# Distributed File Systems

A distributed file system (DFS) is a distributed implementation of a classical time-sharing model of a file system.

The purpose of a DFS is to support the same kind of sharing when the files are physically dispersed among several computers.

## Distributed File Systems

**Definitions**

**Distributed system** - A number of loosely coupled machines connected by a (local area) network.

**Service** - A service is a program executing on one or several computers providing services to unknown clients.

**Server** - A specific machine running the server program.

**Client** - A process demanding service from the server.

**Client interface** - the (carefully specified) routines that the client use to contact the server.

**Primitive file system operations** - The routines included in the client interface for a distributed file system service.

**Component unit** - The smallest set of files that can be stored on a single machine, independently from other units. All files in a unit must be stored in the same location.

## Naming and Transparency

The user of a file system refers to a file by its external name (usually a text string).

The file system translates the external name to an internal name (usually a numerical identifier). This identifier in turn is mapped to disk blocks.

This multilevel mapping hides the details of where on the disk the file is stored.

In a transparent DFS a new dimension is added to the abstraction: that of hiding where in the network the file is located.

Definitions

- **Location transparency**. The name does not reveal any hint of the file's physical storage location.
- **Location independence**. The name of the file need not be changed, when the file's physical storage location is changed.

Both definitions are relative to the level of naming. A file system can be location transparent relative to external names but not location transparent relative to internal names.

## Naming and Transparency

- A location independent naming scheme is a dynamic mapping, since it can map a file to different locations at different times.
- This requires a data base to keep track of the current storage location for the *component units*.
- Therefor *location independence* is a stronger property than *location transparency*.
- Most current DFS:s provide a static location transparent mapping for user level names.
- These systems do not support **file migration**.
- Only AFS and a few experimental file systems support *location independence* and file mobility.

## Naming Schemes

There are three different approaches to naming schemes in a DFS:

1. host:local-name. This naming scheme is neither *location transparent* nor *location independent*.
2. As NFS. A client can mount a remote filesystem at an arbitrary location in its filesystem tree. Only previously mounted remote directories can be reached in a transparent way (unless automount is used).
3. A single global file system tree that looks the same on all machines. Some local files are still needed to interface local hardware units.

From an administrative point of view, NFS is the most complex of these methods. The only reliable way to make all clients look the same is to only allow mounting of a few central file servers.

## Implementation Techniques

- Implementation of transparent naming requires a provision for the mapping of a file name to a physical location.
- To keep the mapping information at a manageable volume, sets of files have to be aggregated into component units and mapping provided at component unit basis. (compare page tables in virtual memories).
- In UNIX-like systems: subtrees in the file system are used to group files into component units.
- To enhance the availability of mapping information we can use replication, local caching, or both.
- Location independence means that the mapping changes over time. If the mapping information is replicated, a simple and consistent update of the information becomes impossible.
- To solve this problem we can use internal low level **location-independent file identifiers**. These low level identifiers identifies to which *component unit* a file belongs and the location within the *component unit*.
- These low level identifiers can be cached and replicated because they never need to be changed.
- The price is the need for a data base to map *component units* to location.

## Consistency Semantics

Assume that two processes A and B have opened the same file.

At what time will process A see changes to the file written by process B?

This depends on which *consistency semantics* the file system use.

There are several possibilities:

**UNIX_Semantics** Changes written by process B is immediately visible to process A

**Session_Semantics** Changes written by B are not immediately visible to A. When the file is closed by B, changes will be visible in sessions started later. Process A that has the file open will still not see the changes.

## Remote Services

When a client needs service from a server on another machine, a message need to be sent to the server demanding the service. The server sends back a message with the requested data.

A common way to achieve this is **Remote Procedure Call (RPC).**

The idea is that an RPC should look like a normal subroutine call to the client.

Another possibility is to use sockets directly. Sockets used in the file system code however, have a few disadvantages:

1. Sockets may not be available in all systems
2. Making a connection using sockets requires knowledge of socket names. This is a type of system configuration data that should not be compiled into file system code.

## RPC

- PRC is actually a programming API (Application Programming Interface). The actual communication still need to use message passing (and sockets).
- An RPC is translated to a message sent to a certain port at the server machine.
- A port is the address to a certain process at the server, for example the file server process.
- When calling local subroutines, the subroutine name is translated to the memory address of the subroutine by the linker.
- When using RPC the RPC subroutine instead is translated to the address of a communication routine and a message is passed as parameter.

But how shall the client know which port number to use?

Two methods:

1. A static port number is compiled into the communication routine.
2. Dynamic translation. The system has a server (portmap) that is called to get the port number for a specified server. When using portmap every service calls portmap at startup to register its port number.

## Caching

- To ensure reasonable performance of a file system, some form of caching is needed.
- In a local file system the the rationale for caching is to reduce disk I/O.
- In a distributed file system (DFS) the rationale is to reduce both network traffic and disk I/O.
- In a DFS the client caches can be located either in the primary memory or on a disk.
- The server will always keep a cache in primary memory in the same way as in a local file system.
- The block size of the cache in a DFS can vary from the size of a disk block to an entire file.

## Cache Location

Where should the cached data be stored - on disk or in main memory?

Disk caches have one clear advantage over main-memory caches: they survive even if the machine crashes.

Main-memory caches have several other advantages:

- They allow diskless workstations.
- Data can be fetched quicker from main memory than from a disk.
- The server caches will always be in main memory. If the client caches are located in main-memory a single caching mechanism can be built for both server and client.

The technology trend towards larger and less expensive memory have reduced the need for disk caches.

If a disk cache is used, a main-memory cache is still needed for performance reasons, thus in this case both types of cache will be used.

## Cache Update Policy

The policy used to write modified data back to the server's master copy has a critical effect on the system's performance and reliability.

Update policies:

- **Write-through.** The simplest and most reliable strategy. Write operations must wait until the data is written to the server. The effect is that the cache is only used for read operations.
- **Delayed write.** Modifications are written to the cache and then written to the server at a later time. Write operations becomes quicker and if data are overwritten before they are sent to the server only the last update need to be written to the server.
- **Write-on-close.** All the time the file is open, the local cache is used. Only when the file is closed, data is written to the file server. For files that are open for long time periods and frequently modified, this gives better performance than delayed write. Used by the Andrew file system.

## Cache Consistency

Whenever caches are used, a method is needed to verify that the content in the cache is consistent with the master copy.

This problem is more difficult in a DFS than in a local filesystem because every client has its own cache. In a local file system all processes shares the cache.

There are two approaches to verifying the validity of cached data:

1. **Client-initiated.** The client initiates a validity check in which it contacts the server and checks whether the cache is consistent with the master copy. Choosing the frequency of validating is the problem. If validation is done to often both the network and the server may be heavily loaded.
2. **Server-initiated.** The server records, for each client, the files that it is caching. Inconsistency is possible every time a file is modified if the file is cached by other clients. Every time a file is updated a message is sent to the clients that have the file cached that the cache is invalid. If session semantics is used validation messages is only needed when a file is closed. If UNIX semantics is used much more frequent validation is needed.

## Comparison of Caching and Remote Service

Remote service means that no cache is used and all requests are sent to the server.

Advantages and disadvantages:

- When caching is used, the access time is reduced. Also the amount of data transported between the client and server is reduced, lowering the load at both the network and the server.
- The network overhead is lower when big blocks are transfered, as is done when caching is used.
- Caching allows for the use of a more optimized inter-machine interface since fixed size big blocks can be used.
- The cache-consistency problem is the major drawback of caching. When writes are frequent, the messages needed to keep the caches consistent may impose a considerable load on the network and the server.

## Stateless Versus Stateful Server

- In all communication protocols that use a connection mechanism, state information for the connection is stored at the server as long as the connection is valid.
- Examples of state information are descriptors for open files and read/write position pointers.
- A *stateless server* do not store state information concerning clients. This requires that a datagram protocol is used where every packet has complete information regarding the request.
- The advantage with storing state information is improved performance.
- The disadvantage with state information is that it creates problems if either a client or a server crashes.
- If a server crashes the state information is lost and have to be recreated in some way.
- If a client crashes the server need to detect this so that it can reclaim space allocated to storing the state of crashed client processes.

## Stateless Server

- A stateless server avoids the problems related to crashes. A client just retransmits requests if it get no response.
- The price for the more robust stateless server is reduced performance and some constraints on the design of the DFS.
- Because a client resends a message if it do not get an answer in a specified time, the server may receive the same request several times.
- This means that the operations must be idempotent, that is they must give the same result if executed several times.
- Self-contained read and write operations have this property if they use an absolute file position.
- Destructive operations such as remove are more problematic.
- Server initiated methods for cache validation are inherently stateful and cannot be used.
- UNIX read/write operations with file descriptors and implicit file positions are inherently stateful and cannot be used directly to a stateless server.