# Networked and Distributed File Systems
# CS 111
# Operating Systems
# Peter Reiher

# Outline

- Goals and challenges of providing file systems over the network
- Basic architectures
- Major issues
  - Authentication and security
  - Performance
- Examples of networked file systems

# Network File Systems: Goals and Challenges

- Sometimes the files we want aren't on our machine
- We'd like to be able to access them anyway
- How do we provide access to remote files?
  - Basic goals
  - Functionality challenges
  - Performance challenges
  - Robustness challenges
  - Manageability challenges

# Basic Goals

- Transparency
  - Indistinguishable from local files for all uses
  - All clients see all files from anywhere
- Performance
  - Per-client: at least as fast as local disk
  - Scalability: unaffected by the number of clients
- Cost
  - Capital: less than local (per client) disk storage
  - Operational: zero, it requires no administration
- Capacity: unlimited, it is never full
- Availability: 100%, no failures or service down-time

# Functionality Challenges

- Transparency
  - Making remote files look just like local files
    - On a network of heterogenous clients and servers
    - In the face of Deutch's warnings
  - Creating global file name-spaces
- Security
  - WAN scale authentication and authorization
- Providing ACID properties
  - Atomicity, Consistency, Isolation, Durability

# Performance Challenges

- Single client response-time
  - Remote requests involve messages and delays
- Aggregate bandwidth
  - Each client puts message processing load on server
  - Each client puts disk throughput load on server
  - Each message loads server's NIC and network
- WAN scale operation
  - Where bandwidth is limited and latency is high
- Aggregate capacity
  - How to transparently grow existing file systems

# Robustness Challenges

- All files should always be available, despite …
  - Failures of the disk on which they are stored
  - Failures of the Remote File Access server
  - Regional catastrophes (flood, earthquake, etc.)
  - Users having deleted the files
- Fail-over should be prompt and seamless
  - A delay of a few seconds might be acceptable
- Recovery must be entirely automated
  - For time, cost, and correctness reasons

# Manageability Challenges

- Storage management
  - Integrating new storage into the system
  - Diagnosing and replacing failed components
- Load and capacity balancing
  - Spreading files among volumes and servers
  - Spreading clients among servers
- Information life cycle management
  - Moving unused files to less expensive storage
  - Archival "compliance," finding archived data
- Client configuration
  - Domain services, file servers, name-spaces, authentication

# Security Challenges

- What meaningful security can we provide for networked file systems?
- Can we guarantee reasonable access control?
- How about secrecy of data crossing the network?
- How can we provide integrity guarantees to remote users?
- What if we can't trust all of the systems requesting files?
- What if we can't trust all of the systems storing files?

# Key Characteristics of Network File System Solutions

- APIs and transparency
  - How do users and processes access remote files?
  - How closely do remote files mimic local files?
- Performance and robustness
  - Are remote files as fast and reliable as local ones?
- Architecture
  - How is solution integrated into clients and servers?
- Protocol and work partitioning
  - How do client and server cooperate?

# Remote File Systems

- The simplest form of networked file system
- Basically, going to a remote machine to fetch files
- Perhaps with some degree of abstraction to hide unpleasant details
- But generally with a relatively low degree of transparency
  - Remote files are obviously remote

# Explicit File Copying

- User-invoked commands to transfer files
  - Copy to local site, then use as a local file
- Typical architecture
  - Client-side: interactive command line interface
    - May include powerful features like wild-cards, multi-file transfer, scheduled delivery, automatic difference detection, GUIs, etc.
  - Server-side: user mode, per client daemon
    - Basically, only this daemon knows file access is remote
- Many protocols are IETF standards
  - Some are very simple and general (FTP, TFTP)
  - Some assume a target OS and/or file system (rcp, rsync)

# Advantages and Disadvantages

- Advantages
  - User-mode client/server implementations
  - Efficient transfers (fast and with little overhead)
  - User directly controls what is transferred when
- Disadvantages
  - Human interfaces, awkward for programs to use
  - Local and remote files are totally different
  - Manual transfers are tedious and error prone
- Contemporary Usage
  - As a last resort
  - Some special applications (like remote boot)

# Remote Access Methods

- Distinct APIs for accessing remote files
  - Standard open/close/read/write are "local only"
  - Use different routines to access remote files
- Distinct user interface for remote files
  - Use a browser instead of a shell or finder
- User-mode implementation
  - Client remote access library, browser command
  - Protocols and servers similar to rcp/FTP
- New file naming schemes (e.g., URLs)

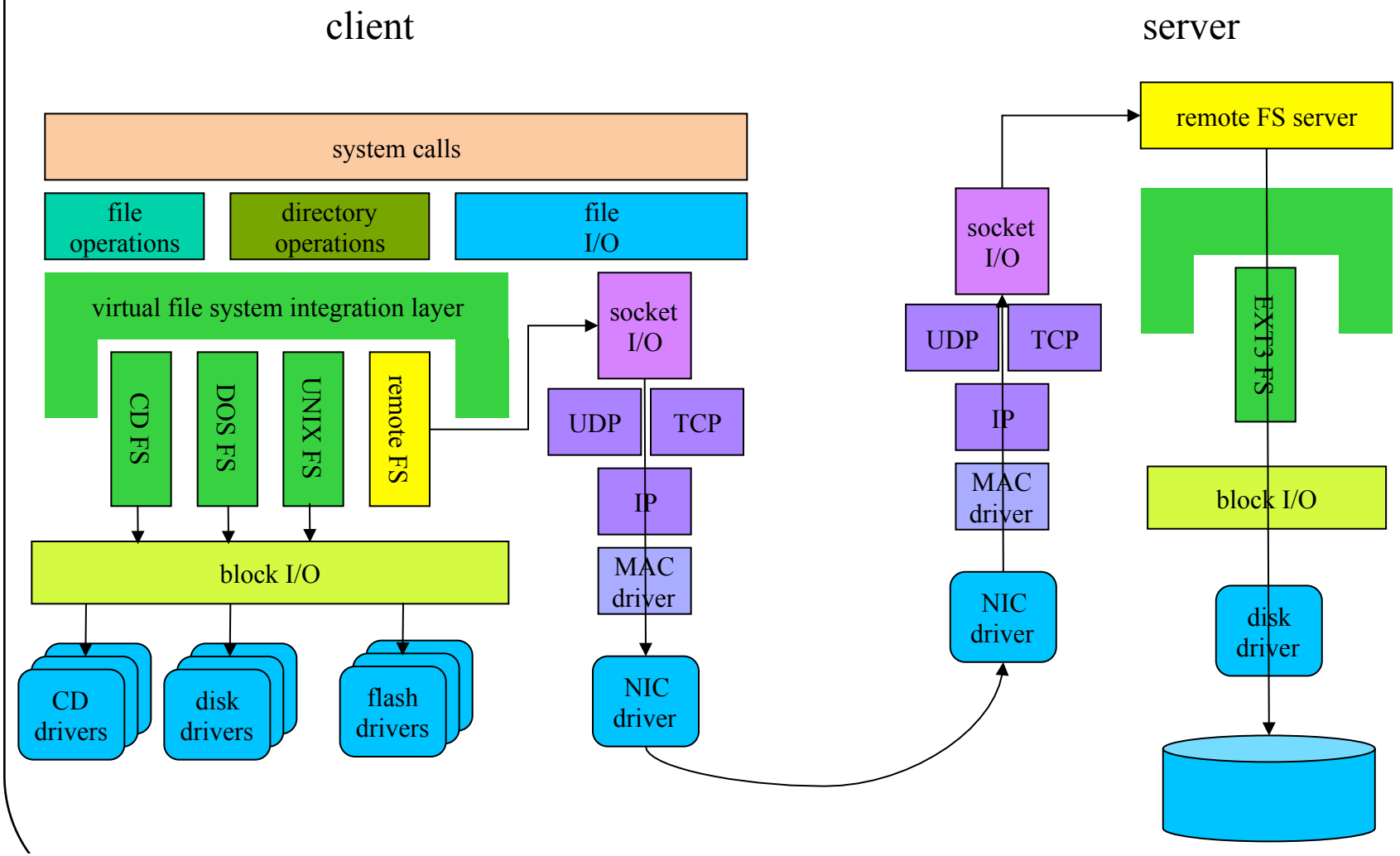
# Advantages and Disadvantages

- Advantages
  - User-mode client/server implementations
  - Services can be designed to suit modes of file use
  - Services encapsulate location of actual data
- Disadvantages
  - Only works for a few programs (e.g., browsers)
  - All other programs (e.g., editors) are "local only"
  - Local and remote files pretty distinct
  - Often no support for writing (or a special interface)
- Contemporary Usage
  - Many key applications: browsers, e-mail, SQL

# Remote File Access Protocols

- Goal: complete transparency
  - Normal file system calls work on remote files
  - Support file sharing by multiple clients
  - High performance, availability, reliability, scalability
- Typical Architecture
  - Uses plug-in file system architecture
  - Client-side file system is merely a local proxy
  - Translates file operations into network requests
  - Server-side daemon receives/process requests
  - Translates them into real file system operations

# Remote File Access Architecture



# The Client Side

- On Unix/Linux, makes use of VFS interface
- Allows plug-in of file system implementations
  - Each implements a set of basic methods
    - create, delete, open, close, link, unlink, etc.
  - Translates logical operations into disk operations
- Remote file systems can also be implemented
  - Translate each standard method into messages
  - Forward those requests to a remote file server
  - RFS client only knows the RFS protocol
    - Need not know the underlying on-disk implementation

# Server Side Implementation

- RFS Server Daemon
  - Receives and decodes messages
  - Does requested operations on local file system
- Can be implemented in user- or kernel-mode
  - Kernel daemon may offer better performance
  - User-mode is much easier to implement
- One daemon may serve all incoming requests
  - Higher performance, fewer context switches
- Or could be many per-user-session daemons
  - Simpler, and probably more secure

# Advantages and Disadvantages

- Advantages
  - Very good application level transparency
  - Very good functional encapsulation
  - Able to support multi-client file sharing
  - Potential for good performance and robustness
- Disadvantages
  - At least part of implementation must be in the OS
  - Client and server sides tend to be fairly complex
- Contemporary use
  - Ubiquitous today, and the wave of the future

# Clustered File Servers

- Use several cooperating file servers in one of the previously discussed ways
- Can aggregate their bandwidth and storage capacity
- Allows client load and file capacity balancing
- Virtualized storage cluster allows us to respond to difficult customer demands
  - Infinite bandwidth
  - Capacity scalability
  - Minimal down-time

# Degrees of Distribution

- Remote file access
  - One server owns disks and implements file systems
  - Clients access files via remote access protocols
- Clustered file servers
  - Multiple servers, each owns disks and file systems
  - Cooperate to provide a single virtual NAS service
- Distributed file systems
  - N servers and M disks
  - Multiple servers can concurrently use same disk
  - "Don't try this one at home, kids"