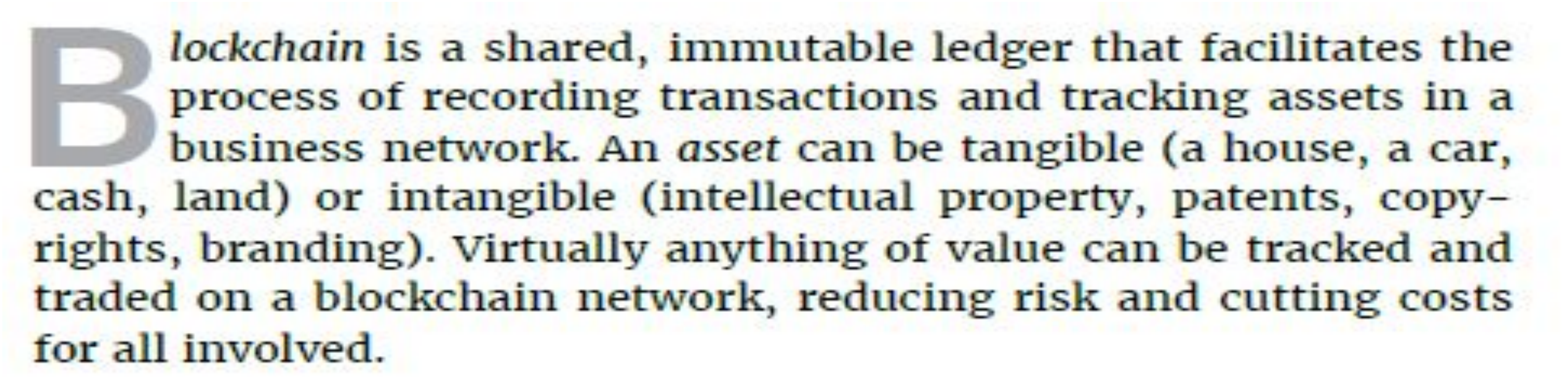
BLOCKCHAIN TECHNOLOGY

Dr.P.Vishvapathi
Professor, CSE Department

I UNIT

1. Two General Problem
2. Byzantine General problem and Fault Tolerance
3. Hadoop Distributed File System

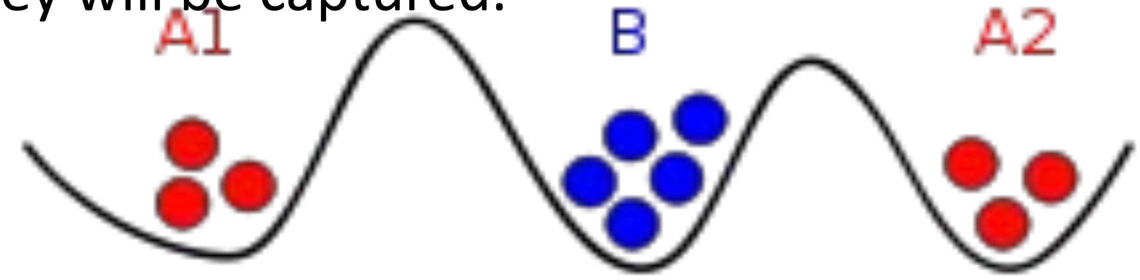


Blockchain is a shared, immutable ledger that facilitates the process of recording transactions and tracking assets in a business network. An *asset* can be tangible (a house, a car, cash, land) or intangible (intellectual property, patents, copy-rights, branding). Virtually anything of value can be tracked and traded on a blockchain network, reducing risk and cutting costs for all involved.

BC IS A DISTRIBUTED LEDGER IN A P2P NETWORK. A GENERAL LEDGER PROVIDES A RECORD OF EACH FINANCIAL TRANSACTION THAT TAKES PLACE DURING THE LIFE OF AN OPERATING COMPANY

1. Two General Problem

- Two armiesTwo armies, each led by a different generalTwo armies, each led by a different general, are preparing to attack a fortified city. The armies are encamped near the city, each in its own valley. A third valley separates the two hills, and the only way for the two generals to communicate is by sending messengers through the valley.
- Unfortunately, the valley is occupied by the city's defenders and there's a chance that any given messenger sent through the valley will be captured.



- While the two generals have agreed that they will attack, they haven't agreed upon a time for attack. It is required that the two generals have their armies attack the city at the same time in order to succeed, else the lone attacker army will die trying.
- They must thus communicate with each other to decide on a time to attack and to agree to attack at that time, and each general must know that the other general knows that they have agreed to the attack plan.
- Because acknowledgement of message receiptBecause acknowledgement of message receipt can be lost as easily as the original message, a potentially infinite series of messages is required to come to consensus.

- The thought experiment involves considering how they might go about coming to consensus. In its simplest form one general is known to be the leader, decides on the time of attack, and must communicate this time to the other general.
- The problem is to come up with algorithms that the generals can use, including sending messages and processing received messages, that can allow them to correctly conclude:
- Yes, we will both attack at the agreed-upon time.
- Allowing that it is quite simple for the generals to come to an agreement on the time to attack (i.e. one successful message with a successful acknowledgement), the argument of the Two Generals' Problem is in the impossibility of designing algorithms for the generals to use to safely agree to the above statement.

- The Two Generals' Problem was the first computer communication problem to be proved to be unsolvable.
- An important consequence of this proof is that generalizations like the Byzantine Generals problem are also unsolvable in the face of arbitrary communication failures.

2. Byzantine General problem and Fault Tolerance

- Byzantine=(of a system or situation) excessively complicated, and typically involving a great deal of administrative detail.
- relating to Byzantium (now Istanbul), the Byzantine Empire, or the Eastern Orthodox Church.

- **What is Byzantine Fault Tolerance?** Byzantine Fault Tolerance(BFT) is the feature of a distributed network to reach consensus(agreement on the same value) even when some of the nodes in the network fail to respond or respond with incorrect information.
- The objective of a BFT mechanism is to safeguard against the system failures by employing collective decision making(both – correct and faulty nodes) which aims to reduce to influence of the faulty nodes. BFT is derived from Byzantine Generals' Problem.

- **Byzantine Generals’ Problem**
  Imagine that several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general.
- The generals can communicate with one another only by messenger. After observing the enemy, they must decide upon a common plan of action.
- However, some of the generals may be traitors, trying to prevent the loyal generals from reaching an agreement.
- The generals must decide on when to attack the city, but they need a strong majority of their army to attack at the same time.
- The generals must have an algorithm to guarantee that (a) all loyal generals decide upon the same plan of action, and (b) a small number of traitors cannot cause the loyal generals to adopt a bad plan. The loyal generals will all do what the algorithm says they should, but the traitors may do anything they wish.
- The algorithm must guarantee condition (a) regardless of what the traitors do. The loyal generals should not only reach agreement, but should agree upon a reasonable plan.

- Byzantine fault tolerance can be achieved if the correctly working nodes in the network reach an agreement on their values. There can be a default vote value given to missing messages i.e., we can assume that the message from a particular node is ‘faulty’ if the message is not received within a certain time limit. Furthermore, we can also assign a default response if the majority of nodes respond with a correct value.
- **Types of Byzantine Failures:**There are two categories of failures that are considered. One is fail-stop(in which the node fails and stops operating) and other is arbitrary-node failure. Some of the arbitrary node failures are given below :
- Failure to return a result ,Respond with an incorrect result
- Respond with a deliberately misleading result,Respond with a different result to different parts of the system

- A **Byzantine fault** (also **interactive consistency**, **source congruency**, **error avalanche**, **Byzantine agreement problem**, **Byzantine generals problem**, and **Byzantine failure**) is a condition of a computer system, particularly distributed computing systems, where components may fail and there is imperfect information on whether a component has failed.

- The term takes its name from an allegory, the "Byzantine Generals Problem", developed to describe a situation in which, in order to avoid catastrophic failure of the system, the system's actors must agree on a concerted strategy, but some of these actors are unreliable.

- In a Byzantine fault, a component such as a server can inconsistently appear both failed and functioning to failure-detection systems, presenting different symptoms to different observers.
- It is difficult for the other components to declare it failed and shut it out of the network, because they need to first reach a consensus regarding which component has failed in the first place.
- Byzantine fault tolerance (BFT) is the dependability of a fault-tolerant computer system to such conditions.

- Formal definition
- **Setting:** Given a system of n components, t of which are dishonest, and assuming only point-to-point channel between all the components.
- Whenever a component A tries to broadcast a value x, the other components are allowed to discuss with each other and verify the consistency of A 's broadcast, and eventually settle on a common value y .
- **Property:**
- The system is said to resist Byzantine Faults if a component A broadcast a value x:
- If A is honest, then all honest components agree on the value x.
- In any case, all honest components agree on the same value y.

- One example of BFT in use is bitcoinOne example of BFT in use is bitcoin, a peer-to-peer digital cash system. The bitcoin networkOne example of BFT in use is bitcoin, a peer-to-peer digital cash system. The bitcoin network works in parallel to generate a blockchainOne example of BFT in use is bitcoin, a peer-to-peer digital cash system. The bitcoin network works in parallel to generate
  a blockchain with proof-of-work allowing the system to overcome Byzantine failures and reach a coherent global view of the system's

3. Hadoop Distributed File System

- The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware.
- Apache HadoopApache Hadoop is an open-source framework that is suited for processing large data sets on commodity hardware.
- Hadoop is an implementation of MapReduce, an application programming model developed by Google, which provides two fundamental operations for data processing: *map* and *reduce*.

- The former transforms and synthesizes the input data provided by the user; the latter aggregates the output obtained by the map operations.
- Hadoop provides the runtime environment, and developers need only provide the input data and specify the map and reduce functions that need to be executed.
- Yahoo!, the sponsor of the Apache Hadoop project, has put considerable effort into transforming the project into an enterprise-ready cloud computing platform for data processing

- Hadoop is an integral part of the Yahoo! cloud infrastructure and supports several business processes of the company.
- Currently, Yahoo! manages the largest Hadoop cluster in the world, which is also available to academic institutions.
- Commodity hardware, in an IT context, is a device or device component that is relatively inexpensive, widely available and more or less interchangeable with other hardware of its type.

- To be interchangeable, commodity hardware is usually broadly compatible and can function on a plug and play basis with other commodity hardware products.
- In this context, a commodity item is a low-end but functional product without distinctive features. A commodity computer, for example is a PC that has no outstanding features and is widely available for purchase.

- RAID (redundant array of independent -- originally inexpensive -- disks) performance typically relies upon an array of commodity hard disks to enable improvements in mean time between failures (MTBF), fault tolerance and failover.

Assumptions and Goals

- **1.Hardware Failure:** An HDFS instance may consist of hundreds or thousands of server machines, each storing part of the file system’s data.
- detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

- **2.Streaming Data Access:** Applications that run on HDFS need streaming access to their data sets.
- They are not general purpose applications that typically run on general purpose file systems.
- HDFS is designed more for batch processing rather than interactive use by users.
- The emphasis is on high throughput of data access rather than low latency of data access.

- **3.Large Data Sets**
- Applications that run on HDFS have large data sets. A typical file in HDFS is gigabytes to terabytes in size.
- Thus, HDFS is tuned to support large files. It should provide high aggregate data bandwidth and scale to hundreds of nodes in a single cluster.
- It should support tens of millions of files in a single instance.

- **4.Simple Coherency Model**
- HDFS applications need a write-once-read-many access model for files.
- A file once created, written, and closed need not be changed.
- This assumption simplifies data coherency issues and enables high throughput data access.

- **5. “Moving Computation is Cheaper than Moving Data”**
- A computation requested by an application is much more efficient if it is executed near the data it operates on.
- This is especially true when the size of the data set is huge. This minimizes network congestion and increases the overall throughput of the system.
- The assumption is that it is often better to migrate the computation closer to where the data is located rather than moving the data to where the application is running.
- HDFS provides interfaces for applications to move themselves closer to where the data is located.

- **6.Portability Across Heterogeneous Hardware and Software Platforms :**HDFS has been designed to be easily portable from one platform to another. This facilitates widespread adoption of HDFS as a platform of choice for a large set of applications.

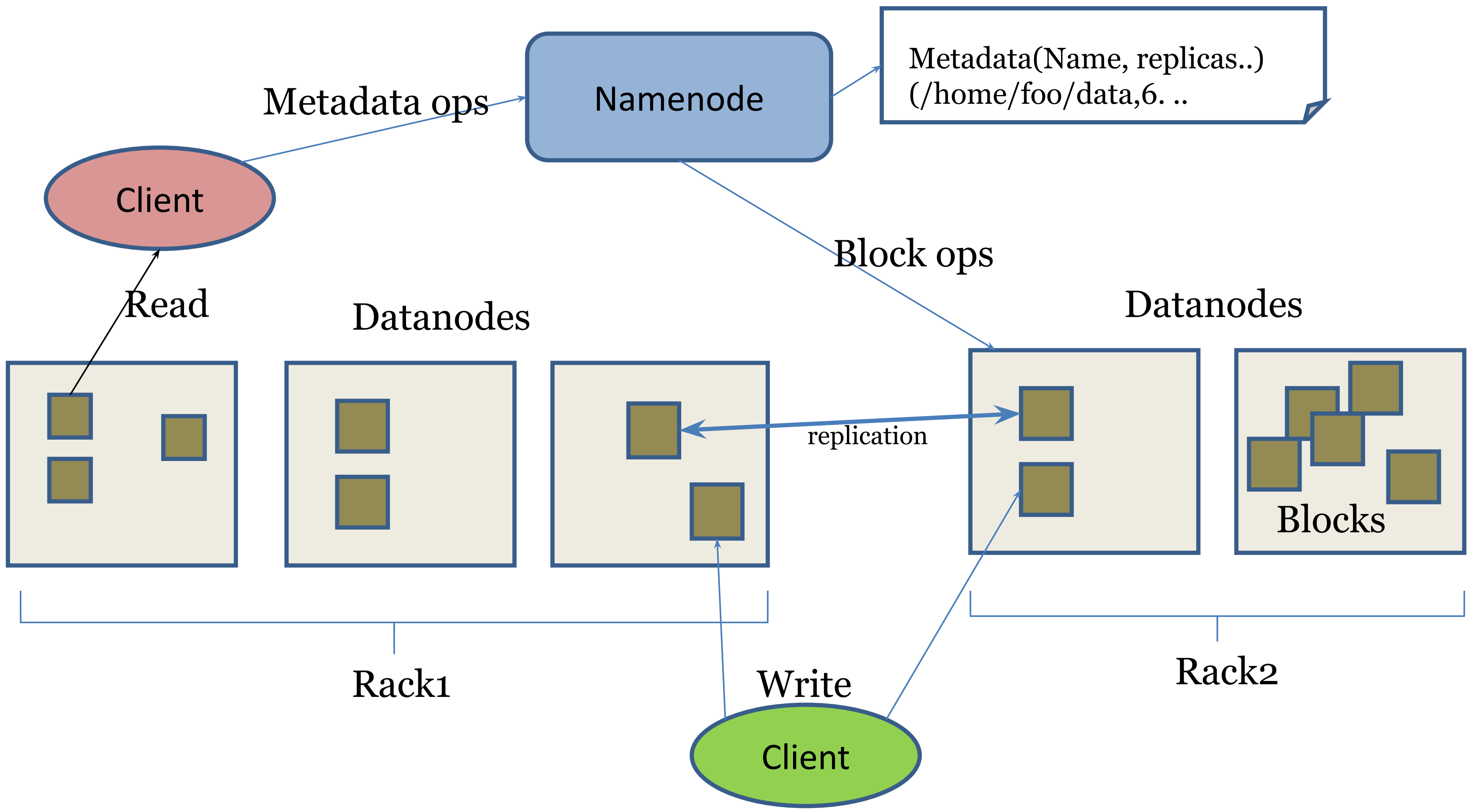
Architecture

- Master/slave architecture
- HDFS cluster consists of a single **Namenode**, a master server that manages the file system namespace and regulates access to files by clients.
- The **NameNode** executes file system namespace operations like opening, closing, and renaming files and directories.
- There are a number of **DataNodes** usually one per node in a cluster.
- The DataNodes manage storage attached to the nodes that they run on.
- HDFS exposes a file system namespace and allows user data to be stored in files.
- A file is split into one or more blocks and set of blocks are stored in DataNodes.
- DataNodes: serves read, write requests, performs block creation, deletion, and replication upon instruction from Namenode.

Namenode and Datanodes

- The NameNode and DataNode are pieces of software designed to run on commodity machines.
- These machines typically run a GNU/Linux operating system (OS). HDFS is built using the Java language; any machine that supports Java can run the NameNode or the DataNode software.
- The system is designed in such a way that user data never flows through the NameNode.

HDFS Architecture



File system Namespace

- Hierarchical file system with directories and files
- Create, remove, move, rename etc.
- Namenode maintains the file system
- Any meta information changes to the file system recorded by the Namenode.
- An application can specify the number of replicas of the file needed: replication factor of the file. This information is stored in the Namenode.

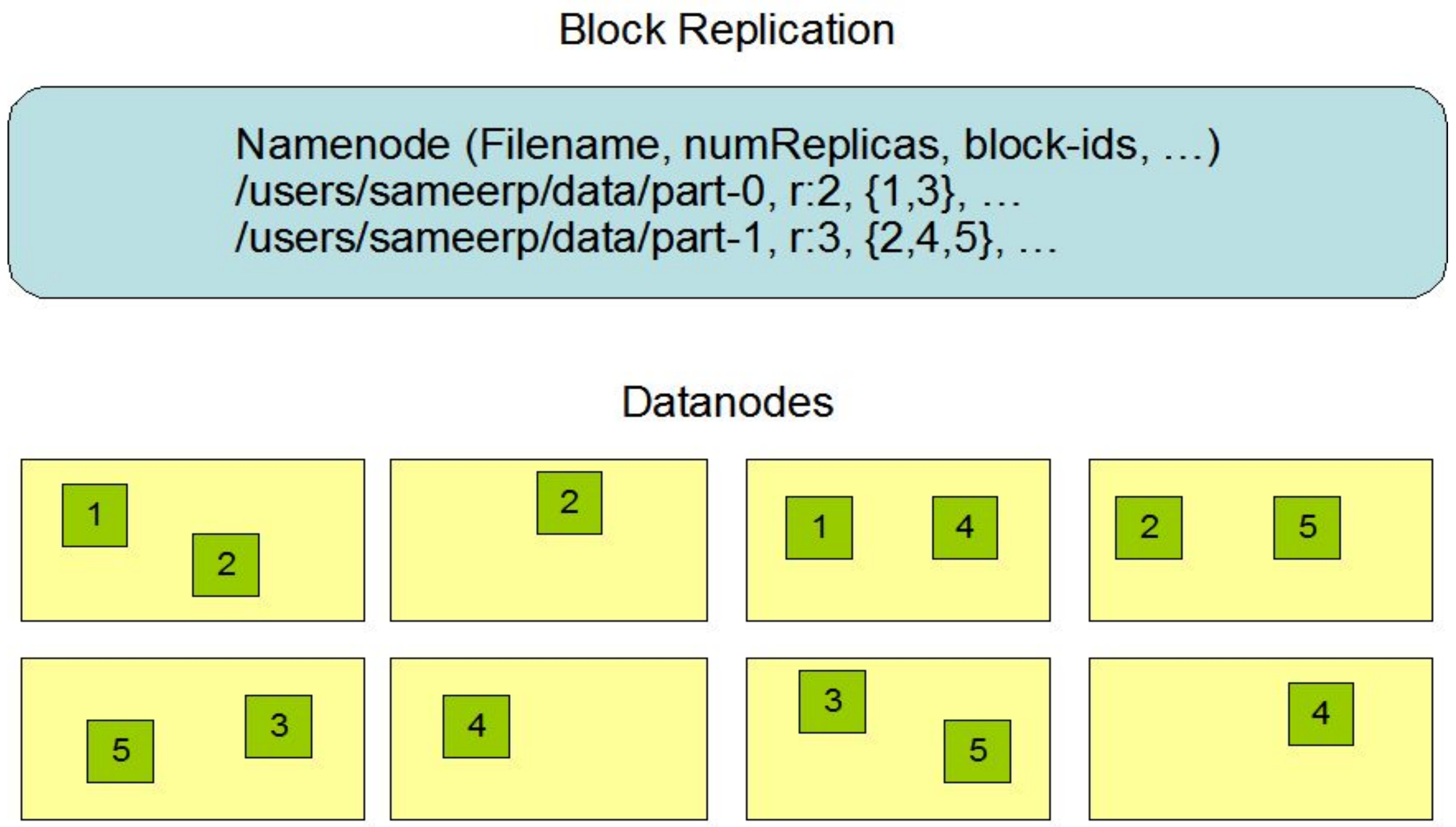
Data Replication

- HDFS is designed to store very large files across machines in a large cluster.
- Each file is a sequence of blocks.
- All blocks in the file except the last are of the same size.
- Blocks are replicated for fault tolerance.
- Block size and replicas are configurable per file.
- The Namenode receives a Heartbeat and a BlockReport from each DataNode in the cluster.
- BlockReport contains all the blocks on a Datanode.

Data Replication

- An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.
- The NameNode makes all decisions regarding 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. A Blockreport contains a list of all blocks on a DataNode.

Block Replication


Block Replication
Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …
Datanodes
1
2
2
1
4
2
5
5
3
4
3
5
4

Replica Placement

- The placement of the replicas is critical to HDFS reliability and performance.
- Optimizing replica placement distinguishes HDFS from other distributed file systems.
- Rack-aware replica placement:
  - Goal: improve reliability, availability and network bandwidth utilization
  - Research topic
- Many racks, communication between racks are through switches.
- Network bandwidth between machines on the same rack is greater than those in different racks.
- Namenode determines the rack id for each DataNode.
- Replicas are typically placed on unique racks
  - Simple but non-optimal
  - Writes are expensive
  - Replication factor is 3(Common Case)
  - Another research topic?
- Replicas are placed: one on a node in a local rack, one on a different node in the local rack and one on a node in a different rack.
- 1/3rd of the replicas on a node, 2/3rd of replicas on a rack and 1/3rd distributed evenly across remaining racks.

Replica Selection

- Replica selection for READ operation: HDFS tries to minimize the bandwidth consumption and latency.
- If there is a replica on the Reader node then that is preferred.
- HDFS cluster may span multiple data centers: replica in the local data center is preferred over the remote one.

Safemode Startup

- On startup Namenode enters Safemode.
- Replication of data blocks do not occur in Safemode.
- Each DataNode checks in with Heartbeat and BlockReport.
- Namenode verifies that each block has acceptable number of replicas
- After a configurable percentage of safely replicated blocks check in with the Namenode, Namenode exits Safemode.
- It then makes the list of blocks that need to be replicated.
- Namenode then proceeds to replicate these blocks to other Datanodes.

Filesystem Metadata

- The HDFS namespace is stored by Namenode.
- Namenode uses a transaction log called the EditLog to record every change that occurs to the filesystem meta data.
  - For example, creating a new file.
  - Change replication factor of a file
  - EditLog is stored in the Namenode's local filesystem
- Entire filesystem namespace including mapping of blocks to files and file system properties is stored in a file FsImage. Stored in Namenode's local filesystem.

Namenode

- Keeps image of entire file system namespace and file Blockmap in memory.
- 4GB of local RAM is sufficient to support the above data structures that represent the huge number of files and directories.
- When the Namenode starts up it gets the FsImage and Editlog from its local file system, update FsImage with EditLog information and then stores a copy of the FsImage on the filesytstem as a checkpoint.
- Periodic checkpointing is done. So that the system can recover back to the last checkpointed state in case of a crash.

Datanode

- A Datanode stores data in files in its local file system.
- Datanode has no knowledge about HDFS filesystem
- It stores each block of HDFS data in a separate file.
- Datanode does not create all files in the same directory.
- It uses heuristics(self discovery) to determine optimal number of files per directory and creates directories appropriately:
  - Research issue?
- 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.

The Communication Protocol

- All HDFS communication protocols are layered on top of the TCP/IP protocol
- A client establishes a connection to a configurable TCP port on the Namenode machine. It talks ClientProtocol with the Namenode.
- The Datanodes talk to the Namenode using Datanode protocol.
- RPC abstraction wraps both ClientProtocol and Datanode protocol.
- Namenode is simply a server and never initiates a request; it only responds to RPC requests issued by DataNodes or clients.

Objectives

Robustness

- Primary objective of HDFS is to store data reliably in the presence of failures.
- Three common failures are: Namenode failure, Datanode failure and network partition.

DataNode failure and heartbeat

- A network partition can cause a subset of Datanodes to lose connectivity with the Namenode.
- Namenode detects this condition by the absence of a Heartbeat message.
- Namenode marks Datanodes without Hearbeat and does not send any IO requests to them.
- Any data registered to the failed Datanode is not available to the HDFS.
- Also the death of a Datanode may cause replication factor of some of the blocks to fall below their specified value.

Re-replication

- The necessity for re-replication may arise due to:
  - A Datanode may become unavailable,
  - A replica may become corrupted,
  - A hard disk on a Datanode may fail, or
  - The replication factor on the block may be increased.

Cluster Rebalancing

- HDFS architecture is compatible with data rebalancing schemes.
- A scheme might move data from one Datanode to another if the free space on a Datanode falls below a certain threshold.
- In the event of a sudden high demand for a particular file, a scheme might dynamically create additional replicas and rebalance other data in the cluster.
- These types of data rebalancing are not yet implemented: research issue.

Data Integrity

- Consider a situation: a block of data fetched from Datanode arrives corrupted.
- This corruption may occur because of faults in a storage device, network faults, or buggy software.
- A HDFS client creates the checksum of every block of its file and stores it in hidden files in the HDFS namespace.
- When a clients retrieves the contents of file, it verifies that the corresponding checksums match.
- If does not match, the client can retrieve the block from a replica.

Metadata Disk Failure

- FsImage and EditLog are central data structures of HDFS.
- A corruption of these files can cause a HDFS instance to be non-functional.
- For this reason, a Namenode can be configured to maintain multiple copies of the FsImage and EditLog.
- Multiple copies of the FsImage and EditLog files are updated synchronously.
- The Namenode could be single point failure: automatic failover is NOT supported! Another research topic.

Data Organization :Data Blocks

- HDFS support write-once-read-many with reads at streaming speeds.
- A typical block size is 64MB (or even 128 MB).
- A file is chopped into 64MB chunks and stored.

Staging

- A client request to create a file does not reach Namenode immediately.
- HDFS client caches the data into a temporary file. When the data reached a HDFS block size the client contacts the Namenode.
- Namenode inserts the filename into its hierarchy and allocates a data block for it.
- The Namenode responds to the client with the identity of the Datanode and the destination of the replicas (Datanodes) for the block.
- Then the client flushes it from its local memory.

Staging (contd.)

- The client sends a message that the file is closed.
- Namenode proceeds to commit the file for creation operation into the persistent store.
- If the Namenode dies before file is closed, the file is lost.
- This client side caching is required to avoid network congestion.

Replication Pipelining

- When a client is writing data to an HDFS file, its data is first written to a local file as explained in the previous section.
- Suppose the HDFS file has a replication factor of three. When the local file accumulates a full block of user data, the client retrieves a list of DataNodes from the NameNode.
- This list contains the DataNodes that will host a replica of that block.

Replication Pipelining

- The client then flushes the data block to the first DataNode.
- The first DataNode starts receiving the data in small portions (4 KB), writes each portion to its local repository and transfers that portion to the second DataNode in the list.
- 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.

Replication Pipelining

- Finally, the third DataNode writes the data to its local repository.
- Thus, a DataNode can be receiving data from the previous one in the pipeline and at the same time forwarding data to the next one in the pipeline.
- Thus, the data is pipelined from one DataNode to the next.

API (ACCESSIBILITY)

Application Programming Interface

- HDFS provides Java API for application to use.
- Python access is also used in many applications.
- A C language wrapper for Java API is also available.
- A HTTP browser can be used to browse the files of a HDFS instance.

FS Shell, Admin and Browser Interface

- HDFS organizes its data in files and directories.
- It provides a command line interface called the FS shell that lets the user interact with data in the HDFS.
- The syntax of the commands is similar to other shells (e.g. bash, csh)
- To Create a directory named /foodir:

```
bin/hadoop dfs -mkdir /foodir
```

- Remove a directory named /foodir

```
bin/hadoop dfs -rmr /foodir
```

- View the contents of a file named /foodir/myfile.txt

```
bin/hadoop dfs -cat /foodir/myfile.txt
```

Admin and Browser Interface

- **Browser Interface:** A typical HDFS install configures a web server to expose the HDFS namespace through a configurable TCP port. This allows a user to navigate the HDFS namespace and view the contents of its files using a web browser.
- The DFSAdmin command set is used for administering an HDFS cluster. These are commands that are used only by an HDFS administrator. Here are some sample action/command pairs:
- Put the cluster in Safemode

```
bin/hadoop dfsadmin -safemode enter
```

- Generate a list of DataNodes

```
bin/hadoop dfsadmin -report
```

- Recommission or decommission DataNode(s)

```
bin/hadoop dfsadmin -refreshNodes
```

Space Reclamation

- **File Deletes and Undeletes**

  When a file is deleted by a user or an application, it is not immediately removed from HDFS. Instead, HDFS first renames it to a file in the /trash directory.

  The file can be restored quickly as long as it remains in /trash. A file remains in /trash for a configurable amount of time.

  After the expiry of its life in /trash, the NameNode deletes the file from the HDFS namespace.

Space Reclamation

- **File Deletes and Undeletes**

  The deletion of a file causes the blocks associated with the file to be freed. Note that there could be an appreciable time delay between the time a file is deleted by a user and the time of the corresponding increase in free space in HDFS.

  A user can Undelete a file after deleting it as long as it remains in the /trash directory.

  If a user wants to undelete a file that he/she has deleted, he/she can navigate the /trash directory and retrieve the file.

Space Reclamation

- **File Deletes and Undeletes**

  The /trash directory contains only the latest copy of the file that was deleted.

  The /trash directory is just like any other directory with one special feature: HDFS applies specified policies to automatically delete files from this directory.

  The current default policy is to delete files from /trash that are more than 6 hours old.

  In the future, this policy will be configurable through a well defined interface.

Space Reclamation

- **Decrease Replication Factor**

  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. The DataNode then removes the corresponding blocks and the corresponding free space appears in the cluster.

  Once again, there might be a time delay between the completion of the setReplication API call and the appearance of free space in the cluster.