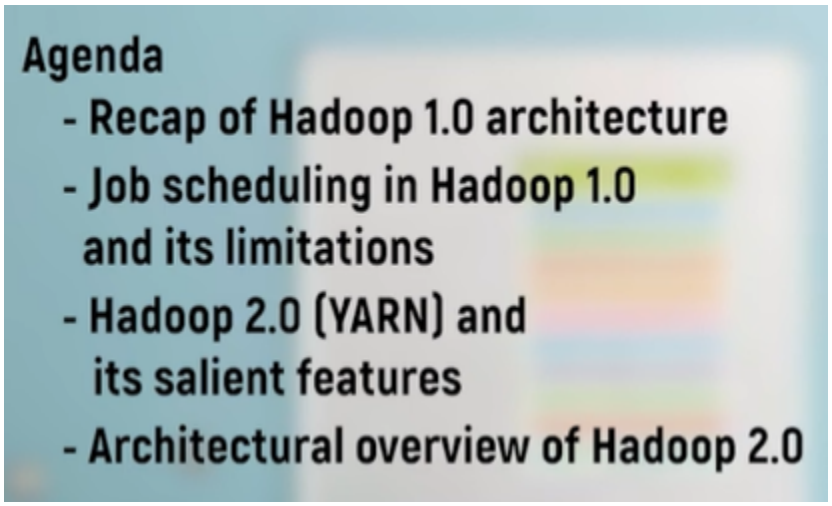
Closer look to Map reduce

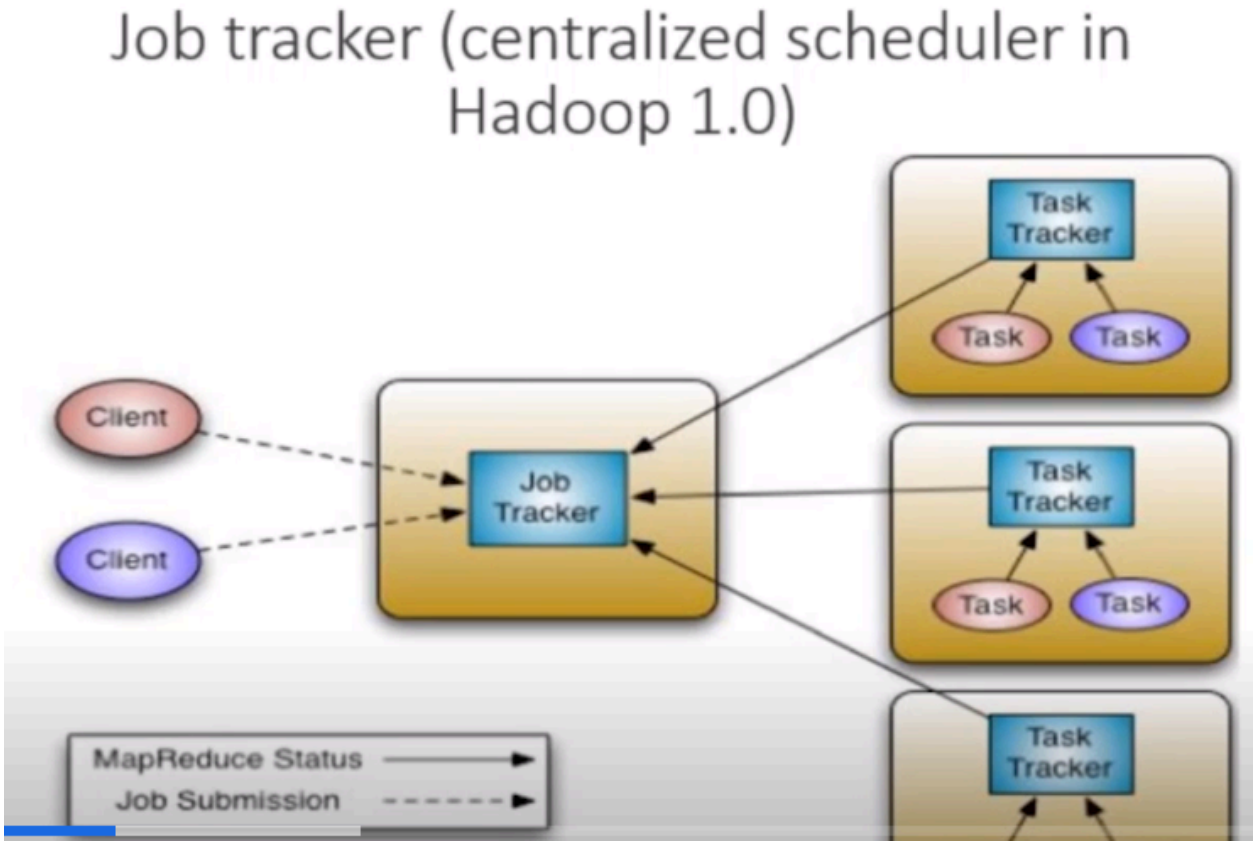
1. We solve anagram problem using map reduce just follow the ppt : https://olympus.mygreatlearning.com/courses/10977/files/745883?module_item_id=449020

Hadoop 2.x - YARN

1. 

2. YARN - yet another resource negotiator

3.


Job tracker (centralized scheduler in Hadoop 1.0)
Task Tracker
Task
Task
Client
Client
Job Tracker
Task Tracker
Task
Task
Task Tracker
MapReduce Status
Job Submission

4. Basically in hadoop 1.0 we have job tracker and task tracker

JobTracker:

- **Function:** Centralized job management and scheduling.
- **Responsibilities:**
  - Accepting job submissions from clients.
  - Splitting jobs into MapReduce tasks.
  - Assigning tasks to available TaskTrackers.
  - Monitoring task progress and status.
  - Re-scheduling failed tasks on different nodes.
  - Tracking resource usage (CPU, memory, disk).

TaskTracker:

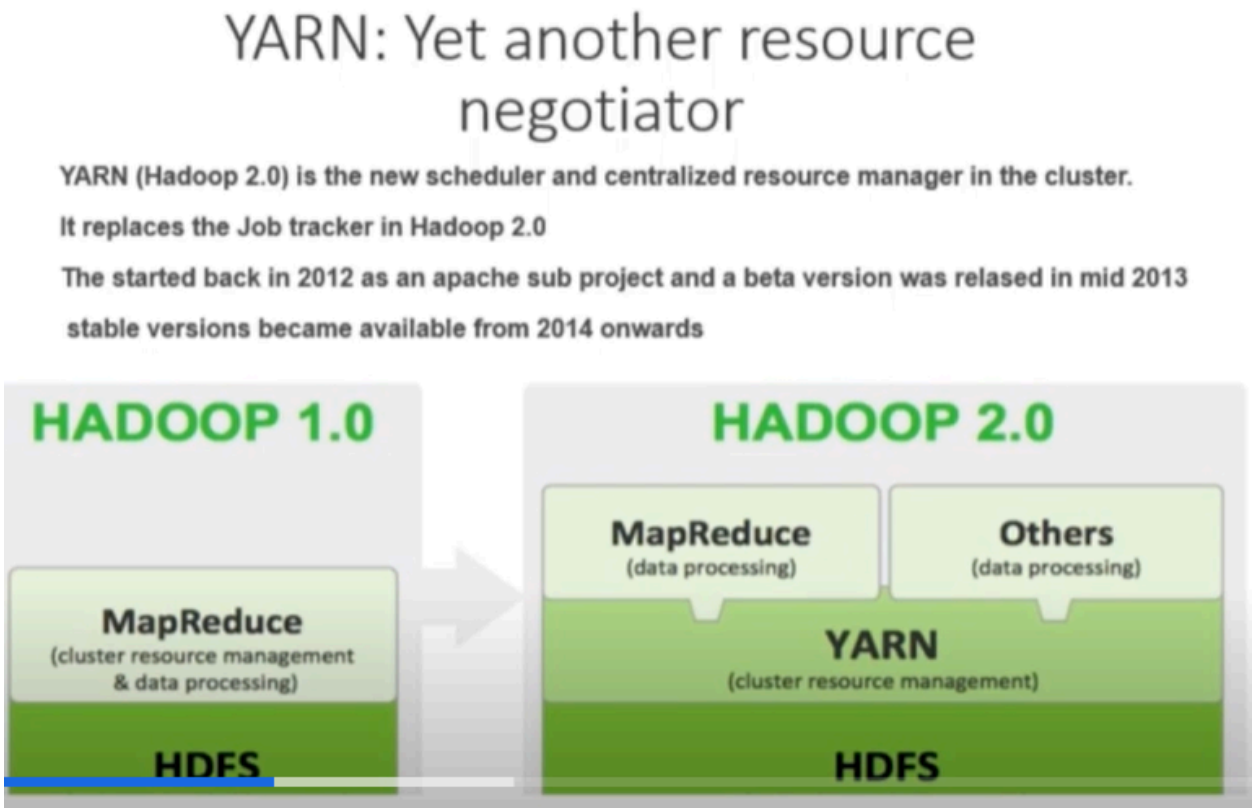
- **Function:** Running assigned MapReduce tasks on a node.
- **Responsibilities:**
  - Downloading necessary files and code for assigned tasks.
  - Launching Map and Reduce tasks as processes.
  - Monitoring task progress and reporting status to JobTracker.
  - Sending task output to HDFS or other designated locations.

Drawbacks:

- **Scalability bottleneck:** Single point of failure, limiting cluster size and performance.
- **Resource Management:** Coarse-grained control, inefficient resource allocation.
- **Flexibility:** Only supports MapReduce framework, hinders adoption of new technologies.
- **Fault Tolerance:** Job restart on failure could be slow and inefficient.
- **Monitoring and Diagnostics:** Limited visibility into job execution and cluster health.

5. And we can only have 4000 datanodes in hadoop 1.0 we cannot beyond that

6. Hadoop 2.0



Improvements in Hadoop 2.0:

- **YARN (Yet Another Resource Negotiator):** Replaces JobTracker with ResourceManager and ApplicationMaster for flexible resource management and support for diverse frameworks beyond MapReduce.
- **NodeManager:** Replaces TaskTracker, running tasks for various frameworks and reporting to ResourceManager.
- **Improved Scalability and Fault Tolerance:** Distributed architecture allows horizontal scaling and efficient job recovery from failures.

7. Here YARN also support other parallel processing framework such as spark etc
8. YARN Components and their Uses:

YARN (Yet Another Resource Negotiator) introduced in Hadoop 2.0, revolutionized resource management, enabling scalability and flexibility for diverse big data processing frameworks. Here's a breakdown of its key components and their practical uses:

1. ResourceManager (RM):

- Function: Oversees the entire cluster, manages resources (CPU, memory, disk), and allocates them to applications.
- Use case: Imagine your cluster has 1000 nodes. When an application submits a job, RM determines available resources, allocates containers (resource units), and monitors overall cluster health.

2. ApplicationMaster (AM):

- Function: Represents an application running on the cluster, negotiates resources with RM, launches containers for tasks, and monitors application progress.
- Use case: Consider a Spark application analyzing 1TB of data. AM requests resources from RM, gets allocated containers on different nodes, launches Spark executors within containers, and tracks their progress.

3. NodeManager (NM):

- Function: Runs on each node, manages and monitors resources, launches containers as instructed by RM, and reports resource usage back.
- Use case: Each node in your cluster has an NM. Upon receiving instructions from RM, NM allocates memory and CPU for containers, launches tasks within them (e.g., Spark executors), and sends resource usage updates to RM.

4. Container:

- Function: Encapsulates resources (memory, CPU, disk) allocated to a specific task in an application.
- Use case: When RM grants resources to AM, it provides containers on specific nodes. Tasks like Spark executors run within these containers, accessing allocated resources for processing.

Components of YARN

Its a 2 tiered model with some components (deamons) in master mode and some operating in slave mode.

Resource manager works in master mode (runs on a dedicated hardware in production setup)

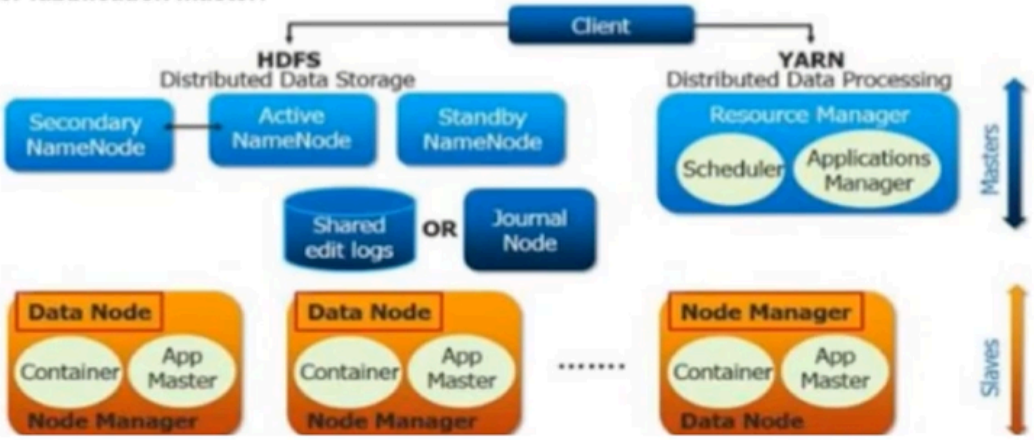
Node manager works in slave mode & its services are run in the data nodes.

Resource manager comprises of 2 components

1)Scheduler

2)Applications Manager

Node manager consists of a container (an encapsulation of resources for running a job) and an app Master (application master)



9. ---

10. We also have active standby namenode in hadoop 2.0 which ensures fault tolerance incase of namenode failure

Hadoop 1.x vs Hadoop 2.x

1. # Agenda

   - HDFS in Hadoop 1.x and its drawbacks
   - HDFS federation in YARN (Hadoop version 2.x)
   - Name node high availability
   - Journal nodes
   - Check pointing

2. # Name node drawbacks

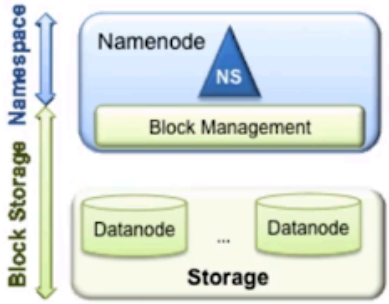


Image courtesy :https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/Federation.html

- **Traditionally there was only one name node and one checkpoint node in Hadoop version 1 (MRv1)**
- **Name-node saves the file system snapshot in the RAM**
- **Every block in HDFS would be an entry in the file system snapshot**
- **There are high chances that the name node's RAM could get exhausted**
- **At this point, there would not be a possibility of adding any more files in the HDFS even if the data nodes have free space**

3. The namenode store all the metadata of cluster in the RAM in hadoop 1.x

Lets do a simple math

Assuming every 128MB block of data on a data node occupies 1MB on the name-node's RAM
Typically a name node has around 100GB RAM = 100 * 1024 = 102400 MB of space
102400 MB of space on name node's RAM maps to 102400 blocks of 128MB files in HDFS
102400 * 128 MB = 1,31,07,200 MB of data = 12,800 GB = 12.5TB

According to this example, our name node can address only 12.5TB of data in the HDFS cluster.
There are hardware limitations to exponentially increasing the capacity of the RAM in a single server
Note : 1MB of RAM space will not be used to save the details of 128MB file block, it would be much lesser in reality, however the above problem still exists.

4.
5. Here we assumed every 128MB block of data occupies 1MB(metadata) on namenodes RAM
6. Here if we have 100gb of ram in namenode..then it can store 12.5TB of Datanode's metadata
7. Now even if our datanodes can store more data..then namenode will not store any metadata and we cannot retrieve this data
8. To solve this problem in hadoop 2.x we have hdfs federation

HDFS Federation

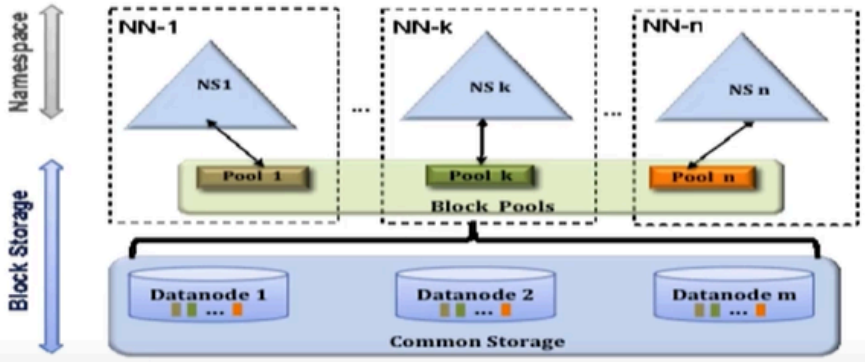


Image courtesy :https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/Federation.html

9.
10. HDFS namespaces

HDFS namespaces

- **HDFS basically sits on native UNIX file system**
- **Name space is just a logical group of files in the file system**
- **The simple example to understand namespace in a Windows environment is that C:\ drive is a separate namespace and D:\ drive is a separate name space**
- **The same idea is extendable with HDFS, i.e. /HR , /IT, /Finance can be considered as a separate name space (these are just directories containing files under them)**
- **Each name node would be assigned one namespace to manage, instead of one name node managing the entire namespace**
- **This helps in balancing the load on name node**

11. Even if our namenode rams get fulled ..we can have a new namenode without interrupting the services

HDFS Federation Explained:

HDFS Federation allows you to create a single logical namespace across multiple independent HDFS clusters. This provides several benefits, including:

- **Scalability:** Break the single name node bottleneck and manage larger datasets.
- **Isolation:** Separate different departments, applications, or data types into individual namespaces.
- **Flexibility:** Integrate HDFS with other systems or cloud storage.
- **High Availability:** Improve fault tolerance by distributing metadata across multiple NameNodes.

Architecture:

A federation consists of multiple independent HDFS clusters, each with its own NameNode and DataNodes. DataNodes register with all NameNodes in the federation and store data blocks for any namespace. Users interact with a specific `Namenode` through a `ViewFs` instance which maps paths to the appropriate underlying `NameNode`.

12.

Imagine a university with three departments: Research, Education, and Administration. Each department has its own data needs and security requirements.

- **Without Federation:** Each department might have its own HDFS cluster, leading to management overhead and resource silos.
- **With Federation:**
  - Three independent HDFS clusters are set up, one for each department.
  - A federation is created, connecting all three clusters.
  - Each department gets its own namespace (e.g., `/research`, `/education`, `/administration`).
  - Users access their specific namespace through `ViewFs` instances.
  - DataNodes store blocks for any namespace, optimizing resource utilization.

This allows each department to manage its data independently while benefiting from the shared storage infrastructure and scalability of the federation.

File read write operations in HDFS federation

- **File read/write request from the data nodes can be sent to all the name nodes in the federation**
- **The block metadata (file system snapshot) always resides in the RAM of the name node**
- **This metadata is replicated across all the name nodes to handle failure recovery**
- **Check pointing could be configured to happen in all the name nodes in the HDFS federation**

Note: This setup is not available out of the box from either Apache Hadoop or any vendors, it's let to the user to have such a setup in place with the help of the infrastructure team

13.

Read Operation:

1. **Client Request:** A client initiates a read request for a file in a specific namespace.
2. **ViewFs Mapping:** The ViewFs layer determines the appropriate NameNode responsible for that namespace.
3. **NameNode Lookup:** The client contacts the designated NameNode to retrieve file metadata (block locations, size, permissions).
4. **DataNode Communication:** The client directly communicates with the DataNodes storing the blocks to fetch data in parallel.
5. **Data Assembly:** The client reassembles the file from the retrieved blocks.

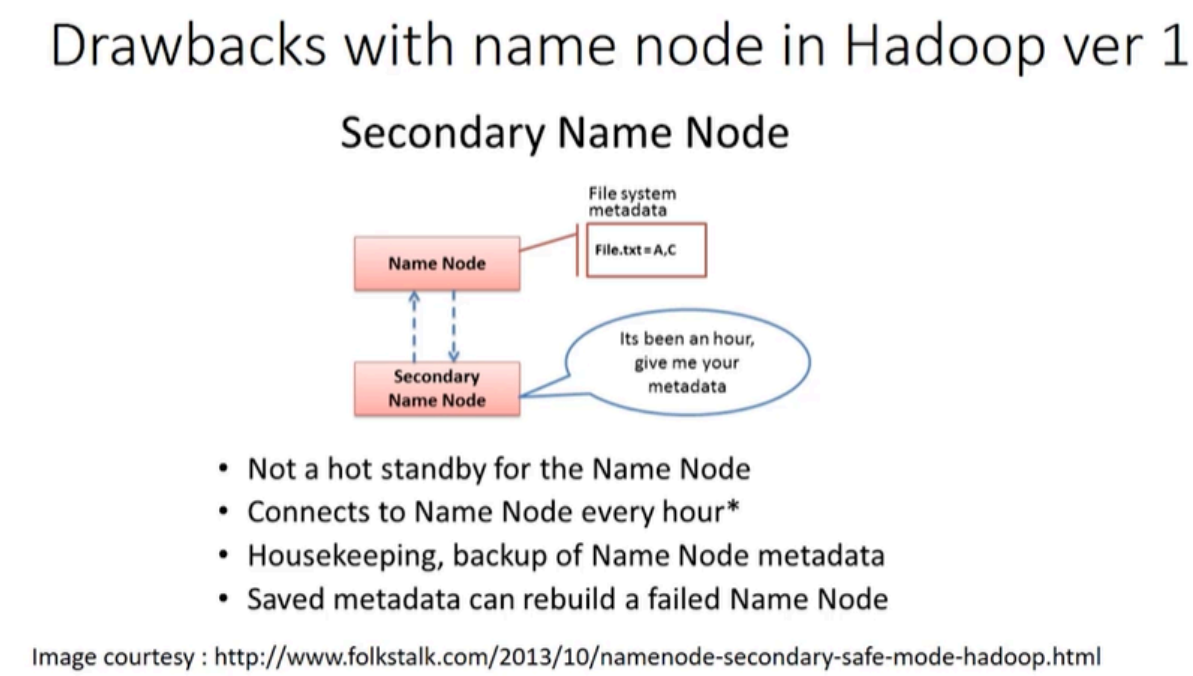
14.

Write Operation:

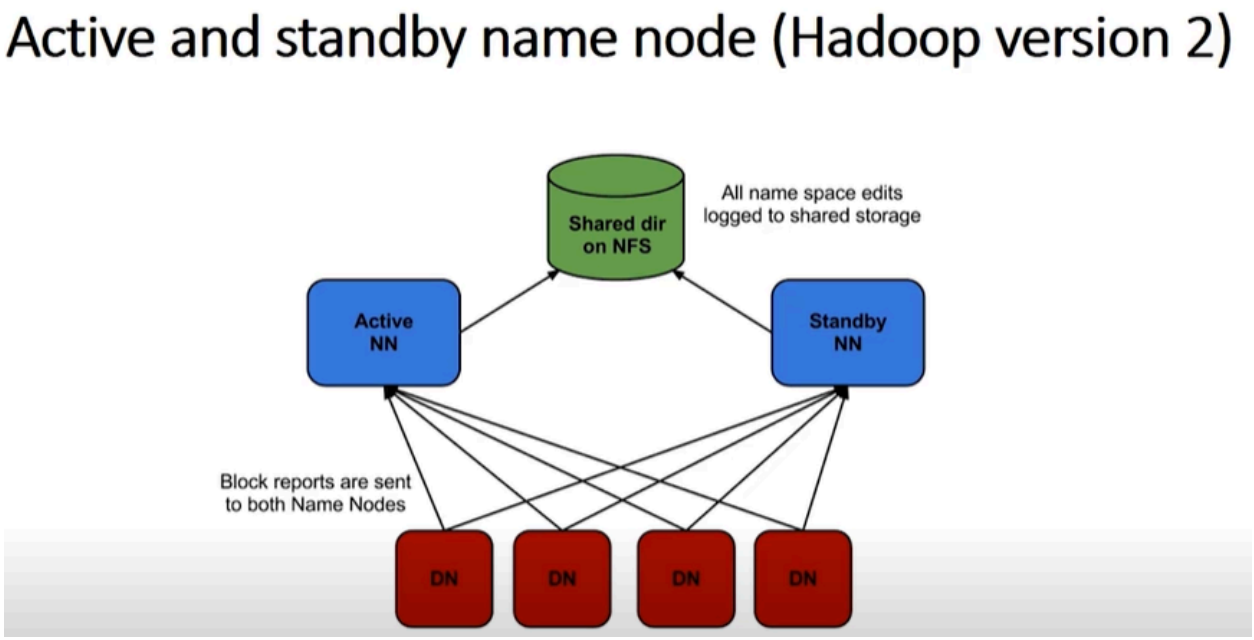
1. **Client Request:** A client initiates a write request for a file in a specific namespace.
2. **ViewFs Mapping:** The ViewFs layer determines the appropriate NameNode.
3. **NameNode Transaction:** The client contacts the designated NameNode to create a new file entry and allocate blocks.
4. **DataNode Selection:** The NameNode selects DataNodes to store blocks based on availability and replication factor.
5. **Data Transfer:** The client directly writes data blocks to the assigned DataNodes.
6. **Transaction Completion:** The client informs the NameNode of completion, updating file metadata.

15.

16. Drawbacks of hadoop 1.0



17. In hadoop 1.0 if the name node crashes ..then we cannot access the files in our hadoop cluster..we have to wait until it restarts again by copying the fsImage from secondary NN
18. This issue was covered in hadoop 2.0



19. The changes made to the cluster will be logged in the edit log of NN

EditLogs:

- **Purpose:** A persistent transaction log that records every change made to the HDFS file system metadata.
- **Management:** Maintained by the NameNode.
- **Contents:** Include operations like creating, deleting, renaming files/directories, changing permissions, replication factors, etc.
- **Structure:** Append-only files, ensuring sequential recording of changes.

Key Uses in Hadoop 2.0:

1. **Faster NameNode Recovery:**
    - On restart, the NameNode quickly replays EditLogs to reconstruct the latest state of the file system, reducing downtime.
    - Compare this to loading the entire file system metadata (FsImage) from scratch, which can be time-consuming for large clusters.
2. **High Availability (HA):**
    - EditLogs are shared between the Active and Standby NameNodes in an HA configuration.
    - The Standby continuously applies EditLogs to maintain a synchronized state, enabling seamless failover if the Active NameNode fails.

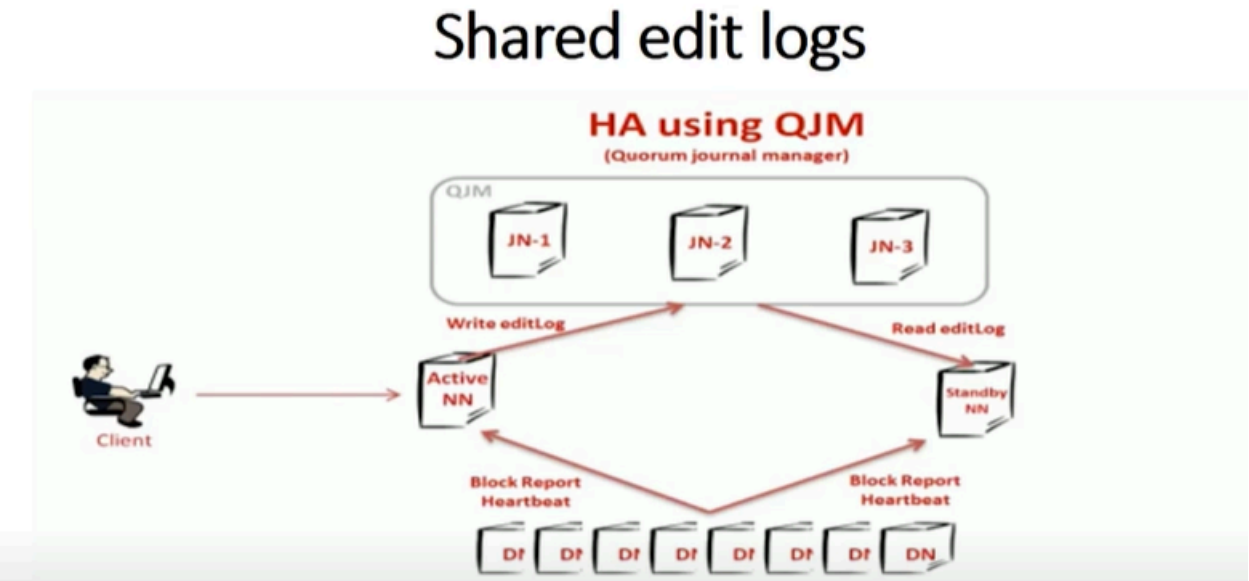
3. Secondary NameNode (BackupNode):

    - Periodically merges EditLogs with the FsImage to create a new checkpoint, reducing the amount of EditLog data the NameNode needs to process on restart.
    - Checkpointing also limits the time needed to recreate the file system state from a crashed NameNode.

4. Auditing and Troubleshooting:

    - EditLogs can be used to track changes in the file system for security auditing, compliance, and troubleshooting purposes.
    - They can help identify unauthorized modifications, accidental deletions, or pinpoint the causes of file system corruption.

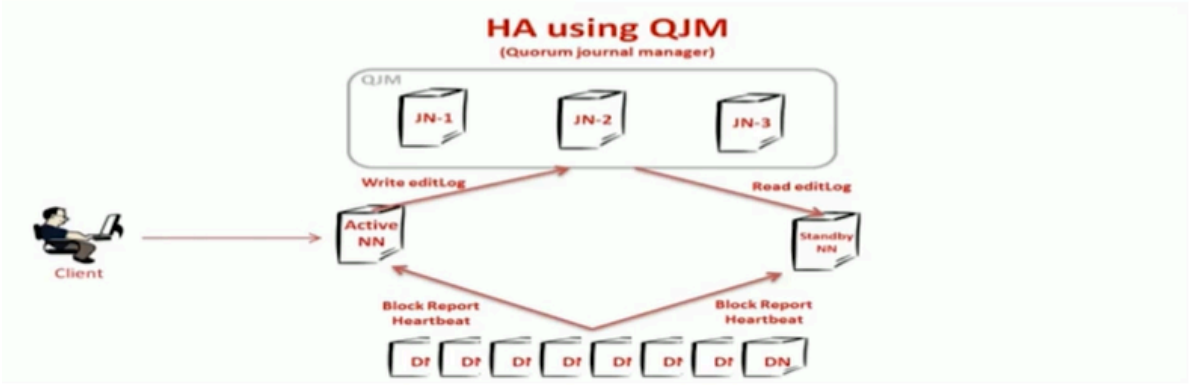
20.
21. Shared edit logs



22. Here we have QJM..which elects one JN(joural node) and replicates the data to other JN's.. incase of one JN fails

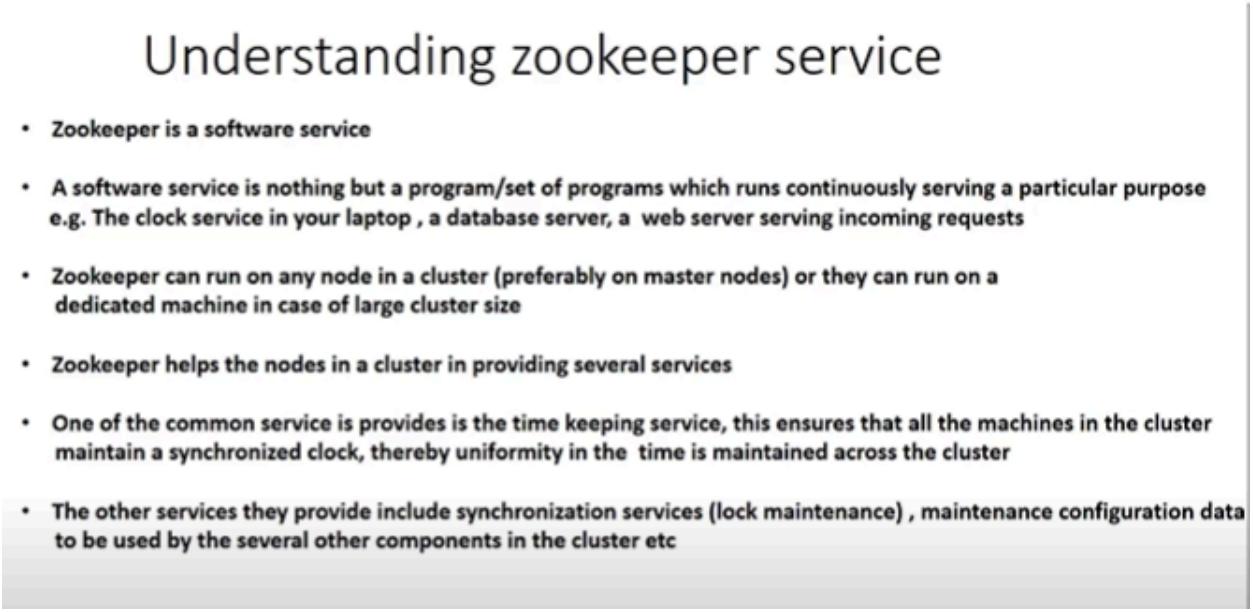
What are these journal nodes ?

- These are machines which host the light weight services (journal node services)
- Typically these services run on multiple machines for failover management
- The active name node backs up their edit logs into these journal nodes
- These edit logs are replicated across multiple journal nodes
- The standby name node periodically uses these edit logs for regular check pointing
- There might not be a need for a separate secondary name node for check pointing purposes



23.

24. Apache ZooKeeper



Imagine you're designing a large-scale online game like World of Warcraft. Hundreds of thousands of players are exploring Azeroth simultaneously, battling dragons and raiding dungeons. How do you ensure everything runs smoothly? This is where **Apache Zookeeper** comes in.

Zookeeper, in simple terms, is a coordination service for distributed systems. It's like a central nervous system for your application, allowing different parts to communicate and work together seamlessly. Here's what it does:

25. Zookeeper is a software service which maintains synchronization(to maintain the same time on all Datanodes etc) across the hadoop cluster
26. Time Keeper is one of the important service which zoo keeper provides us
27. It also helps in lock maintenance (for example ..if we write a file on datanode and a user requests a read on that file ..before the file is completely copied(wrote) to disk..then zoo keeper wont allow it as it provides lock maintenance)

28. It also provides failover controller(ZKFC)

The Apache Zookeeper Failover Controller (ZKFC) is a dedicated component within the Hadoop ecosystem that specifically focuses on ensuring high availability for the **NameNode**, the central management component in HDFS (Hadoop Distributed File System).

Here's how ZKFC works:

1. **Monitoring:** ZKFC continuously monitors the health of the active NameNode, checking for issues like crashes or unresponsiveness.

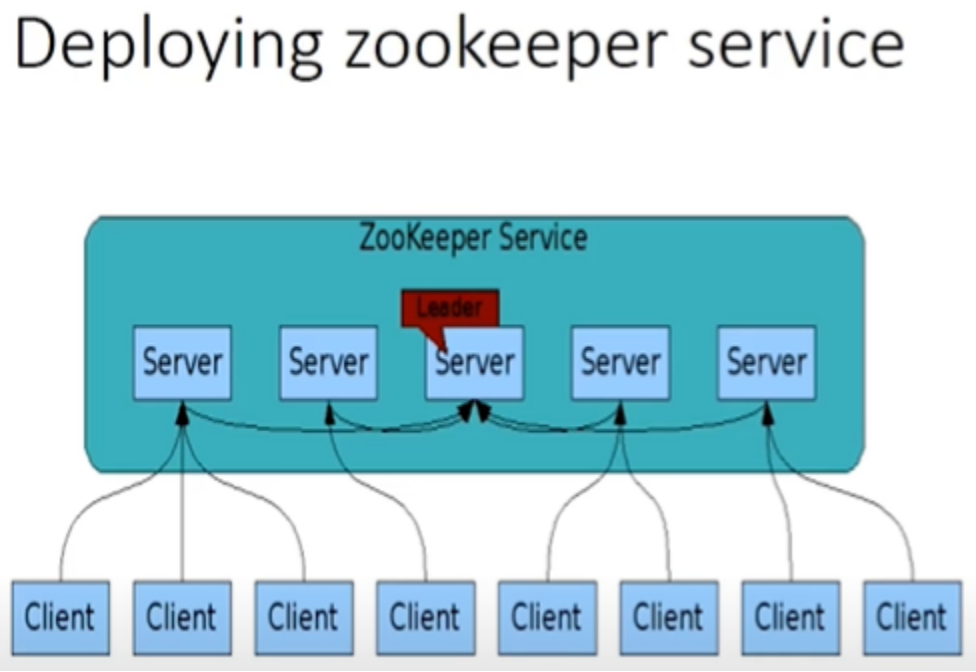
2. **Failover Trigger:** If ZKFC detects a problem with the active NameNode, it triggers a failover process.

3. **Leader Election:** ZKFC initiates a leader election among standby NameNodes within the cluster. Remember, in a typical Hadoop setup, you have one active NameNode and multiple standby NameNodes for redundancy.

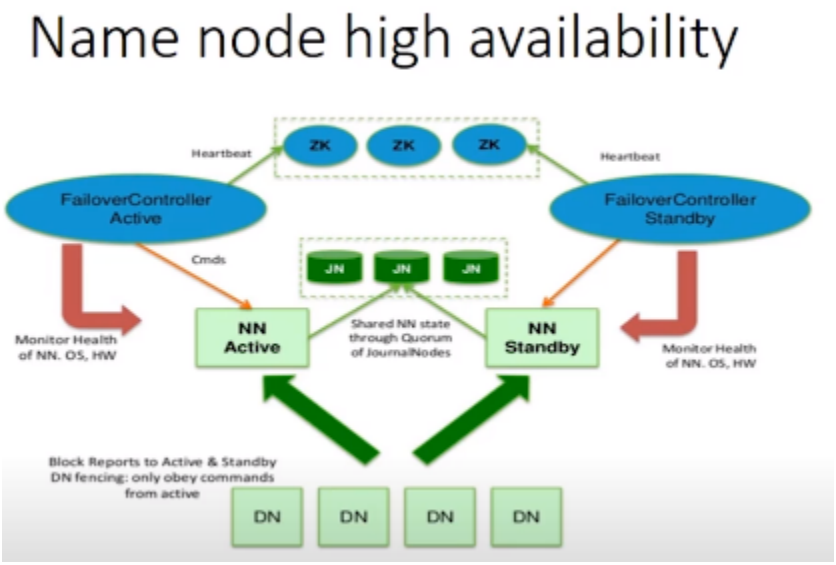
4. **Standby Activation:** Based on the election results, ZKFC promotes the elected standby NameNode to become the new active NameNode.

5. **Configuration Update:** ZKFC updates the Zookeeper leader information and informs other cluster components about the new active NameNode.

29. Here leader is the active namenode..if it crashes then it will elect another NN as a leader



30. Name Node high avalability



31. Here zookeeper has failovercontrollerActive..which actively checks the health of active NN

Hadoop 3.0

1.

Salient features Hadoop version 3

- **Stable version 3.0.0 available as on 13th December 2017**
- **Minimum required Java version is Java 8**
- **Support for erasure coding in HDFS**
- **YARN Timeline Service v.2**
- **Shell script rewrite**
- **Shaded client jars**
- **Intra-data node balancer**

2.
3. Erasure coding

No need to save 3 replicas of the data

- Data was replicated 3 times in earlier version of Hadoop i.e. Hadoop version 1.x and 2.x
- The idea of replication this was to ensure data backup in the event of data node failure
- The drawback of this is that it reduces the storage space in the cluster by 1/3rd
- Data need not be replicated 3 times, instead a mechanism called "erasure coding" is used to re-construct the lost data block using parity bits

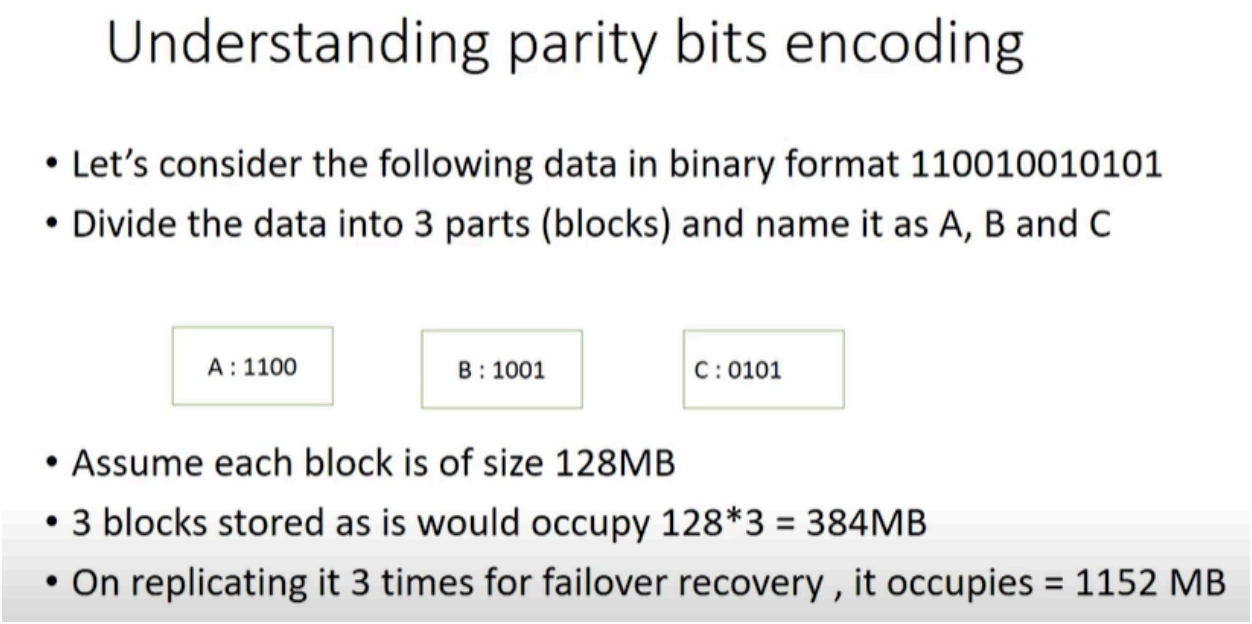
4. Prev in hadoop 1.x and hadoop 2.x we used to replicate our blocks of data in 3 places…this inturn used to occupy more cluster size(if we have 10 TB of data..then it would occupy 30 tb on hadoop)

Hadoop version 3 uses "erasure coding"

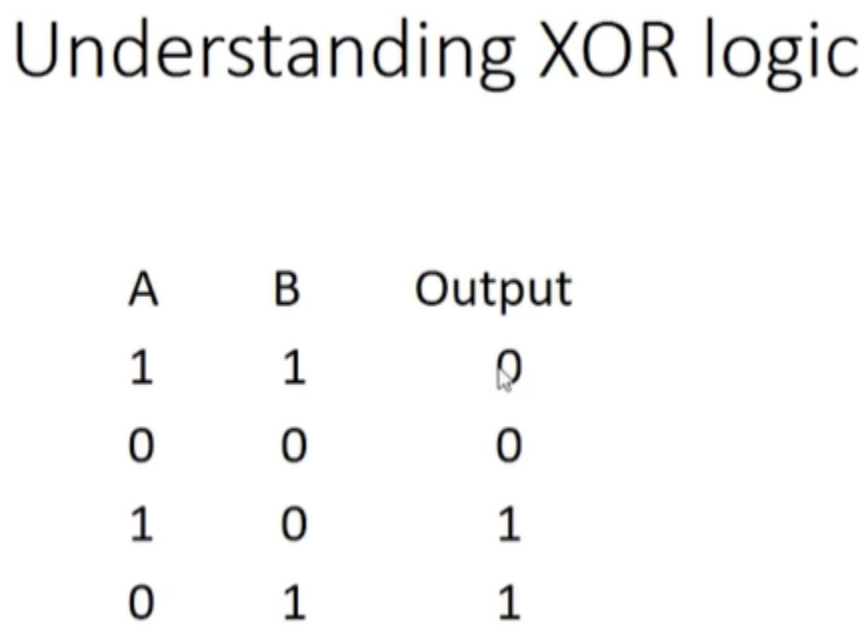
- Lost data can be reconstructed using parity bits concept
- This relives a maximum of 50% extra storage space in the cluster

5.

6. Before understanding erasure coding ..frst we have to understand parity bit encoding

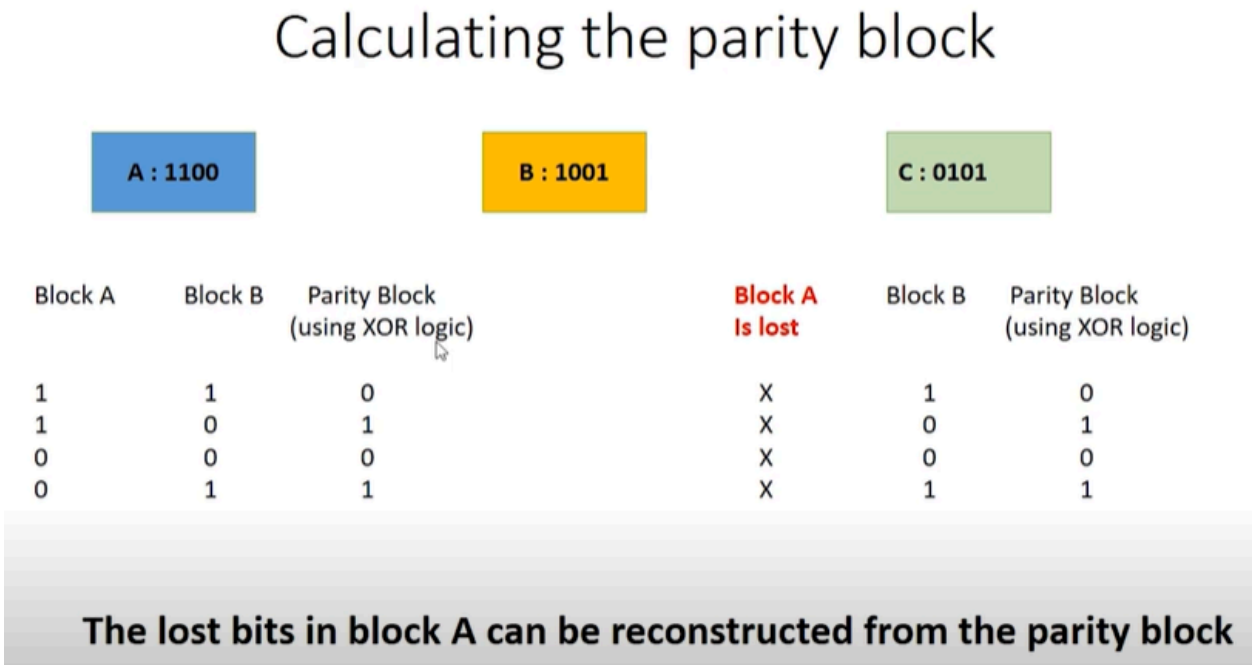


7. Here lets assume each block occupies around 128MB on our hard drive
8. Now using old hadoop version ..the storage occupied on cluster is around 1152MB
9. To understand parity bit..lets understand XOR Logic

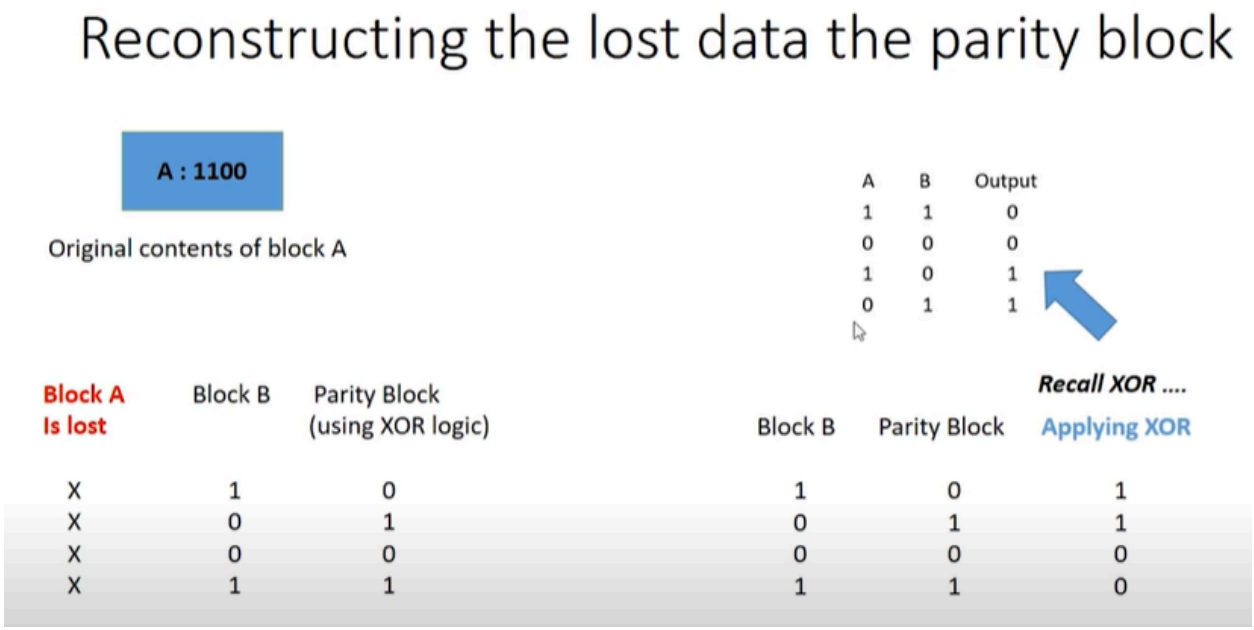


10.

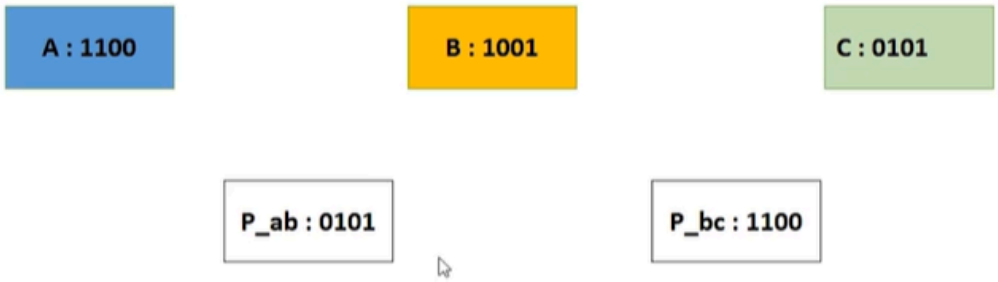
11. Calculating parity block



12. Here each block is stored in different data nodes
13. Now even if we have lost block A..we can retrieve the block A by using parity block
14. If apply XOR on parity block and block B..we can get back block A



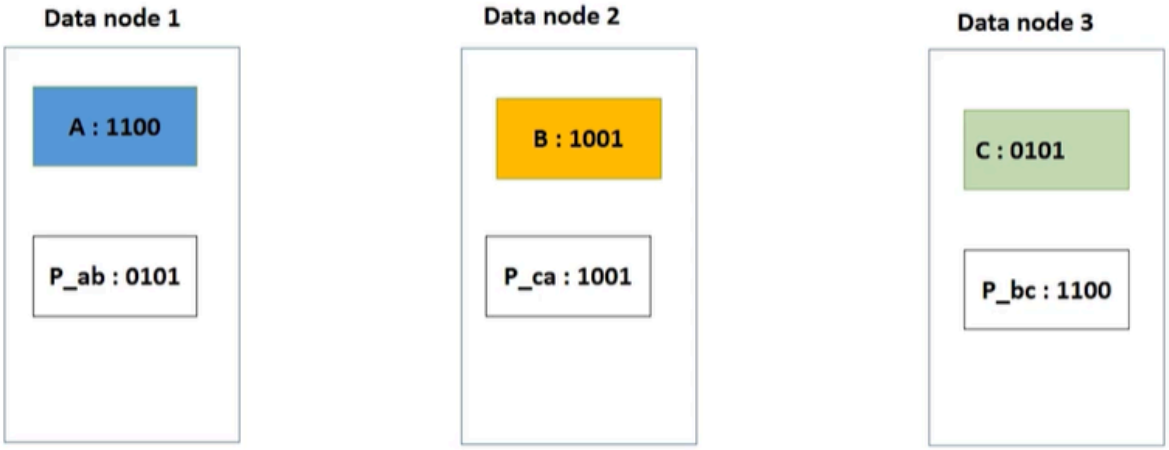
Saving the data & the parity blocks



- We now have 5 blocks of information eventually (3 data blocks and 2 parity blocks)
- Each block occupies 128MB of data on the disc of a data node
- The 3 data blocks would occupy 128 * 3 = 384 and replicating it 3 times occupies 1152 MB
- Saving the above 5 blocks would occupy 128 * 5 = 640MB (almost 40% reduction in disc space)
- Parity blocks are in fact replicated directly or indirectly which would still give up to 30% space saving
- Any savings in disc space would directly reflect on the hardware cost savings

15.
16. We may ask…does parity blocks get replicated?(google)

Saving the data blocks and the parity blocks



Note: Erasure coding uses a more complex technique to handle failure recovery using this parity bit concept. The discussion here is limited to basic understanding of block reconstruction.

17. Intra data node balancer

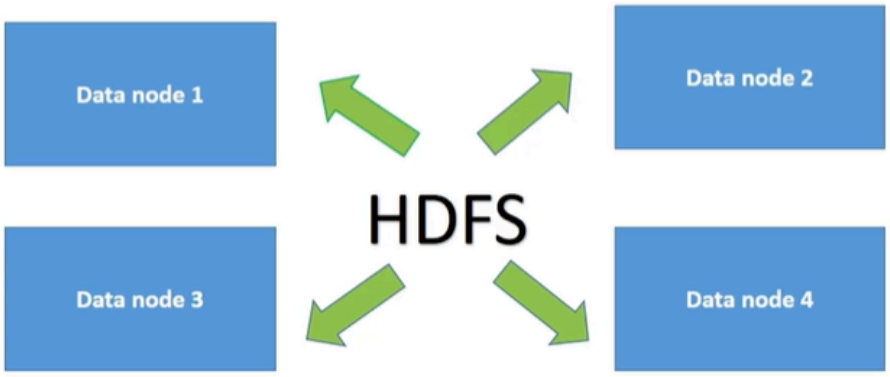
Intra data node balancer

- Earlier versions of Hadoop had an inter data node balancer (HDFS balancer)
- This ensured that the data in the cluster was always evenly distributed among the nodes
- In case we had 10GB of data and 5 data nodes, each data node would get 2GB of data
- Typically each data node in the Hadoop cluster has several hard discs
- Intra data node balancer ensures that data is evenly distributed among these nodes

18. HDFS balancer would evenly data among the datanodes present in the cluster but it never ensured all the hard drives present in the data node evenly distributing the data
19. HDFS balancer

HDFS balancer

- HDFS balancer ensures that the data is evenly distributed across the data nodes in a cluster
- An example, 12TB of data would be distributed as 3 TB on each of the four data nodes
- HDFS balancer would ensure deleting existing data or adding new data would still result in a balanced cluster



20.

21. Drawbacks of HDFS balancer

Drawbacks of HDFS balancer

- A data node has multiple hard drives, usually in the order of 10 to 20 separate hard drives each of capacity anywhere between 1TB to 4TB
- HDFS balances does not ensure if the data in all the hard drives is evenly distributed
- The skewed data distribution in the hard drives of a data node is due to deletion of data and addition of new data into HDFS

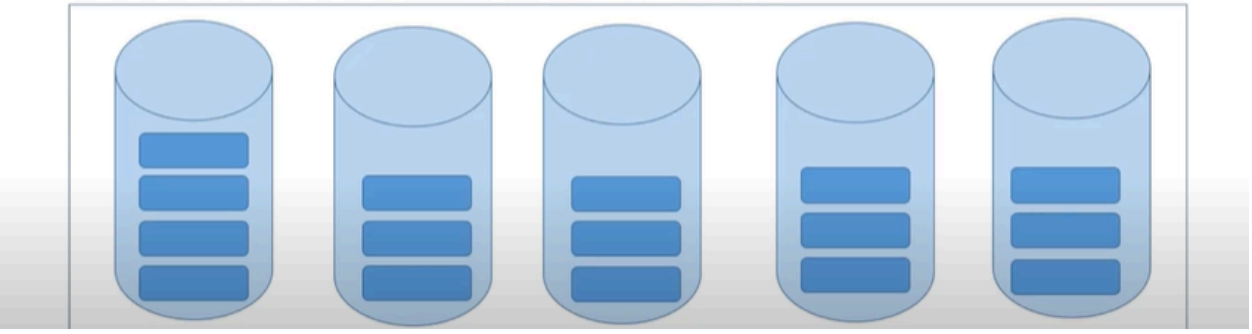
Data node with multiple hard drives

22. Here we can see one hard drive filled with data and other are half empty
23. Intra data node balancer

Intra data node balancer

- This component ensures that the data is almost uniformly distributed among the all the discs of a data node
- This results in an improved data read performance

Data node with multiple hard drives

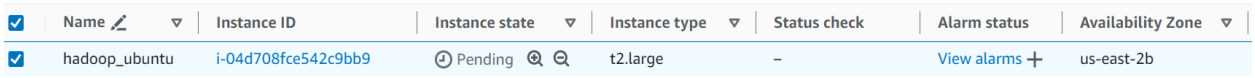


24. Also IDNB gives good read and write speeds as the data is well structured

- Understand the 2 important features of Hadoop version 3
- Basics of erasure coding
- Intra data node balancer

25.
26. Ppt for hadoop 3.0 : https://olympus.mygreatlearning.com/courses/10977/files/745881?module_item_id=449027

Hadoop 3.0 installation

1. Started an Ec2 ubuntu instance on aws to run hadoop
2. 

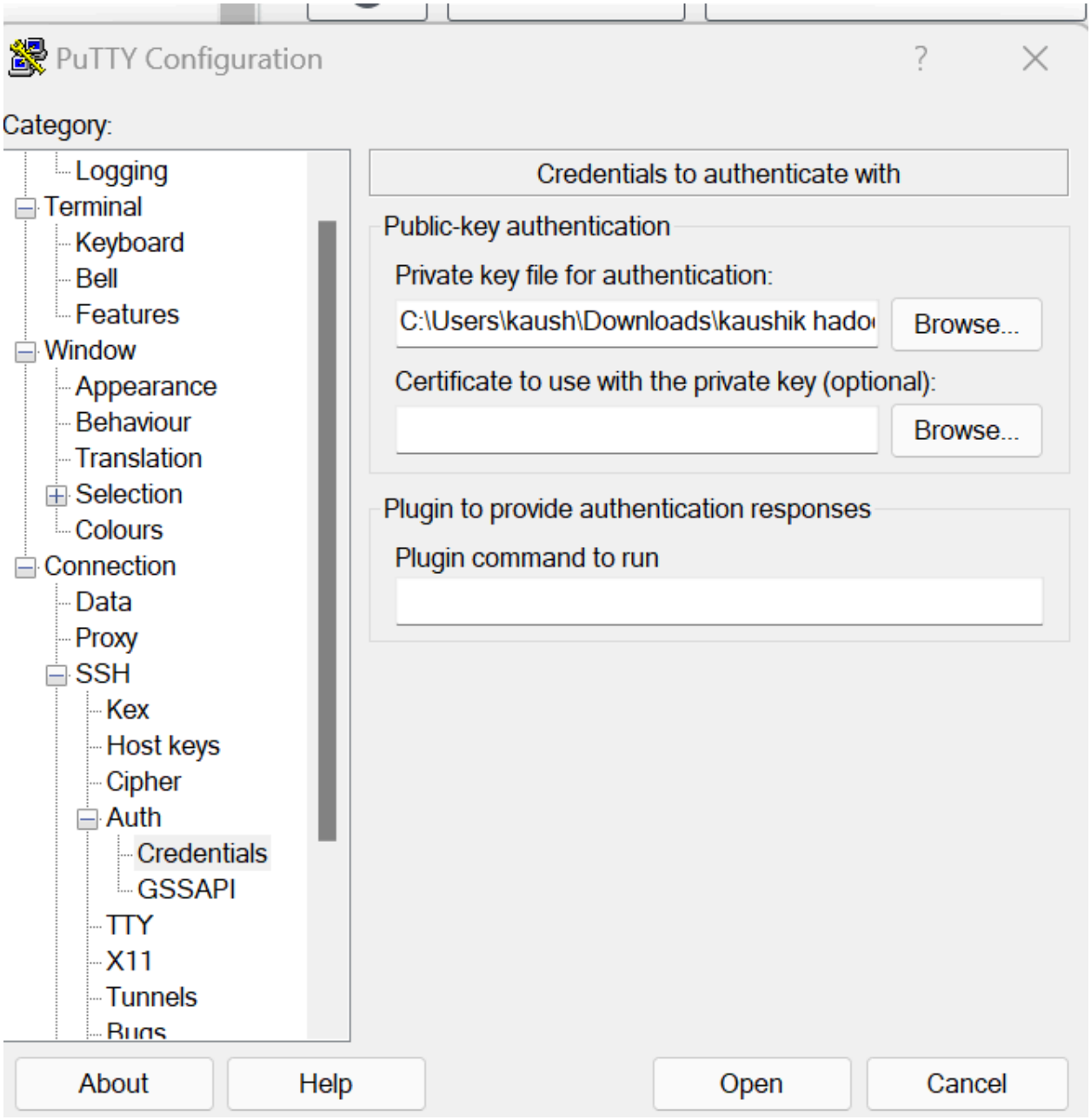
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
hadoop_ubuntu	i-04d708fce542c9bb9	Pending	t2.large	–	View alarms	us-east-2b

3. If we using windows then to access our machine ..we have to use putty
4. Putty cannot read the aws key pair's key file..so we have to download puttygen and convert our pem file to ppk using puttygen and then use aws key for accessing our machine

Hadoop 3.0 Installation part-2

1. Here we need to connect with the instance and next have install hadoop and all
2. To connect with our instance..first we have to copy our ipv4 address of instance

3. Next we have copy our ip address in putty and next go SSH-AUTH and paste our private ppk key



4. Next in the terminal we login as “Ubuntu”

5. Then we create a hadoop group called hadoop and a new user “kaushik”

```
ubuntu@ip-172-31-17-212:~$ sudo addgroup hadoop
Adding group `hadoop' (GID 1001) ...
Done.
ubuntu@ip-172-31-17-212:~$ sudo adduser -- ingroup hadoop hduser
adduser: Only one or two names allowed.
ubuntu@ip-172-31-17-212:~$ sudo adduser --ingroup hadoop hduser
Adding user `hduser' ...
Adding new user `hduser' (1001) with group `hadoop' ...
Creating home directory `/home/hduser' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for hduser
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
ubuntu@ip-172-31-17-212:~$
```

6. Now we need to give root permissions to this user
7. To do that we have to become a root user..to become a root user

```
ubuntu@ip-172-31-17-212:~$ sudo su
root@ip-172-31-17-212:/home/ubuntu#
```

8. We need to add this user to sudoers..to go there “sudo visudo” then under #user privilege specification

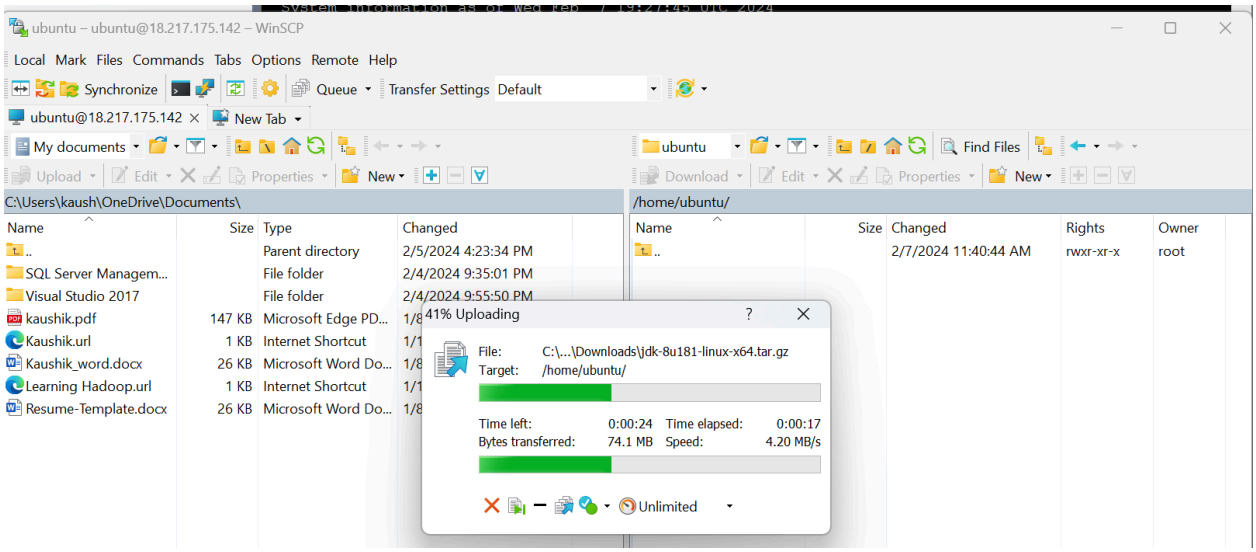
```
# User privilege specification
root    ALL=(ALL:ALL) ALL
kaushik ALL=(ALL:ALL) ALL
```

9. Now we’ll use kaushik …to change to another user we use

```
root@ip-172-31-23-165:/home/ubuntu# su - kaushik
```

10. Now we have to install java

11. We have used help of WinScp to transfer the java tar file from our local disk to Ec2 Instance



12. Next we copy the file from local to home/ubuntu first and then from ubuntu to usr/local

```
kaushik@ip-172-31-23-165:/home/ubuntu$ cd /usr/local
kaushik@ip-172-31-23-165:/usr/local$ sudo su
root@ip-172-31-23-165:/usr/local# cp /home/ubuntu/jdk-8u181-linux-x64.tar.gz /usr/local/
root@ip-172-31-23-165:/usr/local# exit
exit
kaushik@ip-172-31-23-165:/usr/local$ ls
bin  etc  games  include  jdk-8u181-linux-x64.tar.gz  lib  man  sbin  share  src
kaushik@ip-172-31-23-165:/usr/local$ sudo tar xvzf jdk-8u181-linux-x64.tar.gz
```

13. Next we install java using sudo tar xvzf
14. After installation ..we rename file to java for our convenience

```
sudo mv jdk1.8.0_181 java
```

15. Just followed the commands on the installation file.

Hadoop 3.0 and Spark Installation