Coda was designed to be a scalable, secure, and highly available distributed file service.

- Supporting the mobile use of computers.
- Files are organized in volumes.
- Coda relies on the replication of volumes.

**Architecture**

**Naming**

**Replication strategy**

**Disconnected operation**

Venus processes provide access to files maintained by the Vice file server.

role is similar to that of an NFS client.

responsible for allowing the client to continue operation even if access to the file servers is (temporarily) impossible.
Coda file system

Coda was designed to be a scalable, secure, and highly available distributed file service, supporting the mobile use of computers. Files are organized in volumes. Coda relies on the replication of volumes.

Architecture
Naming
Replication strategy
Disconnected operation

Naming

Each file is contained in exactly one volume. Distinction between physical volumes:

- logical volume (represents all replicas of a volume)
- RID (Replicated Volume Identifier): identifier of a logical volume
- VID (Volume Identifier): identifier of a physical volume

File identifier

Server replication

Coda allows file server to be replicated, the unit of replication is a volume.

- Volume Storage Group (VSG): collection of servers that have a copy of a volume
- AVSG (Accessible Volume Storage Group): list of those servers in the volume's VSG that the client can contact
- AVSG - []: client is disconnected
- Coda uses a variant of the "read-one, write-all" update protocol
- Coda version vector

Replication strategy

Coda relies on replication to achieve high availability. It distinguishes between two types of replication:

- Client caching
- Server replication
Message passing model
  variables have to be marshalled from one process, transmitted and unmarshalled into other variables at the receiving process.

Distributed shared memory
  the involved processes access the shared variables directly; no marshalling necessary;
  processes may communicate via DSM even if they have non-overlapping lifetimes.

Implementation approaches
  in hardware
    shared memory multiprocessor architectures, e.g. NUMA architectures.
  in middleware
    language support such as Linda tuple space or JavaSpaces.