CS162
Operating Systems and Systems Programming
Lecture 23

Remote Procedure Call

April 24, 2013

Anthony D. Joseph

http://inst.eecs.berkeley.edu/~cs162

Goals for Today

- Remote Procedure Call
- Examples using RPC and caching
  - Distributed File Systems
  - World-Wide Web

Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne. Many slides generated from my lecture notes by Kubiatowicz.

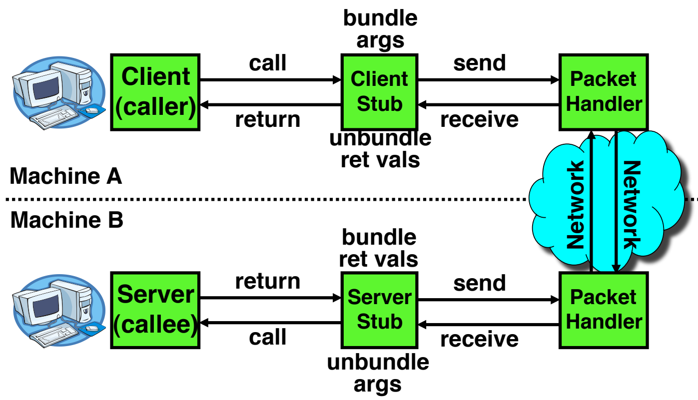
Remote Procedure Call

- Raw messaging is a bit too low-level for programming
- Another option: Remote Procedure Call (RPC)
  - Looks like a local procedure call on client:
    `file.read(1024);`
  - Translated automatically into a procedure call on remote machine (server)
- Implementation:
  - Uses request/response message passing "under the covers"

RPC Details

- Client and server use "stubs" to glue pieces together
  - Client stub is responsible for "marshalling" arguments and "unmarshalling" the return values
  - Server-side stub is responsible for "unmarshalling" arguments and "marshalling" the return values
- Marshalling involves (depending on system) converting values to a canonical form, serializing objects, copying arguments passed by reference, etc.
  - Needs to account for cross-language and cross-platform issues
- Technique: compiler generated stubs
  - Input: interface definition language (IDL)
    - » Contains, among other things, types of arguments/return
  - Output: stub code in the appropriate source language

RPC Information Flow



RPC Binding

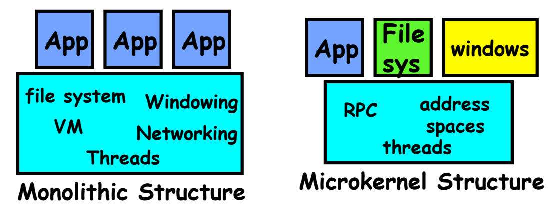
- How does client know which machine to send RPC?
  - Need to translate name of remote service into network endpoint (e.g., host:port)
  - Binding: the process of converting a user-visible name into a network endpoint
    - » This is another word for "naming" at network level
    - » Static: fixed at compile time
    - » Dynamic: performed at runtime
- Dynamic Binding
  - Most RPC systems use dynamic binding via name service
  - Why dynamic binding?
    - » Access control: check who is permitted to access service
    - » Fail-over: If server fails, use a different one

Cross-Domain Communication/Location Transparency

- How do address spaces communicate with one another?
  - Shared Memory with Semaphores, monitors, etc…
  - File System
  - Pipes (1-way communication)
  - "Remote" procedure call (2-way communication)
- RPC's can be used to communicate between address spaces on different machines or the same machine
  - Services can be run wherever it's most appropriate
  - Access to local and remote services looks the same
- Examples of modern RPC systems:
  - CORBA (Common Object Request Broker Architecture)
  - DCOM (Distributed COM)
  - RMI (Java Remote Method Invocation)

Microkernel Operating Systems

- Example: split kernel into application-level servers using RPC
  - File system looks remote, even though on same machine

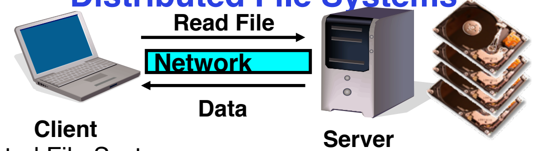


- Why split the OS into separate domains?
  - Fault isolation: bugs are more isolated (build a firewall)
  - Enforces modularity: allows incremental upgrades of pieces of software (client or server)
  - Location transparent: service can be local or remote
    - » For example in the X windowing system: Each X client can be on a separate machine from X server; Neither has to run on the machine with the frame buffer

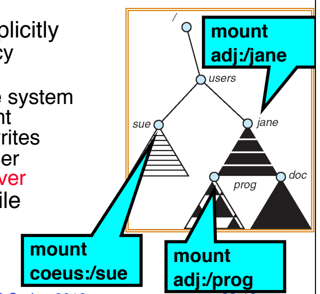
Problems with RPC

- Handling failures
  - Different failure modes in distributed system than on a single machine
  - Without RPC a failure within a procedure call usually meant whole application would crash/die
  - With RPC a failure within a procedure call means remote machine crashed, but local one could continue working
  - Answer? Distributed transactions can help
- Performance
  - Cost of Procedure call « same-machine RPC « network RPC
  - Means programmers must be aware they are using RPC (so much for transparency!)
    - » Caching can help, but may make failure handling even more complex

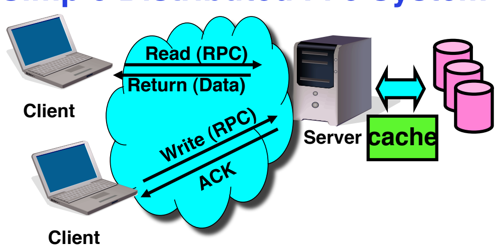
Distributed File Systems



- Distributed File System:
  - Transparent access to files stored on a remote disk
- Naming choices (always an issue):
  - *Hostname*:localname: Name files explicitly
    - » No location or migration transparency
  - *Mounting* of remote file systems
    - » System manager mounts remote file system by giving name and local mount point
    - » Transparent to user: all reads and writes look like local reads and writes to user e.g. /users/sue/foo→/sue/foo on server
  - *A single, global name space:* every file in the world has unique name
    - » Location Transparency: servers can change and files can move without involving user

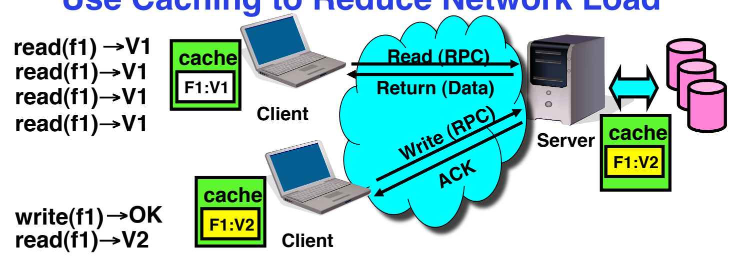


Simple Distributed File System



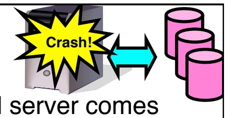
- EVERY read and write gets forwarded to server
- Advantage: Server provides completely consistent view of file system to multiple clients
- Problems? Performance!
  - Going over network is slower than going to local memory
  - Server can be a bottleneck

Use Caching to Reduce Network Load



- Advantage: if open/read/write/close can be done locally, don't need to do any network traffic…fast!
- Problems:
  - Failure:
    - » Client caches have data not committed at server
  - Cache consistency!
    - » Client caches not consistent with server/each other

Failures



- What if server crashes? Can client wait until server comes back up and continue as before?
  - Any data in server memory but not on disk can be lost
  - Shared state across RPC: What if server crashes after seek? Then, when client does “read”, it will fail
  - Message retries: suppose server crashes after it does UNIX “rm foo”, but before acknowledgment?
    - » Message system will retry: send it again
    - » How does it know not to delete it again? (could solve with two-phase commit protocol, but NFS takes a more ad hoc approach)
- Stateless protocol: A protocol in which all information required to process a request is passed with request
  - Server keeps no state about client, except as hints to help improve performance (e.g. a cache)
  - Thus, if server crashes and restarted, requests can continue where left off (in many cases)
- What if client crashes?
  - Might lose modified data in client cache

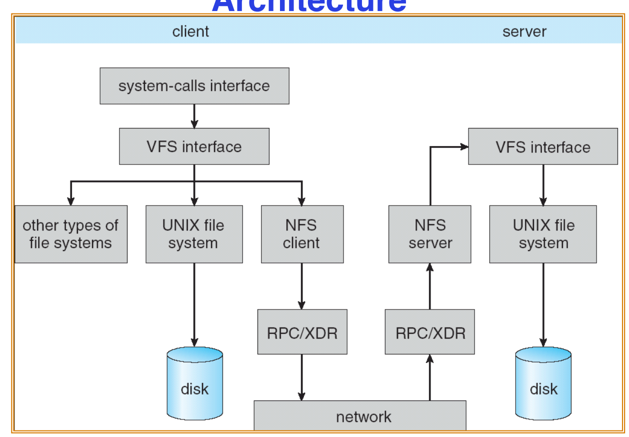
Network File System (NFS)

- Three Layers for NFS system
  - UNIX file-system interface: open, read, write, close calls + file descriptors
  - VFS layer: distinguishes local from remote files
    - » Calls the NFS protocol procedures for remote requests
  - NFS service layer: bottom layer of the architecture
    - » Implements the NFS protocol
- NFS Protocol: RPC for file operations on server
  - Reading/searching a directory
  - Manipulating links and directories
  - Accessing file attributes/reading and writing files
- Write-through caching: Modified data committed to server’s disk before results are returned to the client
  - Lose some of the advantages of caching
  - Time to perform write() can be long
  - Need some mechanism for readers to eventually notice changes! (more on this later)

NFS Continued

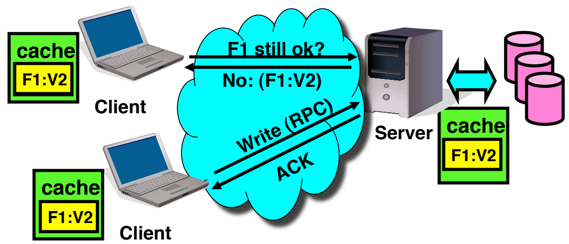
- NFS servers are stateless; each request provides all arguments require for execution
  - E.g. reads include information for entire operation, such as `ReadAt(inumber,position)`, not `Read(openfile)`
  - No need to perform network open() or close() on file – each operation stands on its own
- Idempotent: Performing requests multiple times has same effect as performing it exactly once
  - Example: Server crashes between disk I/O and message send, client resend read, server does operation again
  - Example: Read and write file blocks: just re-read or re-write file block – no side effects
  - Example: What about “remove”? NFS does operation twice and second time returns an advisory error
- Failure Model: Transparent to client system
  - Is this a good idea? What if you are in the middle of reading a file and server crashes?
  - Options (NFS Provides both):
    - » Hang until server comes back up (next week?)
    - » Return an error. (Of course, most applications don’t know they are talking over network)

Schematic View of NFS Architecture



NFS Cache consistency

- NFS protocol: weak consistency
  – Client polls server periodically to check for changes
    » Polls server if data hasn't been checked in last 3-30 seconds (exact timeout it tunable parameter).
    » Thus, when file is changed on one client, server is notified, but other clients use old version of file until timeout.



  – What if multiple clients write to same file?
    » In NFS, can get either version (or parts of both)
    » Completely arbitrary!

NFS Pros and Cons

- NFS Pros:
  – Simple, Highly portable
- NFS Cons:
  – Sometimes inconsistent!
  – Doesn't scale to large # clients
    » Must keep checking to see if caches out of date
    » Server becomes bottleneck due to polling traffic

Administrivia

- Updated Project 4 spec and skeleton will be posted by Friday
- Final Exam Review
  – Monday 5/6, 2-5pm in 100 Lewis Hall
- Final Exam
  – Friday 5/17, 8-11am in 1 Pimentel
  – All material from the course
    » With slightly more focus on second half, but you are still responsible for all the material
  – Two sheets of notes, both sides
  – Dumb calculator allowed

Quiz 23.1: RPC and NFS

- Q1: True _ False _ RPC requires special networking support and functionality
- Q2: True _ False _ The client and server for RPC must use the same hardware architecture (e.g., little endian)
- Q3: True _ False _ Local procedure call << same-machine RPC << remote machine RPC
- Q4: True _ False _ NFS provides weak client-server data consistency

5min Break

Quiz 23.1: RPC and NFS

- Q1: True _ False X RPC requires special networking support and functionality
- Q2: True _ False X The client and server for RPC must use the same hardware architecture (e.g., little endian)
- Q3: True X False _ Local procedure call << same-machine RPC << remote machine RPC
- Q4: True X False _ NFS provides weak client-server data consistency

Andrew File System

- Andrew File System (AFS, late 80's) → DCE DFS (commercial product)
- Callbacks: Server records who has copy of file
  - On changes, server immediately tells all with old copy
  - No polling bandwidth (continuous checking) needed
- Write through on close
  - Changes not propagated to server until close()
  - Session semantics: updates visible to other clients only after the file is closed
    - » As a result, do not get partial writes: all or nothing!
    - » Although, for processes on local machine, updates visible immediately to other programs who have file open
- In AFS, everyone who has file open sees old version
  - Don't get newer versions until reopen file

Andrew File System (con't)

- Data cached on local disk of client as well as memory
  - On open with a cache miss (file not on local disk):
    - » Get file from server, set up callback with server
  - On write followed by close:
    - » Send copy to server; tells all clients with copies to fetch new version from server on next open (using callbacks)
- What if server crashes? Lose all callback state!
  - Reconstruct callback information from client: go ask everyone "who has which files cached?"
- AFS Pro: Relative to NFS, less server load:
  - Disk as cache ⇒ more files can be cached locally
  - Callbacks ⇒ server not involved if file is read-only
- For both AFS and NFS: central server is bottleneck!
  - Performance: all writes→server, cache misses→server
  - Availability: Server is single point of failure
  - Cost: server machine's high cost relative to workstation

World Wide Web

- Key idea: graphical front-end to RPC protocol
- What happens when a web server fails?
  - System breaks!
  - Solution: Transport or network-layer redirection
    - » Invisible to applications
    - » Can also help with scalability (load balancers)
    - » Must handle "sessions" (e.g., banking/e-commerce)
- Initial version: no caching
  - Didn't scale well – easy to overload servers

WWW Caching

- Use client-side caching to reduce number of interactions between clients and servers and/or reduce the size of the interactions:
  - Time-to-Live (TTL) fields – HTTP "Expires" header from server
  - Client polling – HTTP "If-Modified-Since" request headers from clients
  - Server refresh – HTML "META Refresh tag" causes periodic client poll
- What is the polling frequency for clients and servers?
  - Could be adaptive based upon a page's age and its rate of change
- Server load is still significant!

WWW Proxy Caches

- Place caches in the network to reduce server load
  - But, increases latency in lightly loaded case
  - Caches near servers called "reverse proxy caches"
    - » Offloads busy server machines
  - Caches at the "edges" of the network called "content distribution networks"
    - » Offloads servers and reduce client latency
- Challenges:
  - Caching static traffic easy, but only ~40% of traffic
  - Dynamic and multimedia is harder
    - » Multimedia is a big win: Megabytes versus Kilobytes
  - Same cache consistency problems as before
- Caching is changing the Internet architecture
  - Places functionality at higher levels of comm. protocols

Conclusion

- Remote Procedure Call (RPC): Call procedure on remote machine
  - Provides same interface as procedure
  - Automatic packing and unpacking of arguments without user programming (in stub)
- Distributed File System:
  - Transparent access to files stored on a remote disk
    - » NFS uses caching for performance
- Cache Consistency: Keeping contents of client caches consistent with one another
  - If multiple clients, some reading and some writing, how do stale cached copies get updated?
  - NFS: check periodically for changes
- WWW: Caching to load balance, reduce latency/costs
  - Server and edge caches