

CSE 451: Operating Systems

Distributed File Systems

Distributed File Systems

- The most common distributed services:
 - printing
 - email
 - Files
 - Computation
- Basic idea of distributed file systems
 - support network-wide sharing of files and devices (disks)
- Generally provide a “traditional” view
 - a centralized shared local file system
- But with a distributed implementation
 - read blocks from remote hosts, instead of from local disks

11/24/2008

2

Issues

- What is the basic abstraction
 - remote file system?
 - open, close, read, write, ...
 - remote disk?
 - read block, write block
- Naming
 - how are files named?
 - are those names **location transparent**?
 - is the file location visible to the user?
 - are those names **location independent**?
 - do the names change if the file moves?
 - do the names change if the user moves?

11/24/2008

3

- Caching
 - caching exists for performance reasons
 - where are file blocks cached?
 - on the file server?
 - on the client machine?
 - both?
- Sharing and coherency
 - what are the semantics of sharing?
 - what happens when a cached block/file is modified
 - how does a node know when its cached blocks are out of date?

11/24/2008

4

- Replication
 - replication can exist for performance and/or availability
 - can there be multiple copies of a file in the network?
 - if multiple copies, how are updates handled?
 - what if there's a network partition and clients work on separate copies?
- Performance
 - what is the cost of remote operation?
 - what is the cost of file sharing?
 - how does the system scale as the number of clients grows?
 - what are the performance limitations: network, CPU, disks, protocols, data copying?

11/24/2008

5

Example: SUN Network File System (NFS)

- The Sun Network File System (NFS) has become a common standard for distributed UNIX file access
- NFS runs over LANs (even over WANs – slowly)
- Basic idea
 - allow a remote directory to be "mounted" (spliced) onto a local directory
 - Gives access to that remote directory and all its descendants as if they were part of the local hierarchy
- Pretty much exactly like a "local mount" or "link" on UNIX
 - except for implementation and performance ...
 - no, we didn't really learn about these, but they're obvious ☺

11/24/2008

6

- For instance:
 - I mount `/u4/levy` on Node1 onto `/students/foo` on Node2
 - users on Node2 can then access this directory as `/students/foo`
 - if I had a file `/u4/levy/myfile`, users on Node2 see it as `/students/foo/myfile`
- Just as, on a local system, I might link `/cse/www/education/courses/451/08au/` as `/u4/levy/451` to allow easy access to my web data from my home directory

11/24/2008

7

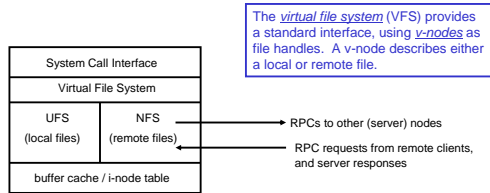
NFS implementation

- NFS defines a set of RPC operations for remote file access:
 - searching a directory
 - reading directory entries
 - manipulating links and directories
 - reading/writing files
- Every node may be both a client and server

11/24/2008

8

- NFS defines new layers in the Unix file system



11/24/2008

9

NFS caching / sharing

- On an open, the client asks the server whether its cached blocks are up to date.
- Once a file is open, different clients can write it and get inconsistent data.
- Modified data is flushed back to the server every 30 seconds.

11/24/2008

10

Example: CMU's Andrew File System (AFS)

- Developed at CMU to support all of its student computing
- Consists of workstation clients and dedicated file server machines (*differs from NFS*)
- Workstations have local disks, used to cache files being used locally (originally whole files, subsequently 64K file chunks) (*differs from NFS*)
- Andrew has a single name space – your files have the same names everywhere in the world (*differs from NFS*)
- Andrew is good for distant operation because of its local disk caching: after a slow startup, most accesses are to local disk

11/24/2008

11

AFS caching/sharing

- Need for scaling required reduction of client-server message traffic
- Once a file is cached, all operations are performed locally
- On close, if the file has been modified, it is replaced on the server
- The client assumes that its cache is up to date, unless it receives a *callback* message from the server saying otherwise
 - on file open, if the client has received a callback on the file, it must fetch a new copy; otherwise it uses its locally-cached copy (*differs from NFS*)

11/24/2008

12

Example: Berkeley Sprite File System

- Unix file system developed for *diskless* workstations with large memories at UCB (*differs from NFS, AFS*)
- Considers memory as a huge cache of disk blocks
 - memory is shared between file system and VM
- Files are permanently stored on servers
 - servers have a large memory that acts as a cache as well
- Several workstations can cache blocks for read-only files
- If a file is being written by more than 1 machine, client caching is turned off – all requests go to the server (*differs from NFS, AFS*)

11/24/2008

13

Example: Google File System (GFS)



NFS, etc.

Independence
Small Scale
Many users
Many programs

GFS

Cooperation
Large Scale
Few users
Few programs (well, many applications)

11/24/2008

14

GFS: Google File System

- Why did Google build its own FS?
- Google has unique FS requirements
 - Huge read/write bandwidth
 - Reliability over thousands of nodes with frequent failures
 - Mostly operating on large data blocks
 - Need efficient distributed operations
- Unfair advantage
 - Google has control over applications, libraries and operating system

11/24/2008

15

GFS Ideology

- Huge amount of data
- Ability to efficiently access data
- Large quantity of Cheap machines
- BW more important than latency
- Component failures are the norm rather than the exception
- Atomic append operation so that multiple clients can append concurrently

11/24/2008

16

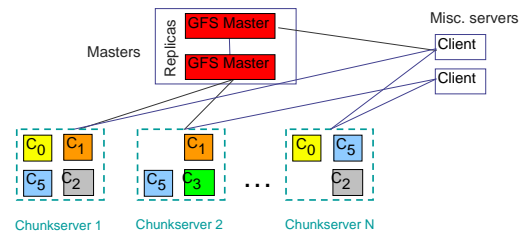
Files in GFS

- Files are huge by traditional standards
- Most files are mutated by appending new data rather than overwriting existing data
- Once written, the files are only read, and often only sequentially.
- Appending becomes the focus of performance optimization and atomicity guarantees

11/24/2008

17

GFS Setup



- Master manages metadata
- Data transfers happen directly between clients/chunkservers
- Files broken into chunks (typically 64 MB)

11/24/2008

18

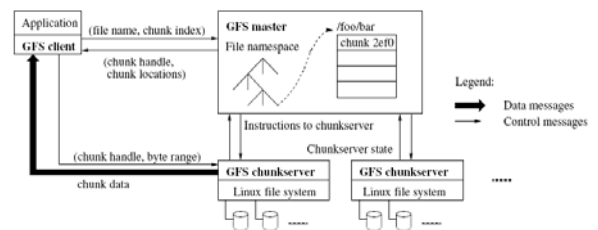
Architecture

- GFS cluster consists of a single *master* and *multiple chunk servers* and is accessed by *multiple clients*.
- Each of these is typically a commodity Linux machine running a user-level server process.
- Files are divided into fixed-size *chunks* identified by an immutable and globally unique 64 bit *chunk handle*
- For reliability, each chunk is replicated on multiple chunk servers
- master maintains all file system metadata.
- The master periodically communicates with each chunk server in *HeartBeat* messages to give it instructions and collect its state
- Neither the client nor the chunk server caches file data eliminating cache coherence issues.
- Clients do cache metadata, however.

11/24/2008

19

Architecture



11/24/2008

20

Read Process

- Single master vastly simplifies design
- Clients never read and write file data through the master. Instead, a client asks the master which chunk servers it should contact.
- Using the fixed chunk size, the client translates the file name and byte offset specified by the application into a chunk index within the file
- It sends the master a request containing the file name and chunk index. The master replies with the corresponding chunk handle and locations of the replicas. The client caches this information using the file name and chunk index as the key.
- The client then sends a request to one of the replicas, most likely the closest one. The request specifies the chunk handle and a byte range within that chunk

11/24/2008

21

Specifications

- Chunk Size = 64 MB
- Chunks stored as plain Unix files on chunk server.
- A persistent TCP connection to the chunk server over an extended period of time (reduce network overhead)
- cache all the chunk location information to facilitate small random reads.
- Master keeps the metadata in memory
- Disadvantages – Small files become Hotspots.
- Solution – Higher replication for such files.

11/24/2008

22

Summary of Distributed File Systems

- There are a number of issues to deal with:
 - what is the basic abstraction
 - naming
 - caching
 - sharing and coherency
 - replication
 - performance
- No right answer! Different systems make different tradeoffs!

11/24/2008

23

- Performance is always an issue
 - always a tradeoff between performance and the semantics of file operations (e.g., for shared files).
- Caching of file blocks is crucial in any file system
 - maintaining coherency is a crucial design issue.
- Newer systems are dealing with issues such as disconnected operation for mobile computers

11/24/2008

24