# Operating Systems Principles

# Distributed File Systems

Mark Kampe
(markk@cs.ucla.edu)

## Distributed File Systems





## Remote Data Access: 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 down-time



## Client/Server Models

- Peer-to-Peer
  - most systems have resources (e.g. disks, printers)
  - they cooperate/share with one-another
- Thin Client
  - few local resources (e.g. CPU, NIC, display)
  - most resources on work-group or domain servers
- Cloud Services
  - clients access services rather than resources
  - clients do not see individual servers



## Remote File Transfer

- explicit commands to copy remote files
  - OS specific: *scp(1)*, *rsync(1),* **S3** tools
  - IETF protocols: FTP, SFTP
- implicit remote data transfers
  - browsers (transfer files with HTTP)
  - email clients (move files with IMAP/POP/SMTP)
- advantages: efficient, requires no OS support
- disadvantages: latency, lack of transparency



## Remote Data Access

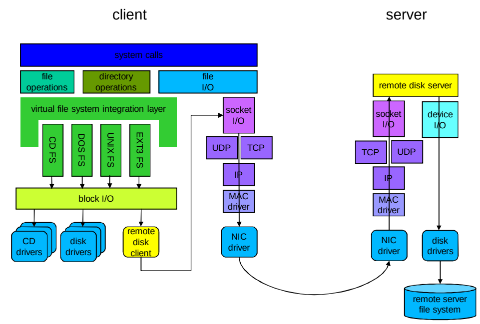
- OS makes remote files appear to be local
  - remote disk access (e.g. Storage Area Network)
  - remote file access (e.g. Network Attached Storage)
  - distributed file systems (NAS on steroids)
- advantages
  - transparency, availability, throughput
  - scalability, cost (capital and operational)
- disadvantages
  - complexity, issues with shared access

## Remote Disk Access

- Goal: complete transparency
  - normal file system calls work on remote files
  - all programs "just work" with remote files
- Typical Architectures
  - Storage Area Network (SCSI over Fibre Chanel)
    - very fast, very expensive, moderately scalable
  - iSCSI (SCSI over ethernet)
    - client driver turns reads/writes into network requests
    - server daemon receives/serves requests
    - moderate performance, inexpensive, highly scalable



## Remote Disk Access Architecture






## Rating Remote Disk Access

- Advantages:
  - provides excellent transparency
  - decouples client hardware from storage capacity
  - performance/reliability/availability per back-end
- Disadvantages
  - inefficient fixed partition space allocation
  - can't support file sharing by multiple client systems
  - message losses can cause file system errors
- This is THE model for Virtual Machines

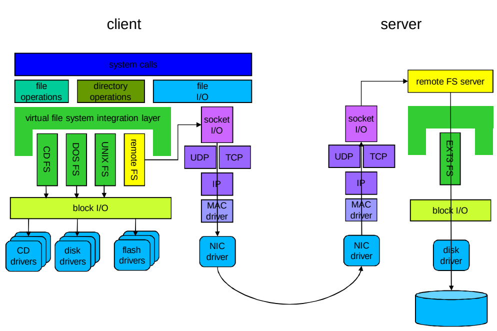


## Remote File Access

- Goal: complete transparency
  - normal file system calls work on remote files
  - support file sharing by multiple clients
  - performance, availability, reliability, scalability
- Typical Architecture
  - exploits plug-in file system architecture
  - client-side file system is 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






## Rating Remote File Access

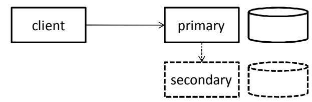
- Advantages
  - very good application level transparency
  - very good functional encapsulation
  - able to support multi-client file sharing
  - potential for good performance and robustness
- Disadvantages
  - at least part of implementation must be in the OS
  - client and server sides tend to be fairly complex
- This is THE model for client/server storage

## Cloud Model

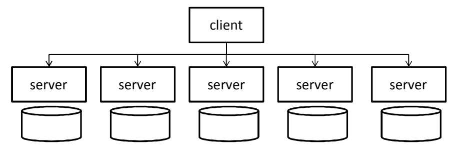
- a logical extension of client/server model
  - all services accessed via standard protocols
- opaque encapsulation of servers/resources
  - resources are abstract/logical, thin-provisioned
  - one, highly available, IP address for all services
  - mirroring/migration happen under the covers
- protocols likely to be WAN-scale optimized
- advantages:
  - simple, scalable, highly available, low cost
  - a very compelling business model



## Remote Disk/File Access




## Distributed File System






## (Remote vs. Distributed FS)

- Remote File Access (e.g. NFS, CIFS)
  - client talks to (per FS) primary server
  - secondary server may take over if primary fails
  - advantages: simplicity
- Distributed File System (e.g. Ceph, RAMCloud)
  - data is spread across numerous servers
  - client may talk directly to many/all of them
  - advantages: performance, scalability
  - disadvantages: complexity++



## Security: Anonymous access

- all files available to all users
  - no authentication required
  - may be limited to read-only access
  - examples: anonymous FTP, HTTP
- advantages
  - simple implementation
- disadvantages
  - incapable of providing information privacy
  - write access often managed by other means



## Peer-to-Peer Security

- client-side authentication/authorization
  - all users are known to all systems
  - all systems are trusted to enforce access control
  - example: basic NFS
- advantages
  - simple implementation
- disadvantages
  - assumes all clients to be trusted
  - doesn't work in heterogeneous OS environment
  - universal user registry is not scalable



## Server Authenticated Sessions

- client agent authenticates to each server
  - session authorization based on those credentials
  - example: CIFS
- advantages
  - simple implementation
- disadvantages
  - may not work in heterogeneous OS environment
  - universal user registry is not scalable
  - no automatic fail-over if server dies

## Domain Authentication Service

- independent authentication of client & server
  - each authenticates with authentication service
  - each knows/trusts only the authentication service
- may issue signed “tickets”
  - assuring each of the others’ identity and rights
  - may be revocable or timed lease
- may establish secure two-way session
  - privacy – nobody else can snoop on conversation
  - integrity – nobody can generate fake messages

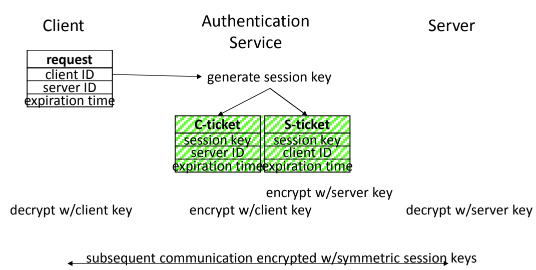


## example: KERBEROS

- establishes secure client/server sessions
- based on digital signatures
  - every agent has a secret (symmetric) key
  - keys are known only to agent, and KERBEROS
- request to KERBEROS encrypted w/client key
  - KERBEROS can decrypt it, authenticating requester
- KERBEROS response is two-part work ticket
  - part 1: encrypted with client's key
    - a symmetric session key
    - part 2 (to be forward, by client, to server)
  - part 2: encrypted with server's key
    - client ID, ticket duration,
    - symmetric session key



## KERBEROS Work Tickets






## Distributed Authorization

- Authentication service returns credentials
  - which server checks against Access Control List
  - advantage: auth service doesn’t know about ACLs
- Authentication service returns Capabilities
  - which server can verify (by signature)
  - advantage: servers do not know about clients
- Both approaches are commonly used
  - credentials: if subsequent authorization required
  - capabilities: if access can be granted all-at-once



## Reliability and Availability

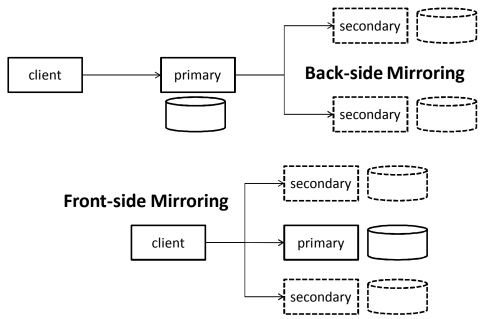
- Reliability … probability of not losing data
  - disk/server failures to not result in data loss
    - RAID (mirroring, parity, erasure coding)
    - copies on multiple servers
  - automatic recovery (of redundancy) after failure
- Availability … fraction of time service available
  - disk/server failures do not impact data availability
    - backup servers with automatic fail-over
  - automatic recovery (back up to date) after rejoin



## Availability: Fail-Over

- data must be mirrored to secondary server
- failure of primary server must be detected
- client must be failed-over to secondary
- session state must be reestablished
  - client authentication/credentials
  - session parameters (e.g. working directory, offset)
- in-progress operations must be retransmitted
  - client must expect timeouts, retransmit requests
  - client responsible for writes until server ACKs

## Reliability: Data Mirroring






## (Mirroring, Parity, Erasure Coding)

- Similar to trade-offs we made in RAID
  - the extra copies mean more network I/O
- Mirroring – multiple copies
  - fast, but requires a great deal of space
- Parity – able to recover from one/two errors
  - lower space overhead
  - requires full strip write buffering
- Erasure coding – recover with N/M copies
  - very space efficient
  - very slow/expensive reads and writes



## Availability: Failure Detect/Rebind

- client driven recovery
  - client detects server failure (connection error)
  - client reconnects to (successor) server
  - client reestablishes session
- transparent failure recovery
  - system detects server failure (health monitoring)
  - successor assumes primary's IP address
  - state reestablishment
    - successor recovers last primary state check-point
    - stateless protocol



## Availability: Stateless Protocols

- a statefull protocol (e.g. TCP)
  - operations occur within a context
  - each operation depends on previous operations
  - successor server must remember session state
- a stateless protocol (e.g. HTTP)
  - client supplies necessary context w/each request
  - each operation is complete and unambiguous
  - successor server has no memory of past events
- stateless protocols make fail-over easy

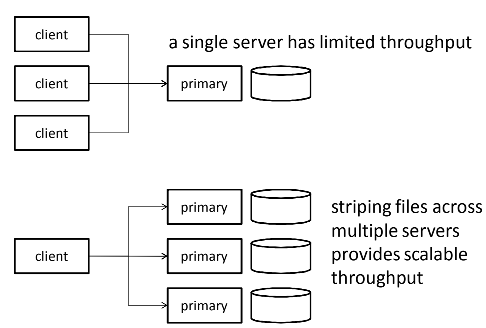


## Availability: Idempotent Operations

- can be repeated many times with same effect
  - read block 100 of file X
  - write block 100 of file X with contents Y
  - delete file X version 3
  - non-idempotent operations
    - read next block of current file
    - append contents Y to end of file X
- if client gets no response, resend request
  - if server gets multiple requests, no harm done
  - works for server failure, lost request, lost response
    - but no ACK does not mean operation did not happen



## Performance: Bandwidth



## Performance: Cost of Reads

- client-side caching
  - eliminate waits for remote read requests
  - reduces network traffic
  - reduces per-client load on server
- whole file (vs. block) caching
  - higher network latency justifies whole file pulls
  - stored in local (cache-only) file system
  - satisfy early reads before entire file arrives



## Performance: Cost of Writes

- write-back cache
  - create the illusion of fast writes
  - combine small writes into larger writes
  - fewer, larger network and disk writes
  - enable local read-after-write consistency
- whole-file updates
  - wait until *close(2)* or *fsync(2)*
  - reduce many successive updates to final result
  - possible file will be deleted before it is written
  - enable atomic updates, close-to-open consistency



## Performance: Cost of Consistency

- caching is essential in distributed systems
  - for both performance and scalability
- caching is easy in a single-writer system
  - force all writes to go through the cache
- multi-writer distributed caching is hard
  - Time To Live is a cute idea that doesn't work
  - constant validity checks defeat the purpose
  - one-writer-at-a-time is too restrictive for most FS
  - change notifications are a reasonable alternative

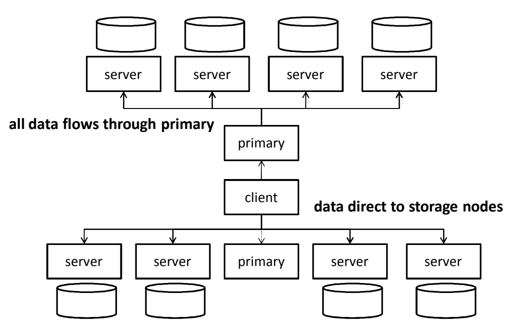


## Performance: Cost of Mirroring

- multi-host vs multi-disk mirroring
  - protects against host and disk failures
  - creates much additional network traffic
- mirroring by primary
  - primary becomes throughput bottleneck
  - replication traffic on back-side network
- mirroring by client
  - data flows directly from client to storage servers
  - replication traffic goes through client NIC
  - parity/erasure code computation on client CPU



## Performance: Direct Data Path



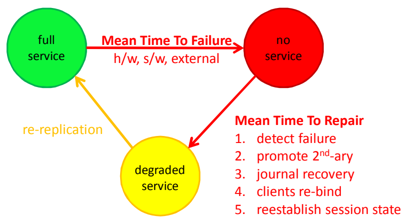



## (benefits of direct data path)

- architecture
  - primary tells clients where which data resides
  - client communicates directly w/storage servers
- throughput
  - data is striped across multiple storage servers
- latency
  - no intermediate relay through primary server
- scalability
  - fewer messages on network
  - much less data flowing through primary servers

## Performance: Recovery Time

$$\text{Availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$






## (availability)

- MTTR (time before service can be restored)
  - primary failure detected
  - secondary promoted to primary role
  - recent/in-progress operations recovered
  - clients learn of change and re-bind
  - session state (if any) has been reestablished
- Degraded service may persist longer
  - restoring lost redundancy may take a while
  - heavily loading servers, disks, and network



## Scalability – Traffic

- network messages are expensive
  - NIC and network capacity to carry them
  - server CPU cycles to process them
  - client delays awaiting responses
- minimize messages/client/second
  - cache results to eliminate requests entirely
  - enable complex operations w/single request
  - buffer up large writes in write-back cache
  - pre-fetch large reads into local cache

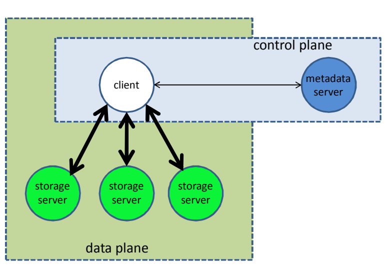


## Scalability - Bottlenecks

- avoid a single control points
  - partition responsibility over many nodes
- separated data- and control-planes
  - control nodes choreograph the flow of data
    - where data should be stored or obtained from
    - ensuring coherency and correct serialization
  - data flows directly from producer to consumer
    - data paths are optimized for throughput/efficiency
- dynamic re-partitioning of responsibilities
  - in response to failures and/or load changes



## Control and Data Planes

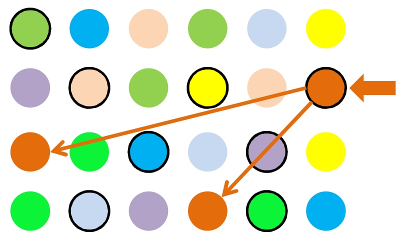





## Scalability: Cluster Protocols

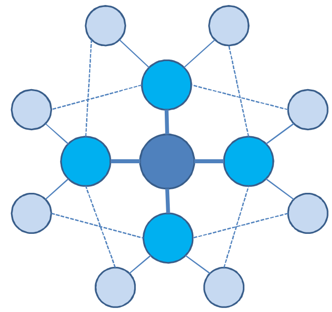
- Consensus protocols do not scale well
  - they only work for small numbers of nodes
- Minimize number of consensus operations
  - elect a single master who makes decisions
  - partitioned and delegated responsibility
- Avoid large-consensus/transaction groups
  - partition work among numerous small groups
- Avoid high communications fan-in/fan-out
  - hierarchical information gathering/distribution

## Small Transaction Clusters





## Hierarchical Communication Structure





## Assignments

- for the next lecture:
  - Symmetric Multi-Processors
  - Clustering Concepts
  - Cloud Concepts
  - Eventual Consistency