Nathan Tung

CS111 Sections of Chapter 4, Article Outline #18

Chapter 4, section 4.5, pages 184-195. What Cloud Computing Really Means, Eric Knorr, Galen Gruman, Infoworld.

- Case Study: Network File System (NFS)
  - Network File System is client/service application that gives shared file storage for clients on the network
    - NFS client puts remote file system onto client's local system much like a local UNIX file system
  - Purpose was to share files for collaboration
    - Simplifies management of workstation collection
      - Admin doesn't have to manage or make backups for each workstation individually
    - Also saves the cost of buying a disk (interface) for every workstation
  - Design goals for NFS
    - Work with existing applications (use same local UNIX semantics)
    - Easily deployable and easy to retrofit into existing UNIX systems
    - Client implementable in other operating systems (can't just be UNIX, but also allow Windows)
    - Efficient enough for users to tolerate, but doesn't need as high performance as in local systems
- Naming Remote Files and Directories
  - Appears the same way as a local UNIX file system
  - A "mounter" program mounts (or grafts) remote file system onto local name space
    - Mounter sends RPC to file server host to obtain a "file handle" (32-byte object name)
    - File handle IDs object on remote NFS server; invisible to applications, but internal to name files
    - File handle is structured: contains file system identifier, inode number, and generation number
      - Generation number: server uses to locate file
      - File system identifier: server IDs file system responsible for file
      - inode number: lets the IDed file system locate file on disk
  - Path names are not included in file handles
    - This is to keep NFS and local file systems consistent in opening files (in case some are renamed)
      - To reuse inode numbers, we increase generation number
      - In ambiguous situations, NFS gives "stale file handle" error for simpler implementation
    - File handles are available even across server failures
- NFS Remote Procedure Calls
  - Opening a remote file "f" involves NFS client sending RPC: LOOKUP(dirfh, "f")
    - NFS server uses extracted identified file system and inode number to locate directory's inode
    - NFS then uses directory's inode to find "f" and creates its file handle for the client
  - Closing a remote file "f" that has not been modified doesn't require RPC
  - NFS RPC hold their own information in order to be stateless (server maintains only on-disk file states)
  - Statelessness makes server failure recovery simple; client just repeats the request (idemponent)
  - Sometimes multiple RPC calls might give inconsistent results if a network reply is lost
    - Example: NFS client sends REMOVE RPC; server removes file, loses reply; NFS REMOVES again, but there's already no file
    - NFS should also avoid issuing multiple RPCs when there's no server failure
      - Use a "soft" state maintained by server that goes away if server fails (AKA a reply cache)
        - Reply cache maintained as soft state, since non-volatile storage is expensive
      - When given a request, server looks up transaction ID in reply cache
      - If it's there, return cache reply; otherwise, process the request
  - UNIX maintains state, but NFS is stateless, as it tries to simplify recovery
    - This is one instance where NFS has different behavior
- Extending the UNIX File System to Support NFS
  - To get NFS as UNIX extension, designers use vnodes (virtual nodes) to split file system
    - vnode is volatile memory structure that abstracts whether file/directory is local or remote
    - File system caller doesn't have to worry about distinguishing local/remote in files or directories
    - Operating on a file involves calling the procedure through vnode interface
  - To open a remote file, caller invokes PATHNODE_TO_VNODE
    - Passes vnode for current working directory and path name for the file

        - LOOKUP is invoked in the vnode layer for each path name component; local-implementation LOOKUP is called for local files and a remote-implementation LOOKPU is called for remote files
        - NFS server extracts file system identifier and inode from file handle; LOOKUP is invoked in the vnode layer and a vnode (if local file is present) is returned to NFS
        - NFS client then creates a vnode with file handle on client computer, returning it to file system call layer on machine
        - On file system call resolution, file descriptor for the file is returned
    - Caches are utilized to store vnode for open files and recently-used vnodes (and their attributes)
    - Caching reduces latency by reducing RPC cost for a client
        - Clients then use fewer RPCs, so a server can support more clients
        - If clients cache same file, we need to ensure R/W coherence
- Coherence
    - NFS could guarantee R/W coherence for either all operations or just some
    - Close-to-open consistency: only provide R/W coherence on OPEN, WRITE, and CLOSE
        - After OPEN/WRITE/CLOSE, a second client using the same file observes the first WRITE
        - Or every READ/WRITE provides coherence; every WRITE results in a READ that observes results
        - Allows higher data rates for reading/writing big files
            - Client can send several reads without waiting for response
            - In the second open-to-close consistency implementation above, no guarantees
                - READ may return data before or after last WRITE when same file is open twice
    - Client WRITEs by modifying local cached version, so no overhead of RPCs
        - CLOSE then sends cached writes to server
        - Delaying modified blocks until CLOSE absorbs overwritten modifications
    - This R/W coherence allows user to edit program on one computer and compile it on a different one
        - After file is modified/written, compiler observes all the edits
        - However, multiclient DB programs that read or write records stored in NFS file might not work correctly due to concurrent execute operations
        - Also, blocks are cached instead of entire files; fetching a file may result in intermixed blocks
- NFS Version 3+ supports 64-bit numbers, adds asynchronous write, etc.
- Version 4 maintains some state, prevents intruders from snooping/modifying network traffic, has more efficient close-to-open consistency scheme, and works well over internet
- What cloud computing really means
    - Cloud computing: virtual servers available over internet (an updated version of utility computing)
        - Adds capacity or capabilities on the fly without new infrastructure, personnel, or licenses
        - May be subscription-based or pay-per-use service; for example, storage services to spam filtering
    - SaaS: Software as a Service
        - Browser delivers an application via multitenant architecture (HR apps, even “desktop” Google Apps)
        - Customer has no investment in services or licenses; providers just have one cheap app to maintain
    - Utility computing
        - Offer solutions like storage and virtual services; often used by IT on demand
        - Used on Amazon, Sun, IBM, etc.
    - Web services in the cloud
        - Like SaaS, these offer APIs for developers to use functionality over internet
        - Range from business services to Google Maps API
    - Platform as a service
        - Another SaaS variation that gives development environment as service (Google App Engine)
    - MSP (managed service providers)
        - IT (not end-user) application like virus scanning, email, or application monitoring service (very old)
    - Service commerce platforms
        - SaaS and MSP hybrid; service hub for user interaction, like a trading environment (travel services)
    - Internet integration
        - Integration of cloud-based services (serving SaaS providers, using in-cloud integration techniques)
- More accurately “sky computing” due to isolated clouds of services which IT customers “plug into individually”
- Loosely coupled services operating on an infrastructure that is agile and scalable