Cambridge File Server

Developed as part of the **Cambridge Distributed Computing System (CDCS)** in the late 1970's.
CDCS was used as the Lab's research environment throughout the 1980's.

http://www.research.microsoft.com/NeedhamBook/cmds.pdf

CFS provides:
- two primitive types: **byte** and **UID**
- two abstractions:
  - **file** - an uninterpreted sequence of bytes named persistently by a **PUID** with a random component
  - **index** - a sequence of PUIDs, itself named by a PUID

Indexes are used by CFS's clients to mirror their directory structures
all index operations are failure atomic (all or nothing is done)

*** existence control:**

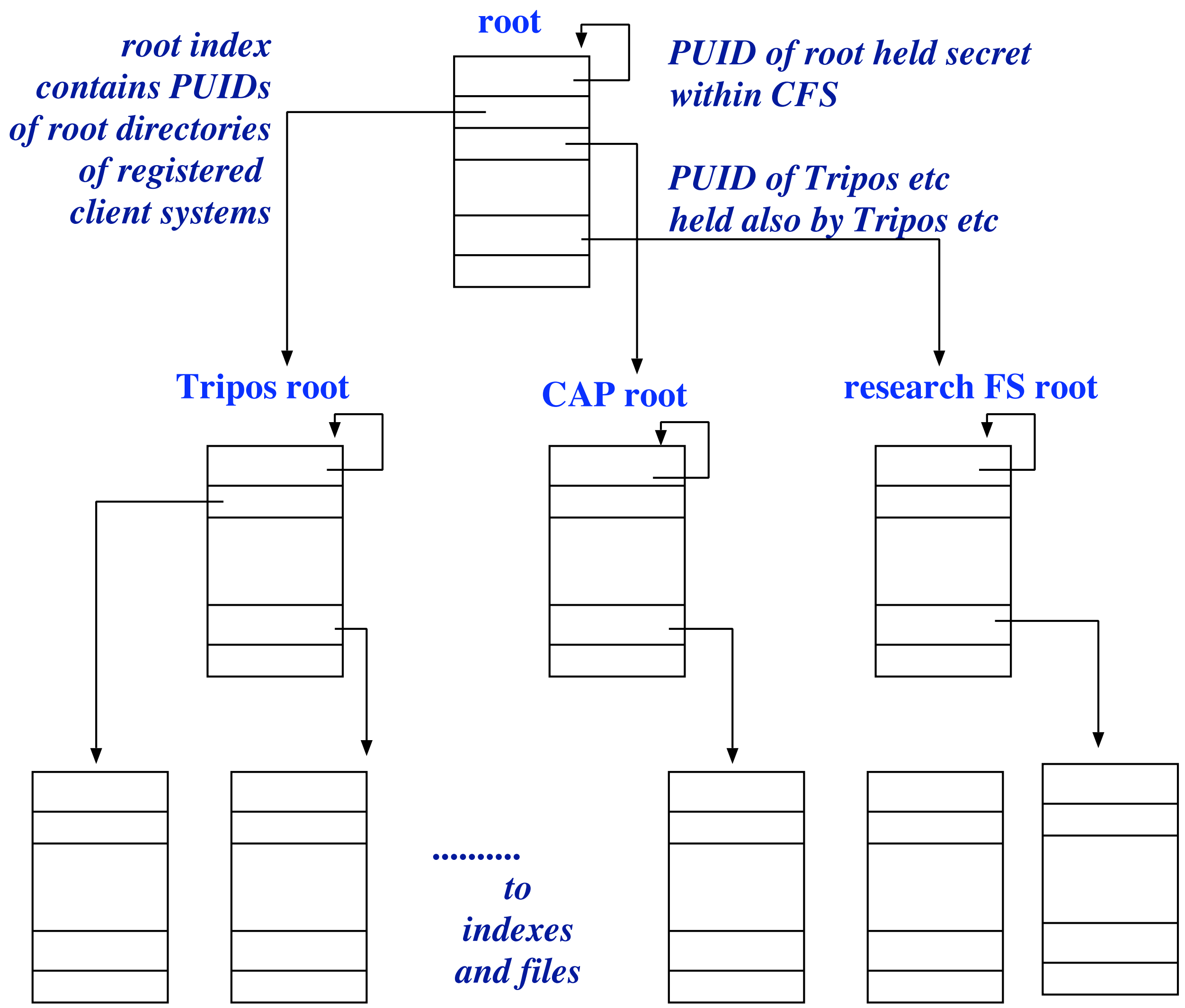
indexes form a general naming network starting from a specific root index

objects are preserved while they are reachable from the root
- reference counts are used (the number of times a UID is included in an index)
- an asynchronous garbage collector is used for cyclic structures

*** concurrency control - just MRSW**

open - a TUID is issued as a handle and the **PUID locked**
TUIDs are timed out (15 mins) reset on access
close - release PUID lock

CFS index structure



some CFS operations

object-ID = file or index ID
PUID = permanent/preserved ID for closed object
TUID = transient ID for open object

open [object-ID, {read/write}] -> TUID

close [TUID, {commit,abort}] -> commit/abort

index operations

create-index [existing index-ID, entry] -> index-ID *note transaction no lost object problem*

preserve [index-ID, entry, object-ID] -> done

retrieve [index-id, entry] -> object-ID

delete [index-ID, entry] -> done

NOTE: no **delete-index** *, garbage collection instead*

file operations

create-file [index-ID, entry, ...] -> file-ID *note transaction no lost object problem*

read [file-ID, offset, length] -> data

write [file-ID, offset, length, data] -> done

NOTE: no **delete-file***, garbage collection instead*

Open, Structured Files, an approach

CFS indexes allow:
- different client operating systems' file services to use CFS
  (their filenames and directory specifications can differ) - openness
- existence control (garbage collection) across file systems
  via reachability from root of index structure

note: for scalability we would need multiple instances of CFS and distributed garbage collection not addressed in CDCS for a LAN-based file service

Can the CFS index approach be generalised to allow:
- embedded links within files, linking to different file systems
  (e.g. to use different media types - video/audio clips)
- still have existence control, so be able to detect these embedded links

Idea: extend the storage type system to specify the storage structure sufficient to locate embedded links

base types: byte-sequence
SFID

generators: sequence
record
union
? other ? set, bag, ...

e.g. a directory might be:
a sequence of records with each record containing a byte-sequence and an SFID

e.g. a thesis might be a loose structure of sequences of variable-length byte-sequences and embedded references.
A typical requirement is to move sections of material around. The structure can be retained.

If the storage structure is stored as metadata for each stored object then
- SFIDs can be located for existence control
- objects can be kept in existence while any link to them remains

contrast with:
- typical network-based file systems model a file as a sequence of bytes identified by a SFID.

- relational databases typically specify records with fixed-length fields
(and have a higher-level type system cf. programming languages)

- embedded URLs (which typically fail after a short time)
the entire Web cannot be searched for embedded URLs before documents are deleted or moved
the storage service level has no knowledge of structure

- DTDs for web documents - more general info
XML type system - more general info - higher level, not concerned with storage

The idea was explored in the project:
Multi-Service Storage Architecture (MSSA)
theses: Sue Thomson 1990 and Sai lai Lo (1994) Lab TR 326 and DD