Hadoop Ecosystem and YARN

- HDFS: Hadoop Distributed File System
- YARN: Yet Another Resource Negotiator
- MapReduce: Programming based Data Processing
- Spark: In-Memory data processing
- PIG, HIVE: Query-based processing of data services
- HBase: NoSQL Database
- Mahout, Spark MLLib: Machine Learning algorithm libraries
- Solar, Lucene: Searching and Indexing
- Zookeeper: Managing cluster
- Oozie: Job Scheduling

NameNode high availability

- High Availability was a new feature added to Hadoop 2.x to solve the Single point of failure problem in the older versions of Hadoop.
- As the Hadoop HDFS follows the master-slave architecture where the NameNode is the master node and maintains the filesystem tree. So HDFS cannot be used without NameNode.
- Before Hadoop 2.0.0, the NameNode was a single point of failure (SPOF) in an HDFS cluster. Each cluster had a single NameNode, and if NameNode fails, the cluster as a whole would be out of services. The cluster will be unavailable until the NameNode restarts or brought on a separate machine.
- Hadoop 2.0 overcomes this SPOF by providing support for many NameNode. HDFS NameNode High Availability architecture provides the option of running two redundant NameNodes in the same cluster in an active/passive configuration with a hot standby.

NameNode high availability

- Active NameNode – It handles all client operations in the cluster.
- Passive NameNode – It is a standby namenode, which has similar data as active NameNode. It acts as a slave, maintains enough state to provide a fast failover, if necessary.
- If Active NameNode fails, then passive NameNode takes all the responsibility of active node and the cluster continues to work.

Issues in maintaining consistency in the HDFS High Availability cluster are as follows:

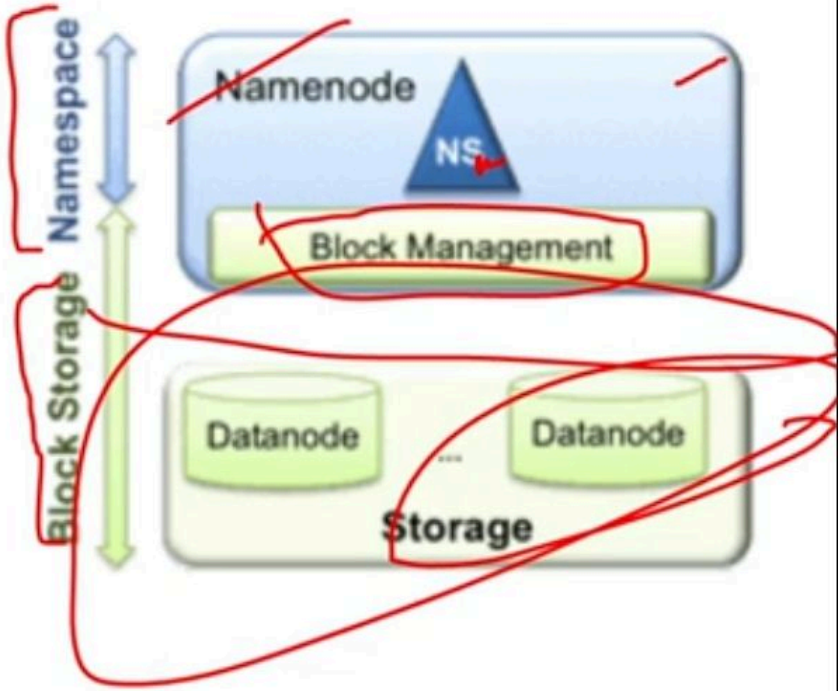
- Active and Standby NameNode should always be in sync with each other, i.e. they should have the same metadata. This permit reinstating the Hadoop cluster to the same namespace state where it got crashed. And this will provide us to have fast failover.
- There should be only one NameNode active at a time. Otherwise, two NameNode will lead to corruption of the data.

HDFS architecture

Namespace: consists of files, blocks, and directories. This layer provides support for namespace related filesystem operations like create, delete, modify, and list files.

Block Storage layer:

- **Block Management:** Block Management provides DataNode cluster membership by handling registrations, and periodic heartbeats. It also maintains locations of blocks, replica placement.
- **Storage:** DataNode manages storage space by storing blocks on the local file system and providing read/write access.



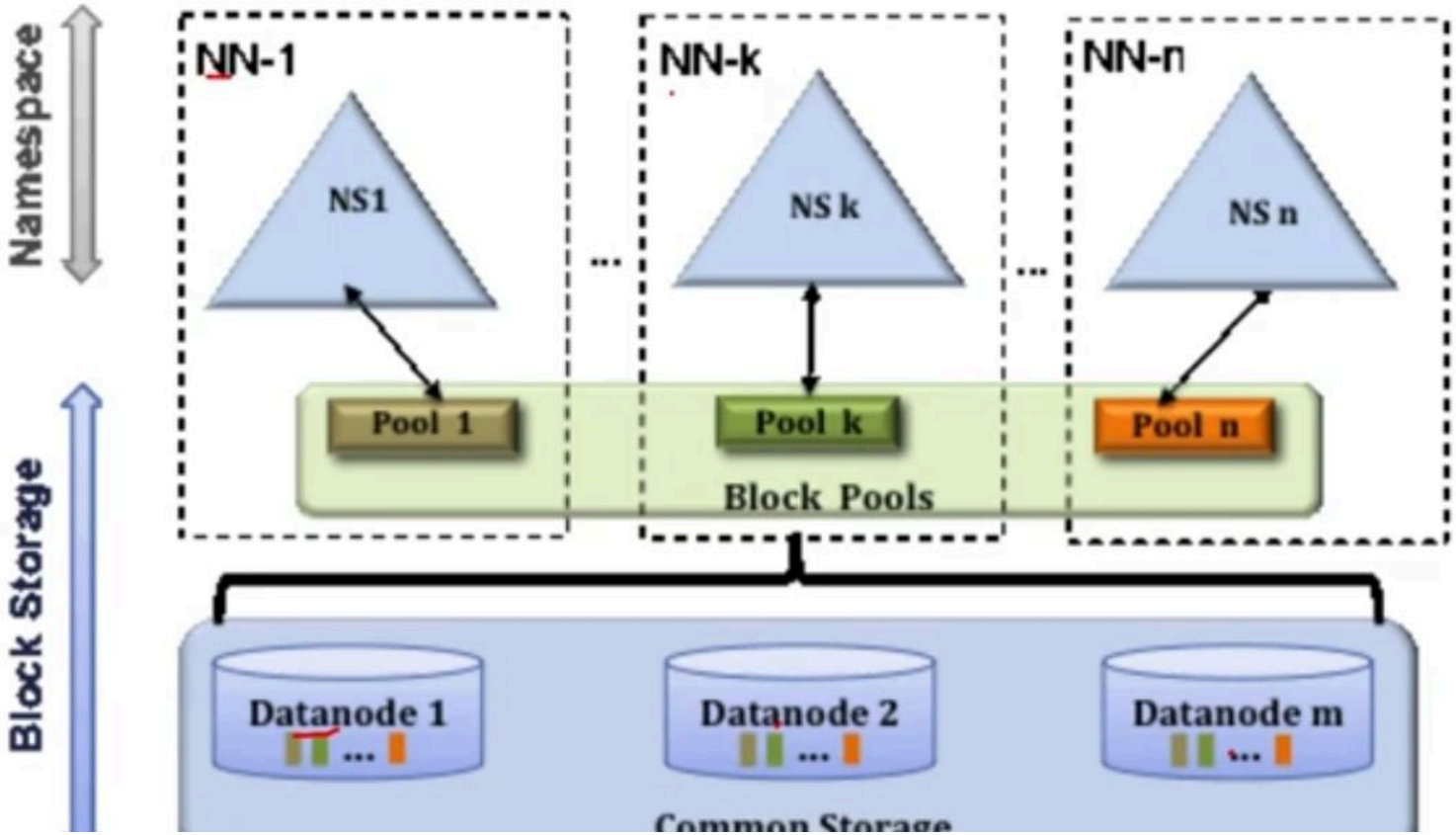
This architecture allows for only single NameNode to maintain the filesystem namespace.

HDFS Federation architecture

HDFS Federation feature introduced in Hadoop 2 enhances the existing HDFS architecture. It overcomes HDFS architecture limitations (discussed above) by adding multiple NameNode/namespaces support to HDFS. This allows the use of more than one NameNode/namespace. Therefore, it scales the namespace horizontally by allowing the addition of NameNode in the cluster.

- In HDFS Federation architecture, there are multiple NameNodes and DataNodes.
- Each NameNode has its own namespace and block pool.
- All the NameNodes uses DataNodes as the common storage.
- Each Datanode gets registered to all the NameNodes in the cluster and store blocks for all the block pools in the cluster.
- Also, DataNodes periodically send heartbeats and block reports to all the NameNode in the cluster and handles the instructions from the NameNodes.

HDFS Federation architecture



HDFS Federation architecture

- There are multiple NameNodes which are represented as NN1, NN2, ..NNn.
- The multiple namespaces managed by their respective NameNode.
- Each namespace has its own block pool.
- Each Datanode store blocks for all the block pools in the cluster. For example, DataNode1 stores the blocks from Pool 1, Pool 2, Pool3, etc.

Summary

HDFS federation feature added to Hadoop 2.x provides support for multiple NameNodes/namespaces. This overcomes the isolation, scalability, and performance limitations of the prior HDFS architecture.

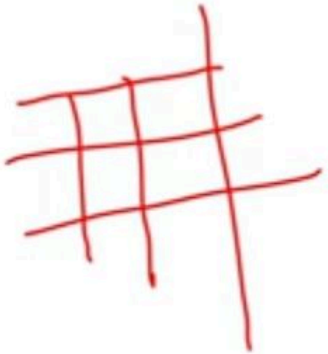
MRv2(mapResuce 2)

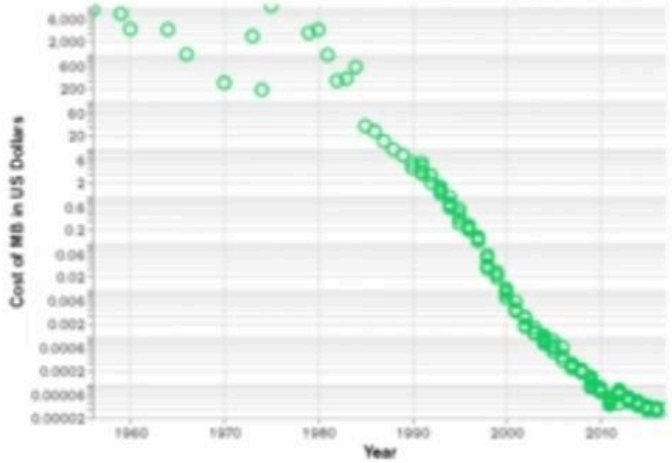
MRv1 uses the JobTracker to create and assign tasks to data nodes, which can become a resource bottleneck when the cluster scales out far enough (usually around 4,000 nodes).

MRv2 (aka YARN, "Yet Another Resource Negotiator") has a Resource Manager for each cluster, and each data node runs a Node Manager.
YARN overcame the scalability shortcomings by splitting the responsibilities of jobtracker into separate entities.
The jobtracker takes care of both job scheduling and task progress monitoring.
YARN separates these two roles into two independent daemons : a resource manager and an application master.
Resource manager is fixed and static.
It performs node management for free and busy nodes for allocating the resource for Map and Reduce phases.
Application manager communicates with the resource manager.

Introduction

NoSQL databases (aka "not only SQL") are non-tabular databases and store data differently than relational tables. NoSQL databases come in a variety of types based on their data model. The main types are document, key-value, wide-column, and graph. They provide flexible schemas and scale easily with large amounts of data and high user loads.
The data came in all shapes and sizes — structured, semi-structured, and polymorphic.





Introduction

SQL VS NoSQL Queries

NoSQL Query:

```
db.users.find(                          ← collection
   { age: { $gt: 18 } },                ← query criteria
   { name: 1, address: 1 }              ← projection
).limit(5)                              ← cursor modifier
```

SQL Query:

```
SELECT _id, name, address   ← projection
FROM   users                ← table
WHERE  age > 18             ← select criteria
LIMIT  5                    ← cursor modifier
```

Advantages and Disadvantages

Advantaegs:

High scalability:

- NoSQL database use sharding for horizontal scaling.
- Vertical scaling is not that easy to implement but horizontal scaling is easy to implement.
- Examples of horizontal scaling databases are MongoDB, Cassandra etc.
- NoSQL can handle huge amount of data because of scalability, as the data grows NoSQL scale itself to handle that data in efficient manner.

High availability:

- Auto replication feature in NoSQL databases makes it highly available because in case of any failure data replicates itself to the previous consistent state.

Advantages and Disadvantages

Disadvantaegs:

Narrow focus:

- It is mainly designed for storage but it provides very little functionality. Relational databases are a better choice in the field of Transaction Management than NoSQL.

Open-source:

- NoSQL is open-source database. There is no reliable standard for NoSQL yet.

GUI is not available

Large document size:

- Some database systems like MongoDB and CouchDB store data in JSON format. Which means that documents are quite large (BigData, network bandwidth, speed), and having descriptive key names actually hurts, since they increase the document size.

Types of NoSQL

- **Document databases** store data in documents similar to JSON (JavaScript Object Notation) objects. Each document contains pairs of fields and values. The values can typically be a variety of types including things like strings, numbers, booleans, arrays, or objects.
- **Key-value databases** are a simpler type of database where each item contains keys and values.
- **Wide-column stores** store data in tables, rows, and dynamic columns.
- **Graph databases** store data in nodes and edges. Nodes typically store information about people, places, and things, while edges store information about the relationships between the nodes.

RDBMS vs NoSQL databases

RDBMS Vs. NoSQL

RDBMS	NoSQL
• Structured and organized data	• No declarative query language
• Structured Query Language (SQL)	• No predefined schema
• Data and its relationships stored in separate tables.	• Key-Value pair storage, Column Store, Document Store, Graph Databases
• Data Manipulation Language, Data Definition Language	• Eventual consistency rather ACID property
• Tight Consistency	• Unstructured and unpredictable data
• BASE Transaction	• CAP Theorem
	• Prioritize high performance, high availability and scalability

NoSql and Relational Database Features

Feature	NoSQL Databases	Relational Databases
Performance	High	Low
Reliability	Poor	Good
Availability	Good	Good
Consistency	Poor	Good
Data Storage	Optimized for huge data	Medium sized to large
Scalability	High	High (but more expensive)

When should NoSQL be used?

- When huge amount of data need to be stored and retrieved .
- The relationship between the data you store is not that important.
- The data changing over time and is not structured.
- Support of Constraints and Joins is not required at database level
- The data is growing continuously and you need to scale the database regular to handle the data.