Chapter 17: Distributed File Systems

CSCI 3753 Operating Systems

Prof. Rick Han

University of Colorado Boulder

Chapter 16: Distributed File Systems

- idea of a DFS is to be able to share remote files
  - instead of manually invoking scp or ftp to transfer files between machines, make remote files appear as if they are part of the local file system
    - this simplifies access to remote files - they appear just as part of your normal directory structure
    - remote files and directories have to first be attached or *mounted* to the local file system and directory structure
    - once mounted, remote files can be accessed and manipulated transparently using normal read/write/open/close function calls
      - you can also copy, rename, move, and change metadata of remote files just like local files
      - the networked file system will properly retrieve files from a remote host or ask the remote host to perform the requested operations on the file
- Examples of distributed file systems
  - Sun NFS (Network File System)
  - CMU's AFS (Andrew File System)
  - Microsoft's SMB/CIFS

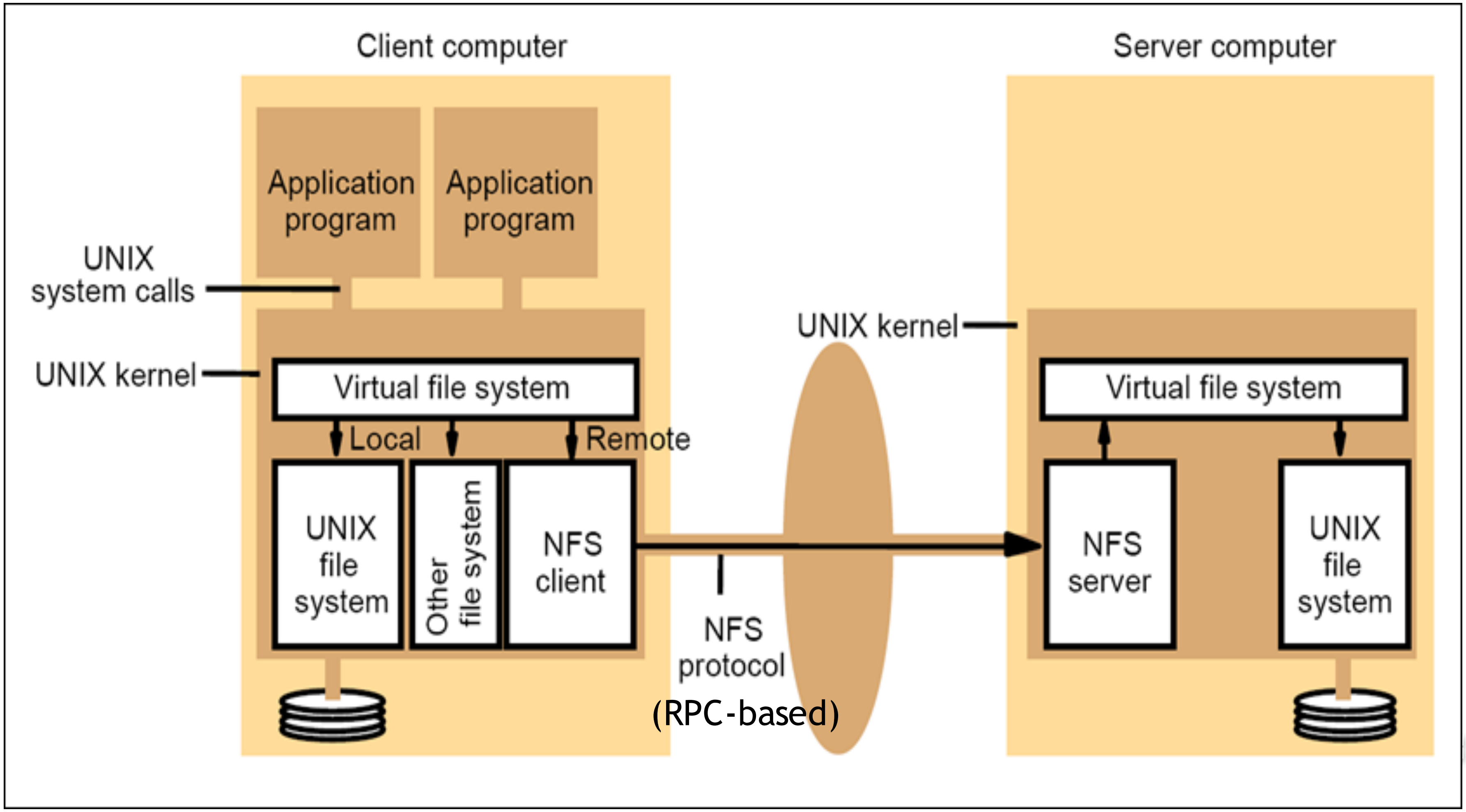
Distributed File Systems

- a DFS is built on top of the TCP/IP network stack
  - realized as an application that calls TCP (or UDP) through the socket API in order to transfer files back and forth across the network
    - NFS can use either TCP or UDP
  - there are usually two applications - a file client and a file server
    - a file client requests a file, modifies a file, or asks for metadata through the networking socket interface
    - a file server responds to a request and may return file data/metadata via network sockets or perform a write operation locally

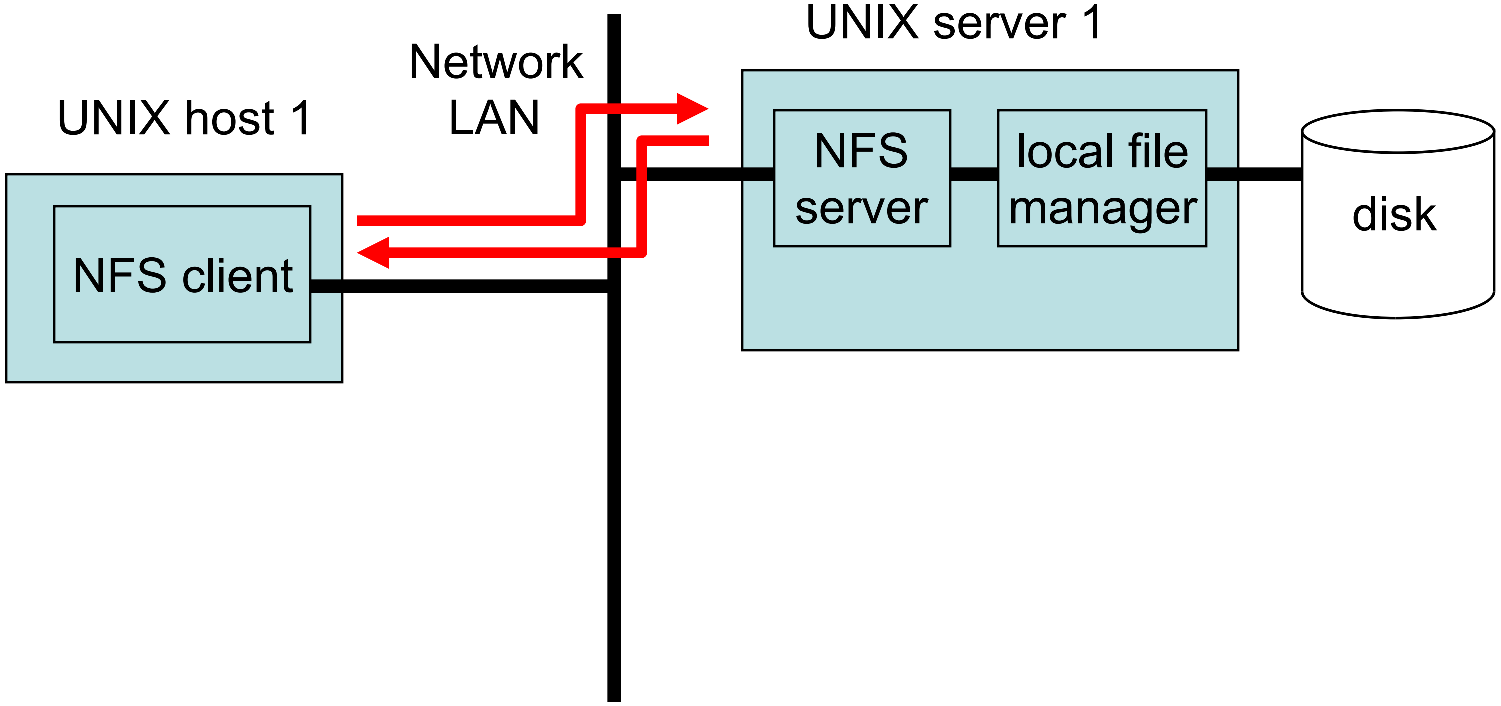
Network File System (NFS)

- a Virtual File System operates on an NFS client
  - files are abstracted as vnodes
  - virtual operations on these vnodes, e.g. read, write, open, rename, chmod, etc.
    - these vnode operations are translated into the specific system calls and formats of the underlying file system and OS
  - the VFS hides the details of the underlying file system (e.g. the particular file header format) and operating system, and also hides the location of files if they're remote
  - multiple different file systems can be supported simultaneously underneath a VFS interface
    - e.g. through the VFS, an NFS client could mount shared files on a Windows NT server running NTFS, a UNIX server running UNIX FS, and another machine running a FAT file system
- NFS is an RPC-based protocol - we'll cover RPC's later

NFS Architecture



NFS Architecture



- NFS client sends a vnode operation (e.g. read/write) on a file or directory to an NFS server daemon (nfsd) process executing on a remote server/host
  - NFS server translates the operation to the local file manager's language
  - depending on the operation (e.g. an open), a *file handle* may be returned to the NFS client which is used in future references to the file - this handle is a "pointer" to the file (essentially contains the inode of the file)
  - there may be multiple roundtrips needed, e.g. to find a remote file /usr/local/bin/emacs, NFS will first request the file handle from the NFS server for the mountpoint (let's say /usr/local was NFS-mounted), then will request the file handle for the bin directory, then the file handle for the emacs file

NFS Architecture

- NFS protocol
  - in earlier versions of NFS (v2 and v3), there was a separate mountd daemon (in addition to nfsd) that was used to mount a remote file system
    - today, the mounting feature is incorporated directly into NFS v4
    - if a client wishes to NFS-mount a remote directory /usr/local on server1, then use the tool mount, and type "mount server1:/usr/local /usr/local"
  - NFS protocol (v3 and earlier, not v4) was *stateless*
    - i.e. the server contains no information about previous NFS client requests
      - implication is that each NFS request must be wholly self-contained, i.e. must contain all the information necessary to process the request
      - e.g. a write() request must contain the file handle, the offset into the file, the length, and the data to be written
    - a stateless approach was chosen to minimize the complexity of crash recovery
    - if a file server crashes, it just needs to be restarted, and can begin servicing new requests immediately
    - if a file client crashes, it can be restarted without requiring any coordination with the file server

NFS Architecture

- NFS protocol:
  - to improve performance, both the NFS client and server can cache data
    - client-side caching
      - if a read() request can be served by the NFS client's cache, then return immediately with data, and no need for network communication
      - NFS client may also send additional read() requests to pre-fetch the next parts of a file in anticipation of future sequential reads - this is call *read-ahead*
      - write()'s can be cached as well and written asynchronously to the NFS server - this is called *write-behind*,
    - server-side caching
      - NFS server caches in RAM recently accessed directory entries, file inodes, and file data, to avoid server-side disk accesses
  - NFS version 4 adds:
    - statefulness at the server - keeps track of open() files
    - more caching at client - because of open() state, can cache more writes at client
    - improved security
    - improved performance
    - Is influenced by the Andrew File System (AFS)

Distributed File Systems

- Samba (SMB) protocol
  - now called CIFS
  - this is another protocol that allows file sharing between UNIX and Windows machines
    - it's a competitor to NFS
  - a UNIX server can export its files to a Windows client by running a Samba server daemon process
    - these files appear as normal files, and you can copy files back and forth between a UNIX server and your local Windows box
    - find the "Map Network Drive" menu in windows to mount the UNIX server's file system
    - smbd and nmbd are the Samba server daemons running on the UNIX server
  - Can also share files in the opposite direction, i.e. a Windows server can export its files to a UNIX client
    - Windows already supports SMB natively and contains a Samba server ready to export its files
    - run `smbclient` on the UNIX workstation to access the Windows box's files

File Hosting Services

- Today, many online services offer remote file hosting
  - These services are typically accessed via the HTTP protocol.
    - HTTP is convenient because it is ubiquitous in our browsers and provides the right functionality to read/write files via HTTP GET and PUT/POST
    - In comparison, distributed file systems are typically accessed via TCP/UDP directly, or via RPC.
  - E.g. Dropbox, Google Drive
  - Question of whether such file hosting systems are built for high performance, as distributed file systems are, or just for the occasional file download

Supplementary Slides

Remote Procedure Call (RPC)

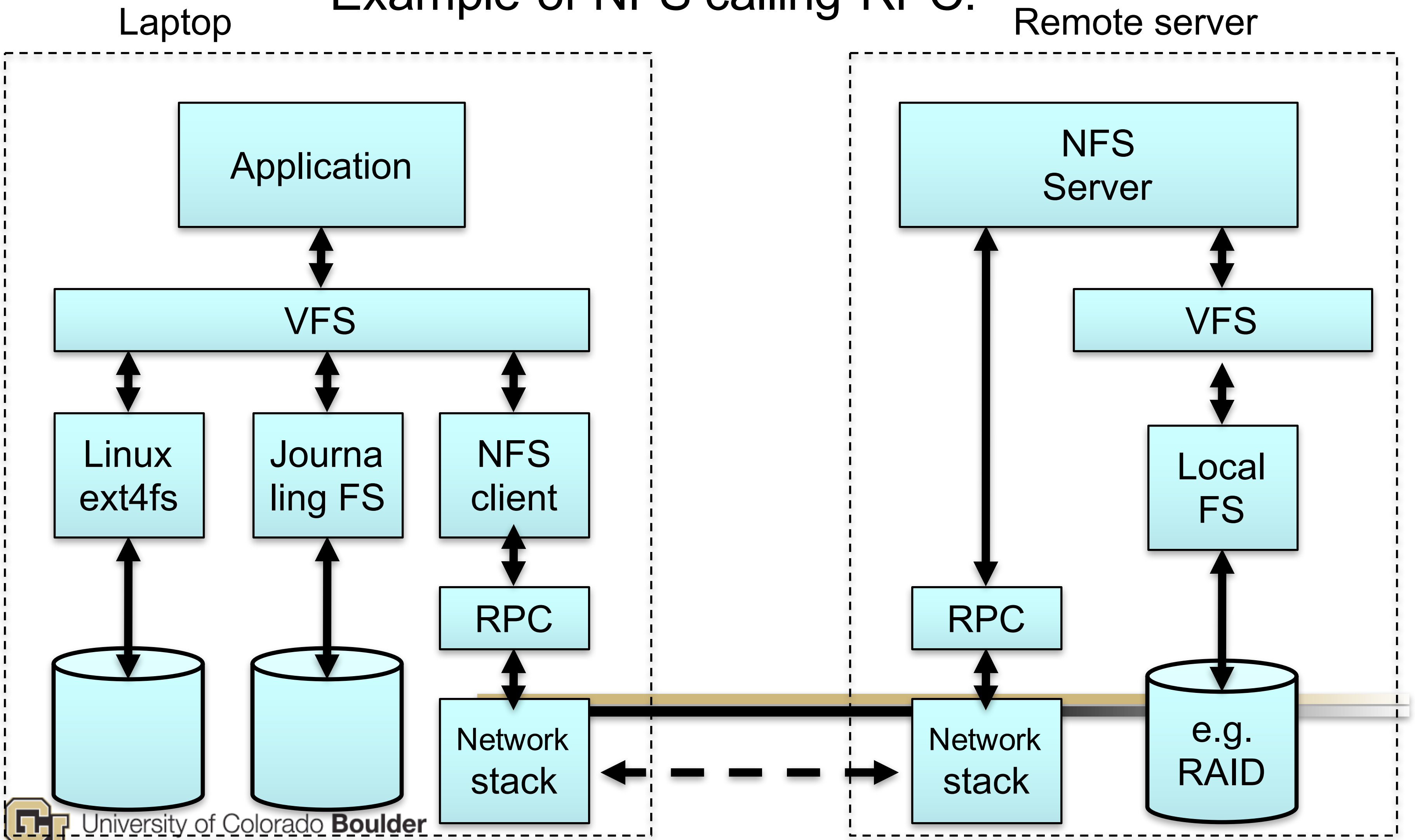
- NFS is actually built on top of RPC, i.e. each file request or vnode operation is sent as a remote procedure call to the NFS server
  - More precisely, the NFS client makes an RPC call, which is then sent over a TCP or UDP socket to the remove server, where the RPC call will be processed and handed over to the NFS server
- an RPC allows remote execution of a function call, e.g. read(fh, size, &data, &metadata) executes on the server and its results are sent back to the client
  - the client's local routine is a *stub routine* that packs the arguments and sends the command to the server - this packing procedure is called *marshalling*
  - the client stub typically blocks until it gets the return results from the server - thus RPC calls are typically *synchronous*
  - the server unpacks the RPC message, performs the operation, and sends a return message back to the client's RPC caller routine
  - the client receives the results and returns from the RPC call

Remote Procedure Call (RPC)

- Note that in the procedure call read(fh, size, &data, &metadata) that the pointers &data and &metadata make no sense in the address space of the remote RPC server.
  - The remote server can't access the data referenced to by these pointers. These pointers only make sense within the address space of the calling process.
  - Therefore, the RPC client must *marshal* the arguments of the RPC call, i.e. convert pointer arguments such as &metadata into actual copies of the data structures, and send the copies of the data structures along with the RPC call. This is unlike local procedure calls, which can easily dereference pointers to access the data.

Remote Procedure Call (RPC)

Example of NFS calling RPC:


Laptop
Remote server
Application
VFS
Linux ext4fs
Journaling FS
NFS client
RPC
Network stack
NFS Server
VFS
Local FS
RPC
Network stack
e.g. RAID