CMPSCI 677 Operating Systems **Spring 2014**

Lecture 21: April 14

Lecturer: Prashant Shenoy *Scribe: Deepak Arora*

21.1 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.

21.1.1 Server Replication

Coda works using servers with replicated volumes. This provides transparency in case of disconnection. To ensure consistency read once write all protocol used in case of replicated servers. In case of network disconnection, network is partitioned.

To maintain consistency when network comes back up, 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.

21.1.2 Disconnected Operation

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.

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)

Illusion of an always connected file system.

21.1.3 Transactional Semantics

Operation at the session level, where session lasts till a reintegration occurs, changes made to the file and then transferred over on reconnection. Disconnected operations should be ensured to be serializable after they

are executed. Transactional semantics which is comparatively strict to the conventional session semantics used by other File Systems. Writes are not visible to other clients until session end. NFS or UNIX provides immediate update hence transactional semantics necessary.

21.2 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. Bottleneck of centralized client server FS is overcome by this truly distributed File System.

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

It uses software RAID (network striping) and a log structured file system. It eliminates server caching using co-operative caching.

21.2.1 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. If data split across multiple disks. All data across different disks becomes unavailable. 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.

21.2.1.1 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. The hardware cost is exactly double and each logical write comprises of 2 physical writes.

21.2.1.2 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. RAID 5 is immune to 1 disk failure. Parity Blocks or the regular blocks can be

reconstructed using the other disks. To handle more than 1 disk failures we need more redundancy or better error correcting schemes.

21.2.2 Software RAID

XFS uses RAID to distribute contents on multiple machines to achieve parallelism and load balance. File System uses software which decides the distribution of data and parity blocks. Performance is lesser since parity computation in software unlike the traditional RAID controller. Since the disks are spread across different machines the system is immune to both disk and machine failures. Redundant data across other machines can be used to recreate the lost data using the parity block.

The machines participating in storage should not be transient for the software RAID to be successful. Again a single machine/disk failure can be handled in case of software RAID(network striping). Software RAID is feasible since the network capacity (bandwidth) is multiple times higher than the access time of a disk, in the current generation, allowing accessing data from distributed disks without a decrease in the overall performance.

21.2.3 Log Structured File System

Typically File System store data on disk with a cache for convenient access to frequently accessed blocks. Large fraction of cache misses are the write requests given equal number of reads and writes.

If we have a system with writes more frequent we need to optimize the File System accordingly to reduce the disk seeks for each write. This is the inspiration for the Log Structured File System which has logs written with invalidating the previous entry for each of the writes instead of overwriting the data at the original location.

Garbage collection needs to be done periodically to continue writing sequentially. Garbage collection tries segmenting the invalidated holes together to achieve sequential memory locations. This infrequent garbage collection is an overhead.

Note that the reads are still random. If we have frequent reads compared to writes then the above technique is not of much use.

21.3 HDFS

Optimized for batch operations(computations on large data sets) rather than interactive. HDFS uses mirroring concept for Fault Tolerance(unlike RAID). This File System assumes write once read many based approach. Once modified using a batch operation the data is not changed frequently. Moving computation to the data is the conventional approach for operations since the computations are on huge amounts of data.

Meta data is separated from the actual data and stored at separate nodes like other high performance File Systems. The location of the replicas is also available at the meta data nodes. Block size and replication factor is configurable in HDFS. Default replication factor is 3 and block size is large compared to interactive File Systems.

21.4 Object Storage Systems

Instead of storing data as files it is stored in the form of containers of arbitrary size called as handles. S3 service of Amazon is based on the Object Storage Systems where the objects are stored in buckets and we use HTTP names/operations instead of the actual file name to access data. Used for archiving or cloud storage where the entire object needs to be relocated.