Distributed File System

Case study with HDFS

1

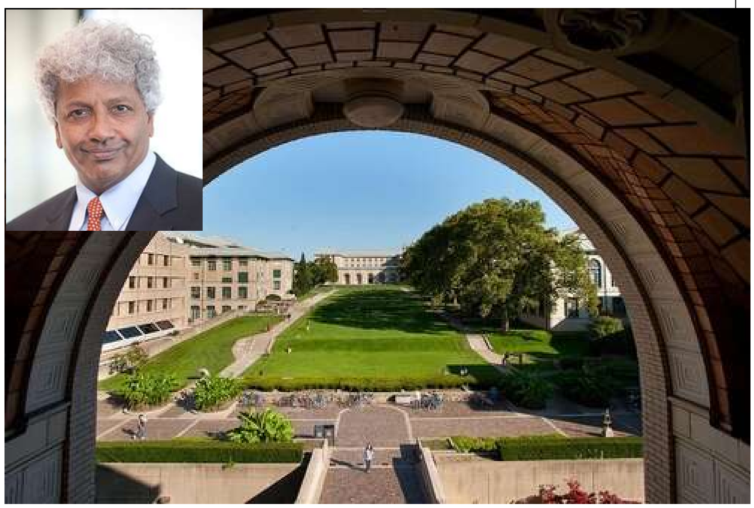
Outline

- Cross-Domain Communication
- System-wide DFS
- Client-Server Architecture
- Case Study: HDFS

16 July 2024

2

Problem of too many…



16 July 2024

3

Outline

- Cross-Domain Communication
- System-wide DFS
- Client-Server Architecture
- Case Study: HDFS

16 July 2024

4

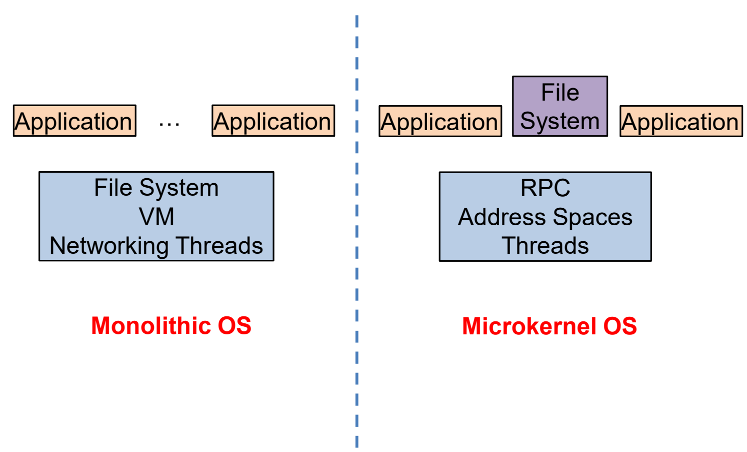
Cross-Domain Communication

- Options for cross-domain communication:
  - Shared memory with Semaphores, monitors, etc.,…
  - Remote Procedure Call
  - System-wide File System
- Examples of RPC based systems:
  - CORBA (Common Object Request Broker Architecture)
  - DCOM (Distributed COM)
  - RMI (Java Remote Method Invocation)

16 July 2024

5

OS Architecture

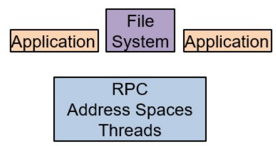


16 July 2024

6

Microkernel Architecture and DFS

- In Microkernel architecture, the File system looks remote.



- Distributed File System (DFS) aims to support transparent access to files on remote disks.
- A single, global name-space, if possible, will support every file with a unique name.
- Location Transparency: Services may migrate, and files can move without involving the user.

16 July 2024

7

Outline

- Cross-Domain Communication
- System-wide DFS
- Client-Server Architecture
- Case Study: HDFS

16 July 2024

8

Architecture Options

- Fully distributed with files in all sites:
  - Issues
    - Performance
    - Implementation complexity
- Client-server Model:
  - File server
    - Dedicated sites storing files perform storage and retrieval operations
  - Client
    - Rest of the sites access the files

16 July 2024

9

Mounting of Remote FS

- System manager mounts remote file system by giving name and local mount point
- Transparency to user - all reads and writes look like local reads and writes to user
  - e.g., /users/sue/foo→lmp/sue/foo on server
- Only previously mounted remote file systems can be accessed transparently

16 July 2024

10

Mounting of Remote FS

- The mount mechanism binds together several filename spaces into a single hierarchically structured name space
- A name space ‘A’ can be mounted (bounded) at an internal node (mount point) of a name space ‘B’
- Kernel requires to maintain the *mount table,* a mapping between mount points to storage devices

16 July 2024

11

Mount Information at Client Side

- A client mounts other file systems
- Different clients may not see the same filename space
- If a file moves to another server, every client needs to update its mount table

16 July 2024

12

Mount Information at Server Side

- Server mounts the file systems of remote nodes
- Every client sees the same filename space
- If a file moves to another server, mounting information at the server only needs to change

16 July 2024

13

Andrew File System (AFS)

- Andrew file system (AFS) is a location-independent file system that uses a local cache to reduce the workload and increase the performance of a distributed computing environment.
  - Introduced by researchers at Carnegie-Mellon University (CMU) in 1983. Team led by Prof. M. Satyanarayanan of CMU.

16 July 2024

14

Why AFS?

- Initial goal of AFS was wide accessibility of computational facilities
- An integrated, campus-wide file system with functional characteristics as close to that of UNIX as possible.
- To enable a student to sit down at any workstation and start using his or her files with as little hassle as possible.

16 July 2024

15

Why AFS?

- They did not want to modify existing application programs, which assume a UNIX file system, in any way.
- Thus, the first design choice was to make the file system compatible with UNIX at the system call level.

16 July 2024

16

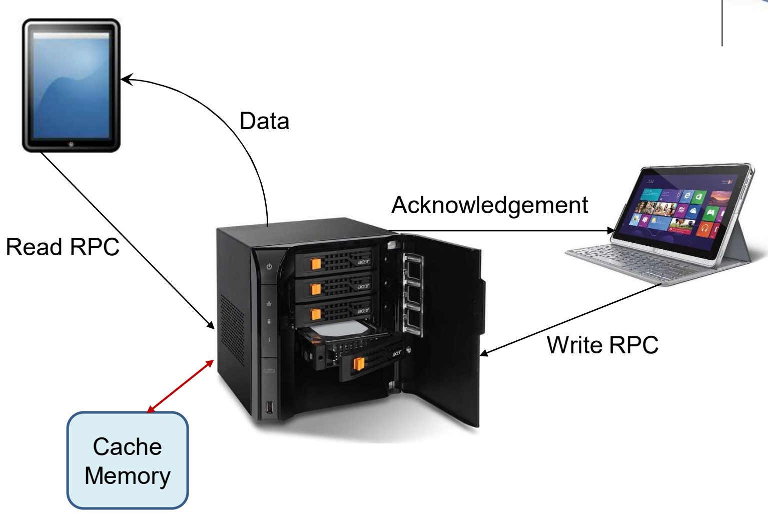
Outline

- Cross-Domain Communication
- System-wide DFS
- Client-Server Architecture
- Case Study: HDFS

16 July 2024

17

Client Server Model using RPC



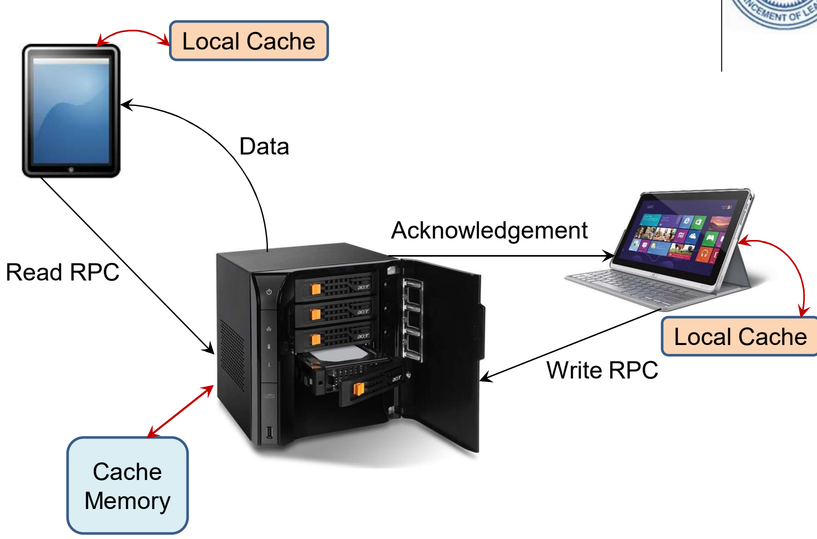
16 July 2024

18

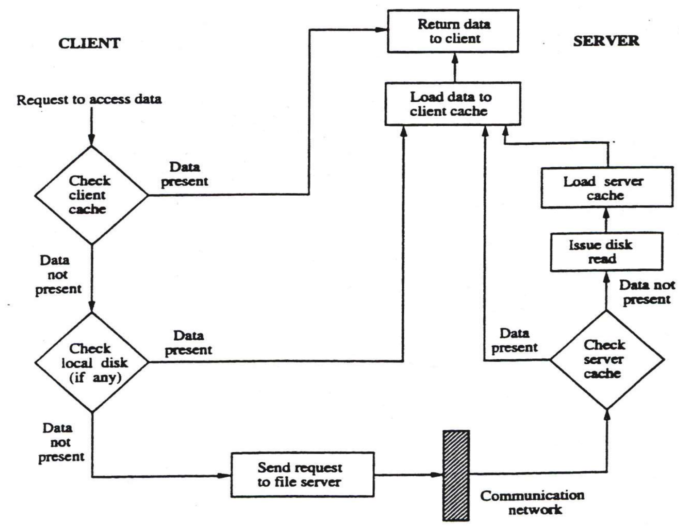
Server-Side Cache

- Reads and writes to server by RPC Calls
- Caching at server-side only
- Advantage:
  - Server provides completely consistent view of file system to multiple clients
- Concerns:
  - High latency
  - Server can be a bottleneck

Using Cache in Both Sides



Accessing DFS



16 July 2024

Source: http://www.slideshare.net/longly/11-distributed-filesystems

21

Does it Solve the Problems?

- Use caching at source and destination
- Advantage:
  - Operations done locally, don't add to network traffic
  - Low latency
- Concerns:
  - Client and Server-side caches may not be mutually consistent
  - Will it be better for a Stateless server?
  - What if multiple clients access a server, some writing and some reading data?

16 July 2024

22

Outline

- Cross-Domain Communication
- System-wide DFS
- Client-Server Architecture
- Case Study: HDFS

What is HDFS?

- Hadoop Distributed File System (HDFS) is a distributed file system designed to run on a huge system built with low-cost hardware components.
  - HDFS was originally built for the Apache Nutch web search engine project.
  - HDFS later transformed into an Apache Hadoop subproject.

Motivations…

- Using a huge number of low-cost computers for storage implies that some of these will always be non-functional.
- Therefore, detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

16 July 2024

25

What’s different for HDFS?

- HDFS is highly fault-tolerant and is designed for low-cost hardware.
- HDFS provides high throughput access to application data and is suitable for applications that have large data sets.
- HDFS relaxes a few POSIX (Portable Operating System Interface for UNIX) requirements for faster access to file system data.

16 July 2024

26

What's different for HDFS?

- Detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.
- HDFS is tuned to support large files.
- HDFS should provide high aggregate data bandwidth and scale to hundreds of nodes in a single cluster.

16 July 2024

27

What's different for HDFS?

- HDFS applications assume and require a write-once-read-many access model for files.
- HDFS provides interfaces for migrating applications closer to data.
- HDFS has been designed to be easily portable from one platform to another.

16 July 2024

28

NameNode and DataNode

- HDFS has a master/slave architecture.
- An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients.
- Number of DataNodes, often one per node in the cluster, manage storage attached to the nodes that they run on.

16 July 2024

29

Role of NameNode

- The NameNode executes file system namespace operations like opening, closing, renaming files and directories.
- It also determines the mapping of blocks to DataNodes.
- NameNode is the arbitrator and repository for all HDFS metadata.

16 July 2024

30

Role of DataNode

- The DataNodes are responsible for serving read and write requests from the file system's clients.
- DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.
- DataNodes have no knowledge about HDFS files.

16 July 2024

31

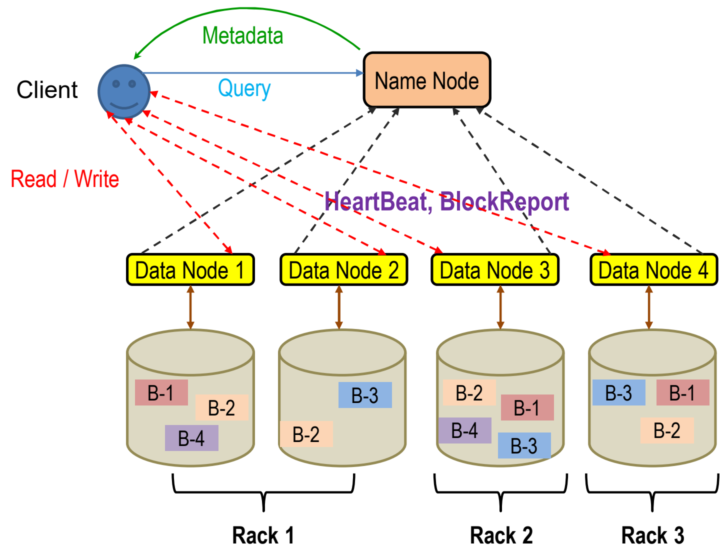
Deploying NameNode & DataNode

- In a typical deployment, there will be one dedicated machine to host only the NameNode software.
- Every other machine in the cluster runs one instance of DataNode software.
- Existence of a single NameNode in a cluster greatly simplifies architecture of the system.

16 July 2024

32

HDFS Architecture



33

Portability

- NameNode and DataNode software run on low-cost computers that may typically run a GNU/Linux OS.
- HDFS is built using Java
  - any machine that supports Java can run the NameNode or DataNode software.
  - Built on Java platform, HDFS can be deployed on a wide range of machines.

16 July 2024

34

NameNode and FS Name Space

- HDFS splits large files into smaller blocks that are stored in DataNodes.
- It is the responsibility of the NameNode to know what blocks on which DataNodes make up the complete file.
- The complete collection of all the files in the cluster is referred as the file system namespace.

16 July 2024

35

NameNode and FS Name Space

- In HDFS, the NameNode maintains the file system namespace.
- Any change to the file system namespace or its properties is recorded by the NameNode.
- HDFS supports a traditional hierarchical file organization.

16 July 2024

36

Replication in HDFS

- In HDFS, blocks of a file are replicated to improve fault tolerance.
- Files in HDFS are write-once and have strictly one writer at any time.

Replication Factor

- In HDFS, an application can specify the number of replicas of a file at file creation time and can be changed later
- The number of replicas to be maintained is called the replication factor of that file – by default this is 3.
- This information is also stored in the NameNode.

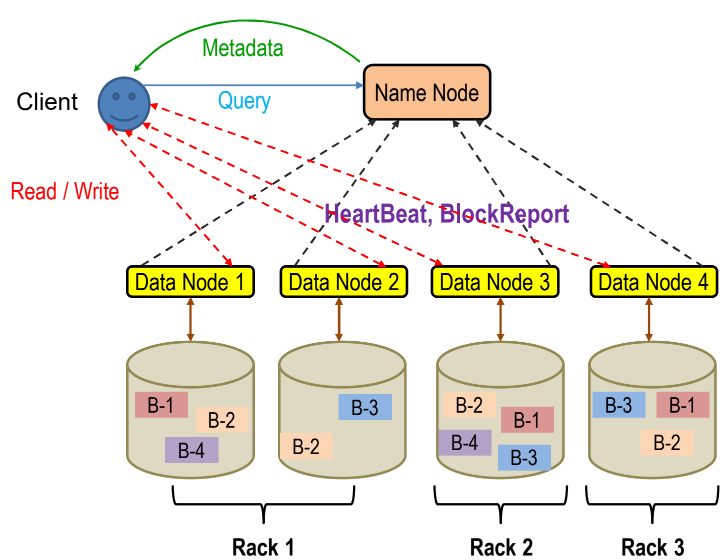
Heartbeat and Blockreport

- NameNode decides on replication of blocks
- It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster.
- Receipt of a Heartbeat implies that the DataNode is functioning properly.
- Blockreport contains a list of all blocks on a DataNode.

16 July 2024

39

HDFS Architecture



40

Replication Placement

- This makes HDFS different from others
- The purpose is to strike a balance between data reliability, availability, and network bandwidth utilization.
- HDFS often runs on a cluster of computers spread across many racks.
- Communication between nodes in different racks goes through switches.

Replication Placement: Policy 1

- A non-optimal yet simple policy is to place replicas on unique racks.
- This prevents losing data when an entire rack fails.
- This evenly distributes replicas in the cluster and eases load balancing on component failure.
- However, cost of writes is high as a write needs to transfer blocks to multiple racks.

Replication Placement: Policy 2

- With replication factor 3, HDFS's placement policy is to
  - put one replica on one node in the local rack,
  - another on a node in a different (remote) rack,
  - and the last on a different node in the same remote rack.
- This policy cuts the inter-rack write traffic which generally improves write performance.

16 July 2024

43

Replication Placement: Policy 2

- This policy does not impact data reliability and availability guarantees as the chance of rack failure is far less than that of node failure.
- However, it reduces the aggregate network bandwidth used when reading data since a block is placed in only two unique racks rather than 3.

16 July 2024

44

Replication Placement: Policy 2

- With this policy, the replicas of a file do not evenly distribute across the racks.
  - 1/3rd of replicas are on one node,
  - 2/3rd of replicas are on one rack, and
  - the other 1/3rd are evenly distributed across the remaining racks.
- This policy improves write performance without compromising data reliability or read performance.

16 July 2024

45

Replication Pipelining

- Suppose the replication factor is 3.
- After the client gets a list of DataNodes from the NameNode, local data is flushed to the first DataNode.
- The first DataNode
  - starts receiving the data in small sizes (4 KB)
  - writes each portion to its local repository and
  - transfers that portion to the second DataNode in the list.

16 July 2024

46

Replication Pipelining

- The second DataNode, in turn,
  - starts receiving each portion of the data block,
  - writes that portion to its repository and then
  - flushes that portion to the third DataNode.
- Finally, the third DataNode writes the data to its local repository.
- Thus, a DataNodes replicate in the pipeline.

16 July 2024

47

Safe Mode

- Initially, the NameNode enters a special state called Safe mode.
- Data blocks are not yet replicated when the NameNode is in Safe mode.
- NameNode gets Heartbeat and Blockreport messages from the DataNodes
- A block is considered safely replicated when minimum number of replicas for that data block is recorded with the NameNode

16 July 2024

48

Safe Mode

- After a configurable percentage of safely replicated data blocks checks in with the NameNode, the NameNode exits the Safe mode state.
- It then determines the list of data blocks (if any) that still have fewer than the specified number of replicas.
- The NameNode then replicates these blocks to other DataNodes.

EditLog and FsImage

- In HDFS, the NameNode uses a transaction log called the EditLog to record every change that occurs to FS metadata.
  - e.g., creating a new file in HDFS causes the NameNode to insert a record into the EditLog indicating this.
  - changing the replication factor of a file causes a new record to be inserted into the EditLog.
- The NameNode uses a file in its local host OS file system to store the EditLog.

EditLog and FsImage

- The entire file system namespace, including the mapping of blocks to files and file system properties, is stored in a file called the FsImage.
- The FsImage is stored as a file in the NameNode's local file system too.
- The NameNode keeps an image of the entire file system namespace and file Blockmap in memory.

16 July 2024

51

Checkpoint

- When the NameNode starts up:
  - it reads the FsImage and EditLog from disk
  - applies all the transactions from the EditLog to the in-memory representation of the FsImage
  - flushes out this new version into a new FsImage on disk, and
  - truncates the old EditLog.
- This process is called a checkpoint.

16 July 2024

52

DataNode and BlockReport

- A DataNode stores HDFS data in files in the local file system.
- When a DataNode starts up,
  - it scans through its local file system,
  - generates a list of all HDFS data blocks that correspond to each of these local files and
  - sends this report to the NameNode.
- This is the Blockreport.

Data Organization

- HDFS is designed to support very large files where data is written only once but read many times at streaming speeds.
- The typical block size used by HDFS is 64 MB.
- An HDFS file is chopped up into 64 MB chunks, and if possible, each chunk will reside on a different DataNode.

Staging

- In HDFS, a client request to create a file does not reach NameNode immediately.
  - Initially, the HDFS client caches the file data into a temporary local file.
  - Application writes are transparently redirected to this temporary local file.
  - When the local file accumulates data more than one HDFS block size, the client contacts the NameNode.

16 July 2024

55

Staging

- The NameNode now inserts the file name into the file system hierarchy and allocates a data block for it.
- The NameNode responds to the client request with the identity of the DataNode and the destination data block.
- Then the client flushes the block of data from the local temporary file to the specified DataNode.

16 July 2024

56

Staging

- When a file is closed,
  - the remaining un-flushed data in temporary local file is transferred to the DataNode.
  - Client tells the NameNode that file is closed
- At this point only, the NameNode commits the file creation operation into a persistent store.
- If the NameNode dies before the file is closed, the file is lost.

Space Reclamation

- In HDFS, a file is not immediately removed when it's deleted by the user.
- HDFS first renames it to a file in the /trash directory.
- User can Undelete a file as long as it remains in the /trash directory.
- A file remains in /trash for a configurable amount of time.

Reducing Replication Factor

- In HDFS, when the replication factor of a file is reduced, the NameNode selects excess replicas that can be deleted.
- The next Heartbeat transfers this information to the DataNode.
- Subsequently, the DataNode removes the corresponding blocks and more free space appears in the cluster.

Thanks for your kind attention

Questions??