CLOUD COMPUTING (SEMTRZG527)

By Sai Sushanth Durvasula

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

innovate achieve lead

BITS Pilani
Hyderabad Campus

Contact Session 15

Recap

- Multi tenancy
  - What is Multi-Tenancy?
  - 4 levels of multi tenancy
  - Cloud security threats
  - Cloud Computing Agreements
  - Service Level Agreement Lifecycle Management
- Amazon EMR
  - What is EMR?
  - Key features of Amazon EMR
  - Understanding clusters and nodes
  - node types in Amazon EMR
- Map Reduce Example 2 – Market Ratings

Agenda

- Fault Tolerance in MapReduce
- Hadoop Eco System

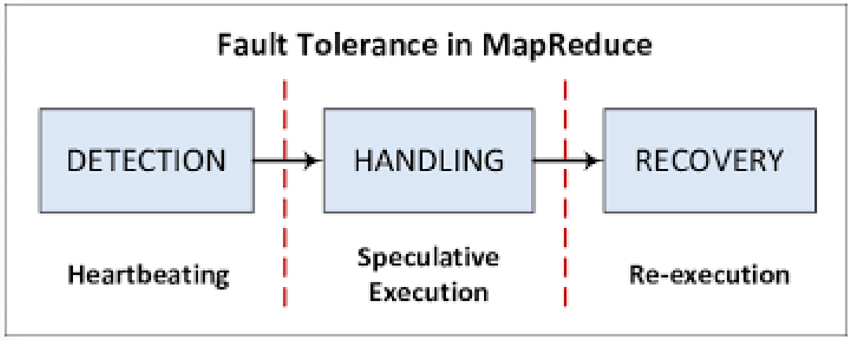
- Apache Pig
  - Introduction
  - Key Features of Pig
  - Word count example using Pig
  - Exercise: Market Ratings

- Apache Hive
  - Introduction
  - Key Features
  - Word count example
  - Exercise: Market Ratings

- Hive vs Pig
- MR vs Hive vs Pig

Fault Tolerance in MapReduce

- Fault tolerance in MapReduce refers to the system's ability to continue functioning properly in the event of failures, such as node crashes or network issues.
- The primary mechanisms for fault tolerance in MapReduce are:
  1) Data Replication
  2) Task Redundancy
  3) Task Monitoring and Restart
  4) Speculative Execution
  5) Job Tracker and Task Tracker High Availability



Fault Tolerance in MapReduce

1) Data Replication: MapReduce replicates input data blocks across multiple nodes in the cluster. This ensures that if a node fails, another node has a copy of the data and can continue processing.

2) Task Redundancy: MapReduce tasks (map and reduce tasks) are replicated across nodes. If a task fails to complete due to node failure, the task can be restarted on another node using the replicated data.

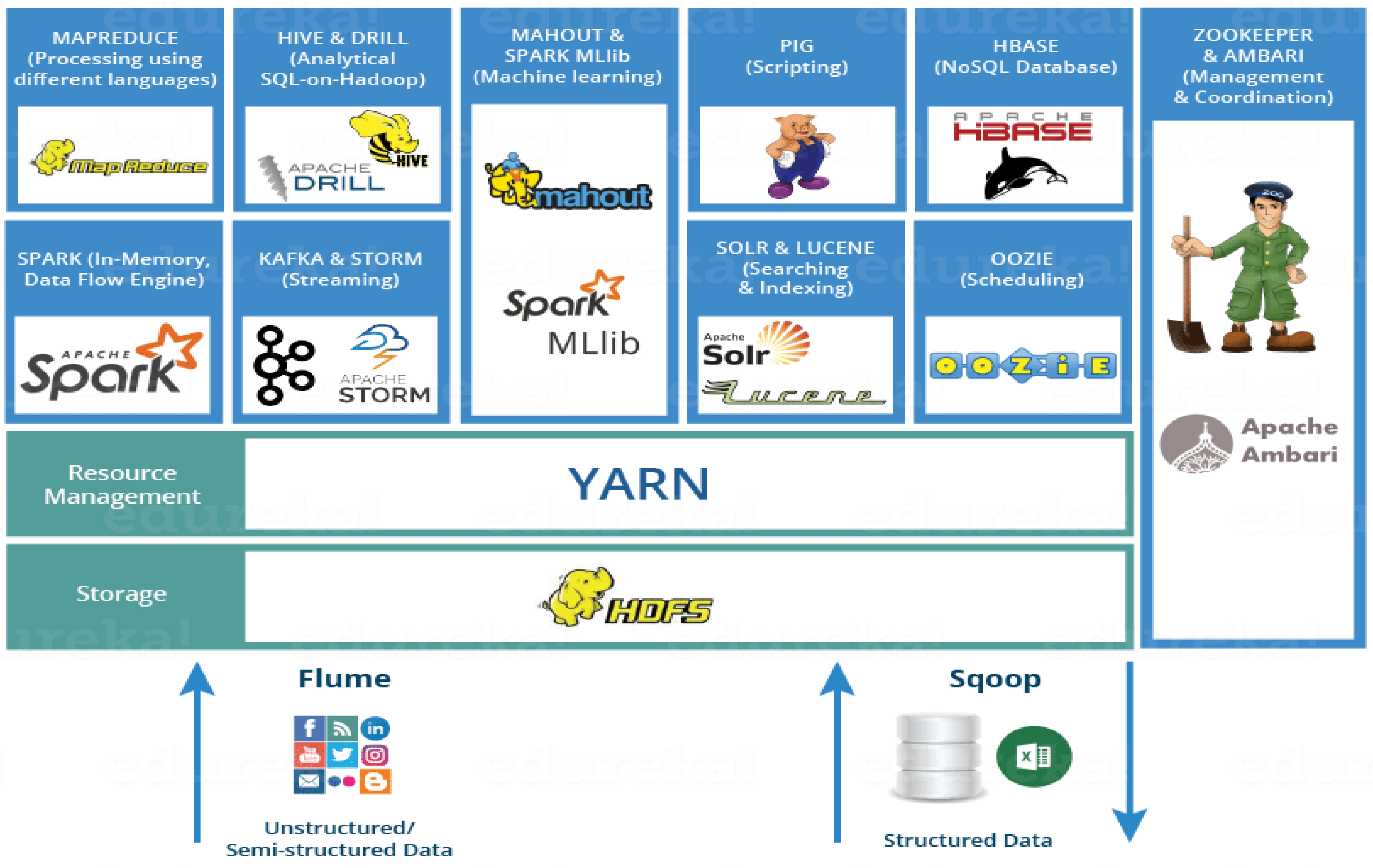
3) Task Monitoring and Restart: MapReduce monitors the progress of tasks and can automatically restart tasks that fail to complete. This helps ensure that the overall job completes even if individual tasks fail.

Fault Tolerance in MapReduce

4) Speculative Execution: MapReduce can run multiple copies of the same task on different nodes simultaneously. If one copy completes significantly faster than the others, the results from that copy are used, and the other copies are killed. This helps mitigate the impact of slow-running tasks.

5) Job Tracker and Task Tracker High Availability: MapReduce maintains a Job Tracker and Task Trackers, which are responsible for coordinating and executing tasks. These components can be configured for high availability to ensure that if one tracker fails, another can take over its responsibilities.

Hadoop Eco System


MAPREDUCE (Processing using different languages)
Map Reduce
HIVE & DRILL (Analytical SQL-on-Hadoop)
APACHE DRILL
HIVE
MAHOUT & SPARK MLlib (Machine learning)
mahout
Spark MLlib
PIG (Scripting)
HBASE (NoSQL Database)
APACHE HBASE
ZOOKEEPER & AMBARI (Management & Coordination)
SPARK (In-Memory, Data Flow Engine)
APACHE Spark
KAFKA & STORM (Streaming)
APACHE STORM
SOLR & LUCENE (Searching & Indexing)
Apache Solr
Lucene
OOZIE (Scheduling)
OOZIE
Apache Ambari
Resource Management
YARN
Storage
HDFS
Flume
Unstructured/ Semi-structured Data
Sqoop
Structured Data

Introduction to Apache Pig

- Apache Pig was developed at Yahoo! to handle their data processing needs.
- It was created to address the challenges of writing complex MapReduce programs for data processing.
- Apache Pig is a high-level scripting language that simplifies the processing of large datasets on Apache Hadoop.
- It provides a way to express data transformations such as ETL (Extract, Transform, Load) operations using a language called Pig Latin.
- Pig Latin abstracts the complexities of writing MapReduce jobs and allows developers to focus on the data manipulation logic.

Key Features of Pig

1) **Ease of Use:** Pig Latin is a scripting language and allows users to write data processing logic in a simple and intuitive way.

2) **Extensibility:** Pig Latin supports user-defined functions (UDFs) in languages like Java, Python, and JavaScript, allowing developers to extend its functionality.

3) **Optimization:** Pig Latin queries are automatically optimized by the Pig framework, improving performance and resource utilization.

4) **Parallel Processing:** Pig Latin operations are executed in parallel, taking advantage of Hadoop's distributed computing capabilities.

5) **Integration:** Apache Pig integrates seamlessly with other Hadoop ecosystem tools like Hive, HBase, and Spark, making it a versatile tool for data processing.

6) **Data Flow Language:** Pig Latin is a data flow language, which means it allows you to specify a series of operations to be applied to your data in a structured way.

7) **Schema Flexibility:** Pig is schema-on-read, meaning it does not enforce a strict schema on data when it is loaded. Instead, the schema is applied when data is read, providing flexibility in data processing.

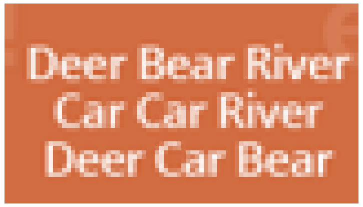
Pig Latin Commands

	Pig commands	Function
Command to Load	Load()	Used to load data from the file system
	PigStorage()	Default load function
	BinStorage()	Used to load data in machine readable format
	TextLoader()	Used to load unstructured text file
	JsonLoader	Used to load file in JSON format
Command to work with data	Filter	Used to access tuples or rows of data.
	Foreach	Used to access columns of data.
	Group	Used to group data in a single relation
	Cogroup	Used to group two or more relations .
	Join	Used to join two or more relations.
	Union	Used to merge the contents of two or more relations.

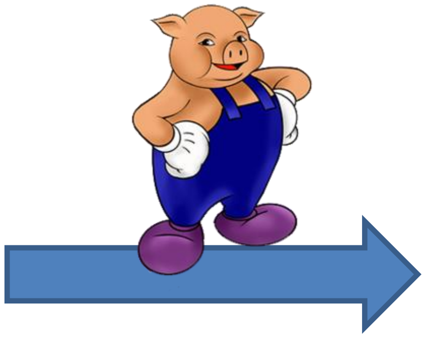
Distributed Word Count Example using Pig Latin

I/P:

input.txt

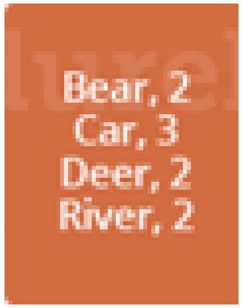


Deer Bear River
Car Car River
Deer Car Bear



O/P:

console



Bear, 2
Car, 3
Deer, 2
River, 2

Word Count example - Pig Latin

Can you write the necessary logic as stated below using pig latin?

-- Load the input file
?

-- Tokenize each line into words
?

-- Group the words and count the occurrences of each word
?
-- Output the result
?

Word Count example – Pig Latin

```
-- Load the input file
lines = LOAD 's3://pig-demo-bucket/input.txt' USING TextLoader()
AS (line:chararray);

-- Tokenize each line into words
words = FOREACH lines GENERATE
FLATTEN(TOKENIZE(LOWER(line))) AS word;

-- Group the words and count the occurrences of each word_groups
= GROUP words BY word;
word_count = FOREACH word_groups GENERATE group AS word,
COUNT(words) AS count;

-- Output the result
STORE word_count INTO 's3://pig-demo-bucket/output' USING
PigStorage('\t');
```

Exercise: Market Ratings using Pig Latin

- Let's consider a scenario where we have a dataset containing information about various products and their ratings given by users.
- We want to calculate the average rating for each product and classify the products into different categories based on their average ratings.
- Ex: I/P:

Product	User	Rating
A	User1	4
A	User2	5
B	User1	3
B	User2	4
B	User3	2

O/P:

Product	Avg Rating
A	4.5
B	3

Introduction to Apache Hive

- Hive was developed by Facebook to provide an SQL-like interface for querying large datasets stored in Hadoop.
- It was later open-sourced and became part of the Apache Software Foundation.
- Apache Hive is a data warehousing infrastructure built on top of Apache Hadoop for querying and analyzing large datasets stored in Hadoop's distributed file system (HDFS)
- Hive provides a SQL-like query language called HiveQL, which allows users to write queries to analyze and process data.
- Hive is commonly used for data warehousing, ETL (Extract, Transform, Load) processes, data analysis, and reporting in organizations dealing with large volumes of data

innovate achieve lead

Key Features of Hive

1) **SQL-Like Query Language:** HiveQL allows users to write queries using familiar SQL syntax, making it easy for SQL users to transition to big data processing.

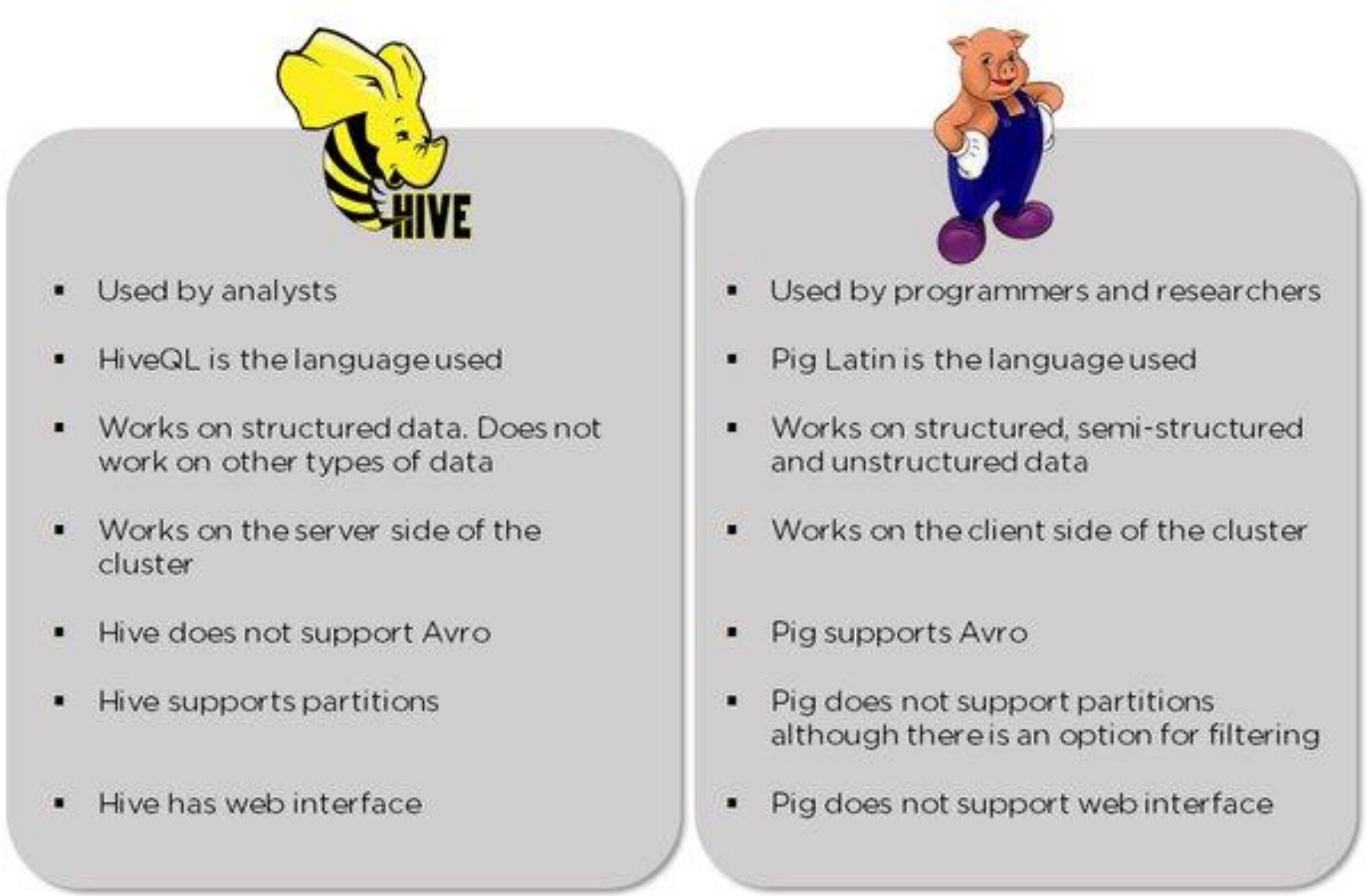
2) **Scalability:** Hive is designed to handle large datasets distributed across a cluster of machines, allowing for scalable data processing.

3) **Schema on Read:** Unlike traditional databases that enforce a schema when data is written, Hive allows data to be stored without a predefined schema and applies the schema when data is read.

4) **Storage Formats:** Hive supports various file formats including text, ORC (Optimized Row Columnar), and Parquet, allowing users to choose the most suitable format for their data.

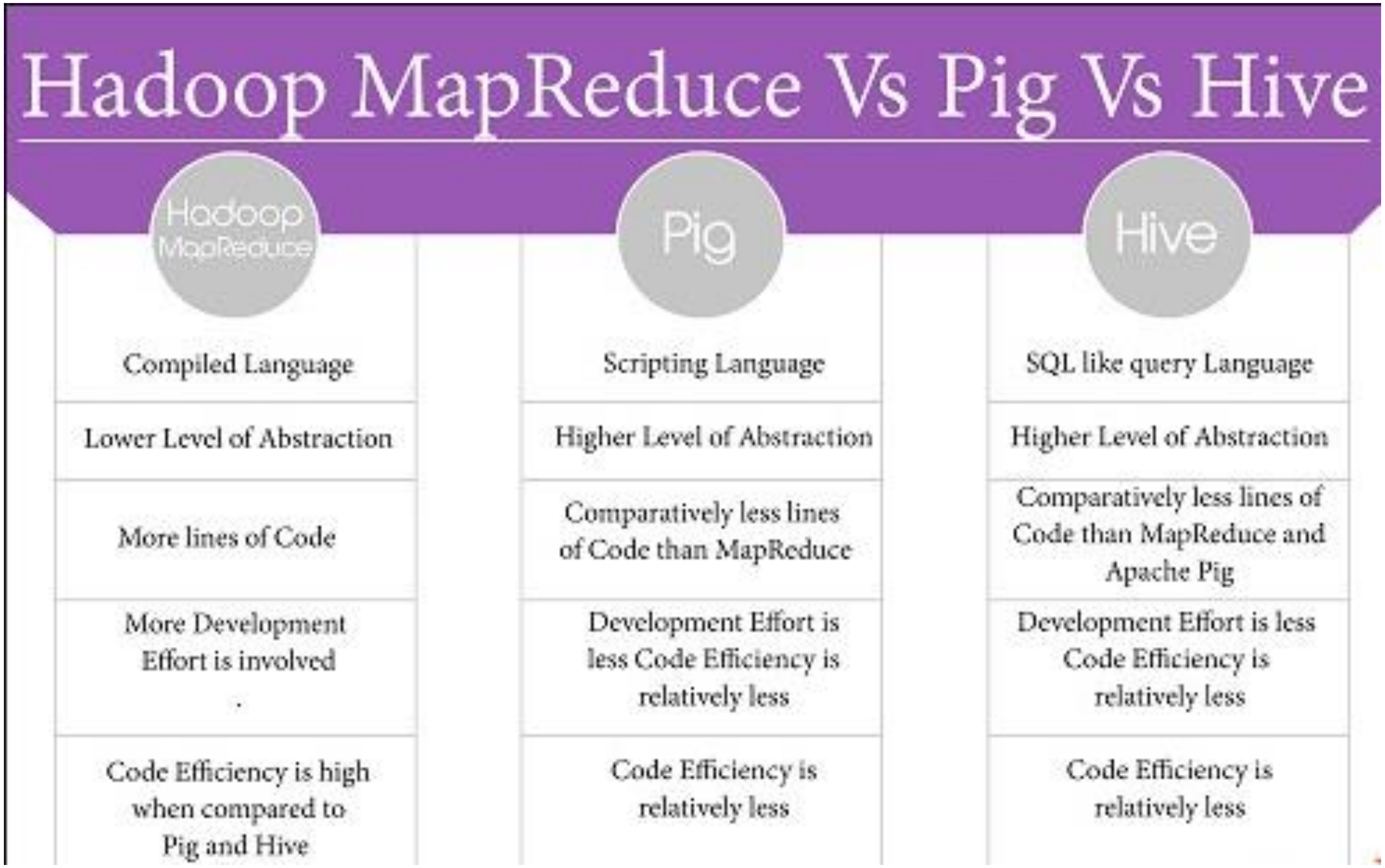
5) **Integration with Hadoop Ecosystem:** Hive integrates with other Hadoop ecosystem tools such as HBase, Spark, and Pig, allowing for seamless data processing workflows.

Hive vs Pig



Used by analysts	Used by programmers and researchers
HiveQL is the language used	Pig Latin is the language used
Works on structured data. Does not work on other types of data	Works on structured, semi-structured and unstructured data
Works on the server side of the cluster	Works on the client side of the cluster
Hive does not support Avro	Pig supports Avro
Hive supports partitions	Pig does not support partitions although there is an option for filtering
Hive has web interface	Pig does not support web interface

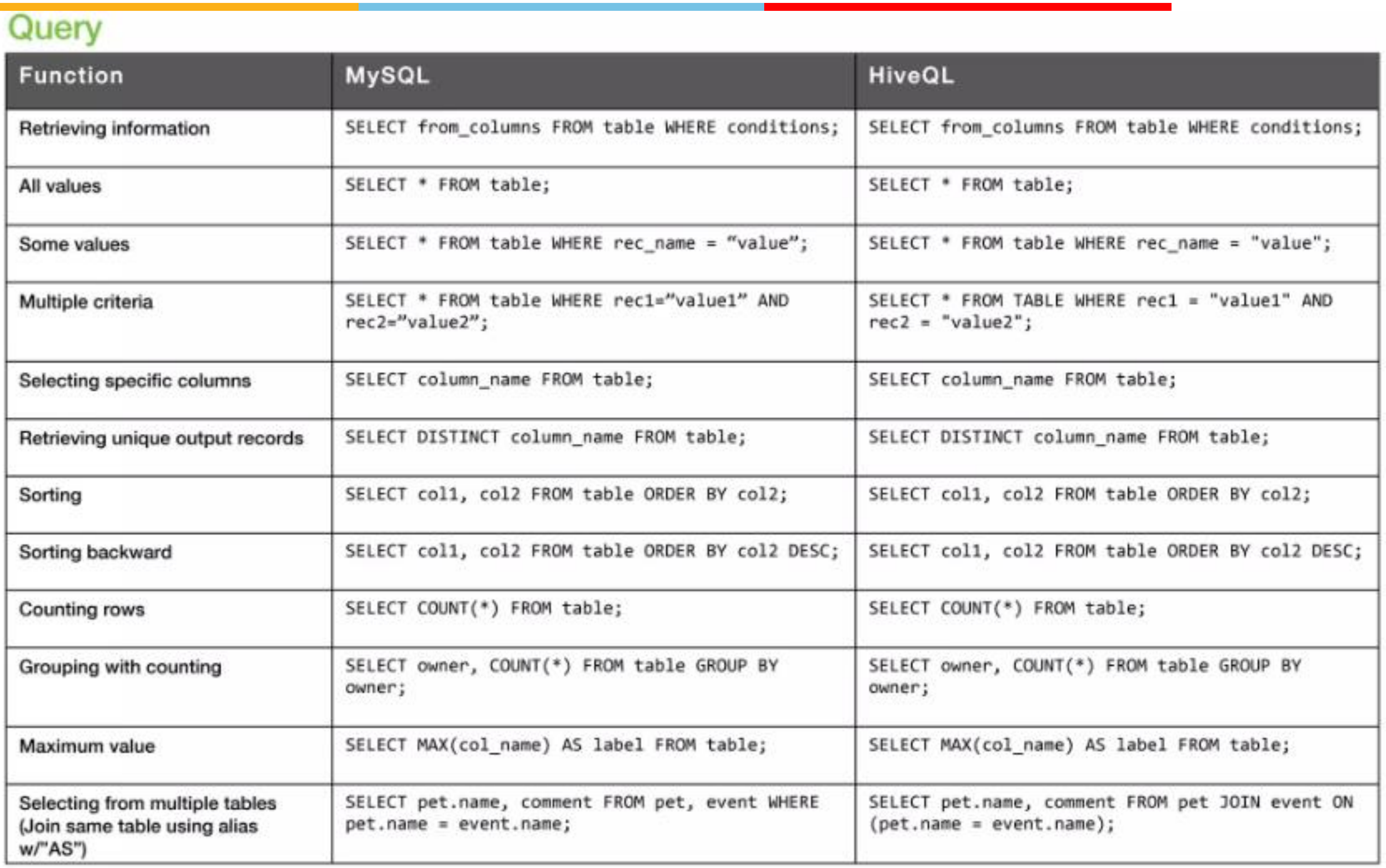
MR vs Hive vs Pig



Hadoop MapReduce Vs Pig Vs Hive

Hadoop MapReduce	Pig	Hive
Compiled Language	Scripting Language	SQL like query Language
Lower Level of Abstraction	Higher Level of Abstraction	Higher Level of Abstraction
More lines of Code	Comparatively less lines of Code than MapReduce	Comparatively less lines of Code than MapReduce and Apache Pig
More Development Effort is involved .	Development Effort is less Code Efficiency is relatively less	Development Effort is less Code Efficiency is relatively less
Code Efficiency is high when compared to Pig and Hive	Code Efficiency is relatively less	Code Efficiency is relatively less

Apache Hive - HQL



Query

Function	MySQL	HiveQL
Retrieving information	`SELECT from_columns FROM table WHERE conditions;`	`SELECT from_columns FROM table WHERE conditions;`
All values	`SELECT * FROM table;`	`SELECT * FROM table;`
Some values	`SELECT * FROM table WHERE rec_name = “value”;`	`SELECT * FROM table WHERE rec_name = "value";`
Multiple criteria	`SELECT * FROM table WHERE rec1=”value1” AND rec2=”value2”;`	`SELECT * FROM TABLE WHERE rec1 = "value1" AND rec2 = "value2";`
Selecting specific columns	`SELECT column_name FROM table;`	`SELECT column_name FROM table;`
Retrieving unique output records	`SELECT DISTINCT column_name FROM table;`	`SELECT DISTINCT column_name FROM table;`
Sorting	`SELECT col1, col2 FROM table ORDER BY col2;`	`SELECT col1, col2 FROM table ORDER BY col2;`
Sorting backward	`SELECT col1, col2 FROM table ORDER BY col2 DESC;`	`SELECT col1, col2 FROM table ORDER BY col2 DESC;`
Counting rows	`SELECT COUNT(*) FROM table;`	`SELECT COUNT(*) FROM table;`
Grouping with counting	`SELECT owner, COUNT(*) FROM table GROUP BY owner;`	`SELECT owner, COUNT(*) FROM table GROUP BY owner;`
Maximum value	`SELECT MAX(col_name) AS label FROM table;`	`SELECT MAX(col_name) AS label FROM table;`
Selecting from multiple tables (Join same table using alias w/"AS")	`SELECT pet.name, comment FROM pet, event WHERE pet.name = event.name;`	`SELECT pet.name, comment FROM pet JOIN event ON (pet.name = event.name);`

Apache Hive – I/O

1) Hive Table Creation:

Ex: to create student table with fields name, roll_no, marks from a csv file as data file input

```
CREATE TABLE IF NOT EXISTS student(
 Name STRING,
 Rollno INT,
 Marks FLOAT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

2) Load Table

```
INSERT INTO TABLE student VALUES ('Dikshant',1,'95'),('Akshat',
2 , '96'),('Dhruv',3,'90');
```
or

Apache Hive – I/O

3) Load data from local file (on laptop):

syntax:

```
LOAD DATA [LOCAL] INPATH '<The table data location>'
[OVERWRITE] INTO TABLE <table_name>;
```

Ex:

```
LOAD DATA LOCAL INPATH
'/home/dikshant/Documents/data.csv' INTO TABLE student;
```

Apache Hive – I/O

4) Load data from HDFS file:

syntax:

```
LOAD DATA INPATH '<The table data location>' [OVERWRITE]
INTO TABLE <table_name>;
```

Ex:

```
LOAD DATA INPATH '/home/dikshant/Documents/data.csv' INTO
TABLE student;
```

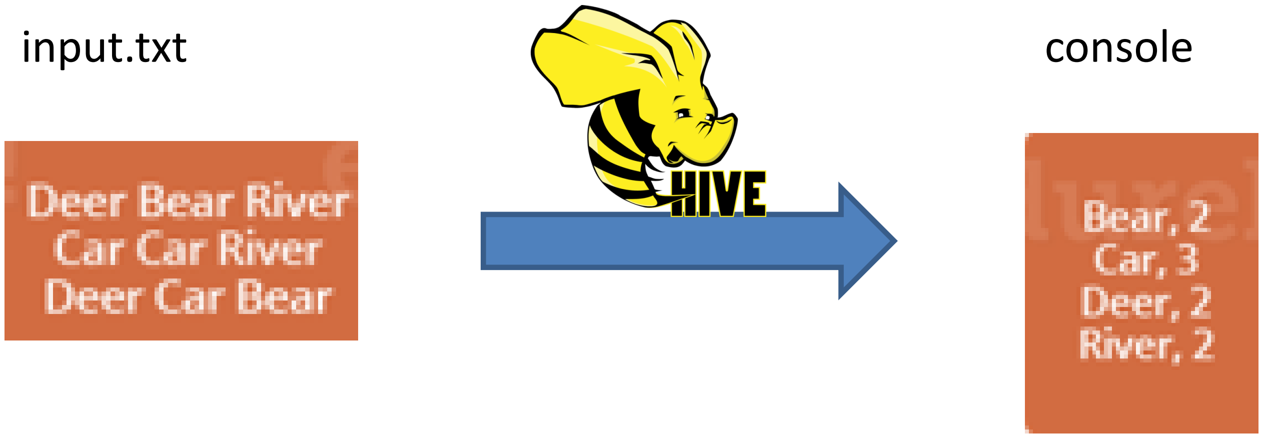
Distributed Word Count using Hive

I/P:

input.txt

O/P:

console



Deer Bear River
Car Car River
Deer Car Bear

Bear, 2
Car, 3
Deer, 2
River, 2

Word Count example – Hive

1) Create a table:

- First, we need to create a table in Hive to store your input data.
- We can create a table in Hive to store this data using the following command:
  - **CREATE TABLE input_table (line STRING);**

2) Load data into the table:

- Load the data from the input.txt file into the input_table using the following command:
  - **LOAD DATA LOCAL INPATH 'input.txt' INTO TABLE input_table;**

Word Count example – Hive

3) Tokenize the text:

```
SELECT explode(split(line, ' ')) AS word
FROM input_table
```

4) Group by words and count:

```
SELECT word, COUNT(1) AS count
FROM (
        SELECT explode(split(lower(line), ' ')) AS word
        FROM input_table
        ) as words
GROUP BY word;
```

The above query will output the count of each word in the input text.

Word Count example - Hive

5) Save the Output to S3 Bucket

```
INSERT OVERWRITE DIRECTORY 's3://pig-demo-bucket/output'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
<SELECT query above>
```

Exercise: Market Ratings using Hive

- Let's consider a scenario where we have a dataset containing information about various products and their ratings given by users.
- We want to calculate the average rating for each product and classify the products into different categories based on their average ratings.
- Ex: I/P: O/P:

Product	User	Rating
A	User1	4
A	User2	5
B	User1	3
B	User2	4
B	User3	2

Product	Avg Rating
A	4.5
B	3

innovate achieve lead

THANK YOU