CLOUD COMPUTING CONCEPTS

with Indranil Gupta (Indy)

DISTRIBUTED FILE SYSTEMS

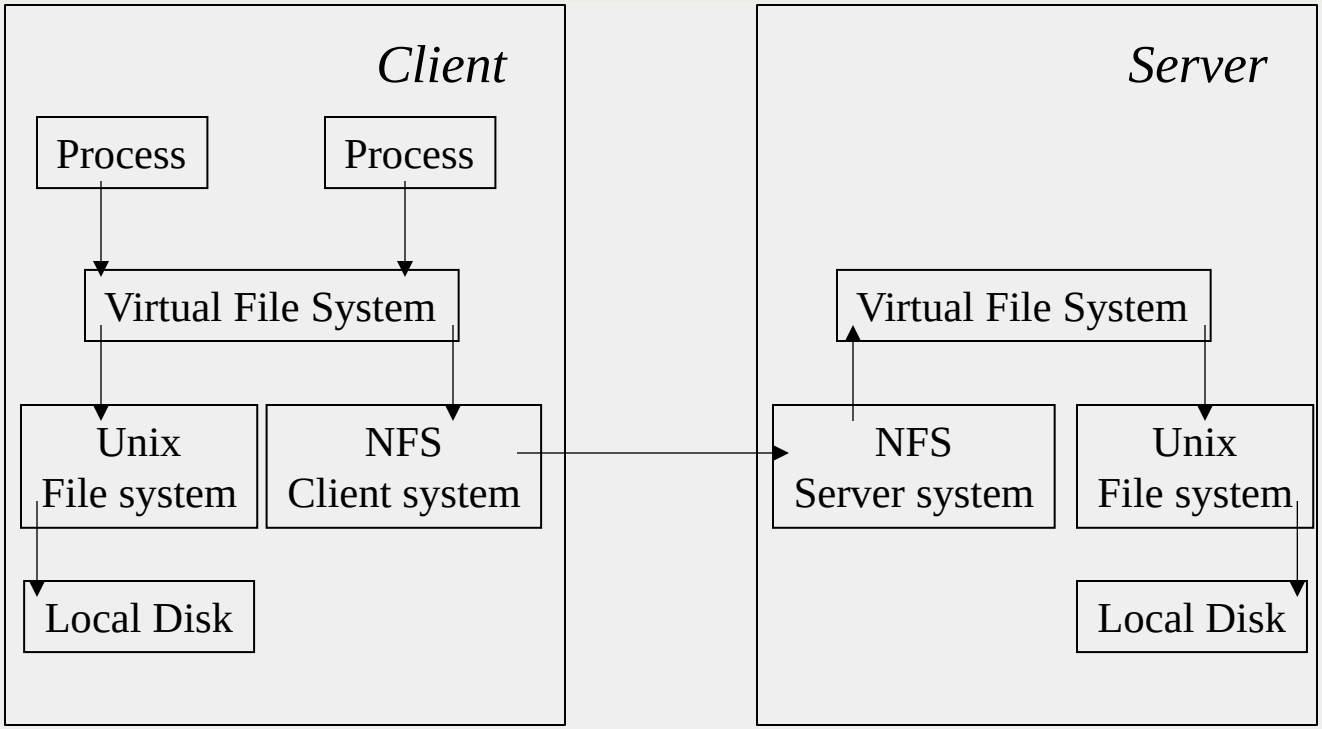
Lecture B

NFS AND AFS

NFS

- Network File System
- Sun Microsystems, 1980s
- Used widely even today

NFS Architecture



NFS Client and Server Systems

- NFS Client system
  - Similar to our “Client service” in our Vanilla DFS
  - Integrated with kernel (OS)
  - Performs RPCs to NFS Server system for DFS operations
- NFS Server system
  - Plays the role of both Flat file service + Directory service from our Vanilla DFS
  - Allows *mounting* of files and directories
    - Mount /usr/edison/inventions into /usr/tesla/my_competitors
    - => Now, /usr/tesla/my_competitors/foo refers to /usr/edison/inventions/foo
    - Mount: Doesn’t clone (copy) files, just point to that directory now

Virtual File System Module

- Allows processes to access files via file descriptors
  - Just like local Unix files! So, local and remote files are indistinguishable (i.e., gives transparency)
  - For a given file access, decides whether to route to local file system or to NFS client system
- Names all files (local or remote) uniquely using “NFS file handles”
- Keeps a data structure for each mounted file system
- Keeps a data structure called v-node for all open files
  - If local file, v-node points to local disk i-node
  - If remote, v-node contains address of remote NFS server

Server Optimizations

- Server caching is one of the big reasons NFS is so fast with reads
  - Server Caching = Store, in memory, some of the recently-accessed blocks (of files and directories)
  - Most programs (written by humans) tend to have *locality of access*
    - Blocks accessed recently will be accessed soon in the future
- Writes: two flavors
  - Delayed write: write in memory, flush to disk every 30 s (e.g., via Unix sync operation)
    - Fast but not consistent
  - Write-through: Write to disk immediately before ack-ing client
    - Consistent but may be slow

Client Caching

- Client also caches recently-accessed blocks
- Each block in cache is tagged with
  - *Tc*: the time when the cache entry was last validated.
  - *Tm*: the time when the block was last modified at the server.
  - A cache entry at time T is valid if

    $(T\text{-}Tc < t)$ *or* $(Tm_{client} = Tm_{server})$.
  - *t=freshness interval*
    - Compromise between consistency and efficiency
    - Sun Solaris: t is set adaptively between 3-30 s for files, 30-60 s for directories
- When block is written, do a delayed-write to server

Andrew File System (AFS)

- Designed at CMU
  - Named after Andrew Carnegie and Andrew Mellon, the “C” and “M” in CMU
- In use today in some clusters (especially University clusters)

Interesting Design Decisions in AFS

- Two unusual design principles:
  - Whole file serving
    - Not in blocks
  - Whole file caching
    - Permanent cache, survives reboots

- Based on (validated) assumptions that
  - Most file accesses are by a single user
  - Most files are small
  - Even a client cache as “large” as 100MB is supportable (e.g., in RAM)
  - File reads are much more frequent than file writes, and typically sequential

AFS Details

- Clients system = *Venus* service
- Server system = *Vice* service
- Reads and writes are optimistic
  - Done on local copy of file at client (Venus)
  - When file closed, writes propagated to Vice
- When a client (Venus) opens a file, Vice:
  - Sends it entire file
  - Gives client a *callback promise*
- Callback promise
  - Promise that if another client modifies then closes the file, a callback will be sent from Vice to Venus
  - Callback state at Venus only binary: valid or canceled

Summary

- Distributed File Systems
  - Widely used today
- Vanilla DFS
- NFS
- AFS
- Many other file systems out there today!