Distributed File Systems

Chapter 17 looks at the current major research and development in distributed-file systems (DFS). The purpose of a DFS is to support the same kind of sharing when the files are physically dispersed among the various sites of a distributed system.

We discuss the various ways a distributed file system can be designed and implemented. First, we discuss common concepts on which distributed file systems are based. Then, we illustrate our concepts by examining AFS—the Andrew distributed file system. By exploring this example system, we hope to provide a sense of the considerations involved in designing an operating system, and also to indicate current areas of operating-system research: network and distributed operating systems.

17.1 NFS provides location transparence since one cannot determine the server hosting the file from its name. (One could however view the mount-tables to determine the file server from which the corresponding file system is mounted, but the file server is not hardcoded in the name of the file.) NFS does not provide location independence since files cannot be moved automatically between different file systems. AFS provides location transparence and location independence.

17.2 Sprite guarantees that clients never see stale data. When a file is opened for write-sharing by multiple clients, all caching for the file is disabled and all operations are sent directly to the server. This scheme guarantees consistency of data. In AFS, writes to files are performed on local disks, and when the client closes the files, the writes are propagated to the server, which then issues callbacks on the various cached copies. During the time the file is written but not closed, the other clients could be accessing stale data. Also, even after the file is closed, the other clients might access stale data if they had performed the open on the file before the updating client had closed it. It is only at the point of time when the next open is performed on a caching client that the server is contacted and the most recent data propagated from the server to the client. NFS uses a more ad-hoc consistency mechanism. Data is flushed from the clients to servers at periodic intervals and on file close operations. Also, a client caching the file data checks for inconsistencies also at periodic intervals. Any updates made and flushed to the server during these intervals are not seen immediately on a client caching the file data.

17.3 A server needs to keep track of what clients are currently caching a file in order to issue a callback when the file is modified. When a server goes down, this state is lost. A server would then have to reconstruct this state typically by contacting all of the clients and having them report to the server what files are currently being cached by each client.

17.4 The advantage is that a single network request is sufficient to perform the path-name translation. Schemes that perform translations one component at a time incur more network traffic. However, when translations are performed one component at a time, the translations of the parent directories are obtained and cached for future reuse, whereas if the translation of the entire path is performed, translations for none of the intermediate elements are available in the cache.

17.5 Location-transparent DFS is good enough in systems in which files are not replicated. Location-independent DFS is necessary when any replication is done.

17.6 The Andrew file system can handle a large, multiclient database as scalability is one of its hallmark features. Andrew is designed to handle up to 5,000 client workstations as well. A database also needs to run in a secure environment and Andrew uses the Kerberos security mechanism for encryption.

17.7 Mapping objects into virtual memory greatly eases the sharing of data between processes. Rather than opening a file, locking access to it, and reading and writing sections via the I/O system calls, memory-mapped objects are accessible as “normal” memory, with reads and writes to locations independent of disk pointers. Locking is much easier also, since one shared memory location can be used as a locking variable for semaphore access. Unfortunately, memory mapping adds complexity to the operating system, especially in a distributed system.

17.8 Caching locally can reduce network traffic substantially as the local cache can possibly handle a significant number of the remote accesses. This can reduce the amount of network traffic and lessen the load on the server. However, to maintain consistency, local updates to disk blocks must be updated on the server using either a write-through or delayed-write policy. A strategy must also be provided that allows the client to determine if its cached data is stale and needs to be updated.

Caching locally provides is obviously more complicated than having a client request all data from the server. But if access patterns indicate heavy writes to the data, the mechanisms for handling inconsistent data may increase network traffic and server load.

17.9 Since the system is totally reliable, a stateful approach would make the most sense. Error recovery would seldom be needed, allowing the features of a stateful system to be used. If the network is very fast as well as reliable, caching can be done on the server side. On a slower network caching on both server and client will speed performance, as would file location-independence and migration. In addition, RPC-based service is not needed in the absence of failures, since a key part of its design is recovery during networking errors. Virtual-circuit systems are simpler and more appropriate for systems with no communications failures.