Raj Kumar Goel Institute of Technology Ghaziabad

BIG DATA

Unit:4

HDFS & Hadoop Environment

Course Details

B.Tech IT

Faculty Passport Size photo

Faculty Name

Ms. Himanshi

Faculty Information

Faculty Passport Size photo

Name: Ms. Himanshi
Designation :Assistant Professor
Branch :IT Department
Qualification : B.Tech , M.Tech

Evaluation Scheme

Subject	L	T	P	CT	TA	TOTAL	PS	TE	PE	TOTAL	CREDIT
Big Data (Department Elective – III)	3	0	0	30	20	50	100			150	3

SUBJECT SYLLABUS

Big Data(KCS-061)

Unit	Topic	Proposed Lectures
I	**Introduction to Big Data**: Types of digital data, history of Big Data innovation, introduction to Big Data platform, drivers for Big Data, Big Data architecture and characteristics, 5 Vs of Big Data, Big Data technology components, Big Data importance and applications, Big Data features – security, compliance, auditing and protection, Big Data privacy and ethics, Big Data Analytics, Challenges of conventional systems, intelligent data analysis, nature of data, analytic processes and tools, analysis vs reporting, modern data analytic tools.	06
II	**Hadoop:** History of Hadoop, Apache Hadoop, the Hadoop Distributed File System, components of Hadoop, data format, analyzing data with Hadoop, scaling out, Hadoop streaming, Hadoop pipes, Hadoop Echo System. **Map Reduce:** Map Reduce framework and basics, how Map Reduce works, developing a Map Reduce application, unit tests with MR unit, test data and local tests, anatomy of a Map Reduce job run, failures, job scheduling, shuffle and sort, task execution, Map Reduce types, input formats, output formats, Map Reduce features, Real-world Map Reduce	08
III	**HDFS (Hadoop Distributed File System):** Design of HDFS, HDFS concepts, benefits and challenges, file sizes, block sizes and block abstraction in HDFS, data replication, how does HDFS store, read, and write files, Java interfaces to HDFS, command line interface, Hadoop file system interfaces, data flow, data ingest with Flume and Scoop, Hadoop archives, Hadoop I/O: compression, serialization, Avro and file-based data structures. **Hadoop Environment:** Setting up a Hadoop cluster, cluster specification, cluster setup and installation, Hadoop configuration, security in Hadoop, administering Hadoop, HDFS monitoring & maintenance, Hadoop benchmarks, Hadoop in the cloud	08

SUBJECT SYLLABUS

Big Data(KCS-061)

IV	**Hadoop Eco System and YARN**: Hadoop ecosystem components, schedulers, fair and capacity, Hadoop 2.0 New Features - NameNode high availability, HDFS federation, MRv2, YARN, Running MRv1 in YARN. **NoSQL Databases:** Introduction to NoSQL **MongoDB:** Introduction, data types, creating, updating and deleing documents, querying, introduction to indexing, capped collections **Spark:** Installing spark, spark applications, jobs, stages and tasks, Resilient Distributed Databases, anatomy of a Spark job run, Spark on YARN **SCALA:** Introduction, classes and objects, basic types and operators, built-in control structures, functions and closures, inheritance.	**09**
V	**Hadoop Eco System Frameworks**: Applications on Big Data using Pig, Hive and HBase **Pig** - Introduction to PIG, Execution Modes of Pig, Comparison of Pig with Databases, Grunt, Pig Latin, User Defined Functions, Data Processing operators,	**09**
	Hive - Apache Hive architecture and installation, Hive shell, Hive services, Hive metastore, comparison with traditional databases, HiveQL, tables, querying data and user defined functions, sorting and aggregating, Map Reduce scripts, joins & subqueries. **HBase** – Hbase concepts, clients, example, Hbase vs RDBMS, advanced usage, schema design, advance indexing, Zookeeper – how it helps in monitoring a cluster, how to build applications with Zookeeper. IBM Big Data strategy, introduction to Infosphere, BigInsights and Big Sheets, introduction to Big SQL.	

Course Objective

BIG DATA

KCS-061

Students will try to learn:

1. To provide an overview of an exciting growing field of big data analytics.

2. To introduce the tools required to manage and analyze big data like Hadoop, NoSql MapReduce.

3. To teach the fundamental techniques and principles in achieving big data analytics with scalability and streaming capability.

4. To enable students to have skills that will help them to solve complex real-world problems in for decision support

Course Outcome

CO 1	Demonstrate knowledge of Big Data Analytics concepts and its applications in business.
CO 2	Demonstrate functions and components of Map Reduce Framework and HDFS.
CO 3	Discuss Data Management concepts in NoSQL environment.
CO 4	Explain process of developing Map Reduce based distributed processing applications.
CO 5	Explain process of developing applications using HBASE, Hive, Pig etc.

Program Outcomes

RAJ KUMAR GOEL INSTITUTE OF TECHNOLOGY, GHAZIABAD

Department of Computer Science & Engineering

Programme Outcomes (PO)

Engineering Graduates will be able to:

PO1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4 Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including

Program Outcomes

prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

CO-PO Mapping

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
C316.1	3	1	1	2						2	2	1	1	3
C316.2	3	2	1	1	2					2	2	1	2	3
C316.3	2	3	2	2	2					3	2	1	2	3
C316.4	3	2	1	2	2					2	2	1	1	3
C316.5	2	3	3	1	2					3	3	3	3	3

End Semester

PAPER ID 420818

Printed Page: 1 of 1
Subject Code: KCS061

Roll No: | | | | | | | | | | | | | |

BTECH
(SEM VI) THEORY EXAMINATION 2021-22
BIG DATA

Time: 3 Hours *Total Marks: 100*

Note: Attempt all Sections. If you require any missing data, then choose suitably.

SECTION A

1. Attempt *all* questions in brief. **2*10 = 20**

Qno	Questions	CO
(a)	List any five Big Data platforms.	1
(b)	Write any two industry examples for Big Data.	1
(c)	What is the role of Sort & Shuffle in Map-Reduce?	2
(d)	Give the full form of HDFS.	2
(e)	What is the block size of a HDFS?	3
(f)	Name the two type of nodes in Hadoop.	3
(g)	Compare and Contrast No SQL Relational Databases.	4
(h)	Does MongoDB support ACID properties? Justify your answer.	4
(i)	Describe schema.	5
(j)	Discuss the different types of data that can be handled with HIVE.	5

SECTION B

2. Attempt any *three* of the following: **10*3 = 30**

Qno	Questions	CO
(a)	Detail about the three dimensions of BIG data.	1
(b)	Illustrate the architecture of Map Reduce.	2
(c)	Examine how a client read and write data in HDFS.	3
(d)	With the help of suitable example, explain how CRUD operations are performed in MongoDB.	4
(e)	Differentiate between Map-Reduce, PIG and HIVE	5

SECTION C

3. Attempt any *one* part of the following: **10*1 = 10**

(a)	Discuss in detail the different forms of BIG data.	1
(b)	Elaborate various components of Big Data architecture.	1

4. Attempt any *one* part of the following: **10 *1 = 10**

(a)	Explain the detailed architecture of Map-Reduce	2
(b)	Differentiate "Scale up and Scale out" Explain with an example How Hadoop uses Scale out feature to improve the Performance.	2

5. Attempt any *one* part of the following: **10*1 = 10**

(a)	Demonstrate the design of HDFS and concept in detail.	3
(b)	Write the benefits and challenges of HDFS	3

6. Attempt any *one* part of the following: **10*1 = 10**

(a)	Classify and detail the different types of NoSQL	4
(b)	Summarize the role of indexing in MongoDB using an example.	4

7. Attempt any *one* part of the following: **10*1 = 10**

(a)	Explore various execution models of PIG.	5
(b)	Design and explain the detailed architecture of HIVE.	5

Topics to be covered

Hadoop Ecosystem and YARN

Hadoop ecosystem components NameNode high availability HDFS federation MRv2

NoSQL Database

Introduction

Advantages and Disadvantages Types of NoSQL

RDBMS vs NoSQL databases

NoSql and Relational Database

MongoDB

Introduction Data Types

SQL vs MongoDB

Create, Read Update and Delete Documents Querying

Indexing

Capped collections

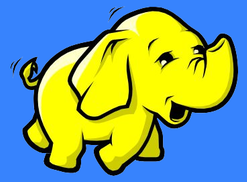
Topics to be covered…

Spark

Introduction
Installing spark
Spark
applications
Jobs, stages and tasks
Resilient Distributed
Database Anatomy of a
Spark job run Spark on
YARN

SCALA

Introduction
Classes and
Objects
Basic types and
Operators Built-in
control structures
Functions and closures
Inheritance



Hadoop Ecosystem and YARN

Hadoop ecosystem components NameNode high availability HDFS federation MRv2

Hadoop ecosystem components

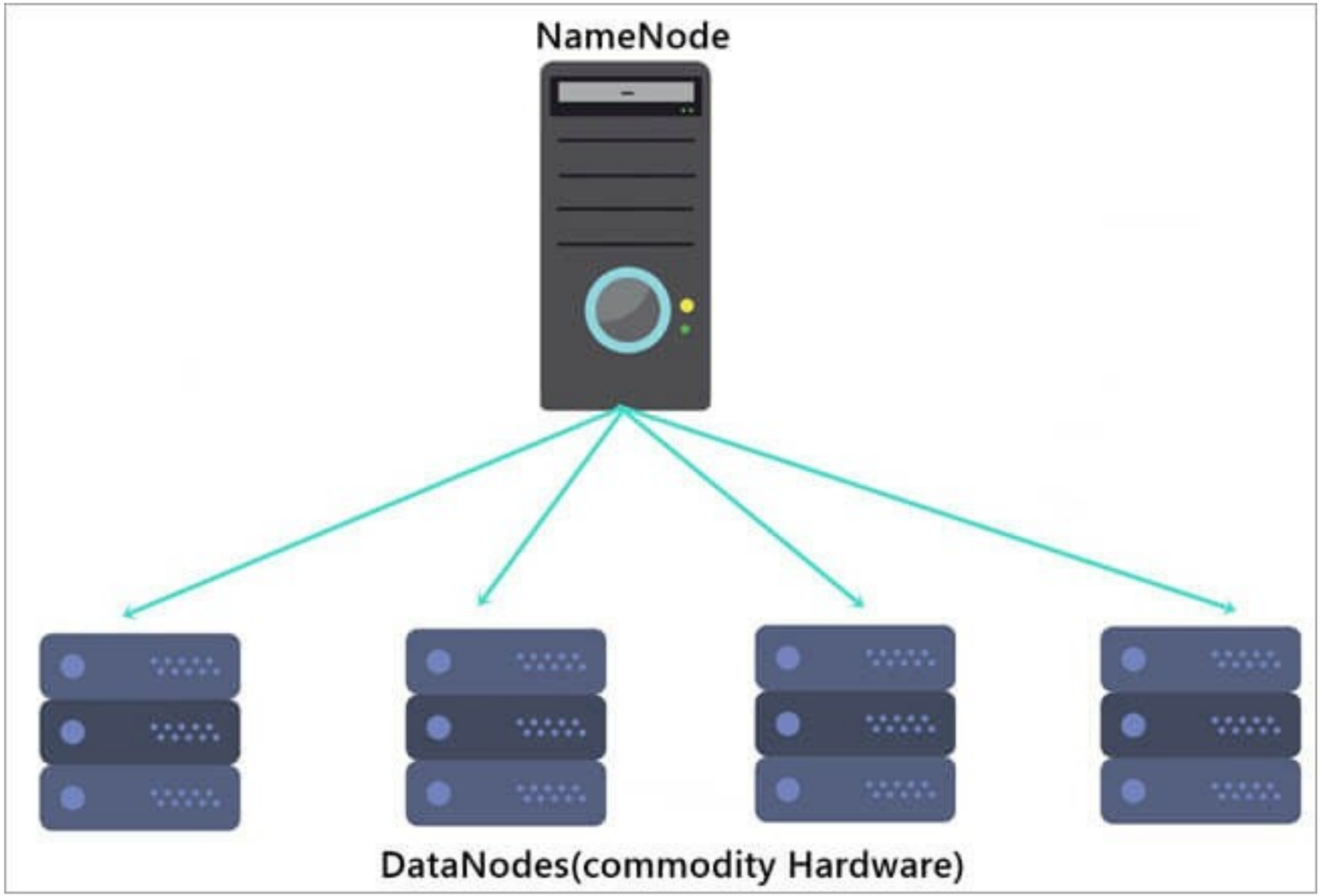
- HDFS: Hadoop Distributed File System
- YARN: Yet Another Resource Negotiator
- MapReduce: Programming based Data Processing
- Spark: In-Memory data processing
- PIG, HIVE: Query-based processing of data services
- HBase: NoSQL Database
- Mahout, Spark MLLib: Machine Learning algorithm
- libraries Solar,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

NameNode high availability

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



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.

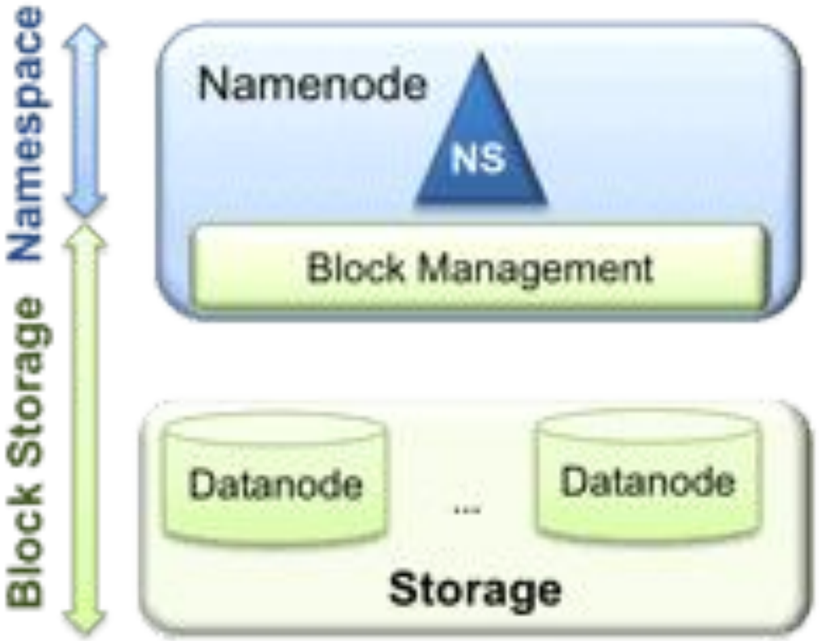
Issues in maintaining consistency in the HDFS High Availability cluster are as follows:

- 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 federation

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.

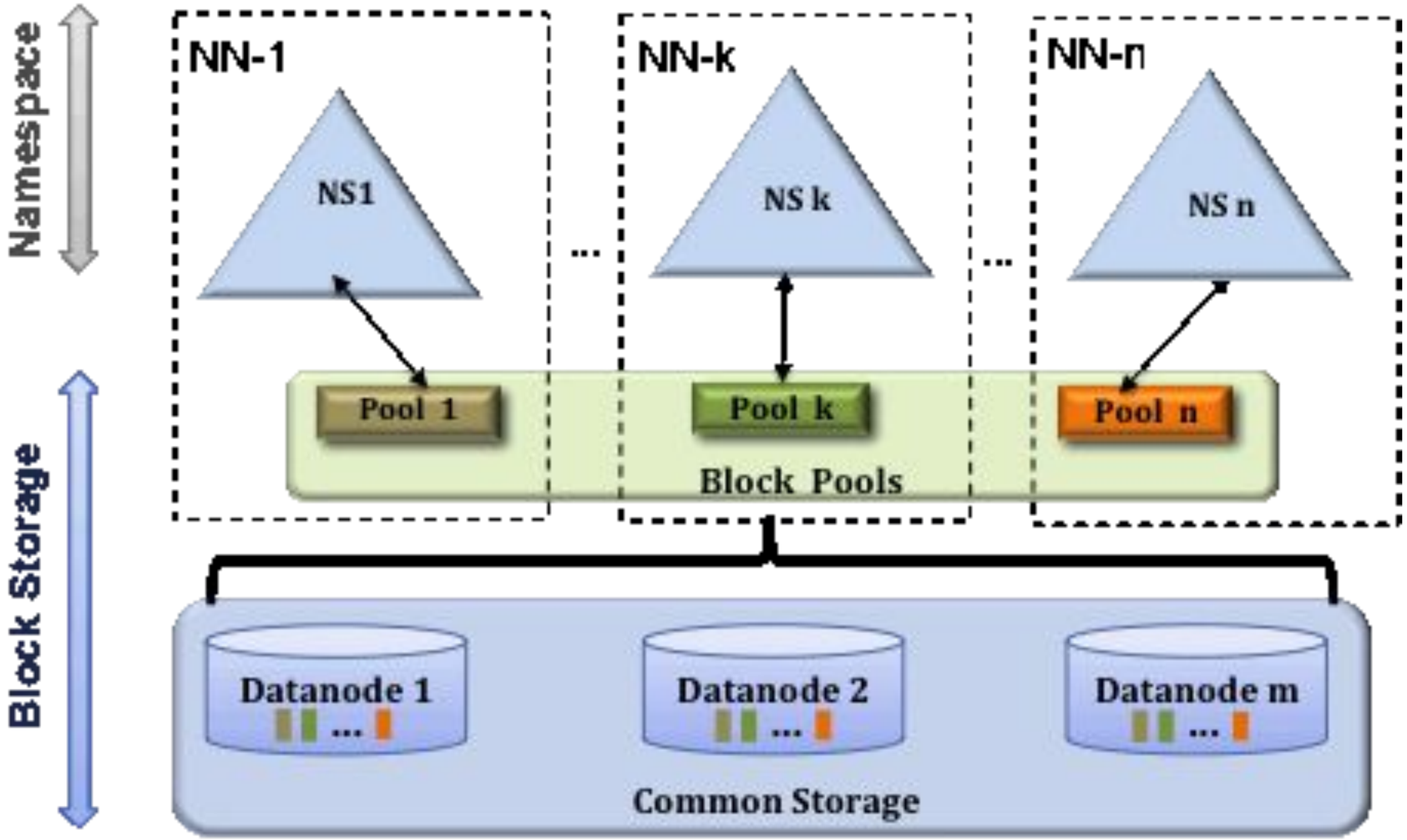
This architecture allows for only single NameNode to maintain the filesystem namespace.

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.

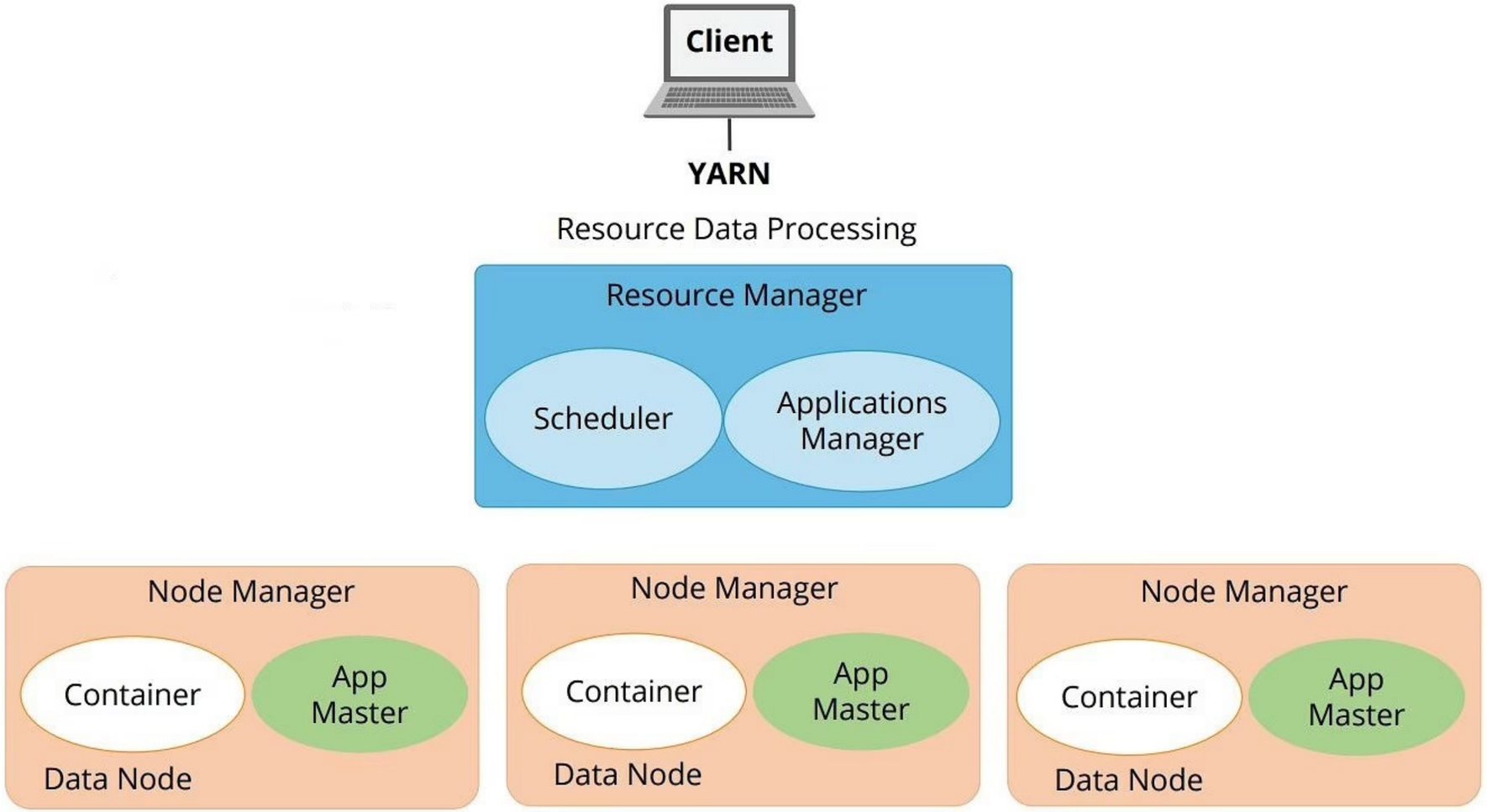
MRv2

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.

YARN


Client
YARN
Resource Data Processing
Resource Manager
Scheduler
Applications Manager
Node Manager
Container
App Master
Data Node
Node Manager
Container
App Master
Data Node
Node Manager
Container
App Master
Data Node

NoSQL Database

Introduction

Advantages and Disadvantages

Types of NoSQL

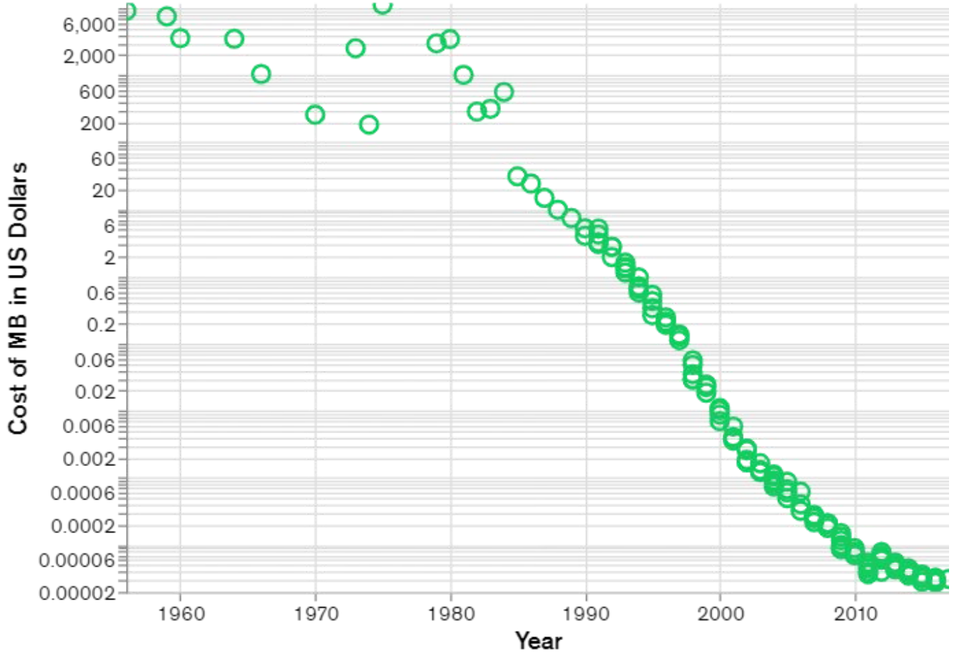
RDBMS vs NoSQL databases NoSql and Relational Database Features

When should NoSQL be used?

Introduction

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

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

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 databases

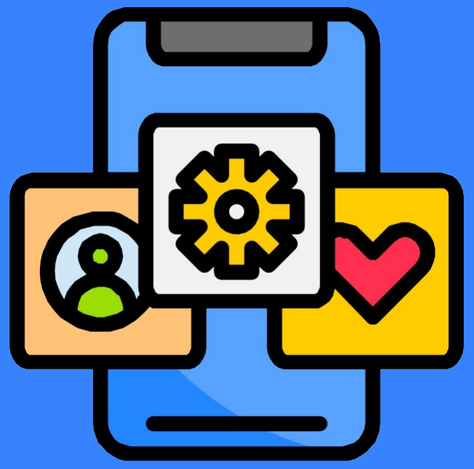
RDBMS Vs. NoSQL

RDBMS

- Structured and organized data
- Structured Query Language (SQL)
- Data and its relationships stored in separate tables.
- Data Manipulation Language, Data Definition Language
- Tight Consistency
- BASE Transaction

NoSQL

- No declarative query language
- No predefined schema
- Key-Value pair storage, Column Store, Document Store, Graph Databases
- Eventual consistency rather ACID property
- Unstructured and unpredictable data
- CAP Theorem
- Prioritize high performance, high availability and scalability



NoSql and Relational Database Features

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 should NoSQL be used?

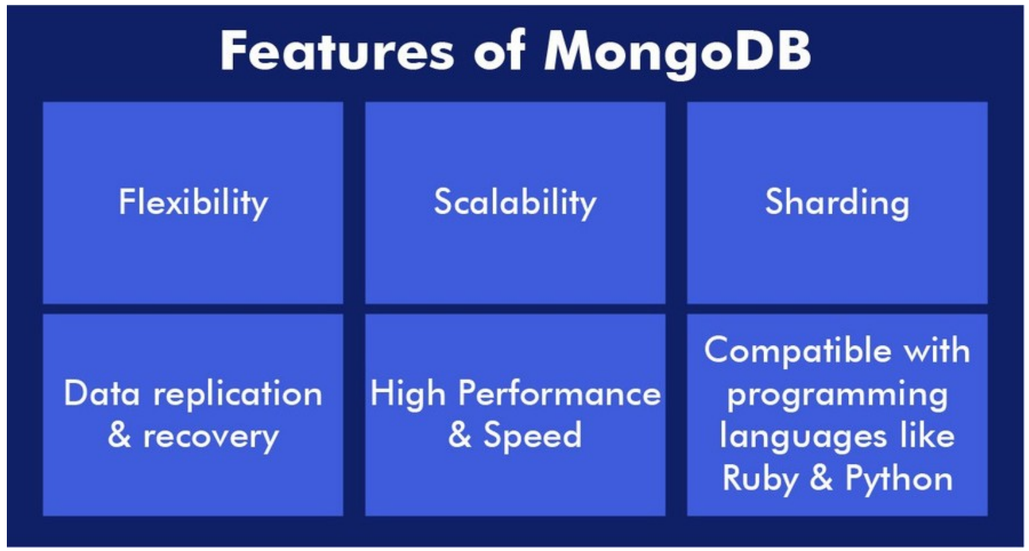
- When huge amount of data need to be stored and retrieved .
- The relationship between the data you store is not that
- important. The data changing over time and is not structured.
- Support of Constraints and Joins is not required at database level
- The data is growing continuously and you need to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

mongoDB

Introductio
n Data
Types
Create, Update and Delete
Documents
Querying
Indexing
Capped collectons

Introduction

- MongoDB is a document-oriented NoSQL database used for high-volume data storage.
- MongoDB is written in C++.
- It uses JSON-like documents with optional schema instead of using tables and rows in traditional relational databases.
- Documents containing key-value pairs are the basic units of data in MongoDB.
- This allows developers to focus on programming the application rather than scaling
- it. MongoDB provides high performance, high availability and automatic scaling.



Data Types

Data Types

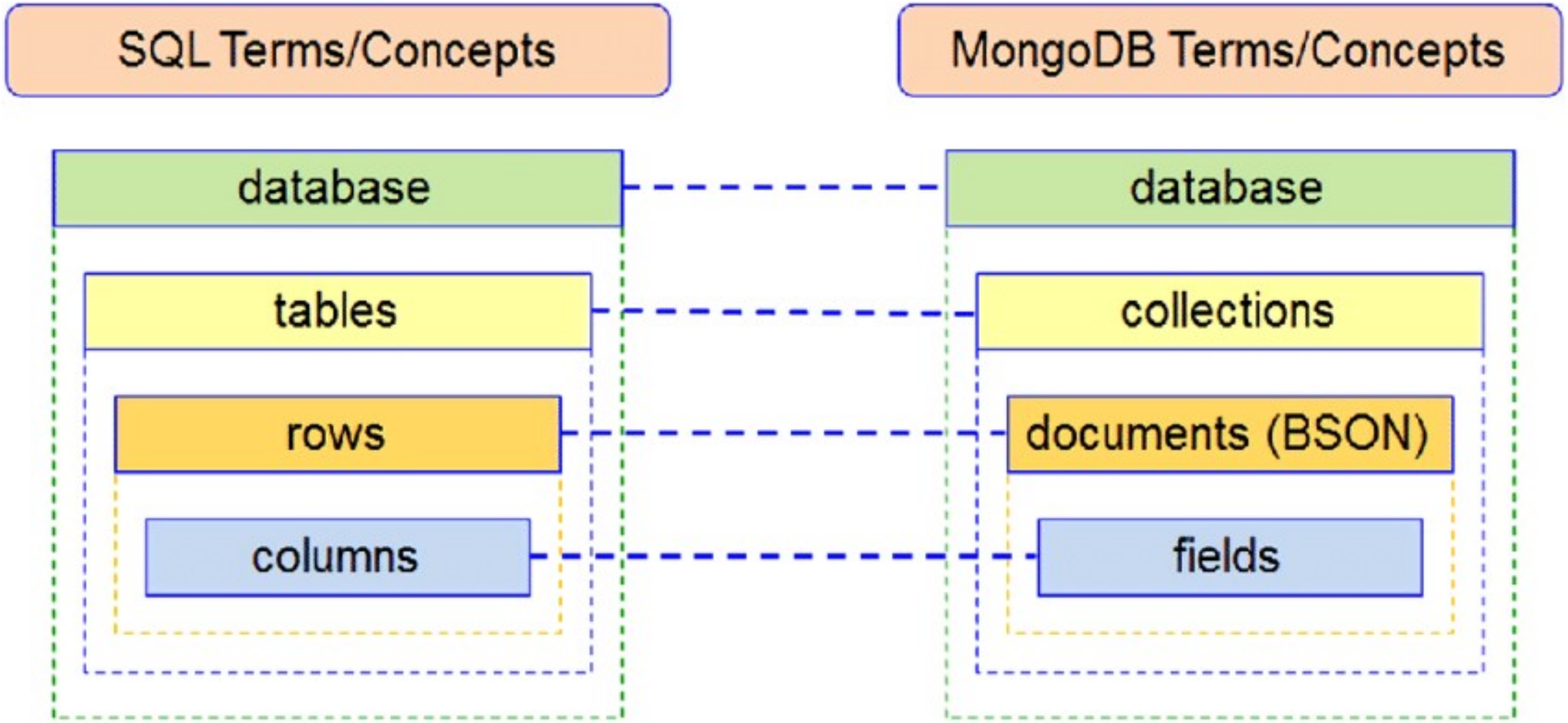
- **String:** String is the most commonly used datatype. It is used to store data.
- **Integer:** Integer is used to store the numeric value. It can be 32 bit or 64 bit depending on the server you are using.
- **Boolean:** True/False.
- **Double:** Stores floating point values.
- Min/Max Keys: This datatype compare a value against the lowest and highest bson elements.
- **Arrays:** This datatype is used to store a list or multiple values into a single key.
- **Object:** Object datatype is used for embedded documents.
- **Null:** It is used to store null values.
- **Date:** This datatype stores the current date or time in unix time format.

Data Types

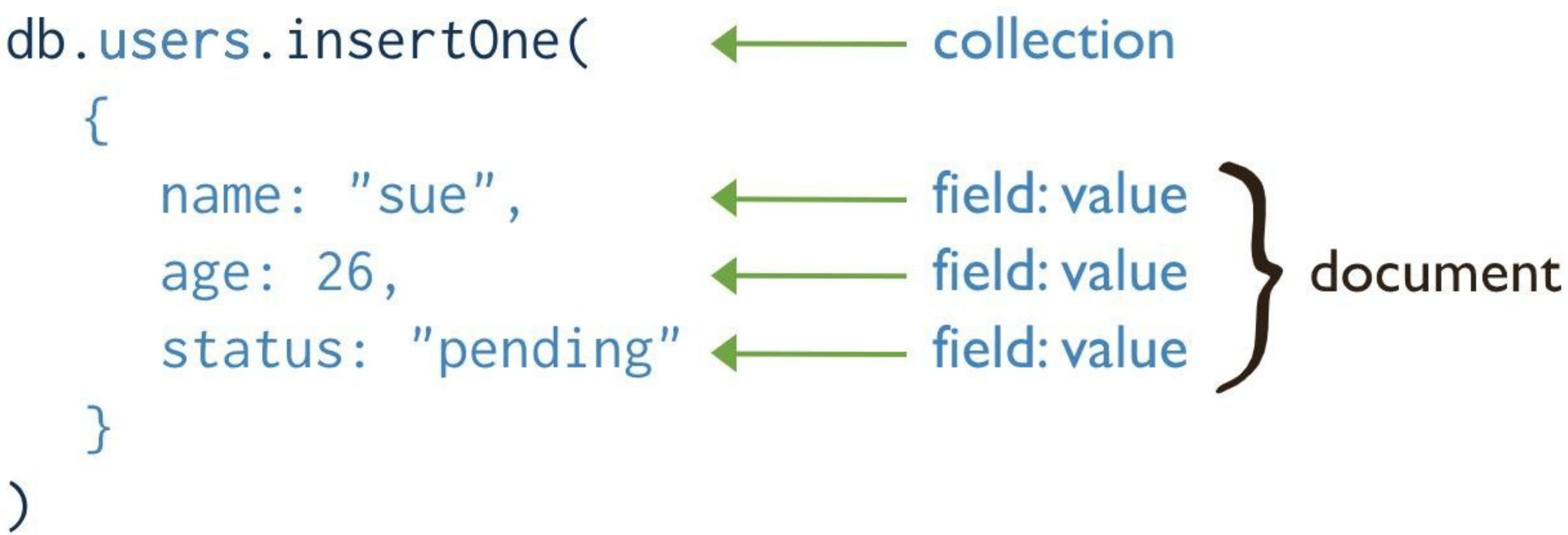
```
1    _id : ObjectId("5a09e59efc1f462097f46536 ")        ObjectId
2    item : "canvas "                                   String
3    qty : 100                                          Int32
4  ⌄ tags : Array                                       Array
5      0 : "cotton "                                    String
6  ⌄ size : Object                                      Object
7      h : 28                                           Int32
8      w : 35.5                                         Double
9      uom : "cm "                                      String
```

SQL vs MongoDB

SQL vs MongoDB



MongoDB Documents



Create, Read, Update and Delete Documents

Create Documents

```
db.collection.insertOne(
  <document>,
  {
    writeConcern: <document>
  }
)
```

```
db.collection.insertMany(
  [ <document 1> , <document 2>,
  ... ],
  {
    writeConcern:
    <document>,  ordered:
    <boolean>
  }
)
```

```
use("test");

db.sales.insertOne(
{ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" : new
Date("2014-03-01T08:00:00Z")}
);
```

db.collection.insertOne(): Inserts a single document into a collection.

db.collection.insertMany(): Inserts multiple documents into a collection.

Read Documents

Read One Document:

```
db.collection.findOn
e(
  { <query> },
  { <projection> }
)
```

```
db.sales.findOne
        (
    { "_id" : 1 },
    { "_id" : 0 }
)
;
db.sales.find(
    { "item" : "abc"
    },
    { "price" : 1 }
);
```

Read Many Documents:

```
db.collection.find(
  { <query> },
  { <projection> }
)
```

Update Documents

updateOne, updateMany and replaceOne

updateOne and updateMany each take a filter documents as their first parameter and a modifier document as the second parameter.
replaceOne also takes a filter as the first parameter, but as the second parameter replaceOne expects a document with which it will replace the document matching the filter.

```
db.RecordsDB.updateOne({name: "Marsh"}, {$set:{ownerAddress: "451 W. Coffee St. A204"}})
db.RecordsDB.updateMany({species:"Dog"}, {$set: {age: "5"}})

{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
> db.RecordsDB.find()
{ "_id" : ObjectId("5fd98ea9ce6e8850d88270b5"), "name" : "Kitana", "age" : "4 years", "species" :  "Cat", "ownerAddress" : "521 E. Cortland", "chipped" : true }
{ "_id" : ObjectId("5fd993a2ce6e8850d88270b7"), "name" : "Marsh", "age" : "5", "species" : "Dog",
"ownerAddress" : "451 W. Coffee St. A204", "chipped" : true }
{ "_id" : ObjectId("5fd993f3ce6e8850d88270b8"), "name" : "Loo", "age" : "5", "species" :
```

Delete Documents

MongoDB has two different methods of deleting records from a collection:

- db.collection.deleteOne()
- db.collection.deleteMany()

deleteOne():

```
db.RecordsDB.deleteOne({name:"Maki"})
```

deleteMany():

```
db.RecordsDB.deleteMany({species:"Dog"}
)
```

Queryin
g

Querying

find() Method
To query data from MongoDB collection, you need to use MongoDB's find() method.
find() method will display all the documents in a non-structured way.

pretty() Method
To display the results in a formatted way, you can use pretty() method.

findOne() method
Apart from the find() method, there is findOne() method, that returns only one document.

AND in MongoDB
OR in MongoDB
NOR in MongoDB
NOT in MongoDB

Indexing

Indexing

- A database index is similar to a book's index.
- A query that does not use an index is called a collection scan, which means that the server has to look through whole database to find a query's results.
- Avoid collection scans because the process is very slow for large
- collections. To create an index, we use createIndex collection method.

MongoDB supports indexes

- At the collection level
- Similar to indexes on RDBMS

Can be used for

- More efficient
- filtering More
- efficient sorting
  Index-only queries (covering index)

Capped collectons

Capped collectons

We can create collections in mongoDb on which we can apply size limit. These special type of collections are called Capped Collections. These are a kind of circular queues, in which if allocated size limit is reached, it makes space for new documents by overwriting the oldest documents in the collection.

How to check if the collection is capped or not?
We can check whether the collection is capped or not with the isCapped() method in MongoDB. This method returns true is the specified collections capped collection. Otherwise, return, false.

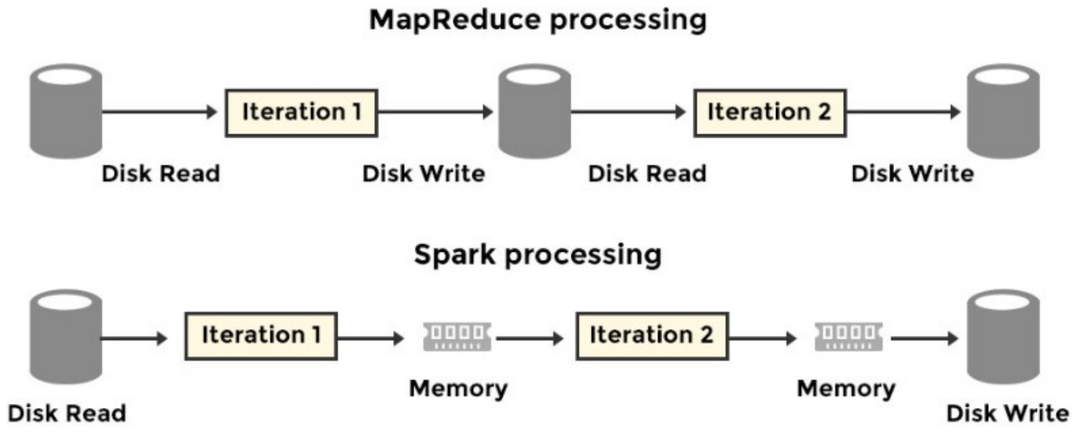
Syntax:
db.Collection_name.isCapped()

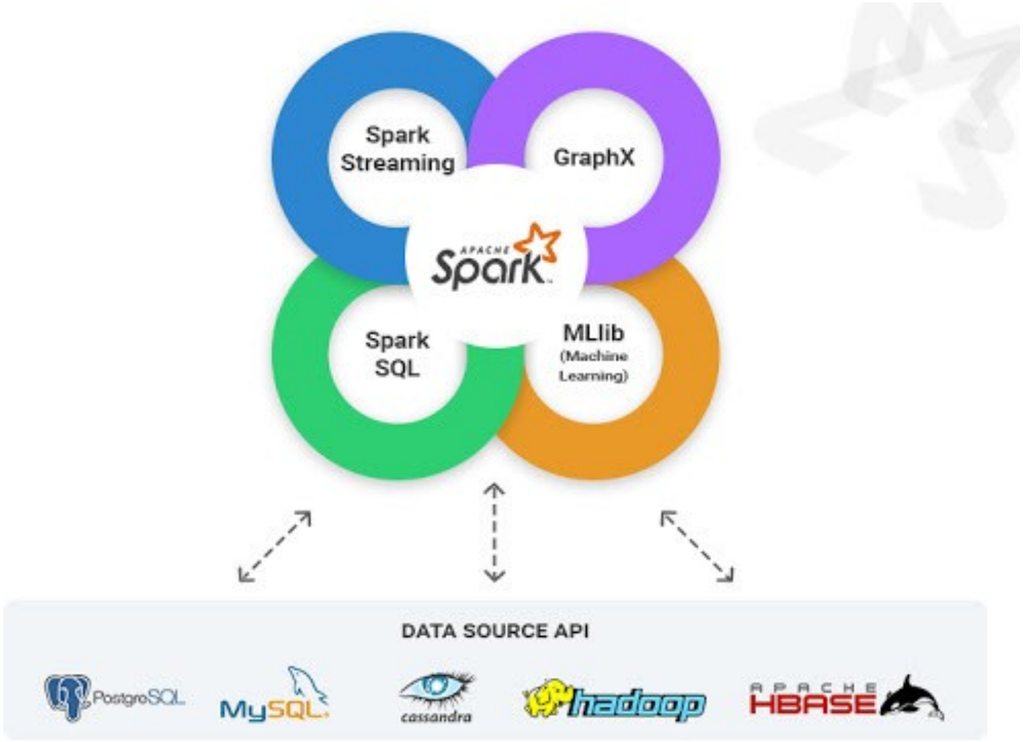
Example:
db.student.isCapped()


APACHE
Spark
TM

Introduction

Apache Spark is an open-source, distributed processing system used for big data workloads. It utilizes in-memory caching and optimized query execution for fast queries against data of any size. Simply put, Spark is a fast and general engine for large-scale data processing.





Installing spark

Step 1: Verifying Java Installation

- Java installation is one of the mandatory things in installing
- Spark.
  $java -version

Step 2: Verifying Scala Installation

- You should Scala language to implement Spark.
  $scala -version

Step 3: Downloading Apache Spark

- Download the latest version of Spark.
  Find the Spark tar file in the download folder.

Step 4: Installing Spark

- Extracting Spark tar - $ tar xvf
  spark-1.3.1-bin-hadoop2.6.tgz Setting up the environment
  for Spark.
-

Step 4: Verifying the Spark Installation

$spark-shell

Spark applications

Machine Learning:

- Apache Spark is equipped with a scalable Machine Learning Library called MLlib that can perform advanced analytics such as clustering

Fog Computing:

- Fog computing, also called fog networking or fogging, describes a decentralized computing structure located between the cloud and devices that produce data.

Processing Streaming Data:

- Stream processing is the processing of data in motion, or in other words, computing on data directly as it is produced or received. The majority of data are born as continuous streams: sensor events, user activity on a website, financial trades, and so on

most popular companies that are utilizing various applications of Apache Spark:

- **Uber:** Uber uses Kafka, Spark Streaming, and HDFS for building a continuous ETL
- pipeline. **Pinterest:** One of the successful web and mobile application companies, Pinterest uses Spark Streaming in order to gain deep insight into customer engagement details.

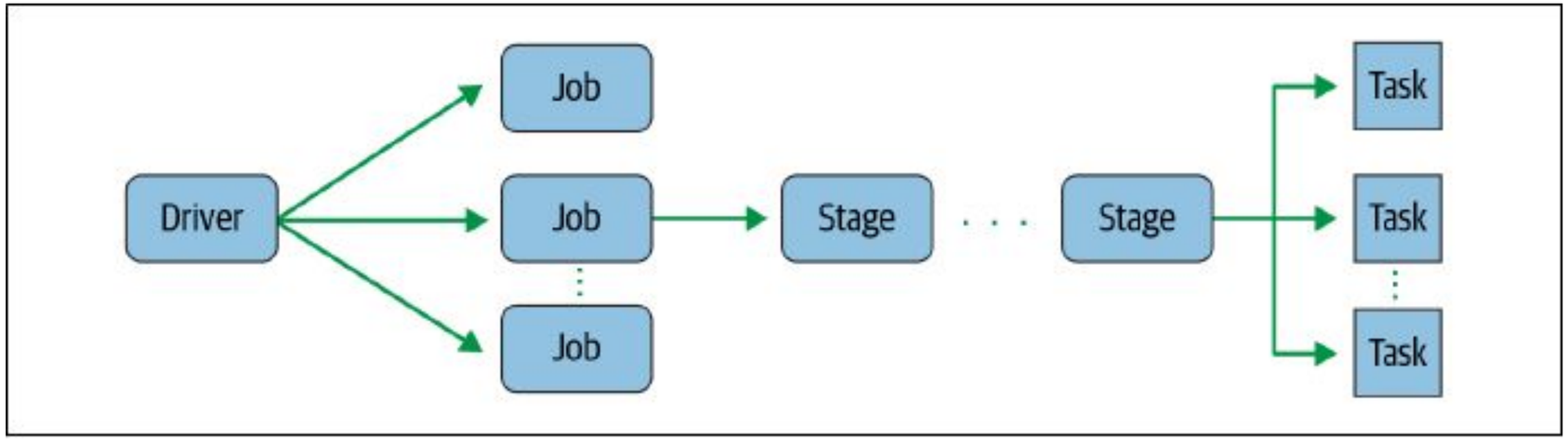
Jobs, stages and tasks

Jobs, stages and tasks

Job - A parallel computation consisting of multiple tasks that get spawned in response to a Spark action (e.g., save(), collect()).

Stage - Each job gets divided into smaller sets of tasks called stages that depend on each other. As part of the DAG nodes, stages are created based on what operations can be performed serially or in parallel. Not all Spark operations can happen in a single stage, so they may be divided into multiple stages.

Task - A single unit of work or execution that will be sent to a Spark executor.



Resilient Distributed Database (RDD)

Resilient Distributed Database(RDD)

- The fundamental abstraction in Spark is the RDD, short for Resilient Distributed
- Dataset. It is a read-only (immutable) collection of objects or records, partitioned across the cluster that can be operated on in parallel.
- A partition can be reconstructed if the hosting node experiences failure.
- RDDs are a lower-level API; the other two Spark data abstractions namely DataFrames and Datasets compile to an RDD.
- The constituent records or objects within an RDD are Java, Python, or Scala objects. Anything can be stored in any format in these objects.

Resilient: means an RDD is fault-tolerant and able to recompute missing or damaged partitions due to node failures.

Distributed: means data making up an RDD is spread across a cluster of machines.

Datasets: refer to representations of the data records we work with. External Data can be loaded using a variety of sources such as JSON file, CSV file, text file or database via JDBC.

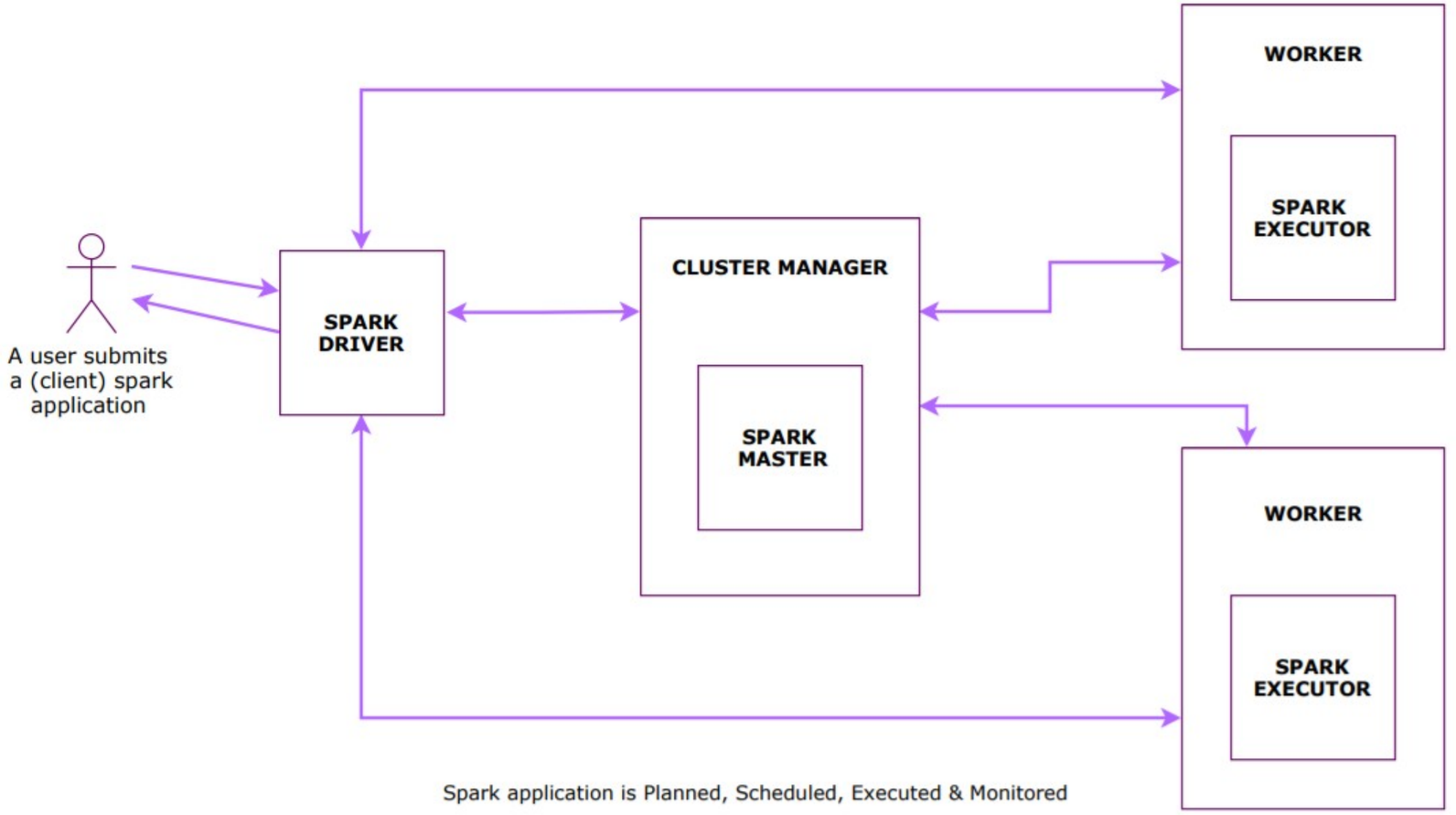
Anatomy of a Spark job run

Spark application contains several components, all of which exists whether you are running Spark on a single machine or across a cluster of hundreds or thousands of nodes. The components of the spark application are Driver, the Master, the Cluster Manager and the Executors.

All of the spark components including the driver, master, executor processes run in java virtual machines(JVMs). A JVM is a cross-platform runtime engine that executes the instructions compiled into java bytecode. Scala, which spark is written in, compiles into bytecode and runs on JVMs.

- **Spark Driver:**
  - The life of Spark programs starts and ends with the Spark Driver. The Spark driver is the process which the clients used to submit the spark program.
- **Spark Executors:**
  - Spark executors are the host processes on which the tasks from Spark DAG runs. Executors reserve CPU and Memory resources on slave nodes or workers in slave node Executors are dedicated to specific spark application and terminated when the application completes. Spark executors can run hundreds of tasks within a spark

Anatomy of a Spark job run

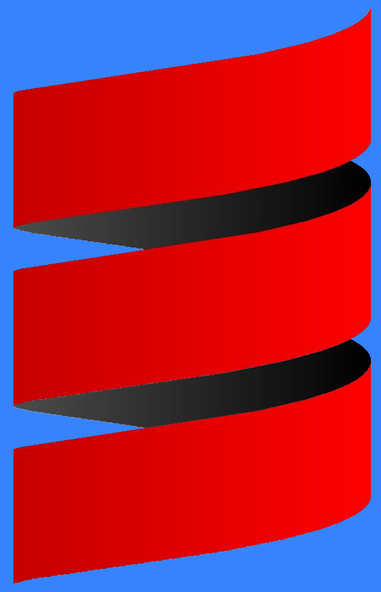


Anatomy of a Spark job run

Spark on YARN

Spark on YARN

- Apache Spark is an in-memory distributed data processing engine and YARN is a cluster management technology.
- **Spark provides two modes:** YARN client mode and YARN cluster mode.
- YARN client mode, the driver runs in the client.
- Client mode is also useful when building Spark programs, since any debugging output is immediately visible.
- In YARN cluster mode, the driver runs on the cluster in the YARN application master. YARN cluster mode is appropriate for production jobs.
- In YARN cluster mode, the entire application runs on the cluster.
- In YARN cluster mode, YARN will also retry the application if the application master fails.



SCALA

Introduction

Scala is a strong statically typed general-purpose programming language which supports both object-oriented programming and functional programming.

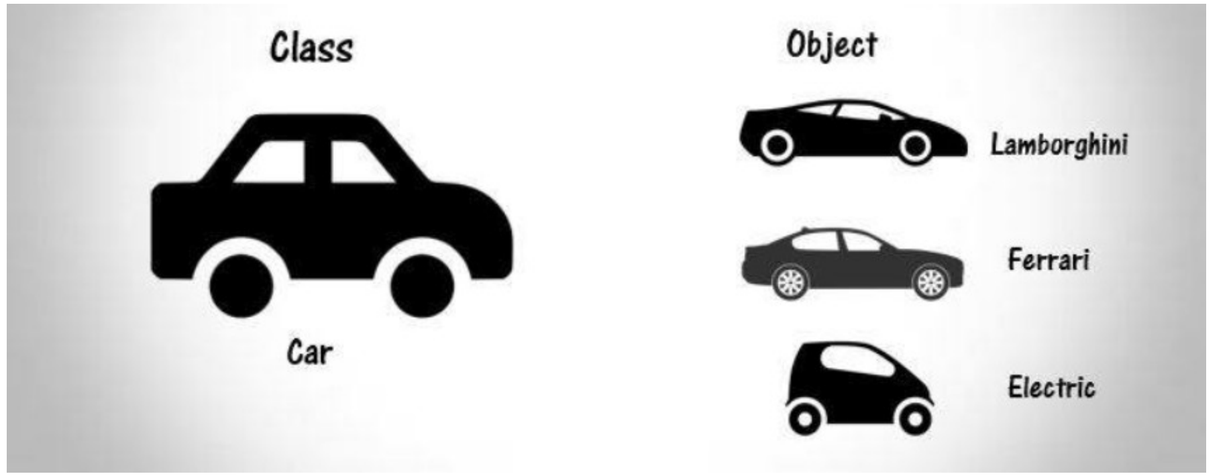
Scala is used in Data processing, distributed computing, and web development. It powers the data engineering infrastructure of many companies.

Classes and Objects

Classes and Objects

A class is a blueprint for objects. Once you define a class, you can create objects from the class blueprint with the keyword new. Through the object, you can use all functionalities of the defined class.

```
class Point(xc: Int, yc:
  Int) {  var x: Int = xc
  var y: Int = yc

  def move(dx: Int, dy:
     Int) {  x = x + dx
     y = y + dy
     println ("Point x location : " +
     x);  println ("Point y location :
     " + y);
  }
```



Classes and Objects

Singleton Objects:

Scala is more object-oriented than Java because, in Scala, we cannot have static members. Instead, Scala has singleton objects. A singleton is a class that can have only one instance, i.e., Object. You create a singleton using the keyword object instead of the class keyword.

```
// Singleton object with named
// as Exampleofsingleton
object Exampleofsingleton
{

    // Variables of singleton object
    var str1 = "Welcome ! EIOV";
    var str2 = "This is Scala language tutorial";

    // Method of singleton object
    def display()
    {
        println(str1);
        println(str2);
    }
}
```

Basic types and Operators

Basic types and Operators

Sr. No.	Data Type	Description
1	Byte	8 bit signed value. Range from -128 to 127
2	Short	16 bit signed value. Range -32768 to 32767
3	Int	32 bit signed value. Range -2147483648 to 2147483647
4	Long	64 bit signed value. -9223372036854775808 to 9223372036854775807
5	Float	32 bit IEEE 754 single-precision float
6	Double	64 bit IEEE 754 double-precision float
7	Char	16 bit unsigned Unicode character. Range from U+0000 to U+FFFF
8	String	A sequence of Chars
9	Boolean	Either the literal true or the literal false
10	Unit	Corresponds to no value
11	Null	null or empty reference
12	Nothing	The subtype of every other type; includes no values
13	Any	The supertype of any type; any object is of type Any
14	AnyRef	The supertype of any reference type

Operators:

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. Scala is rich in built-in operators and provides the following types of operators –

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators

Operators

Assignment Operators

There are the following assignment operators supported by Scala language –

Operator	Description
=	Simple assignment operator, Assigns values from right side operands to left side operand
+=	Add AND assignment operator, It adds the right operand to the left operand and assigns the result to the left operand
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assigns the result to left operand
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assigns the result to the left operand
/=	Divide AND assignment operator, It divides left operand with the right operand and assigns the result to left operand
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to the left operand
<<=	Left shift AND assignment operator
>>=	Right shift AND assignment operator
&=	Bitwise AND assignment operator
^=	bitwise exclusive OR and assignment operator
|=	bitwise inclusive OR and assignment operator

→ Relational Operators

The following relational operators are supported by the Scala language

Operator	Description
==	Checks if the values of two operands are equal or not, if yes then the condition becomes true.
!=	Checks if the values of two operands are equal or not, if values are not equal then the condition becomes true.
>	Checks if the value of the left operand is greater than the value of the right operand, if yes then the condition becomes true.
<	Checks if the value of the left operand is less than the value of the right operand, if yes then the condition becomes true.
>=	Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes then the condition becomes true.
<=	Checks if the value of the left operand is less than or equal to the value of the right operand, if yes then the condition becomes true.

Operators

→ Logical Operators

The following logical operators are supported by the Scala language.

Operator	Description
&&	It is called Logical AND operator. If both the operands are non zero then the condition becomes true.
||	It is called Logical OR Operator. If any of the two operands is non zero then the condition becomes true.
!	It is called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then the Logical NOT operator will make it false.

→ Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. The truth tables for &, |, and ^ are as follows –

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

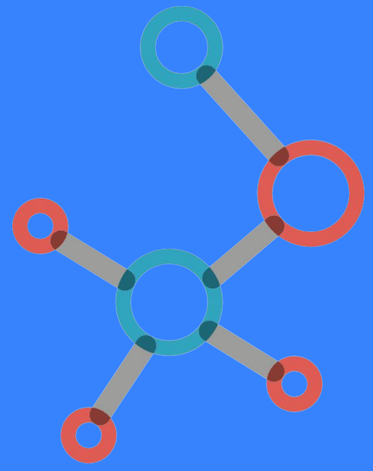
Operator	Description
&	Binary AND Operator copies a bit to the result if it exists in both operands.
|	Binary OR Operator copies a bit if it exists in either operand.
^	Binary XOR Operator copies the bit if it is set in one operand but not both.
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.
<<	Binary Left Shift Operator. The bit positions of the value of the left operand are moved left by the number of bits specified by the right operand.
>>	Binary Right Shift Operator. The bit positions of the left operand value are moved right by the number of bits specified by the right operand.
>>>	Shift right zero-fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.

Operators

Assignment Operators

There are the following assignment operators supported by Scala language –

Operator	Description
=	Simple assignment operator, Assigns values from right side operands to left side operand
+=	Add AND assignment operator, It adds the right operand to the left operand and assigns the result to the left operand
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assigns the result to left operand
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assigns the result to the left operand
/=	Divide AND assignment operator, It divides left operand with the right operand and assigns the result to left operand
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to the left operand
<<=	Left shift AND assignment operator
>>=	Right shift AND assignment operator
&=	Bitwise AND assignment operator
^=	bitwise exclusive OR and assignment operator
|=	bitwise inclusive OR and assignment operator



Built-in control structures

Built-in control structures

- **If expressions**
- **While loops**
- **For**
- **expressions**
- **Exception handling with try**

  expressions Match expressions

- **If expressions:**

  Scala's if works just like in many other languages. It tests a condition and then executes one of two code branches depending on whether the condition holds true. Here is a common example, written in an imperative style:

```
var filename =
  "default.txt" if
  (!args.isEmpty)
    filename = args(0)
```

Built-in control structures

While loops:

Scala's while loop behaves as in other languages. It has a condition and a body, and the body is executed over and over as long as the condition holds true.

example:

```
def gcdLoop(x: Long, y: Long): Long =
  {  var a = x
  var b = y
  while (a != 0)
  {  val temp =
  a
  a = b % a
  b = temp
 }
 b
}
```

Built-in control structures

For expressions:

```
val filesHere = (new
        java.io.File(".")).listFiles  for (file <-
        filesHere)
              println(file)
```

Exception handling with try expressions:

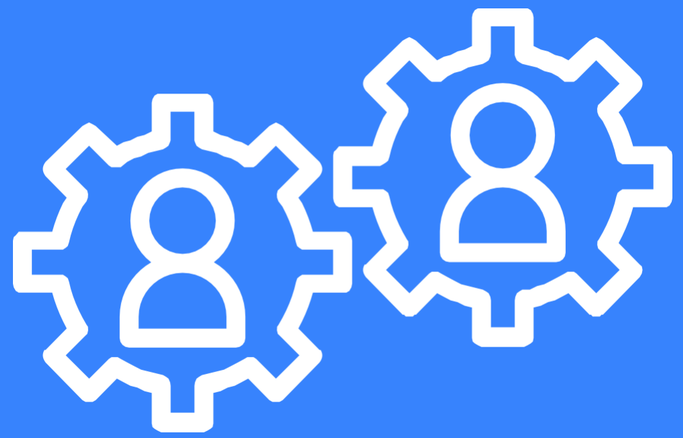
```
import java.io.FileReader
 import
 java.io.FileNotFoundException
 import java.io.IOException

try {
  val f = new FileReader("input.txt") // Use and close file
} catch {
  case ex: FileNotFoundException => // Handle missing
  file  case ex: IOException => // Handle other I/O error
```

Built-in control structures

Match expressions:

Scala's match expression lets you select from several alternatives, just like switch statements in other languages.

```
val firstArg = if (args.length > 0) args(0) else
 ""  firstArg match {
   case "salt" => println("pepper")
   case "chips" => println("salsa")
   case "eggs" => println("bacon")
   case _ => println("huh?")
 }
```



Functions and closures

Functions

Scala has both functions and methods and we use the terms method and function interchangeably with a minor difference.

Function Declarations: def functionName ([list of parameters]) : [return type]

Function Definitions:

```
def functionName ([list of parameters]) :
[return  type] = {
 function
  body  return
  [expr]
}
```

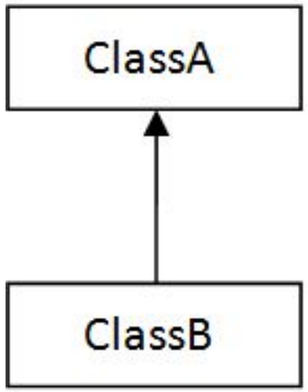
Calling Functions: functionName(list of parameters)

Closures

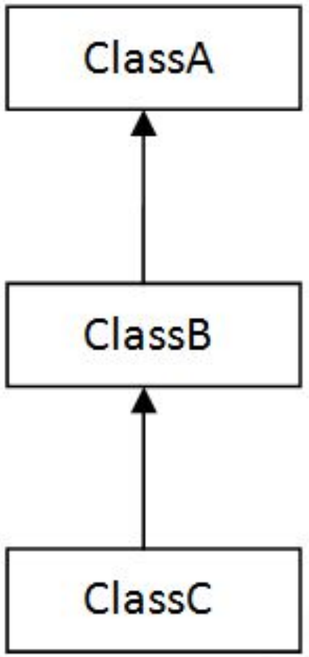
Difference between a closure function and a normal function is the free variable. A free variable is any kind of variable which is not defined within the function and not passed as the parameter of the function. A free variable is not bound to a function with a valid value. The function does not contain any values for the free variable.

Inheritance

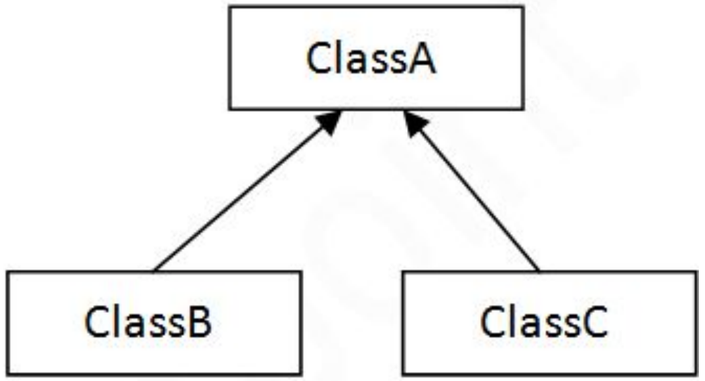
Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in Scala by which one class is allowed to inherit the features(fields and methods) of another class.



1) Single



2) Multilevel



3) Hierarchical

```
class parent_class_name extends
child_class_name{
// Methods and fields
}
```