Lecture 2 – Distributed Filesystems

922EU3870 – Cloud Computing and Mobile Platforms, Autumn 2009

2009/9/21

Ping Yeh (葉平), Google, Inc.

Outline

- Get to know the numbers
- Filesystems overview
- Distributed file systems
  - Basic (example: NFS)
  - Shared storage (example: Global FS)
  - Wide-area (example: AFS)
  - Fault-tolerant (example: Coda)
  - Parallel (example: Lustre)
  - Fault-tolerant and Parallel (example: dCache)
- The Google File System
- Homework

Numbers real world engineers should know

L1 cache reference	0.5	ns
Branch mispredict	5	ns
L2 cache reference	7	ns
Mutex lock/unlock	100	ns
Main memory reference	100	ns
Compress 1 KB with Zippy	10,000	ns
Send 2 KB through 1 Gbps network	20,000	ns
Read 1 MB sequentially from memory	250,000	ns
Round trip within the same data center	500,000	ns
Disk seek	10,000,000	ns
Read 1 MB sequentially from network	10,000,000	ns
Read 1 MB sequentially from disk	30,000,000	ns
Round trip between California and Netherlands	150,000,000	ns

The Joys of Real Hardware

Typical first year for a new cluster:

~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)

~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)

~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)

~1 network rewiring (rolling ~5% of machines down over 2-day span)

~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)

~5 racks go wonky (40-80 machines see 50% packetloss)

~8 network maintenances (4 might cause ~30-minute random connectivity losses)

~12 router reloads (takes out DNS and external vips for a couple minutes)

~3 router failures (have to immediately pull traffic for an hour)

~dozens of minor 30-second blips for dns

~1000 individual machine failures

~thousands of hard drive failures

slow disks, bad memory, misconfigured machines, flaky machines, etc.

File Systems Overview

- System that permanently stores data
- Usually layered on top of a lower-level physical storage medium
- Divided into logical units called “files”
  - Addressable by a filename (“foo.txt”)
- Files are often organized into directories
  - Usually supports hierarchical nesting (directories)
  - A path is the expression that joins directories and filename to form a unique “full name” for a file.
- Directories may further belong to a volume
- The set of valid paths form the *namespace* of the file system.

What Gets Stored

- User data itself is the bulk of the file system's contents
- Also includes meta-data on a volume-wide and per-file basis:

Volume-wide:	Per-file:
Available space	name
Formatting info	owner
character set	modification date
…	physical layout…

High-Level Organization

- Files are typically organized in a “tree” structure made of nested directories
- One directory acts as the “root”
- “links” (symlinks, shortcuts, etc) provide simple means of providing multiple access paths to one file
- Other file systems can be “mounted” and dropped in as sub-hierarchies (other drives, network shares)
- Typical operations on a file: create, delete, rename, open, close, read, write, append.
  - also lock for multi-user systems.

Low-Level Organization (1/2)

- File data and meta-data stored separately
- File descriptors + meta-data stored in inodes (Un*x)
  - Large tree or table at designated location on disk
  - Tells how to look up file contents
- Meta-data may be replicated to increase system reliability

Low-Level Organization (2/2)

- “Standard” read-write medium is a hard drive (other media: CDROM, tape, ...)
- Viewed as a sequential array of blocks
- Usually address ~1 KB chunk at a time
- Tree structure is “flattened” into blocks
- Overlapping writes/deletes can cause fragmentation: files are often not stored with a linear layout
  – inodes store all block numbers related to file

Fragmentation

A	B	C	(free space)

A	B	C	A	(free space)

A	(free space)	C	A	(free space)

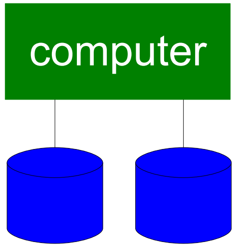
A	D	C	A	D	(free)

Filesystem Design Considerations

- Namespace: physical, logical
- Consistency: what to do when more than one user reads/ writes on the same file?
- Security: who can do what to a file? Authentication/ACL
- Reliability: can files not be damaged at power outage or other hardware failures?

Local Filesystems on Unix-like Systems

- Many different designs
- Namespace: root directory “/”, followed by directories and files.
- Consistency: “sequential consistency”, newly written data are immediately visible to open reads (if...)
- Security:
  - uid/gid, mode of files
  - kerberos: tickets
- Reliability: superblocks, journaling, snapshot
  - more reliable filesystem on top of existing filesystem: RAID



Namespace

- Physical mapping: a directory and all of its subdirectories are stored on the same physical media.
  - /mnt/cdrom
  - /mnt/disk1, /mnt/disk2, … when you have multiple disks
- Logical volume: a logical namespace that can contain multiple physical media or a partition of a physical media
  - still mounted like /mnt/vol1
  - dynamical resizing by adding/removing disks without reboot
  - splitting/merging volumes as long as no data spans the split

Journaling

- Changes to the filesystem is logged in a *journal* before it is committed.
  - useful if an supposedly atomic action needs two or more writes, e.g., appending to a file (update metadata + allocate space + write the data).
  - can play back a journal to recover data quickly in case of hardware failure. (old-fashioned “scan the volume” fsck anyone?)
- What to log?
  - changes to file content: heavy overhead
  - changes to metadata: fast, but data corruption may occur
- Implementations: xfs3, ReiserFS, IBM's JFS, etc.

RAID

- Redundant Array of Inexpensive Disks
- Different RAID levels
  - RAID 0 (striped disks): data is distributed among disk for performance.
  - RAID 1 (mirror): data is mirrored on another disk 1:1 for reliability.
  - RAID 5 (striped disks with distributed parity): similar to RAID 0, but with a parity bit for data recovery in case one disk is lost. The parity bit is rotated among disks to maximize throughput.
  - RAID 6 (striped disks with dual distributed parity): similar to RAID 5 but with 2 parity bits. Can lose 2 disks without losing data.
  - RAID 10 (1+0, striped mirrors)

Snapshot

- A snapshot = a copy of a set of files and directories at a point in time.
  - read-only snapshots, read-write snapshots
  - usually done by the filesystem itself, sometimes by LVMs
  - backing up data can be done on a read-only snapshot without worrying about consistency
- Copy-on-write is a simple and fast way to create snapshots
  - current data is the snapshot.
  - a request to write to a file creates a new copy, and work from there afterwards.
- Implementation: UFS, Sun's ZFS, etc.

- Should the file system be faster or more reliable?
- But faster at what: Large files? Small files? Lots of reading? Frequent writers, occasional readers?
- Block size
  - Smaller block size reduces amount of wasted space
  - Larger block size increases speed of sequential reads (may not help random access)

Distributed File Systems

Distributed Filesystems

- Support access to files on remote servers
- Must support concurrency
  - Make varying guarantees about locking, who “wins” with concurrent writes, etc...
  - Must gracefully handle dropped connections
- Can offer support for replication and local caching
- Different implementations sit in different places on complexity/feature scale

DFS is useful when...

- Multiple users want to share files
- Users move from one computer to another
- Users want a uniform namespace for shared files on every computer
- Users want a centralized storage system – backup and management

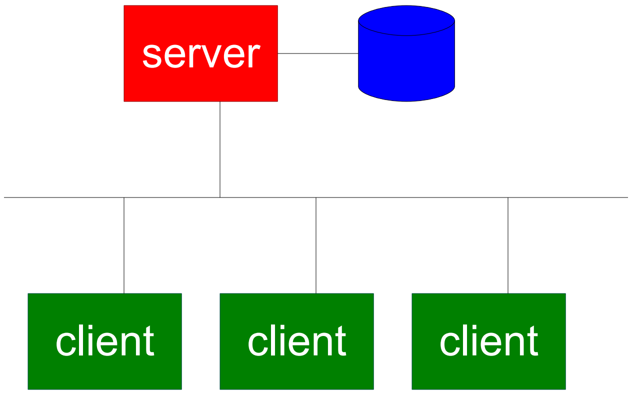
- Note that a “user” of a DFS may actually be a “program”

Features of Distributed File Systems

- Different systems have different designs and behaviors on the following features.
  - Interface: file system, block I/O, custom made
  - Security: various authentication/authorization schemes.
  - Reliability (fault-tolerance): can continue to function when some hardware fail (disks, nodes, power, etc).
  - Namespace (virtualization): can provide logical namespace that can span across physical boundaries.
  - Consistency: all clients get the same data all the time. It is related to locking, caching, and synchronization.
  - Parallel: multiple clients can have access to multiple disks at the same time.
  - Scope: local area network vs. wide area network.

NFS

- First developed in 1980s by Sun
- Most widely known distributed filesystem.
- Presented with standard POSIX FS interface
- Network drives are mounted into local directory hierarchy
  - Your home directory at CINC is probably NFS-driven
  - Type 'mount' some time at the prompt if curious



NFS Protocol

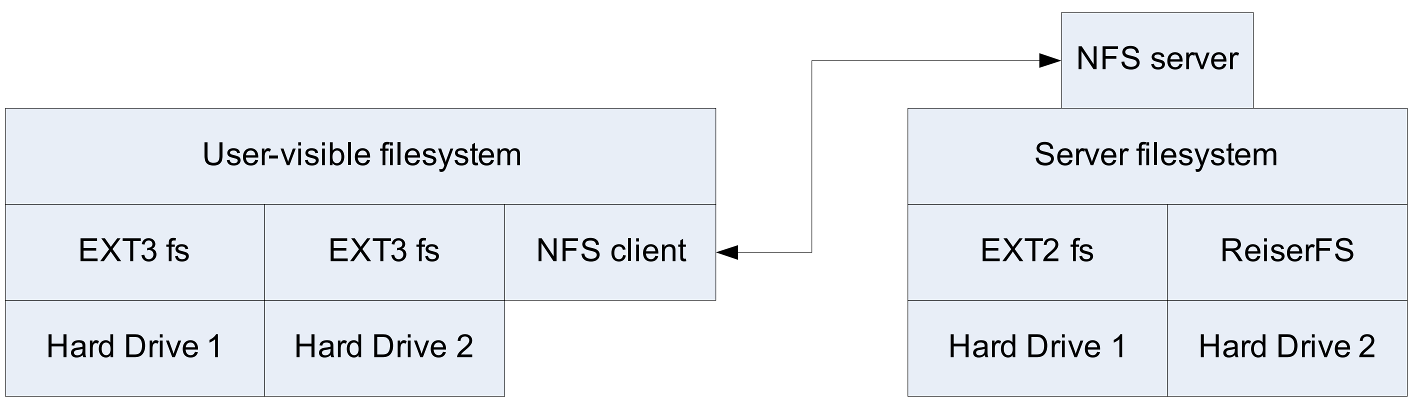
- Initially completely stateless
  - Operated over UDP; did not use TCP streams
  - File locking, etc, implemented in higher-level protocols
  - Implication: All client requests have enough info to complete op
    - example: Client specifies offset in file to write to
    - one advantage: Server state does not grow with more clients
- Modern implementations use TCP/IP & stateful protocols
- RFCs:
  - RFC 1094 for v2 (3/1989)
  - RFC 1813 for v3 (6/1995)
  - RFC 3530 for v4 (4/2003)

NFS Overview

- Remote Procedure Calls (RPC) for communication between client and server
- Client Implementation
  - Provides transparent access to NFS file system
    - UNIX contains Virtual File system layer (VFS)
    - Vnode: interface for procedures on an individual file
  - Translates vnode operations to NFS RPCs
- Server Implementation
  - Stateless: Must not have anything only in memory
  - Implication: All modified data written to stable storage before return control to client
    - Servers often add NVRAM to improve performance

Server-side Implementation

- NFS defines a virtual file system
  - Does not actually manage local disk layout on server
- Server instantiates NFS volume on top of local file system
  - Local hard drives managed by concrete file systems (ext*, ReiserFS, ...)
  - Export local disks to clients



NFS Locking

- NFS v4 supports stateful locking of files
  - Clients inform server of intent to lock
  - Server can notify clients of outstanding lock requests
  - Locking is lease-based: clients must continually renew locks before a timeout
  - Loss of contact with server abandons locks

NFS Client Caching

- NFS Clients are allowed to cache copies of remote files for subsequent accesses
- Supports *close-to-open* cache consistency
  - When client A closes a file, its contents are synchronized with the master, and timestamp is changed
  - When client B opens the file, it checks that local timestamp agrees with server timestamp. If not, it discards local copy.
  - Concurrent reader/writers must use flags to disable caching

Cache Consistency

- Problem: Consistency across multiple copies (server and multiple clients)

- How to keep data consistent between client and server?
  - If file is changed on server, will client see update?
  - Determining factor: Read policy on clients

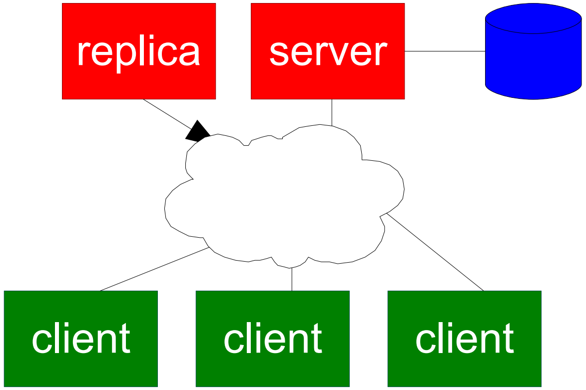
- How to keep data consistent across clients?
  - If write file on client A and read on client B, will B see update?
  - Determining factor: Write and read policy on clients

NFS: Summary

- Very popular
- Full POSIX interface means it “drops in” very easily.
- No cluster-wide uniform namespace: each client decides how to name a volume mounted from an NFS server.
- No location transparency: data movement means remount
- NFS volume managed by single server
  - Higher load on central server, bottleneck on scalability
  - Simplifies coherency protocols
- Challenges: Fault tolerance, scalable performance, and consistency

AFS (The Andrew File System)

- Developed at Carnegie Mellon University since 1983
- Strong security: Kerberos authentication
  - richer set of access control bits than UNIX: separate "administer"/"delete" bits, application-specific bits.
- Higher scalability compared with distributed file systems at the time. Supports 50,000+ clients at enterprise level.
- Global namespace: /afs/cmu.edu/vol1/..., WAN scope
- Location transparency: moving files just works.
- Heavily use client side caching.
- Support read-only replication.

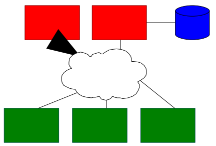


Morgan Stanley's comparison: AFS vs. NFS

- Better client/server ratio
  - NFS (circa 1993) topped out at 25:1
  - AFS: 100s
- Robust volume replication
  - NFS servers go down, and take their clients with them
  - AFS servers go down, nobody notices (OK, RO data only)
- WAN file sharing
  - NFS couldn't do it reliably
  - AFS worked like a charm
- Security was never an issue (kerberos4)

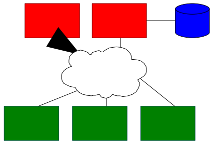
http://www-conf.slac.stanford.edu/AFSBestPractices/Slides/MorganStanley.pdf

Local Caching



- File reads/writes operate on locally cached copy
- The modified local copy is sent back to server when file is closed
- Open local copies are notified of external updates through callbacks
  - but not updated
- Trade-offs:
  - Shared database files do not work well on this system
  - Does not support write-through to shared medium

Replication



- AFS allows read-only copies of filesystem volumes
- Copies are guaranteed to be atomic checkpoints of entire FS at time of read-only copy generation (“snapshot”)
- Modifying data requires access to the sole r/w volume
  - Changes do not propagate to existing read-only copies

AFS: Summary

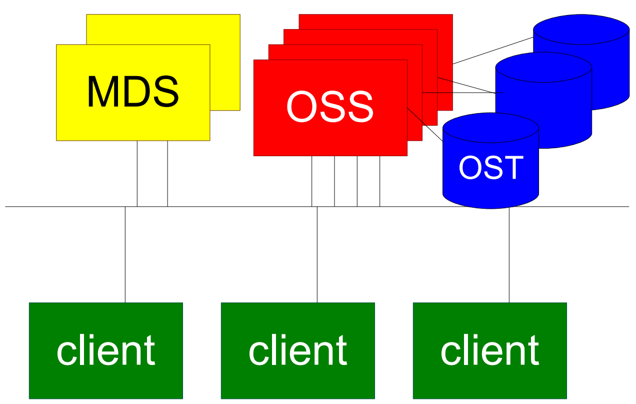
- Not quite POSIX
  - Stronger security/permissions
  - No file write-through
- High availability through replicas and local caching
- Scalability by reducing load on servers
- Not appropriate for all file types

ACM Trans. on Computer Systems, 6(1):51-81, Feb 1988.

Global File System

Lustre Overview

- Developed by Peter Braam at Carnegie Mellon since 1999
  - then Cluster File System in 2001, acquired by Sun in 2007
  - Goal: removing bottlenecks to achieve scalability
- Object-based: separate metadata and file data
  - metadata server(s): MDS (namespace, ACL, attributes)
  - object storage server(s): OSS
  - client read/write directly with OSS
- Consistency: Lustre distributed lock manager
- Performance: data can be striped
- POSIX interface



Key Design Issue : Scalability

- I/O throughput
  - How to avoid bottlenecks
- Metadata scalability
  - How can 10,000’s of nodes work on files in same folder
- Cluster Recovery
  - If sth fails, how can transparent recovery happen
- Management
  - Adding, removing, replacing, systems; data migration & backup

The Lustre Architecture

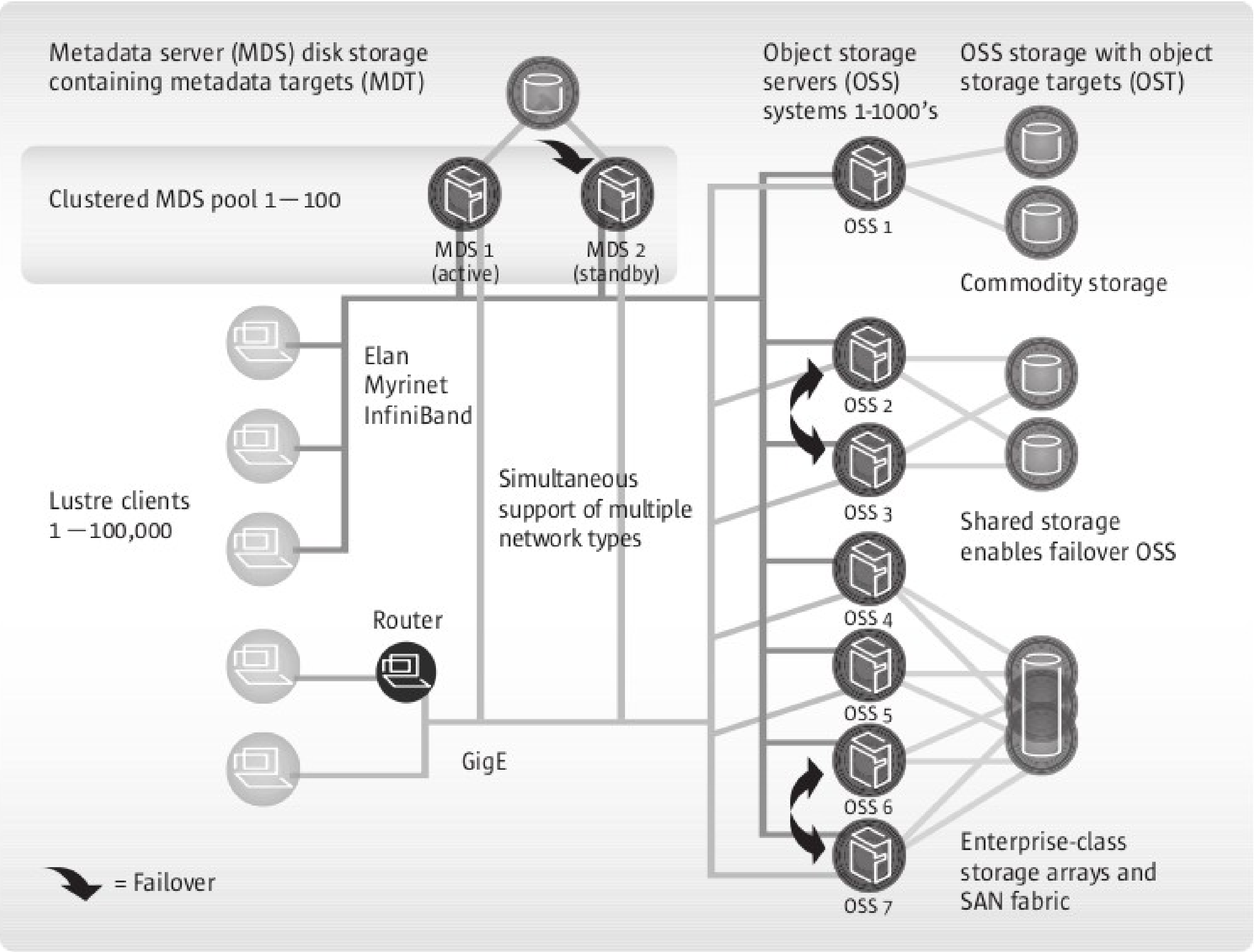


Figure 1. Lustre architecture for clusters

Metadata Service (MDS)

- All access to the file is governed by MDS which will directly or indirectly authorize access.
- To control namespace and manage inodes
- Load balanced cluster service for the scalability (a well balanced API, a stackable framework for logical MDS, replicated MDS)
- Journaled batched metadata updates

Object Storage Targets (OST)

- Keep file data objects
- File I/O service: Access to the objects
- The block allocation for data obj., leading distributed and scalability
- OST s/w modules
  - OBD server, Lock server
  - Obj. storage driver, OBD filter
  - Portal API

Distributed Lock Manager

- For generic and rich lock service
- Lock resources: resource database
  - Organize resources in trees
- High performance
  - node that acquires resource manages tree
- Intent locking: utilization dependent locking modes
  - write-back lock for low utilization files
  - no lock for high contention files, write to OSS by DLM

Metadata

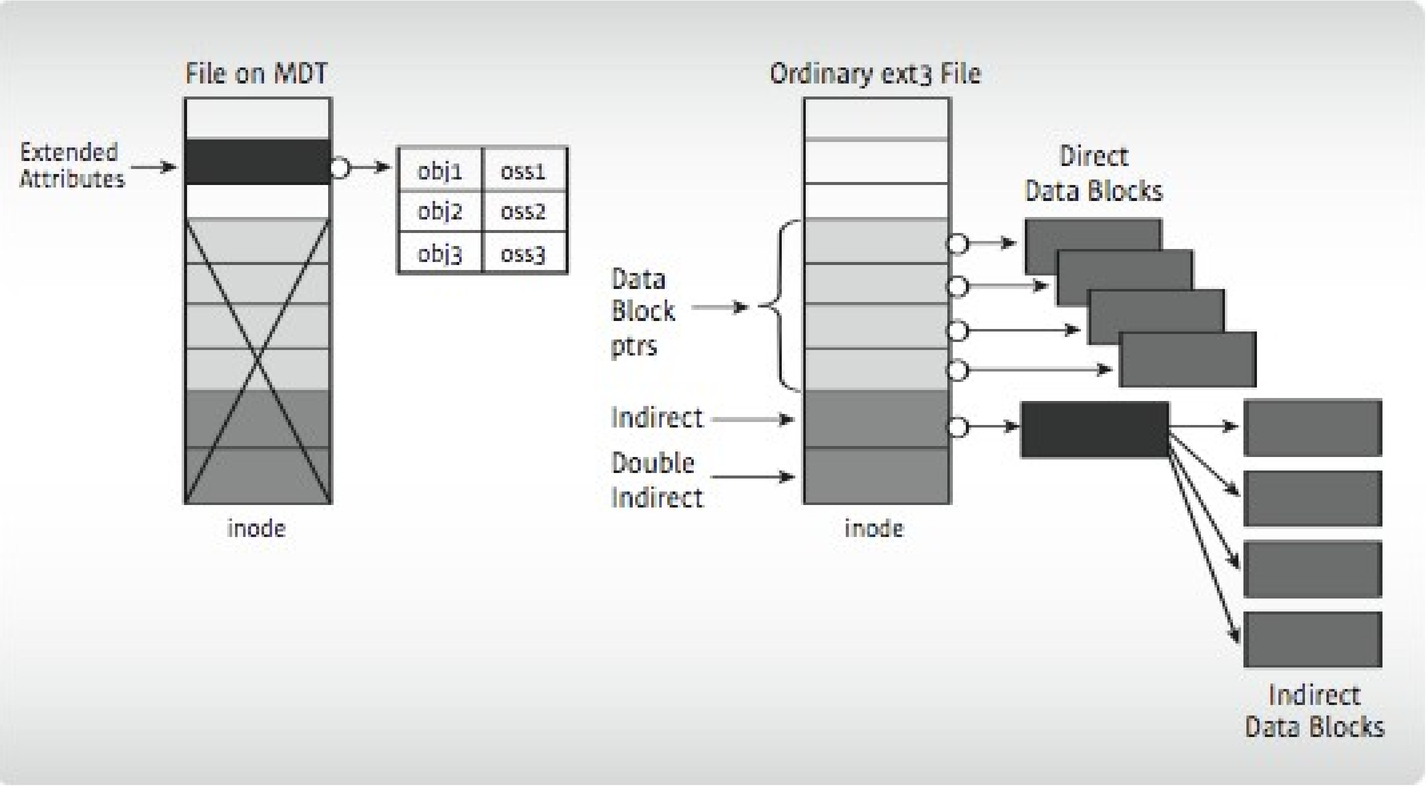


Figure 5. MDS inodes point to objects; ext3 inodes point to data

File I/O

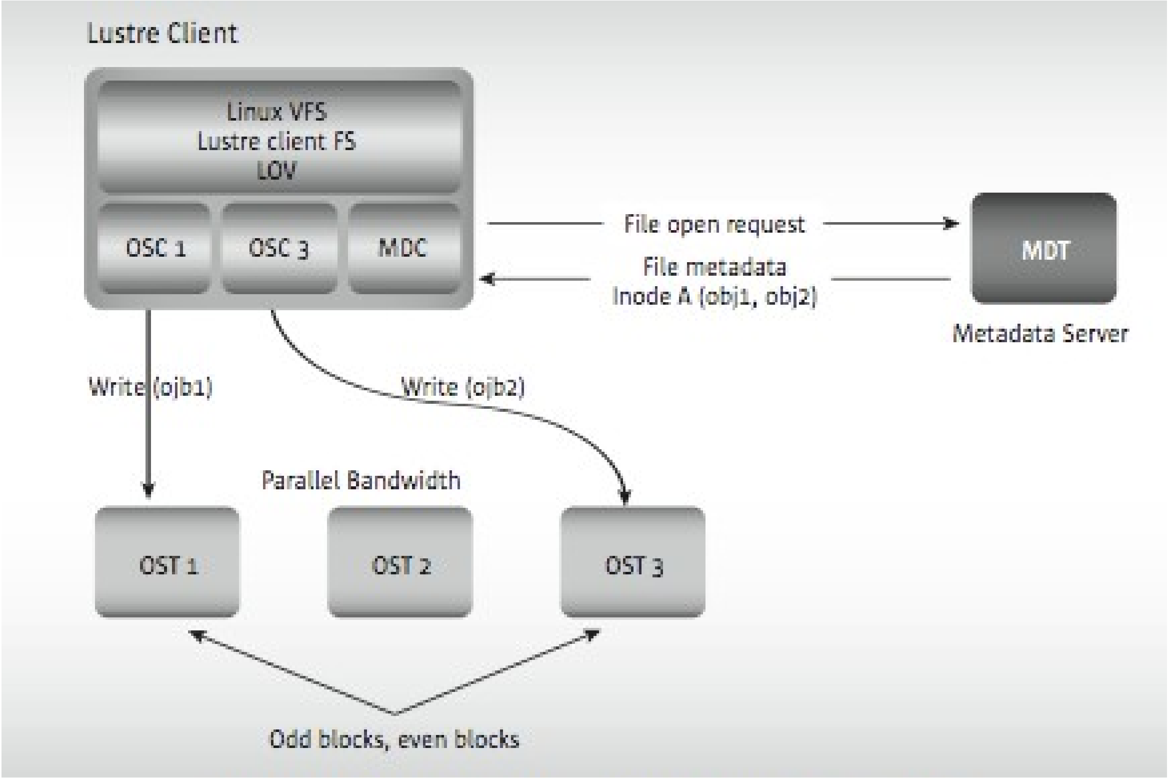


Figure 6. File open and file I/O in the Lustre file system

Striping

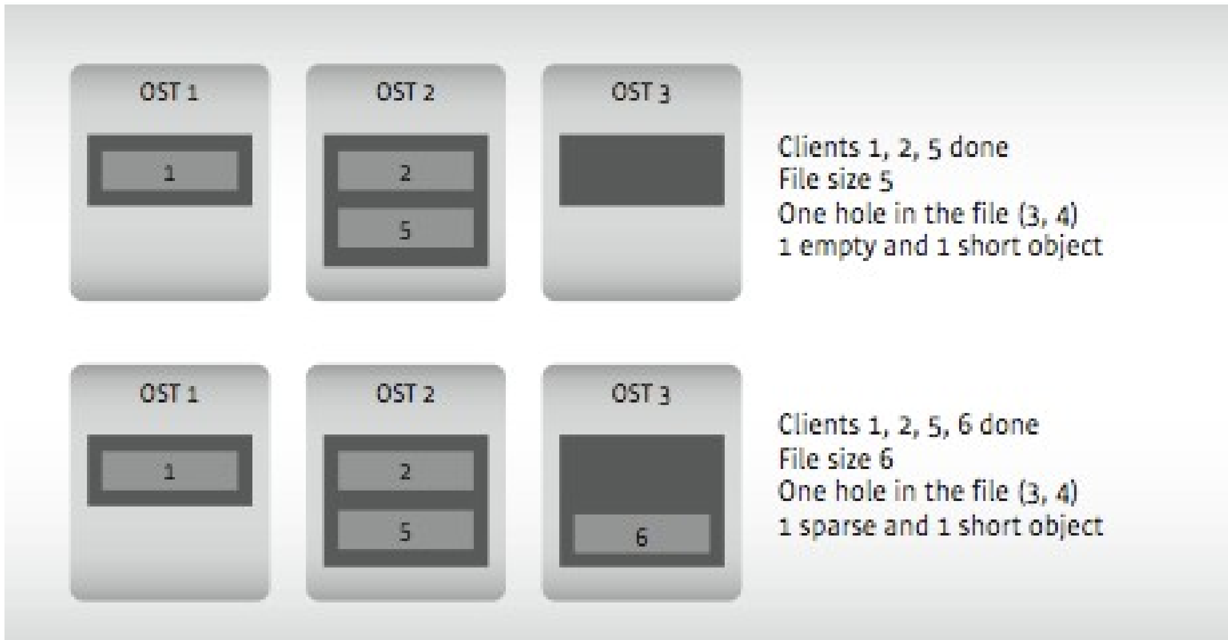
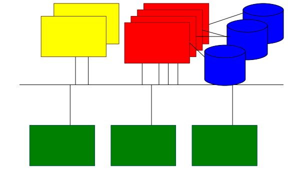


Figure 8. A compute application rendering a movie file with three objects



Lustre Summary

- Good scalability and high performance
  - Lawrence Livermore National Lab BlueGene/L has 200,000 clients processes access 2.5PB Lustre fs through 1024 IO nodes over 1Gbit Ethernet network at 35 GB/sec.
- Reliability
  - Journaled metadata in metadata cluster
  - OSTs are responsible for their data
- Consistency

Other file systems worth mentioning

- Coda: post-AFS, pre-Lustre, from CMU
- Global File System: shared disk file systems
- PVFS by IBM
- ZFS by Sun