CMPSCI 677 Operating Systems **Spring 2013**

Lecture 23: April 24

Lecturer: Prashant Shenoy *Scribe: Steve Li*

23.1 Distributed file system

A Distributed file system is a system that allows a machine to access files on another machine. There are two canonical accessing models for distributed file systems:

Remote access model
All read/write requests to a file are processed on the server.

Upload/download model
When a client opens a file, a copy of the file is transferred (downloaded) to the client. All the subsequent read/write operations are done on the local machine. Once the file is closed, the result will be sent back (uploaded) to the server.

Consistency is simpler to achieve in the remote access model than in the upload/download model, because in the former model all the requests are handled on the server, which ensures consistency is attained. As for the upload/download model, there might be multiple clients opening the same file simultaneously. Keeping multiple copies of the same file on different machines makes consistency more challenging.

Distributed file system server can be categorized into two types: stateless server and stateful server. A stateful server keeps information such as which client is currently accessing a file and which file is opened on the server side; while a stateless server keeps no information with that regard.

23.2 Network file system

Network File System (NFS) is a distributed file system protocol developed by Sun. In the case the file being accessed is local, the file accessing mechanism on an NFS machine sends a request to the local file system interface to access the local storage. On the other hand, if the file is sitting on the remote machine, a request will be sent to the NFS client, which in turn makes an RPC call to the NFS server. Once the NFS server received the RPC call it will access the local file system and send the results back to the client. As a matter of fact, RPC was invented by Sun for implementing NFS.

Currently, the NFS versions still in use are version 3 and version 4. NFS version 3 follows a stateless server design, whereas version 4 is a stateful server design. NFS protocol defines several file accessing operations. Some of the operations are not supported by both version 3 and version 4. For example, there is no "open" operation in NFS version 3, which is stateless, meaning that servers do not need to keep track of file open states of individual clients.

In particular, opening a file in NFS version 3 is translated into a lookup operation to check if the specified file exists and the client has sufficient permission to access the file. Followed by a lookup, subsequent read requests will then be sent to the server in order to read data. For reading data in NFS version 4, the server only needs to keep track of which client is opening which file. As a result, clients can group three operations *lookup, open, read* together as one compound RPC request. This decreases overall RPC latency and leads to faster, performance optimized file accessing.

23.2.1 Mount protocol

Mount protocol establishes a mapping of a file system subtree on the server to local file hierarchy of the client.

Notice that NFS version 3 does not support transitive exports. For instance, say server B exports a subtree T_1 to server A, and server A exports a subtree T_2 that contains T_1 to some client. For security reason, the client is not allows to directly access T_1.

Mount can be performed on demand instead of at the boot time. In particular, when user tries to access an NFS directory, it will be mounted automatically – this is called automounting.

23.2.2 Consistency and caching

There are several types of file sharing semantics. UNIX semantics means that each operation is performed immediately and is guaranteed to be visible to all processes accessing that file. UNIX semantics are expected When operations are done locally.

NFS provides weaker semantics because of consistency reasons. When a process A writes to a file on the server, and a process B on another machine reads that file right after that, what process B reads is not guaranteed to include the change made by A because the modification might be cached locally at A and not yet updated with the server.

NFS protocol does not specify consistency requirements but leave it to the implementation. In the current implementation, clients synchronize changes with server every 30 seconds.

23.2.3 File locking

OS allows processes acquire locks when accessing files to prevent multiple processes from writing to a file at the same time. For instance, the inbox file in UNIX is protected with file lock. Because NFS version 3 is stateless, file locking is not supported. On the other hand, NFS version 4 is stateful and it allows client lock a file or even just a part of the file (a range of bytes in the file).

23.2.4 Delegation

Delegation is the act of transferring responsibility over an object (e.g.: a file), which is tightly related to the idea of transferring files in the download/upload access model. NFS by default only supports remote server mode in version 3. Delegation was introduced from version 4. When a client asks for a file, the server transfers a copy of the file to the client, called *master copy*, and all the subsequent changes are made locally and therefore no RPC requests are sent since then until the client closes the file.

If multiple processes from different machines want to access the same file, the server can recall delegation by asking the delegated client send the master copy back and the server then switch to remote access mode. NFS version 4 supports both remote access mode and upload/download mode.

As we can see, delegation impacts scalability. On the one hand, delegation aims at optimizing performance.

On the other hand, being stateful hurts performance and scalability. In general, NFS version 4 is less scalable, not for its stateful design but because it supports a much richer set of features that complicate the entire system.

23.2.5 RPC failures

Original NFS runs on UDP, while more recent versions of NFS run on TCP and hence deal with packet losses at the TCP level (although it is still possible to set it to work over UDP).

To use UDP, NFS has to handle potential packet loss by itself. It is common to use a time-out to determine if a packet is lost or not. Say if an NFS client does not receive the response from the server within a period of time, it will then re-issue the request. To guarantee correctness, operations have to be idempotent, that is, repeating an operation multiple times will result in the same outcome as being done exactly once.

To achieve this, all the requests are logged when processed and the results are cached. Once a request comes in, the server checks the log if the request has already been processed. If that is the case, the server simply returns the cached output. By doing so, exactly once semantic is provided.

23.3 Coda

Coda is a distributed file system developed as a research project at CMU. It was designed for mobile clients that disconnect as their machines move. To make disconnects transparent, each client keep a copy of remote files once connecting to the server. When disconnected, clients can work on the local copy without accessing to server. Once connecting back to the server, clients synchronize the updated contents with the server and download new files on the server to its cache.

Notice that not all files on the server are cached. Usually, the system would guess which files the user might be more interested in and cache only those subset. To maintain consistency, a mechanism called version vector is used: every update to a file will increment the corresponding entry of the version vector. By comparing version vector, server/clients can figure out which files are most up-to-date. However, when write-write conflict happens, users are responsible to resolve the conflict similar to the situation using source code control systems.

Clients basically transit among the following three states:

Hoarding client is connected and actively downloads files from the server and keep a cache on local machine

Emulation client disconnects but acts like it is connected by working on the cached contents

Reintegration client/server compare version vector and synchronize with each other. Client stays in reintegration state until synchronization is complete and then transitions to hoarding state (if connected)

23.4 XFS

XFS is a serverless file system that is similar to a P2P file system. Every machine involved in xFS can become server to some files and also client to some other files.

XFS uses the storage technology called RAID (redundant array of independent disks) to spread data on multiple disks.

23.4.1 RAID

The basic idea of RAID is file striping: each file is partitioned into multiple pieces and each of them is stored on different disk. The advantage of file striping includes the gain of parallelism. When accessing a file, all the pieces of that single file on different disks can be accessed in parallel, which could result in linear speedup. In addition, automatic load balance comes for free because popular files are distributed across multiple disks and therefore no single disk will be overloaded.

However, file striping comes with disadvantages: when a disk fails, all the files with a part on the disk will be gone. Assuming independent fails, the mean time to failure drops significantly as the number of disks increases. Therefore, redundancy must be added to make the system usable in practice.

23.4.1.1 RAID 1

RAID level 1 is simply mirroring: every primary disk comes with a corresponding secondary disk, and each write will send to both disks so they will have identical content. The main drawback is that RAID 1 is costly (in terms of hardware and cost to keep consistency) due to its high redundancy.

23.4.1.2 RAID 4 and RAID 5

RAID 4 introduces a smarter redundancy mechanism that involves a parity block for each striped file.

For example, the parity of striped blocks D_0, D_1, D_2, D_3 is computed by $p = D_0 \otimes D_1 \otimes D_2 \otimes D_3$ where $\otimes$ denotes XOR operation. Using the property $a \otimes b = c \Rightarrow c \otimes a = b, c \otimes b = a$, if one partition, say D_0, fails, it can be recovered by $D_0 = D_1 \otimes D_2 \otimes D_3 \otimes p$.

In RAID 4, all parity blocks are stored on the same dedicated disk. As a result, the parity disk becomes a bottleneck because every write will goes to the parity disk. To fix this, RAID 5 distributes parity blocks evenly to all the disks.

XFS uses RAID to distribute contents on multiple machines to achieve parallelism and load balance.