Introduction to Cloud Computing

Distributed File Systems

15-319, spring 2010

12th Lecture, Feb 18th

Majd F. Sakr

Lecture Motivation

- Quick Refresher on Files and File Systems
- Understand the importance of File Systems in handling data
- Introduce Distributed File Systems
- Discuss HDFS

Files

- **File in OS?**
  - Permanent Storage
  - Sharing information since files can be created with one application and shared with many applications
  - Files have data and attributes

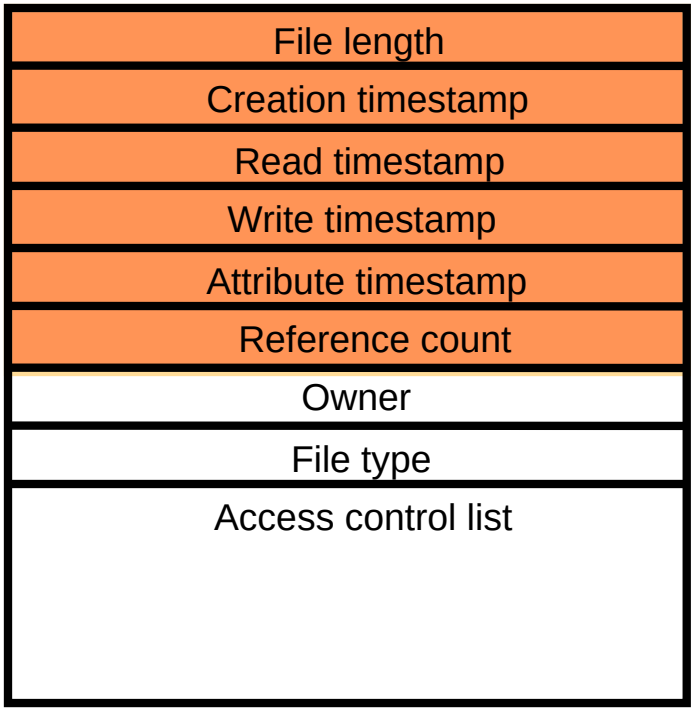


Figure 2: File attribute record structure

Couloris,Dollimore and Kindberg Distributed Systems: Concepts & Design Edn. 4 , Pearson Education 2005

File System

- The OS interface to disk storage
- Subsystem of the OS
- Provides an abstraction to storage device and makes it easy to store, organize, name, share, protect and retrieve computer files
- A typical layered module structure for the implementation of a Non-DFS in a typical OS:

Directory module:	relates file names to file IDs
File module:	relates file IDs to particular files
Access control module:	checks permission for operation requested
File access module:	reads or writes file data or attributes
Block module:	accesses and allocates disk blocks
Device module:	disk I/O and buffering

Couloris,Dollimore and Kindberg Distributed Systems: Concepts & Design Edn. 4 , Pearson Education 2005

Great! Now how do you Share Files?

- **1980s: Sneakernet**

  Copy files onto floppy disks, physically carry it to another computer and copy it again.
  - **We still do it today with Flash Disks!**

- **Networks emerged**
  - Started using **FTP**

    - Save time of physical movement of storage devices.

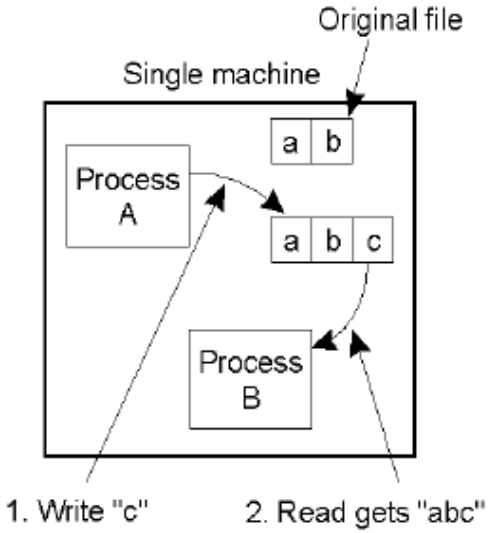
    - Two problems:
      - Needed to copy files twice: from source computer onto a server, and from the server onto the destination computer.
      - Users had to know the physical addresses of all computers involved in the file sharing.

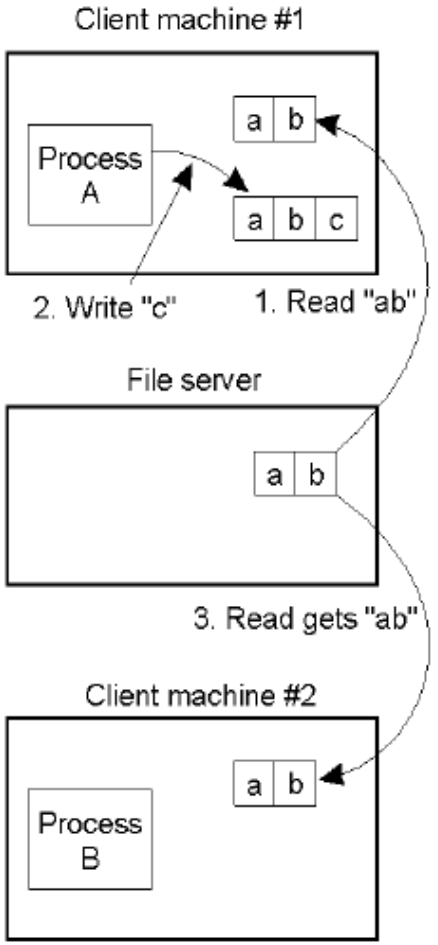
History of Sharing Computer Files

- **Networks emerged (contd.)**
  - Computer companies tried to solve the problems with **FTP**, new systems with new features were developed.
    - Not as a replacement for the older file systems but represented an additional layer between the disk, FS and user processes.
  - Example:

    Sun Microsystem's **Network File System (NFS).**

File Sharing (1/7)





- **On a single processor**, when a write is followed by a read, the read data is the accurate written one
- **On a distributed system with caching**, the read data might not be the most up to date.

http://www.nmc.teiher.gr/activities/MASTERS/JOINT/Material/Vall/DSC_2.pdf

File Sharing (2/7)

- How to deal with shared files on a distributed system with caches?

  There are 4 ways!

Method	Comment
UNIX semantics	Every operation on a file is instantly visible to all processes
Session semantics	No changes are visible to other processes until the file is closed
Immutable files	No updates are possible; simplifies sharing and replication
Transaction	All changes occur atomically

File Sharing (3/7)

- **UNIX semantics**
  - Every file operation is instantly visible to all users. So, any read following a write returns the correct value.
  - A total global order is enforced on all file operations to return the most recent value.
    - In a single physical machines, a shared l-Node is used to achieve this control.
    - Files data is a shared data structure among all users.
  - In Distributed file server, same behavior needs to be done!
    - Instant update cause performance implications.
    - Fine grain operations increase overhead.

File Sharing (4/7)

- **UNIX semantics**

  - Distributed UNIX semantics
    - Could use centralized server that can serialize all file operations.
    - Poor performance under many use patterns.

  - Performance constraints require that the clients cache file blocks, but the system must keep the cached blocks consistent to maintain UNIX semantics.
    - Writes invalidate cached blocks.
    - Read operations on local copies “after” the write according to a global clock happened “before” the write.
      - Serializable operations in transaction systems.
      - Global virtual clock orders on all writes, not reads.

File Sharing (5/7)

- **Session semantics**
  - Changes become visible when the session is finished.
  - When modified by multiple parties, the final file state is determined by who closes last.
  - When two processes modify the same file, session semantics would produce one process' changes or the other but not both.
    - Many processes keep files open for long periods.
  - This approach is different from most of programmers experience, so must be used with caution.

Good for process whose file modification is transaction oriented (connect, modify, disconnect).

Bad for series of open operations.

File Sharing (6/7)

- **Immutable Files**
  - No updates are possible.
    - Both file sharing and replication are simplified.
  - No way to open a file for writing or appending.
  - Only directory entries may be modified.
  - To replace or change an old file, a new one must be created.
  - Fine for many applications. However, it's different enough that it must be approached with caution.
  - Design Principle:
    - Many applications of distribution involve porting existing non-distributed code along with its assumptions.

File Sharing (7/7)

- **Atomic transactions**
  - Changes are all or nothing
    - Begin-Transaction
    - End-Transaction
  - System responsible for enforcing serialization.
  - Ensuring that concurrent transactions produce results consistent with some serial execution.
    - Transaction systems commonly track the read/write component operations.
  - Familiar aid of atomicity provided by transaction model to implementers of distributed systems.
    - Commit and rollback both very useful in simplifying implementation.

Distributed File System

- File System for a physically distributed set of files.
- Usually within an network of computers.
- Allows clients to access files on remote hosts as if the client is actually working on the host.
- Enables maintaining data consistency.
- Acts as a common data store for distributed applications.

Simple Example of a DFS

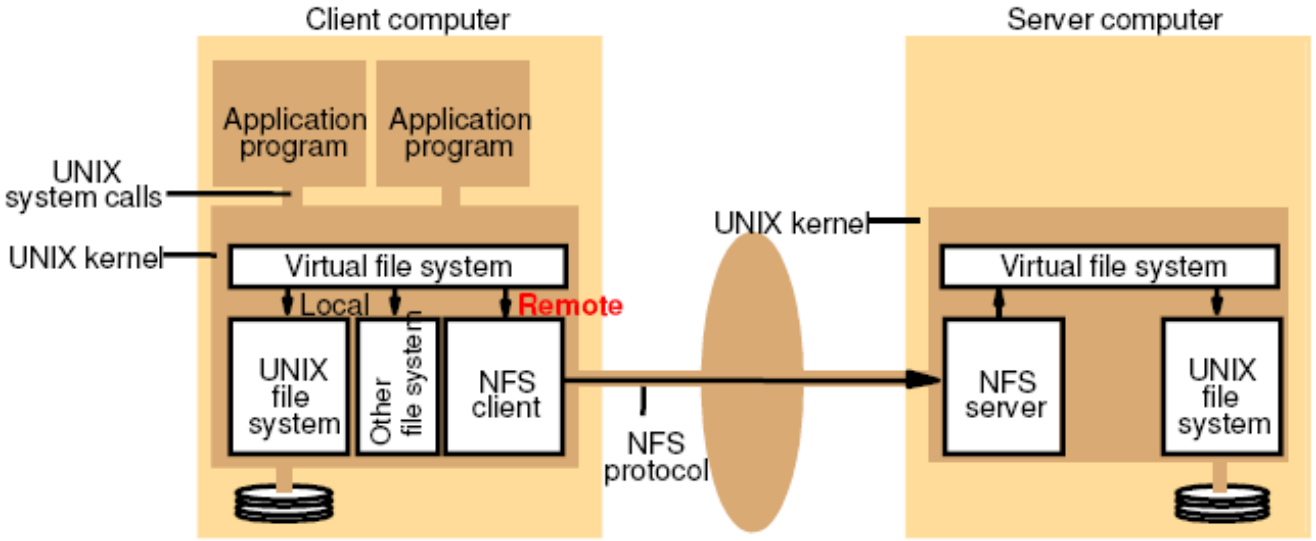
- **Dropbox**
  - Keeps your files synced across many computers.
  - Is a “transparent” DFS implementation.
  - Transparent to the OS (Windows, Mac, Linux)
  - Keeps track of consistency, backups etc.
  - ACL is not mature – personal experiences were near catastrophic.

DFS Requirements

- **Most attributes inherited from Distributed Systems**
- **Transparency**
  - Location
  - Access
  - Scaling
  - Naming
  - Replication
- **Concurrency**
  - Concurrent updates
  - Locking
- **Fault-Tolerance**
- **Scalability**
- **Heterogeneity**
- **Consistency**
- **Efficiency**
- **Location Independence**
- **Security**

Network File System (NFS)

- An industry standard by Sun Microsystems for file sharing on local networks since the 1980s.
- An open and popular standard with clear and simple interfaces.
- Supports many of the DFS design requirements (EX: transparency, heterogeneity, efficiency).
- Limited achievement of: concurrency, replication, consistency and security.



http://www.cs.uwaterloo.ca/~iaib/cs454/notes/5.FileSystems.pdf

More on NFS

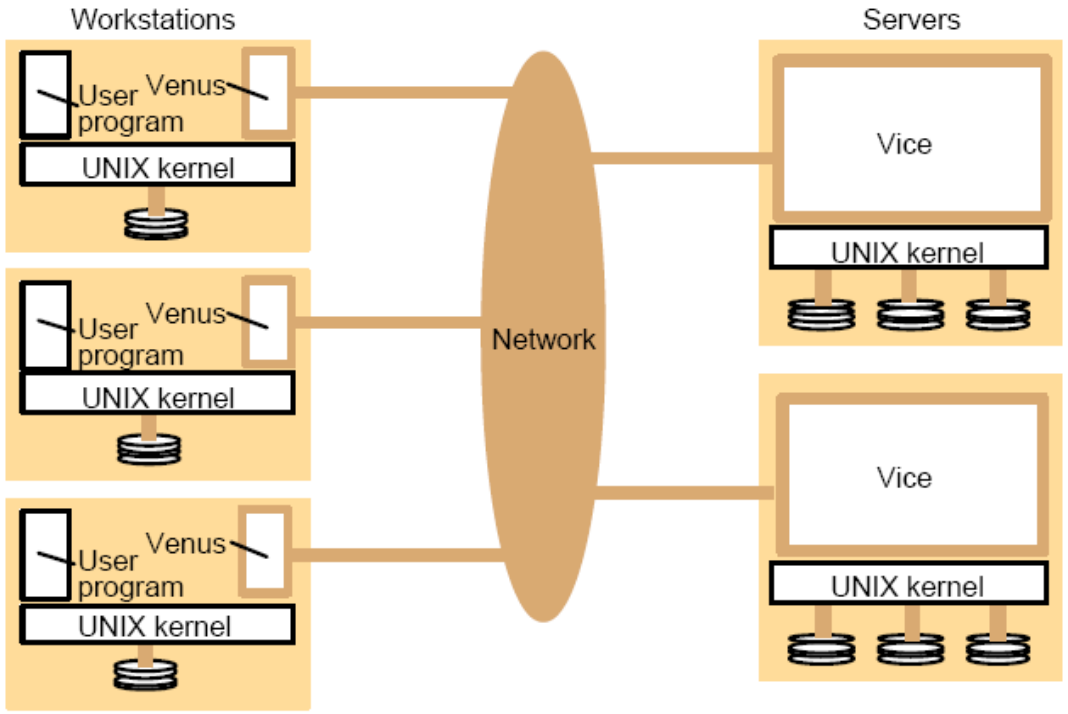
- **Supports directory and file access via remote procedure calls (RPCs)**
- **All UNIX system calls supported other than open & close**
- **Open and close are intentionally not supported**
  - For a read, client sends lookup message to server
  - Server looks up file and returns handle
  - Unlike open, lookup does not copy info in internal system tables
  - Subsequently, read contains file handle, offset and num bytes
  - Each message is self-contained
- **Pros: server is stateless, i.e. no state about open files**
- **Cons: Locking is difficult, no concurrency control**

NFS Tradeoffs

- **NFS Volume is Managed by a Single Server**
  - Higher Load on a Single Server
  - Simplified Coherency Protocols
- **Not Fault Tolerant**
  - Scalability is a real issue
  - Multiple Points of Failure in large (1000+) systems
    - App Bugs, OS
    - Hardware / Power Failure
    - Network Connectivity
- **Monitor, fault tolerance, auto-recovery essential in Cloud Computing.**

Andrew File System (AFS)

- Under development since 1983 at CMU.
- Andrew is highly scalable; the system is targeted to span over 5000 workstations.



http://www.cs.uwaterloo.ca/~iaib/cs454/notes/5.FileSystems.pdf

- NFS compatible.
- Distinguishes between client machines and dedicated server machines. Servers and clients are interconnected by an inter-net of LANs.

AFS Details

- **Based on the upload/download model**
  - Clients download and cache files
  - Server keeps track of clients that cache the file
  - Clients upload files at end of session
- **Whole file caching is central idea behind AFS**
  - Later amended to block operations
  - Simple, effective
- **AFS servers are stateful**
  - Keep track of clients that have cached files
  - Recall files that have been modified

Google File System (GFS)

- **Google has had issues with existing file systems on their huge distributed systems**
- **They created a new file system that matched well with MapReduce (also by Google)**
- **They wanted to have :**
  - The ability to detect, tolerate, recover, from failures automatically
  - Large Files, >= 100MB in size each
  - Large, streaming reads (each read being >= 1MB in size)
  - Large sequential writes that append
  - Concurrent appends by multiple clients
  - Atomicity for appends without synchronization overhead among clients

Architecture of GFS

- **Single master to coordinate access, keep metadata**
  - Simple centralized management
  - Fixed size chunks (64MB)
- **Many Chunk Servers (100 – 1000s)**
  - Files stored as chunks
  - Each chunk identified by 64-bit unique id
- **Reliability through replication**
  - Each chunk replicated across 3+ chunk servers
- **Many clients accessing same and different files stored on same cluster**
  - No data caching
  - Little benefit due to large data sets, streaming reads

GFS Architecture (Continued)

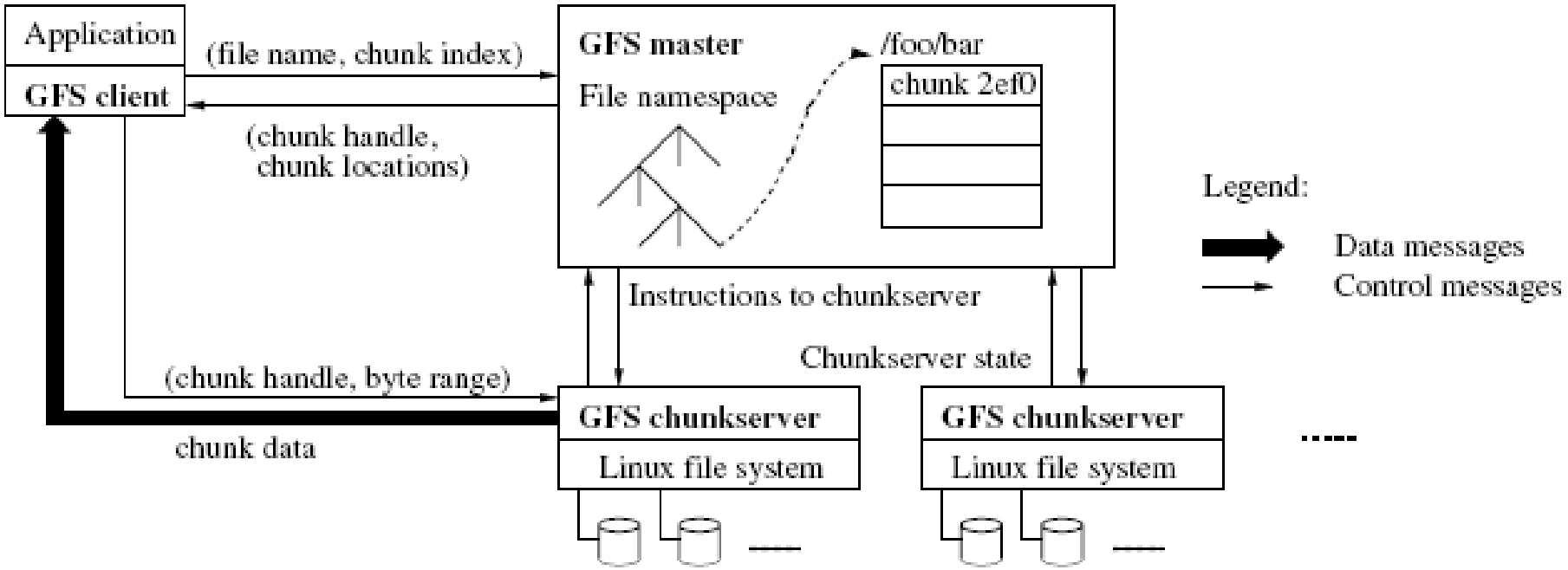


Figure from "The Google File System," Ghemawat et. al., SOSP 2003

Master and Chunk Server Responsibilities

- **Master Node**
  - Holds all metadata
    - Namespace
    - Current locations of chunks
    - All in RAM for fast access
  - Manages chunk leases to chunk servers
  - Garbage collects orphaned chunks
  - Migrates chunks between chunk servers
  - Polls chunk servers at startup
  - Use heartbeat messages to monitor servers
- **Chunk Servers**
  - Simple
  - Stores Chunks as files
  - Chunks are 64MB size
  - Chunks on local disk using standard filesystem
  - Read write requests specify chunk handle and byte range
  - Chunks replicated on configurable chunk servers

The Design Tradeoff

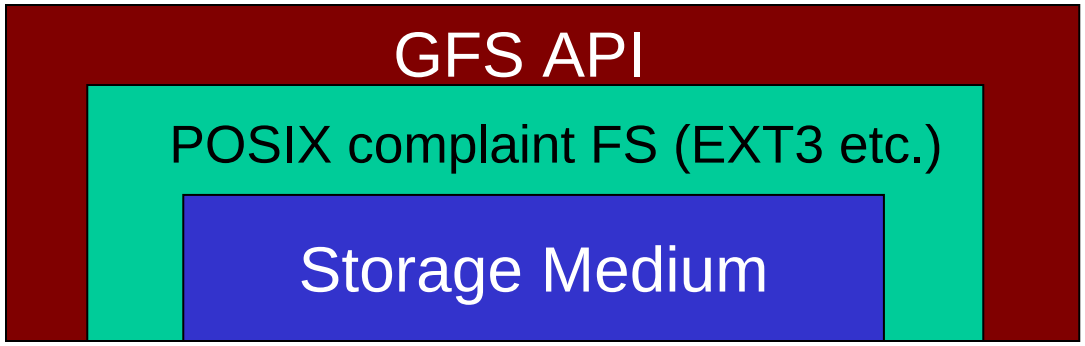
- **Can have small number of Large Files**
  - Less Metadata, the GFS Masternode can handle it
  - Fewer Chunk Requests to Masternode
  - Best for Streaming Reads
- **Large number of Small Files**
  - 1 chunk per file
  - Waste of Space
  - Pressure on Masternode to index all the files

How about Clients who need Data?

- **GFS clients**
  - Consult master for metadata
  - Access data from chunk servers
  - No caching at clients and chunk servers due to the frequent case of streaming
- **A client typically asks for multiple chunk locations in a single request**
- **The master also predicatively provide chunk locations immediately following those requested**

Differences in the GFS API

- **Not POSIX compliant**
  - Cannot mount an HDFS system in Unix directly, no Unix file semantics
  - An API over your existing File System (EXT3, RiserFS etc.)
- **API Operations**
  - Open, Close, Create and delete
  - Read and write
  - Record append
  - Snapshot (Quickly create a copy of the file)



Replication in GFS

- **The Data Chunks on the Chunk Servers are replicated (Typically 3 times)**
- **If a Chunk Server Fails**
  - Master notices missing heartbeats
  - Master decrements count of replicas for all chunks on dead chunk server
  - Master replicates chunks missing replicas in background
  - Highest priority of chunks missing greatest number of replicas

Consistency in GFS

- **Changes to Namespace are atomic**
  - Done by Single Master Server
  - Master uses logs to define global total order of namespace-changing operations
- **Data changes are more complicated**
  - Consistent: File regions all see as same, regardless of replicas they read from
  - Defined: after data mutation, file region is consistent and all clients see that entire mutation

Mutation in GFS

- **Mutation = write or append**
  - must be done for all replicas
- **Goal: minimize master involvement**
- **Lease mechanism:**
  - Master picks one replica as primary; gives it a “lease” for mutations
  - Primary defines a serial order of mutations
  - All replicas follow this order
- **Data flow decoupled from control flow**

Hadoop Distributed File System (HDFS)

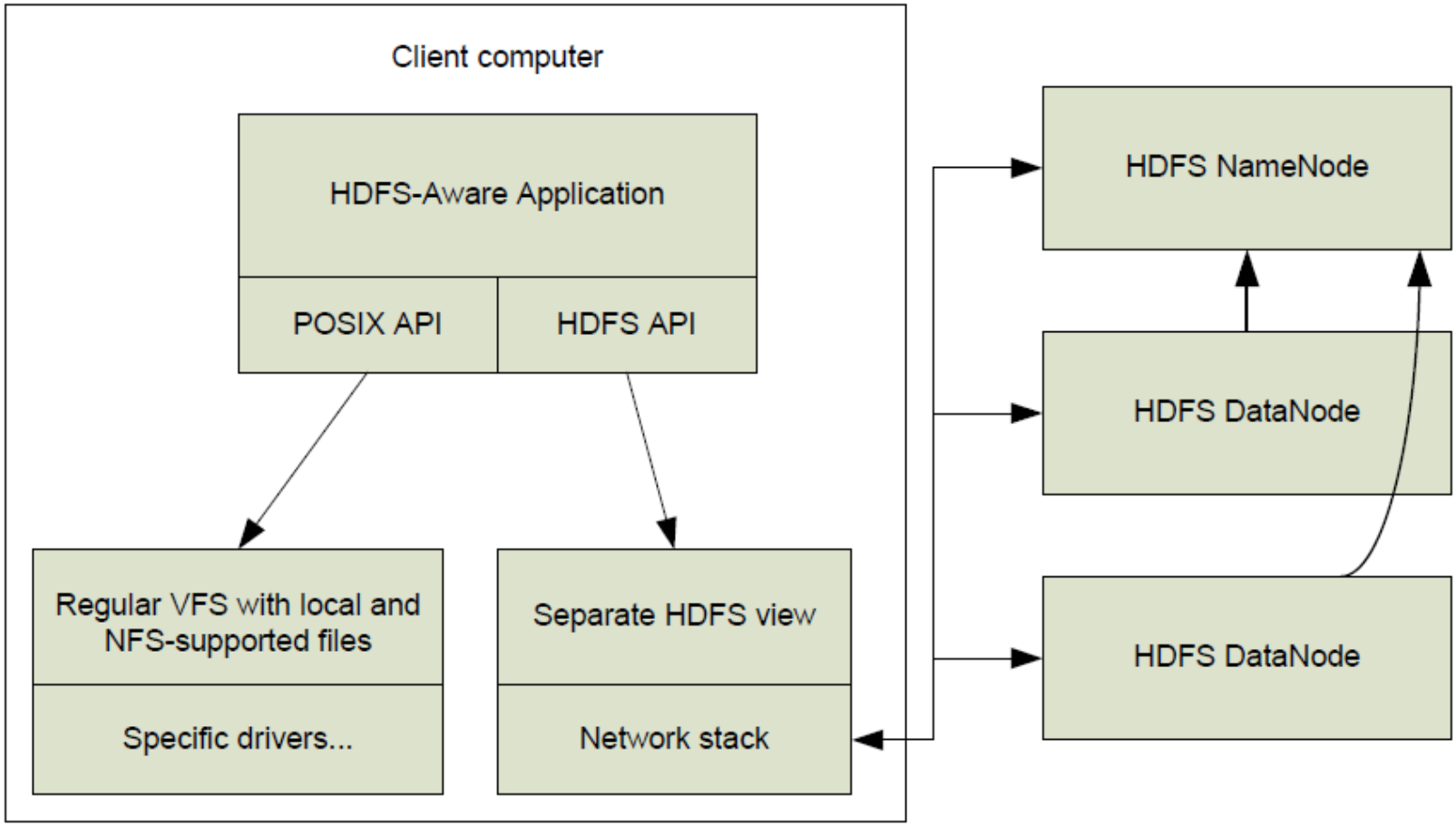
- **So GFS is super cool. I want it now!**
  - Not possible, It's Google's proprietary technology.
  - Good thing they published the technology though.
  - Now you can get the next best thing: **HDFS**

- **HDFS is open-source GFS under Apache**
  - Same design goals
  - Similar architecture, API and interfaces
  - Compliments Hadoop MapReduce

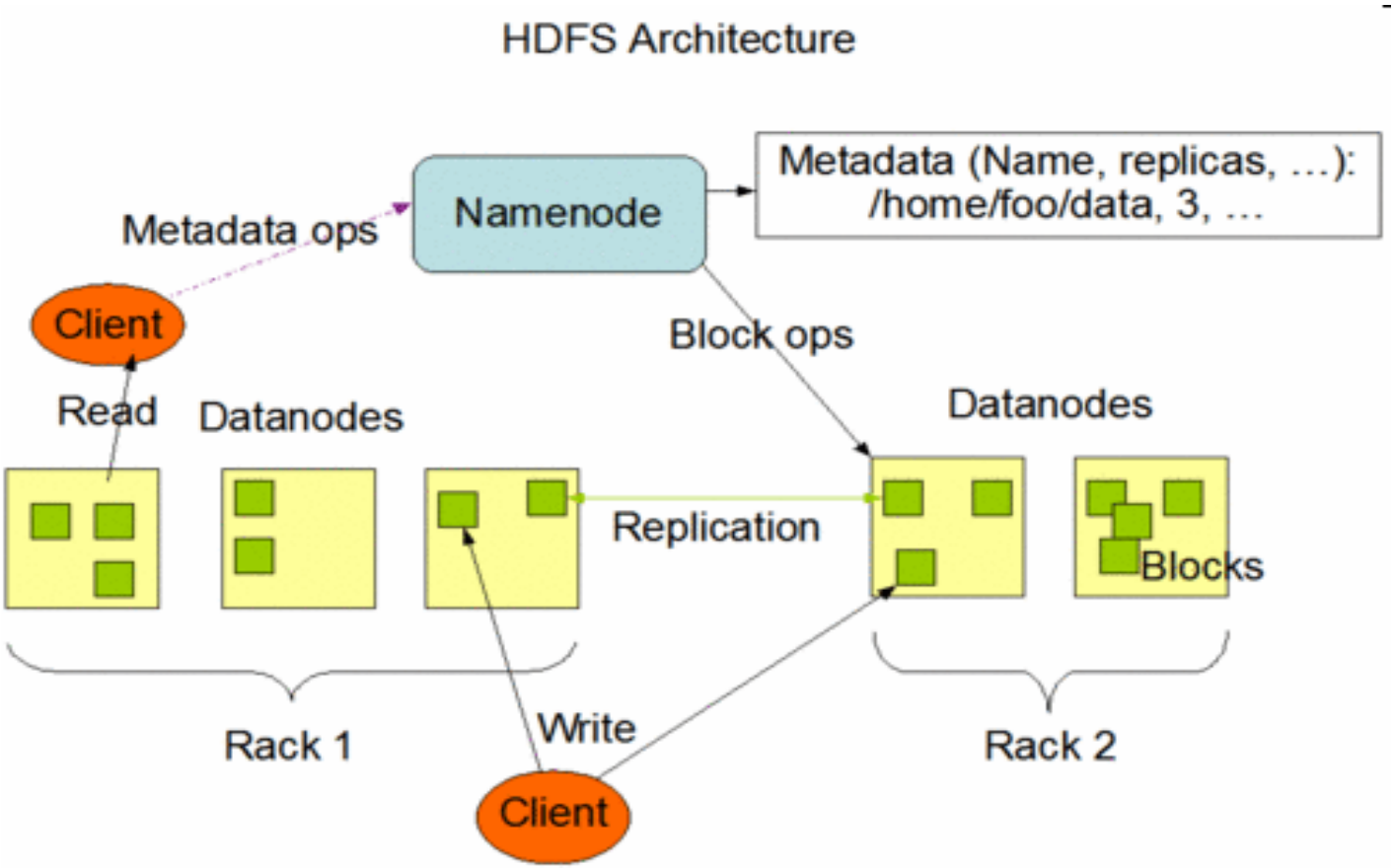
HDFS Design

- **Files stored as blocks**
  - Much larger size than most filesystems (default is 64MB)
- **Reliability through replication**
  - Each block replicated across 3+ DataNodes
- **Single master (NameNode) coordinates access, metadata**
  - Simple centralized management
- **No data caching**
  - Little benefit due to large data sets, streaming reads
- **Familiar interface, but customize the API**
  - Simplify the problem; focus on distributed apps

HDFS Architecture I



HDFS Architecture II

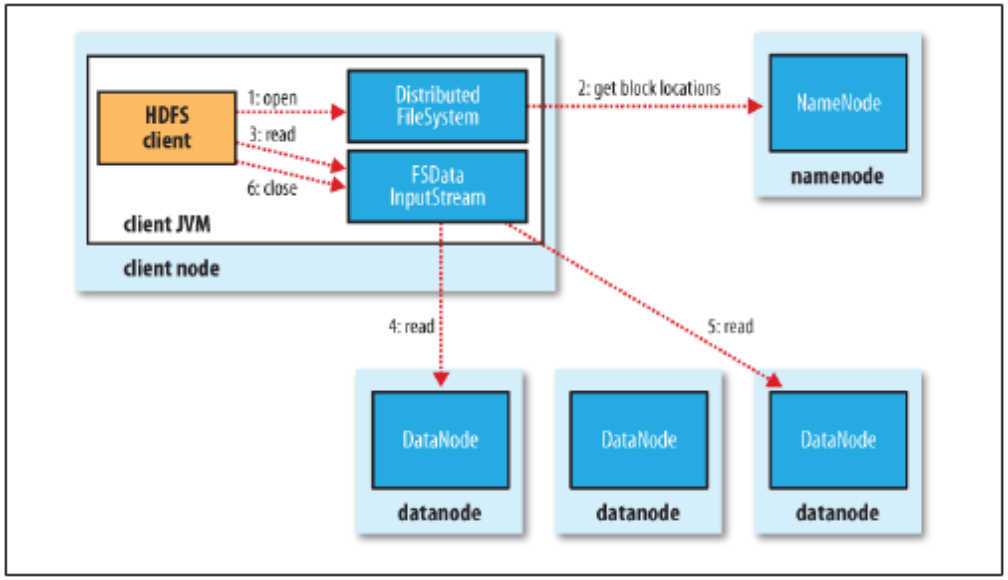


How to Talk to HDFS

- **Hadoop and HDFS are written in Java**
  - Primary Interface to HDFS is in Java
  - Other Interfaces do exist: C, FUSE,WebDAV, HTTP, FTP (buggy)
- **In Java, we use the `FileSystem` and `DistributedFileSystem` classes:**
  - Best practice is to use the URI class and `hdfs://`
  - Try out the example code in Chapter 3 of Tom White's Hadoop Book.

Anatomy of an HDFS File Read

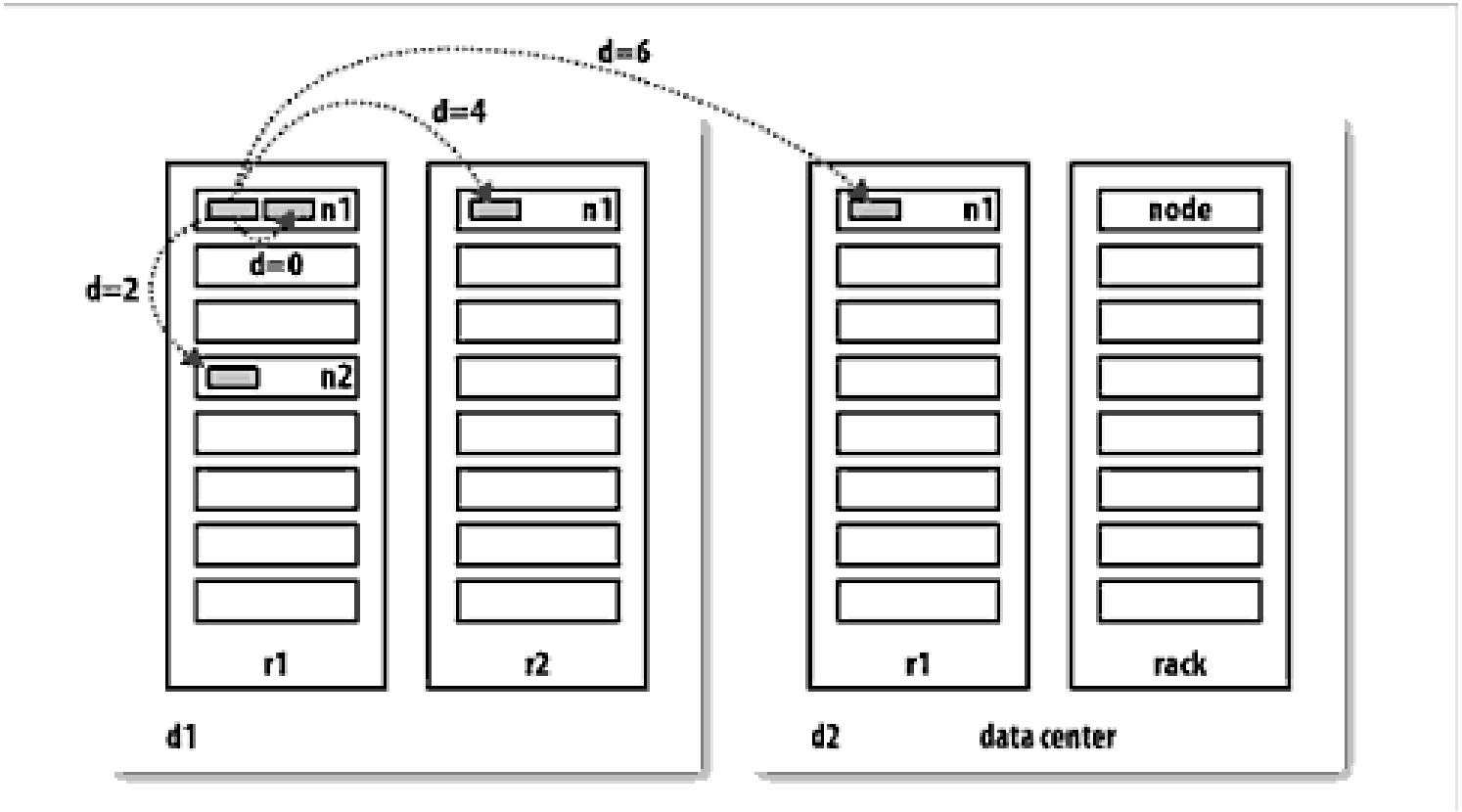
- A Client Program requests for data from the NameNode using the filename
- The NameNode returns the block locations on the DataNodes
- The Client then accesses each block Individually



Data Locality and "Rack Awareness"

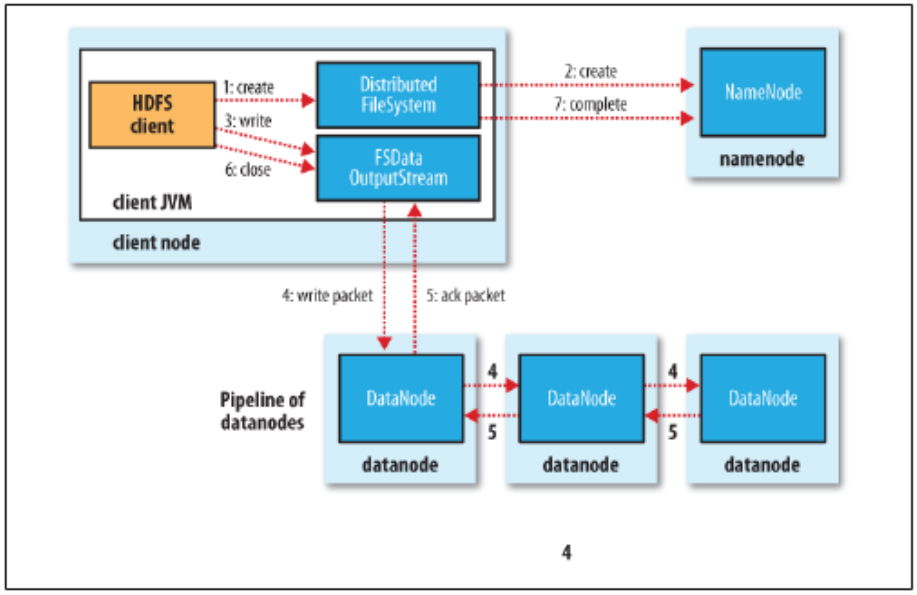
- **Bandwidth between two nodes in a Datacenter depends primarily on the "distance" between the two nodes**
- **The Distance can be calculated as follows:**
  - Suppose node *n1* is on rack *r1* in data center *d1 (d1/r1/n1)*
  - *distance(/d1/r1/n1 , /d1/r1/n1) = 0* (Procceses in the same node)
  - *distance(/d1/r1/n1 , /d1/r1/n2) = 2* (Nodes in the same rack)
  - *distance(/d1/r1/n1 , /d1/r2/n3) = 4* (Nodes across racks)
  - *distance(/d1/r1/n1 , /d2/r3/n4) = 6* (Nodes across Different DCs)
- **If configured properly with this information, Hadoop will optimize file reads and writes and sends jobs closest to the data.**

Rack Awareness in HDFS



Anatomy of an HDFS File Write

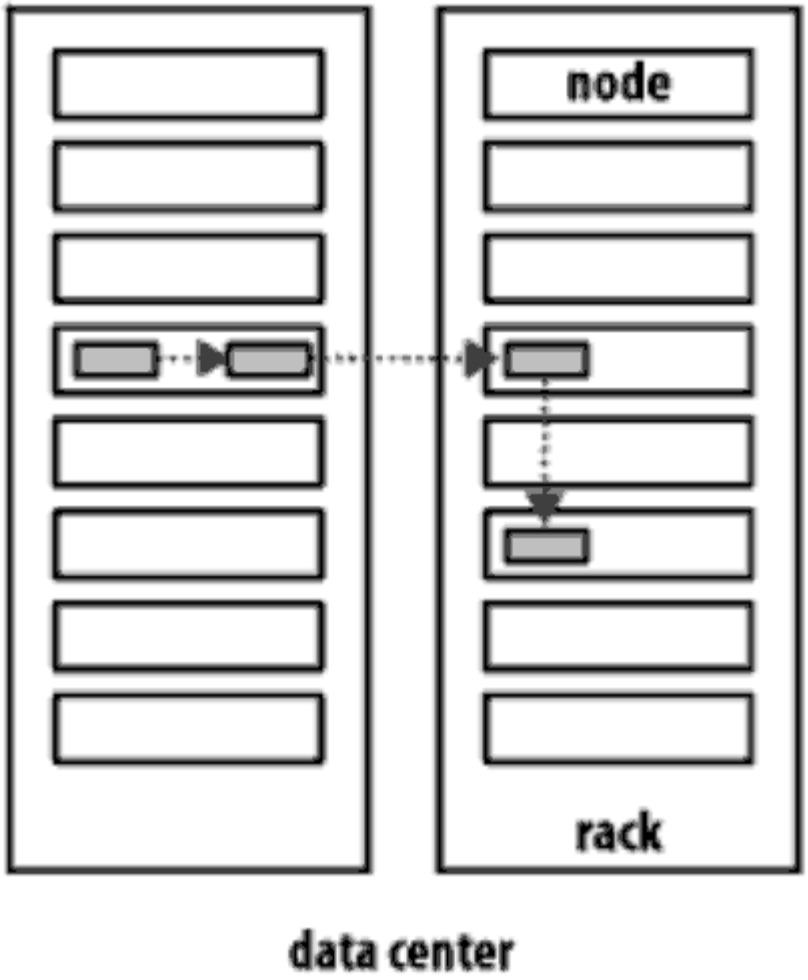
- The client issues a write request to the NameNode
- The client then writes individual blocks to the DataNodes and the DataNode pipeline the data for replication.
- After a block is written, the Data node sends an ACK
- After the File is written, the Client informs the NameNode



Replica Placement

- **Tradeoff between Reliability and Bandwidth**
  - Replicas on Same node will be faster but unreliable
  - Replicas across nodes will be reliable but slower
- **Hadoop's Replica Placement Strategy**
  - First replica is placed on the same node as client process (or chosen at random if client is outside the cluster)
  - Second replica is placed on different rack from the first (*off-rack*)
  - Third replica is placed on a different node on the same rack as the first.

Replica Placement



Coherency in HDFS

- After a File is Created, it will be visible in the FS namespace
- However writes to HDFS are not guaranteed to be visible, even after a `flush();`

```
Path p = new Path("p");
OutputStream out = fs.create(p);
out.write("content".getBytes("UTF-8"));
out.flush();
assertThat(fs.getFileStatus(p).getLen(), is(0L));
```

- File contents are visible only after the stream has been closed in HDFS. Also any file block that is currently being written in HDFS will not be visible
- MUST sync

Current Shortcomings of HDFS

- Write appends not stable and currently disabled (Hadoop 0.20)
- Coherency Model is an issue for application developers
- Handling of Corrupt Data (Sorry for the recent server downtime folks! ☹)
- No Authentication for HDFS users – You are who you say you are in HDFS
- Fault Tolerance is still Faulty

Applications

- **What does it have to do with cloud computing?**
  - Data is at the Heart of Cloud Computing Services
  - Need File Systems that can fit the bill for large, scalable hardware and software
  - GFS/HDFS and similar Distributed File Systems are now part and parcel of Cloud Computing Solutions.

References

- www.scs.ryerson.ca/~aabhari/DS-CH8.ppt
- www.cs.umd.edu/class/fall2002/.../Distributed_File_Systems.pdf
- web.cs.wpi.edu/.../Week%203%20--%20Distributed%20File%20Systems.ppt
- http://www.slideworld.com/slideshows.aspx/Chapter-17-DistributedFile-Systems-ppt-2113201
- www.cs.uga.edu/~laks/ADCS-Materials/DFS.ppt
- http://www.cs.uwaterloo.ca/~iaib/cs454/notes/5.FileSystems.pdf
- http://www.cs.chalmers.se/~tsigas/Courses/DCDSeminar/Files/afs_report.pdf
- http://www.nmc.teiher.gr/activities/MASTERS/JOINT/Material/Vall/DSC_2.pdf
- http://www.cs.umd.edu/~hcma/818g/
- https://vpn.qatar.cmu.edu/+CSCO+00756767633A2F2F636265676E792E6E707A2E626574++/citation.cfm?id=98169
- http://www.cs.rice.edu/~gw4314/lectures/dfs.ppt
- George Coulouris, JeanDollimore and Tim Kindberg. Distributed Systems:Concepts and Design(Edition3).Addison-Wesley2001 http://www.cdk3.net/
- AndrewS.Tanenbaum,Maartenvan Steen. Distributed Systems:Principles and Paradigms. Prentice-Hall2002. http://www.cs.vu.nl/~ast/books/ds1/
- P. K.Sinha, P.K. "Distributed Operating Systems, Concepts and Design", IEEE Press, 1993
- Couloris,Dollimore and Kindberg Distributed Systems: Concepts & Design Edn. 4 , Pearson Education 2005