# DSC650: DATA TECHNOLOGY AND FUTURE EMERGENCE

Lecture 4: **Data Munging**

Lecturer: Dr Khairul Anwar Sedek

Faculty of Computer and Mathematical Sciences

Universiti Teknologi MARA, Perlis Branch

# Lecture 4: **Data Munging**

- Different Type of Data Processing: Parallel, Distributed, Batch, Transactional, Cluster and etc
- MapReduce Framework, Algorithm and Process Data
- Real-Time Data Analysis using Apache Spark
- Scalability and Fault Tolerance
- Optimization and Data Locality
- Real World Cases

At the end of the lecture, students should be able to;

- CLO1: Demonstrate an understanding on the basic **concepts and practices** of big data technology

## DATA PROCESSING

- Process of collecting, processing, manipulating, and managing the data to generate meaningful information to the end user.
- Data may be originated from diversified sources in the form of transactions, observations, and so forth – data capture
- Once data is captured, data processing begins.
- There are basically two different types of data processing, namely, centralized and distributed data processing.

## DATA PROCESSING

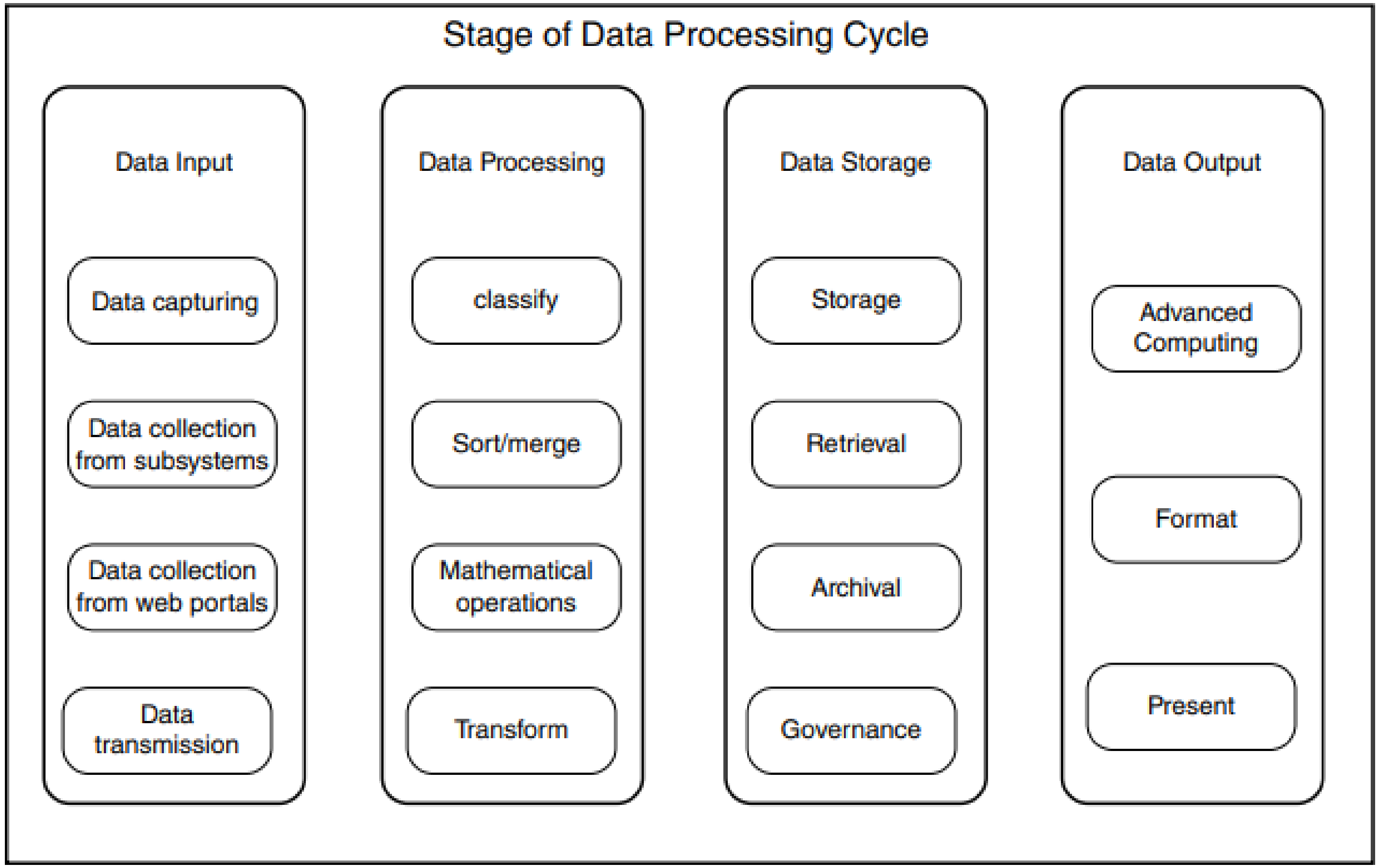
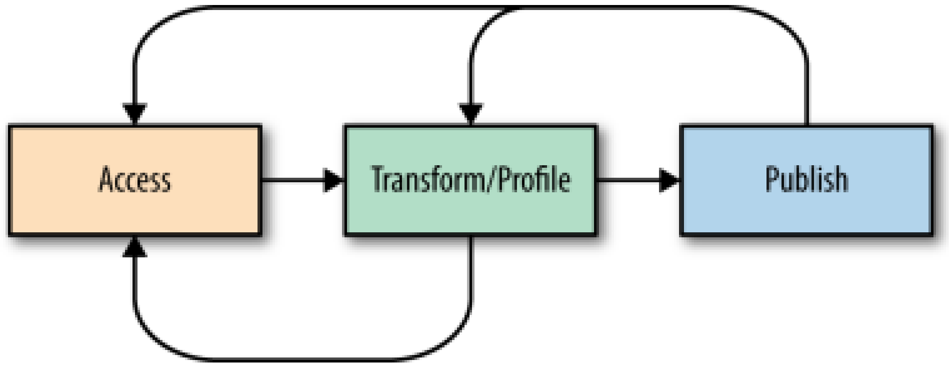


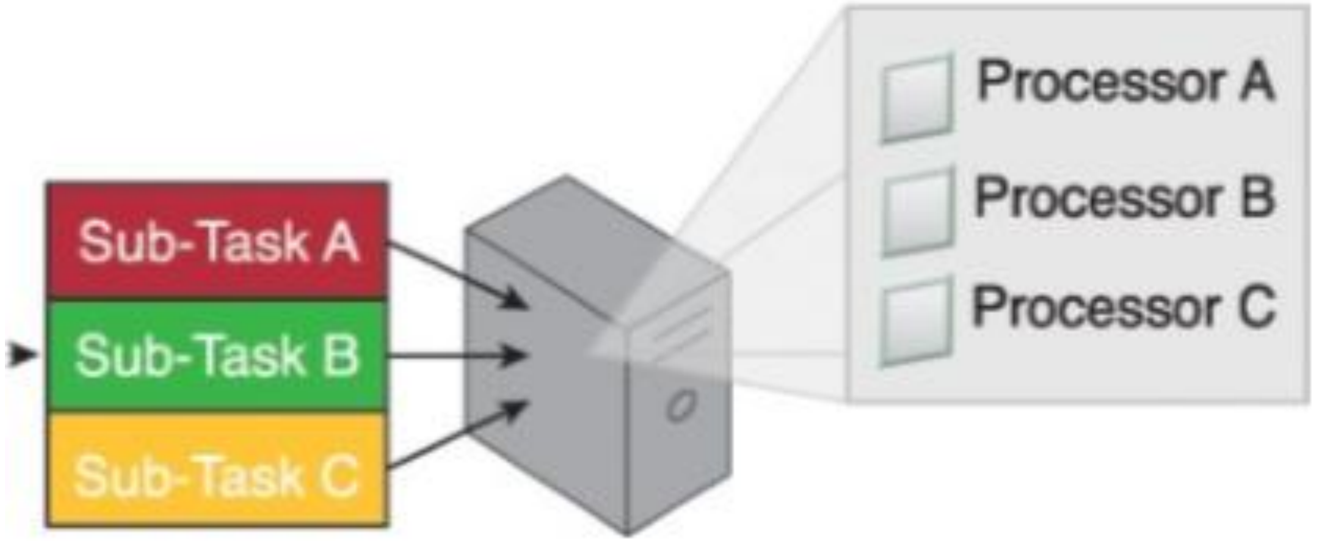
**Figure 4.1** Data processing cycle.

# DATA MUNGING

- **aka. Data wrangling**
- process of **transforming and mapping data** from one **"raw" data form into another format** with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics.
- typically follows a set of general steps which begin with
  - **Access**. extracting the data in a raw form from the data source,
  - **Transform**. "munging" the raw data using algorithms (e.g. sorting) or parsing the data into predefined data structures, and
  - **Publish**. finally depositing the resulting content into a data sink for storage and future use

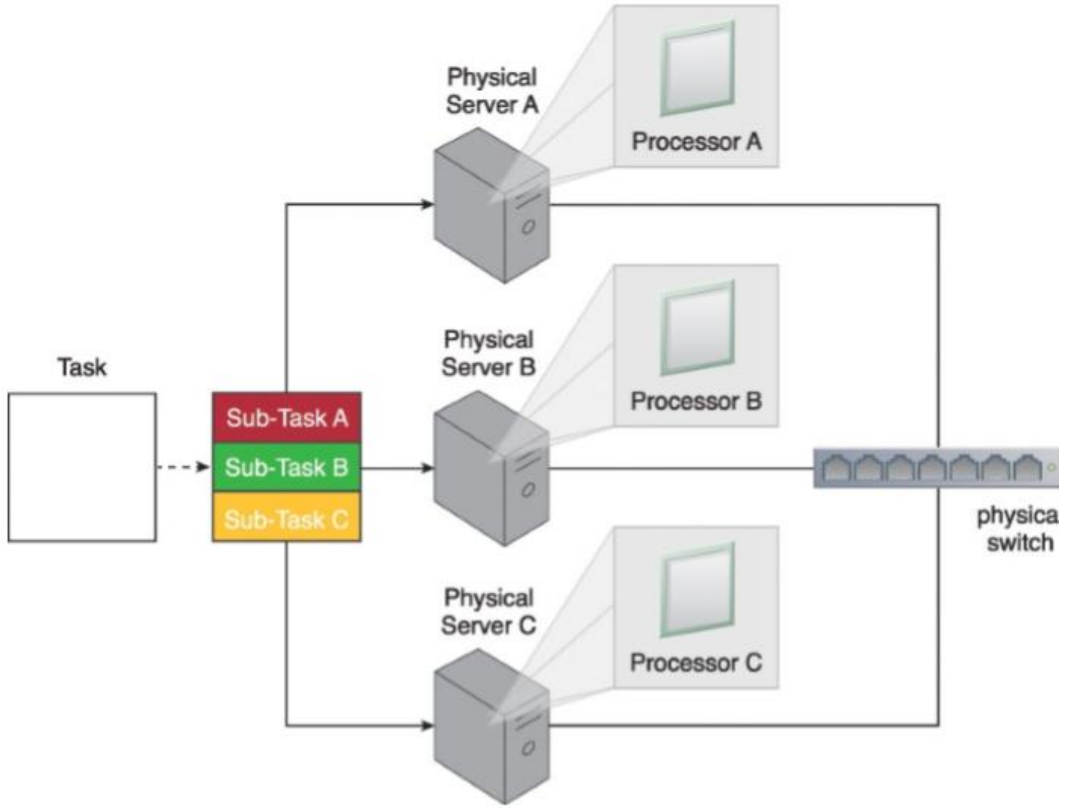


# PARALLEL DATA PROCESSING



- Simultaneous execution of multiple sub-tasks that collectively comprise a larger task.
- The goal is to reduce the execution time by dividing a single larger task into multiple smaller tasks that run concurrently.
- Typically achieved within the confines of a single machine with multiple processors.

## DISTRIBUTED DATA PROCESSING



- Similar to parallel data processing in that the same principle of "divide-and-conquer".
- However, distributed data processing is always achieved through physically separate machines that are networked together as a cluster.

## PROCESSING WORKLOADS

Processing workload defined as the amount and nature of data that is processed within a certain amount of time.

Workloads are usually divided into two types:

- Batch
- Transactional

## PROCESSING WORKLOADS

**Batch**

- *Offline processing*
- Processing data in batches and usually imposes delays, which in turn results in high-latency responses.
- Batch workloads typically involve large quantities of data with sequential read/writes and comprise of groups of read or write queries.
- Queries can be complex and involve multiple joins.
- (Online Analytical Processing) OLAP systems commonly process workloads in batches.
- Strategic BI and analytics are batch-oriented as they are highly read-intensive tasks involving large volumes of data.

## PROCESSING WORKLOADS

**Transactional**

- *Online processing*
- Transactional workload processing follows an approach whereby data is processed interactively without delay, resulting in low–latency responses.
- Transaction workloads involve small amounts of data with random reads and writes.
- Online Transaction Processing (OLTP) and operational systems, which are generally write-intensive.
- Although these workloads contain a mix of read/write queries, they are generally more write-intensive than read-intensive.
- Transactional workloads comprise random reads/writes that involve fewer joins than business intelligence and reporting workloads.

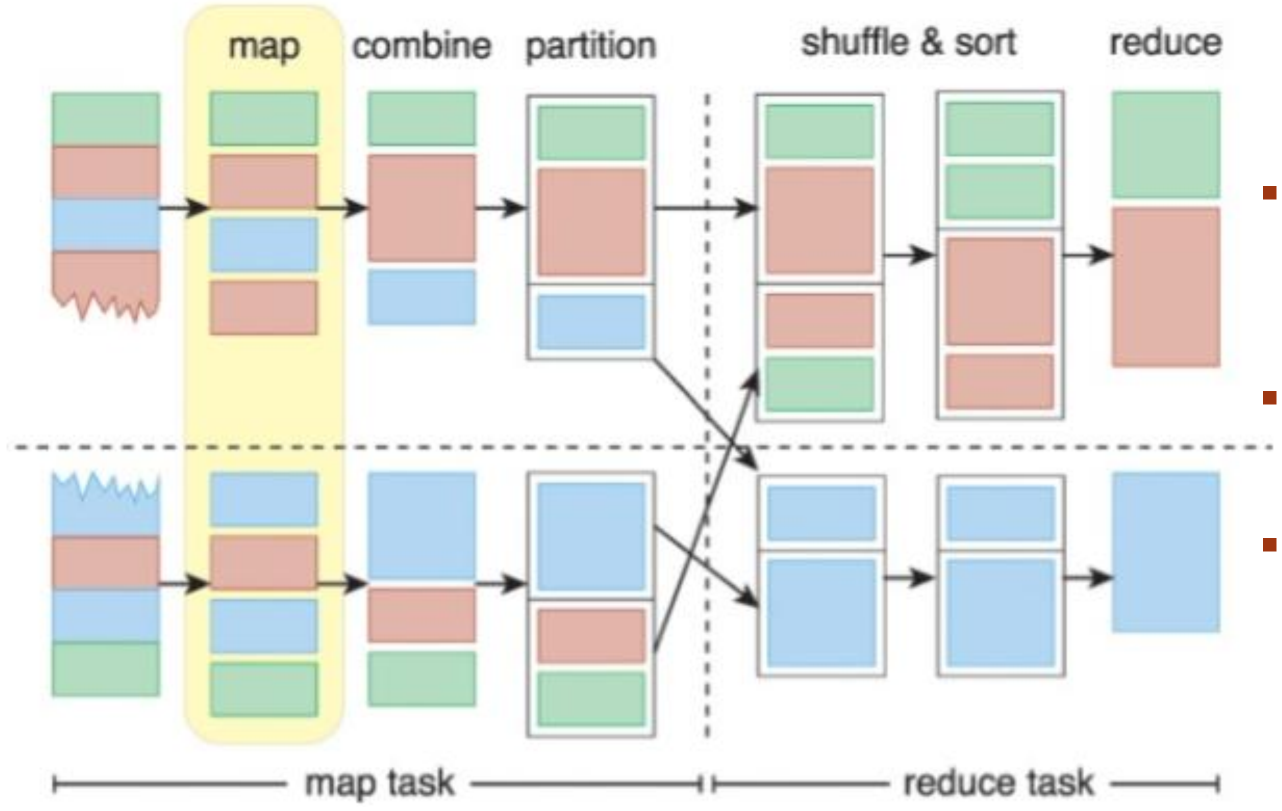
## CLUSTER

- Horizontally scalable storage solutions.
- Clusters also provides the mechanism to enable distributed data processing with linear scalability.
- Since clusters are highly scalable, Big Data processing as large datasets can be divided into smaller datasets and then processed in parallel in a distributed manner.
- When leveraging a cluster, Big Data datasets can either be processed in batch mode or real-time model.
- Ideally, a cluster will be comprised of low-cost commodity nodes that collectively provide increased processing capacity.
- Other benefits; redundancy and fault tolerance
  - Consist of physically separate nodes.
  - Redundancy and fault tolerance allow resilient processing and analysis to occur if a network or node failure occurs.

## MAPREDUCE

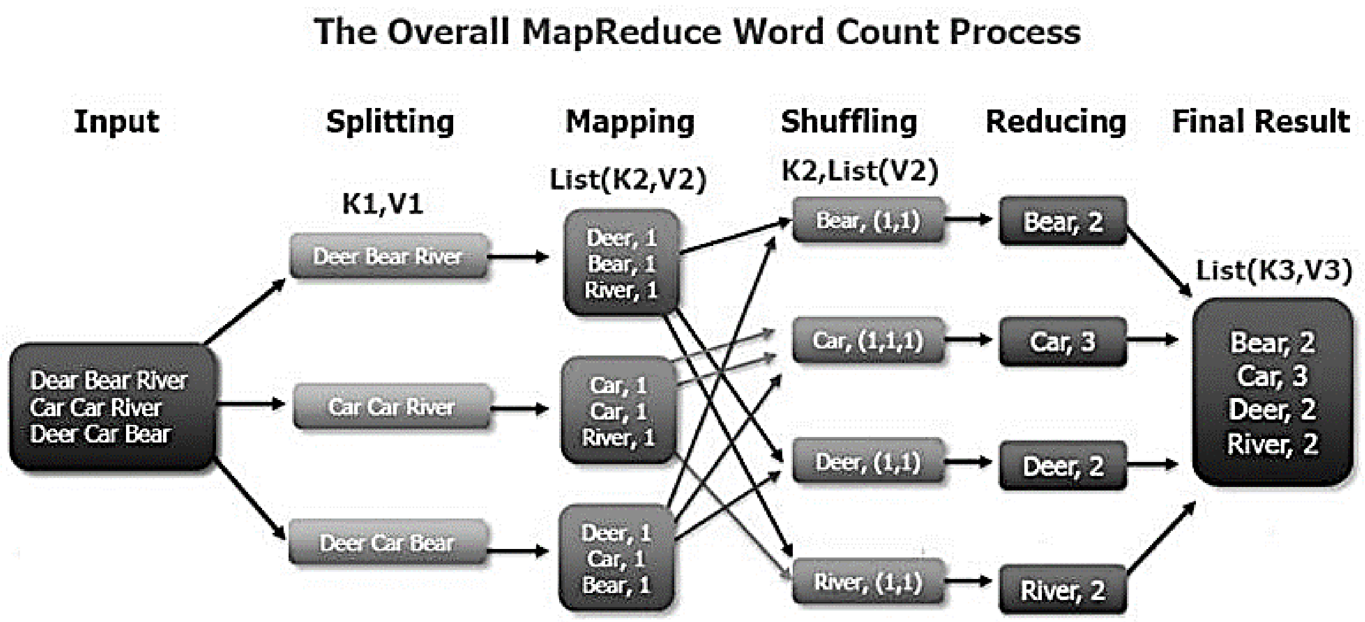
- Batch processing framework.
- Highly scalable and reliable
- Principle of divide-and-conquer – provides built-in fault tolerance and redundancy.
- Has roots in both distributed and parallel computing.
- Process schema-less datasets.
- A dataset is broken down into multiple smaller parts, and operations are performed on each part independently and in parallel.

# MAP AND REDUCE TASK

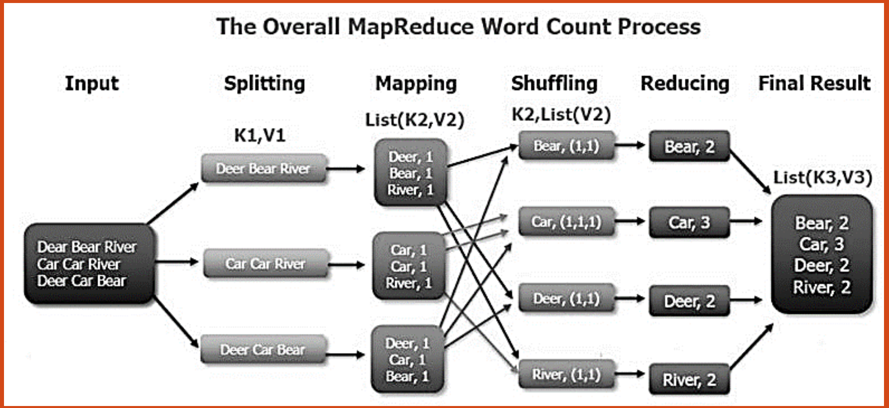


- A single processing run of the MapReduce processing engine is known as MapReduce job.
- Each MapReduce job is composed of a map task and a reduce task
- Each task consists of multiple stages.

# MAP AND REDUCE TASK



# MAP AND REDUCE TASK



- **Step 1**: Take the file as input for processing purpose. Any file will consist of group of lines. These lines containing key-value pair of data. Whole file can be read out with this method.
- **Step 2**: In next step file will be in "splitting" mode. This mode will divide file into key, value pair of data. This time key will be offset and data will be value part of program. Each line will be read individually so there is no need to split data manually.
- **Step 3**: Further step is to process the value of each line with associate from counting number. Each individual that is separated from a space counted with number and that number is written with each key. This is the logic of "mapping" that programmer need to write.
- **Step 4**: After that shuffling is performed and with this each key get associated with group of numbers that involved in mapping section. Now scenario become key with string and value will be list of numbers. This will go as input to reducer.
- **Step 5**: In reducer phase whole numbers are counted and each key associated with final counting is the sum of all numbers which leads to final result.
- **Step 6**: Output of reducer phase will lead to final result. This final result will have counting of individual wordcount.

# MAPREDUCE ALGORITHMS

## Task Parallelism

- Parallelization of data processing by **dividing a task into sub-tasks and running each sub-task on a separate processor**, generally on a separate node in a cluster.
- Each sub-task generally executes a different algorithm, with its own copy of the same data or different data as its input, in parallel.
- Generally, the output from multiple sub-tasks is joined together to obtain the final set of results.

## Data Parallelism

- Parallelization of data processing by **dividing a dataset into multiple datasets and processing each sub-dataset in parallel**.
- The sub-datasets are spread across multiple nodes and are all processed using the same algorithm.
- Generally, the output from each processed sub-dataset is joined together to obtain the final set of results.

## REAL-TIME PROCESSING

- In real-time mode, data is **processed in-memory** - it is captured before being persisted to the disk.
- Response time generally ranges from a sub-second to under a minute.
- Realtime mode addresses the velocity characteristic.
- Also called **event** or **stream** processing as the data either arrives continuously (stream) or at intervals (event).
- The individual event/stream datum is generally small in size, but its continuous nature results in very large datasets.
- Another related term, **interactive** mode.
- Interactive mode generally refers to query processing in realtime.
- Operational BI/analytics are generally conducted in realtime mode.

## DISTRIBUTED DATA PROCESSING PRINCIPLE

- Fundamental principle - Speed, Consistency and Volume (SCV) principle.
- **Speed** – refers to how quickly the data can be processed once it is generated.
  - In the case of realtime analytics, data is processed comparatively faster than batch analytics.
  - This generally excludes the time taken to capture data and focuses only on the actual data processing, such as generating statistics or executing an algorithm.

- **Consistency** – refers to the accuracy and precision of results.

  - Results are deemed accurate if they are close to the correct value and precise if close to each other.

- **Volume** – refers to the amount of data that can be processed.

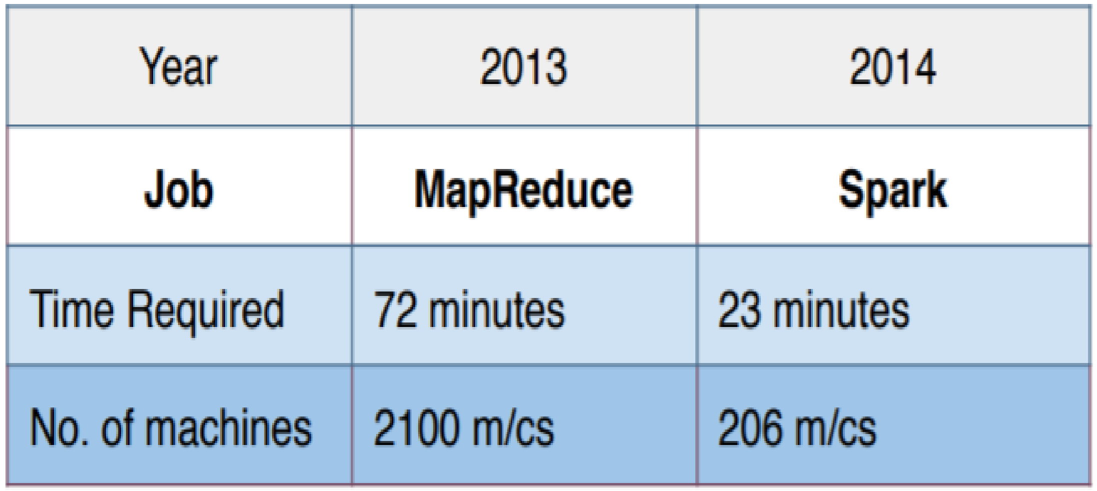
  - huge volumes of data that need to be processed in a distributed manner.

  - Processing such voluminous data in its entirety while ensuring speed and consistency is not possible.

# REALTIME PROCESSING - SPARK

❑ Runtime performance is much better than MR

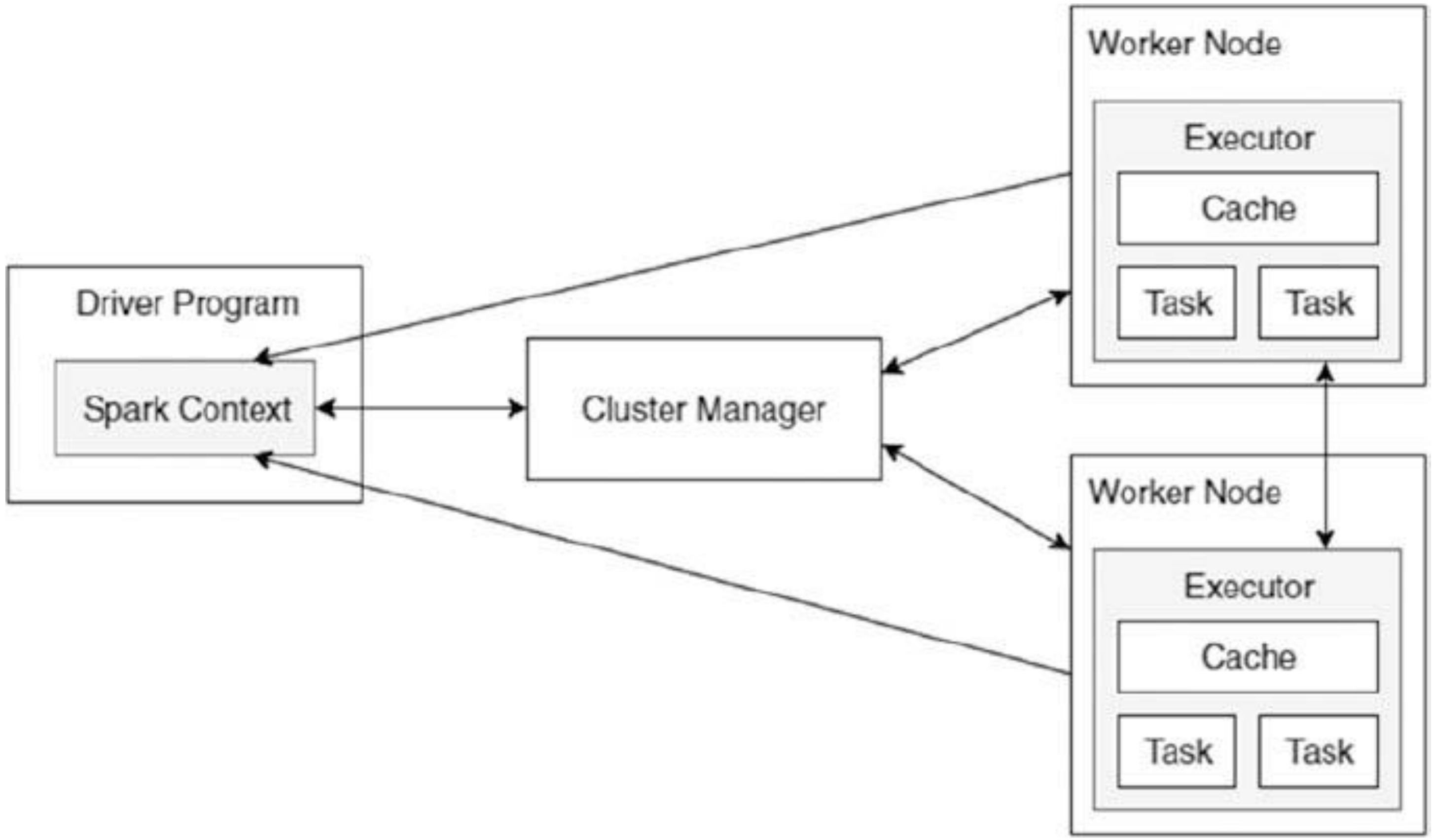
Sorting 100TB data



| Year | 2013 | 2014 |
|---|---|---|
| **Job** | **MapReduce** | **Spark** |
| Time Required | 72 minutes | 23 minutes |
| No. of machines | 2100 m/cs | 206 m/cs |

Reference: https://databricks.com/blog/2014/10/10/spark-petaby

- for data processing
- designed to be fast and general purpose
- based on cluster computing platform
- uses in-memory distributed computing
- can run on top of existing Hadoop environment / can also run as standalone
- provides shell support (interactive programming environment)
- supports different types of workloads (batch and streaming data)



Spark Architecture

## REALTIME PROCESSING - SPARK

- Apache Spark uses a master/slave/worker architecture.
- A driver program runs on the master node and talks to an executor on worker node.
- Spark applications run as independent sets of processes, which is coordinated by the SparkContext object and created by the driver program.
- Spark can run in standalone mode or on a cluster of many nodes.
- SparkContext talks to the cluster manager to allocate resources across applications.
- Spark acquires executors on nodes in the cluster, and then sends application code to them.
- Executors are processes that actually run the application code and store data for these applications.

# REALTIME PROCESSING - SPARK

- Spark's core concept is **Resilient Distributed Dataset** (RDD).
- RDD is a fault-tolerant collection of elements.
- RDD can be operated on in parallel.
- It is a resilient and distributed collection of records, which can be at one partition or more, depending on the configuration.
- RDD is an **immutable** distributed collection of objects, which implies that **you cannot change data in RDD** but you can apply transformation on one RDD to get another one as a result.
- It abstracts away the complexity of working in parallel.
- **Resilient:** Fault tolerant, able to re-compute when it has missing records or damaged partitions due to node failures.
- **Distributed:** Data resides on multiple nodes in a cluster.
- **Dataset:** A collection of partitioned data with a key-value pair or primitive values called tuples. Represents the records of data you work with.

# REALTIME PROCESSING - SPARK

- Additional traits;
  - **Immutable:** RDD never changes once created; they are read-only and can only be transformed to another RDD using transformation operations.
  - **Lazy evaluated:** you first load data into RDD, then apply a filter on it, and ask for a count.
  - **In-memory:** Spark keeps RDD in memory as much size as it can and for as long as possible.
  - **Typed:** RDD records are strongly typed, like Int in RDD[Int] or tuple (Int, String) in RDD[(Int, String)].
  - **Cacheable:** RDD can store data in a persistent storage.
  - **Partitioned:** Data is split into a number of logical partitions based on multiple parameters, and then distributed across nodes in a cluster.
  - **Parallel:** RDDs are normally distributed on multiple nodes, which is the abstraction it provides; after partitioning, it is acted upon in parallel fashion.
  - **Location aware:** RDDs has location preference, Spark tries to create them as close to data as possible provided resources are available (**data locality**).

# MAPREDUCE VS. SPARK

**MapReduce**

- Amount of overhead associated with MapReduce – job creation and coordination.
- batch-oriented processing of large amounts of data that has been stored to disk.
- Cannot process data incrementally and can only process complete datasets.
- It therefore requires all input data to be available in its entirety before the execution of the data processing job.

**Spark**

- Realtime processing
- Suitable for smaller set data
- In-memory processing (not permanent but faster)
- Uses parallel computing BUT NOT mapper & reducer

# HADOOP SCALABILITY

- Scalability means expanding or contracting the cluster. We can scale Hadoop HDFS in 2 ways.
- **Vertical Scaling**
  - We can add more disks on nodes of the cluster.
    For doing this, we need to edit the configuration files and make corresponding entries of newly added disks. Here we need to provide downtime though it is very less. So people generally prefer the second way of scaling, which is horizontal scaling.
- **Horizontal Scaling**
  - Another option of scalability is of adding more nodes to the cluster on the fly without any downtime. This is known as horizontal scaling. We can add as many nodes as we want in the cluster on the fly in real-time without any downtime. This is a unique feature provided by Hadoop.

Big Data Part-3

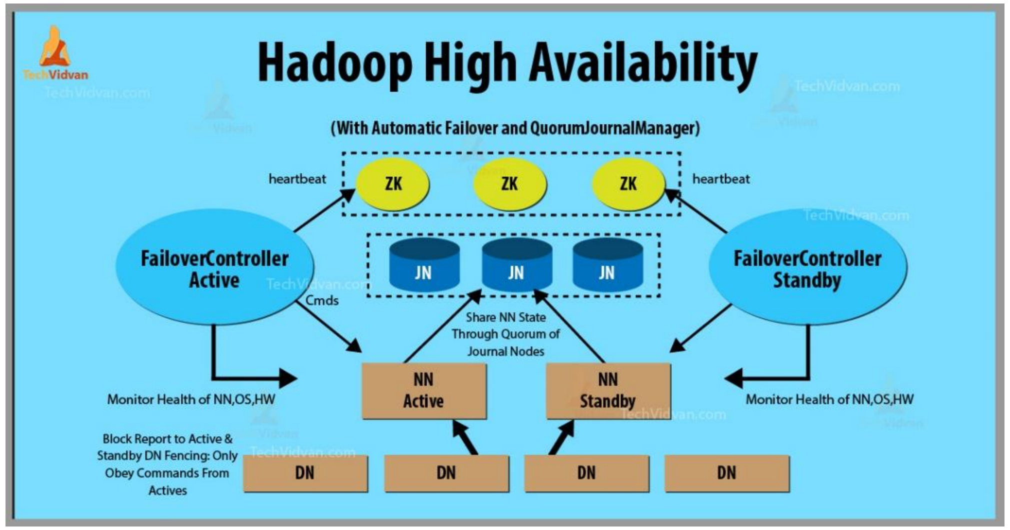
1. What is Scalability?
2. Types of Scalability
3. Vertical Scalability
4. Horizontal Scalability
5. RDBMS vs Distributed
6. Big Data vs Hadoop?
7. Limitations of Big Data
8. Challenges of Big data
9. Industry wise Comparisons of Data

# HADOOP HIGH AVAILABILITY & NAMENODE HIGH AVAILABILITY ARCHITECTURE

- 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. This NameNode becomes a bottleneck. HDFS high availability feature addresses this issue.
- **High availability** refers to the availability of system or data in the wake of component failure in the system
- The high availability feature in Hadoop ensures the availability of the Hadoop cluster without any downtime, even in unfavorable conditions like NameNode failure, DataNode failure, machine crash, etc.

# HOW HADOOP HDFS ACHIEVES HIGH AVAILABILITY?

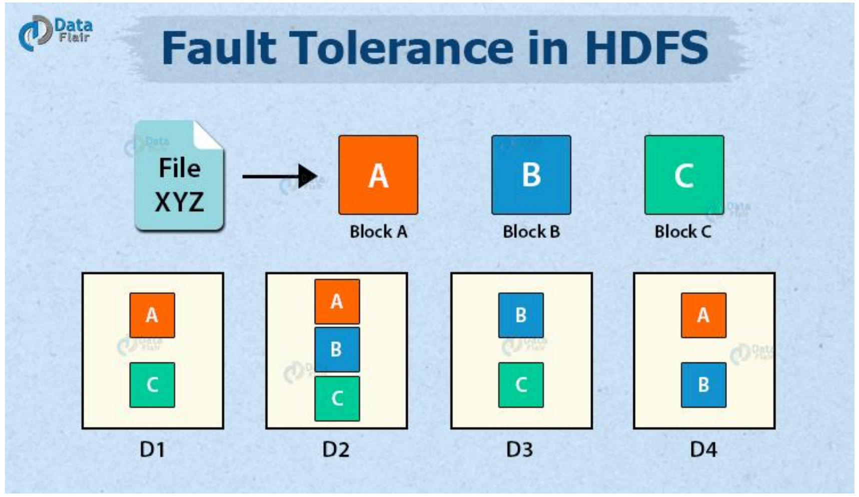
- HDFS stores users' data in files and internally, the files are split into fixed-size blocks.
  - These blocks are stored on DataNodes. NameNode is the master node that stores the metadata about file system i.e. block location, different blocks for each file, etc.
- Availability if DataNode fails
  - If DataNode fails, the NameNode redirects the user to the other DataNode containing a copy of the same data. The user requesting for data read, access the data from other DataNodes containing a copy of data, without any downtime. Thus cluster is available to the user even if any of the DataNodes fails.
- Availability if NameNode fails
  - The Hadoop HA cluster consists of two NameNodes (or more after Hadoop 3) running in a cluster in an active/passive configuration with a hot standby.
  - So, if an active node fails, then a passive node becomes the active NameNode, takes the responsibility, and serves the client request.

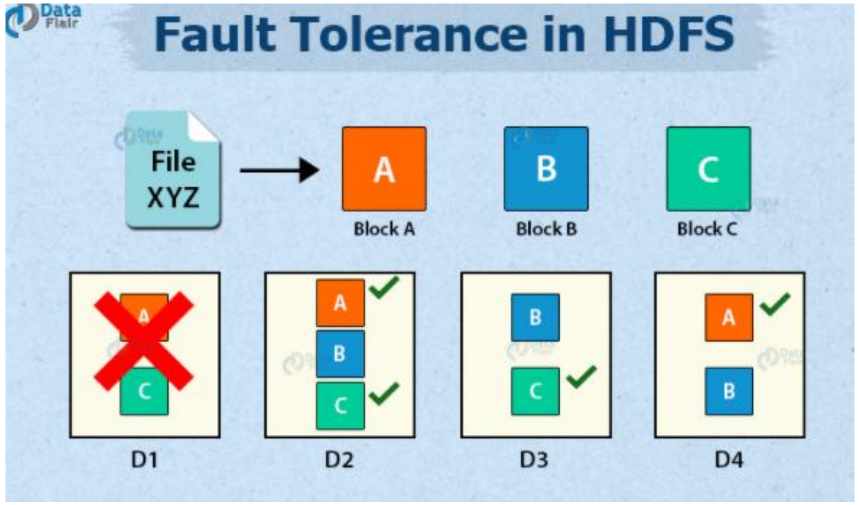


TechVidvan
Hadoop High Availability
(With Automatic Failover and QuorumJournalManager)
heartbeat
ZK
ZK
ZK
heartbeat
FailoverController Active
JN
JN
JN
FailoverController Standby
Cmds
Share NN State Through Quorum of Journal Nodes
Monitor Health of NN,OS,HW
NN Active
NN Standby
Monitor Health of NN,OS,HW
Block Report to Active & Standby DN Fencing: Only Obey Commands From Actives
DN
DN
DN
DN

## HOW HDFS ACHIEVES FAULT TOLERANCE?

- **Fault tolerance** refers to the ability of the system to work or **operate even in case of unfavorable conditions** (like components failure).
- Fault tolerance in Hadoop HDFS refers to the working **strength of a system in unfavorable conditions** and how that system can handle such a situation.
- HDFS is **highly fault-tolerant**. Before Hadoop 3, it handles faults by the process of **replica creation**. It creates a replica of users' data on different machines in the HDFS cluster. So whenever if any machine in the cluster goes down, then data is accessible from other machines in which the same copy of data was created.



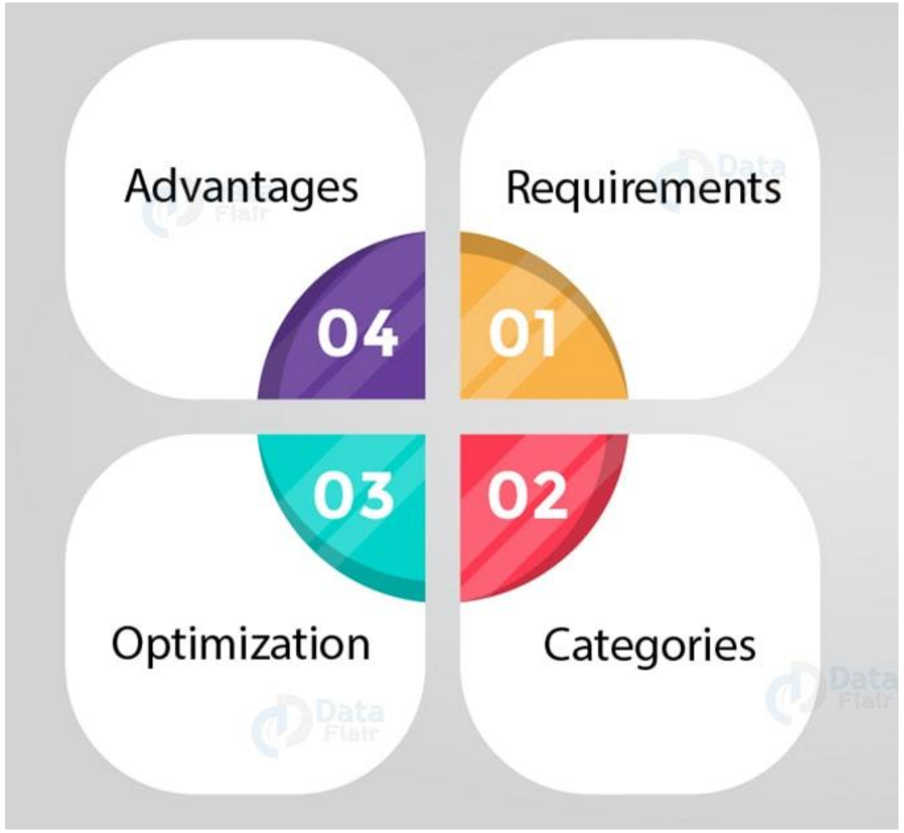


## A DEMO TO SHOW THE FAULT TOLERANCE EFFECT OF HADOOP

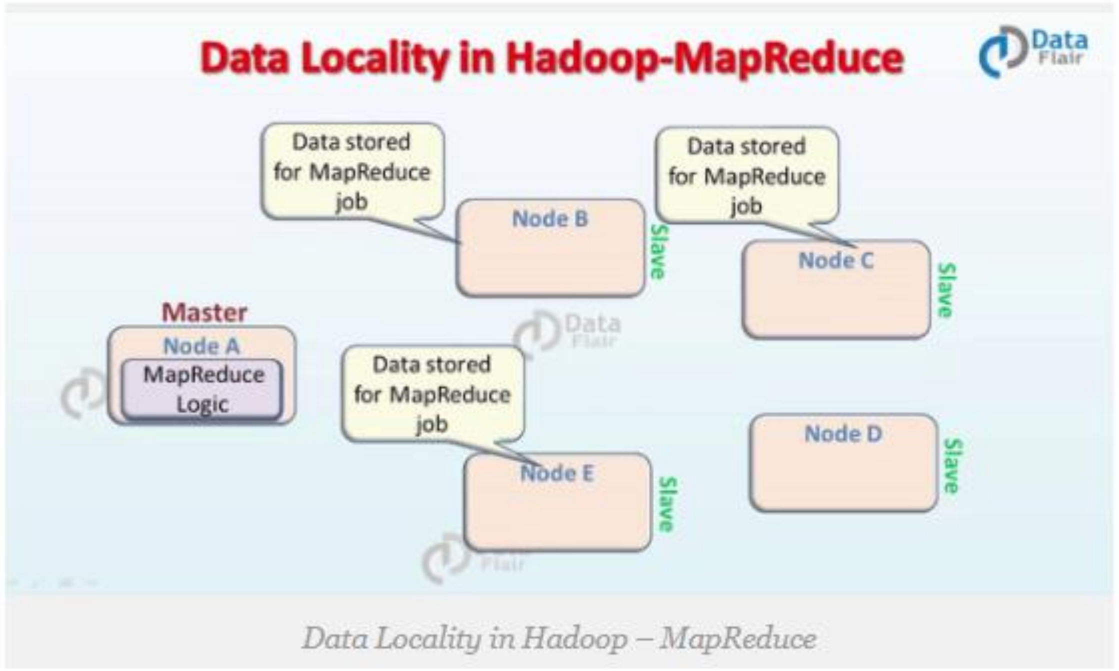
## HADOOP OPTIMIZATION TECHNIQUES

- Proper configuration of your cluster
- LZO compression usage
- Proper tuning of the number of MapReduce tasks
- Combiner between mapper and reducer
- Usage of most appropriate and compact writable type for data
- Reusage of Writables

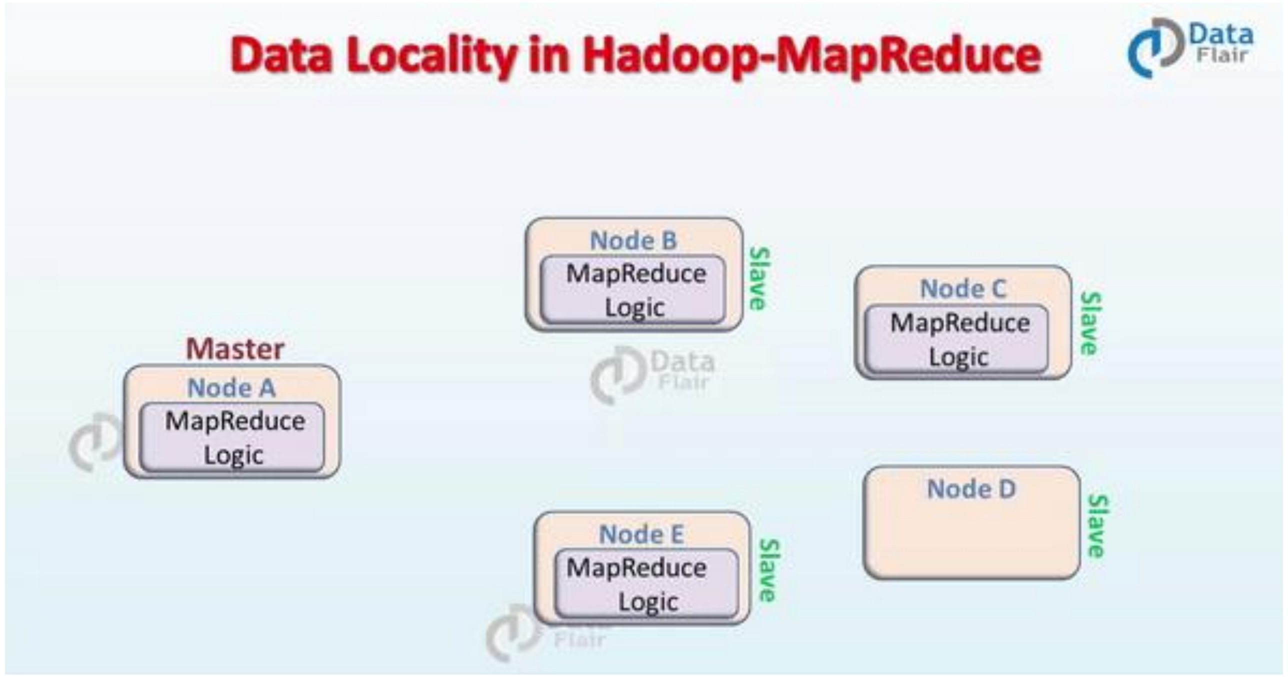
## DATA LOCALITY IN HADOOP



- The major drawback of Hadoop was cross-switch network traffic due to the huge volume of data.
- To overcome this drawback, **Data Locality** in Hadoop came into the picture.
- Data locality in MapReduce refers to the ability to move the computation close to where the actual data resides on the node, instead of moving large data to computation..
- Benefits:
  - minimizes network congestion and increases the overall throughput of the system


Data Locality in Hadoop-MapReduce
Data Flair
Data stored for MapReduce job
Node B
Slave
Data stored for MapReduce job
Node C
Slave
Master
Node A
MapReduce Logic
Data stored for MapReduce job
Node E
Slave
Node D
Slave

*Data Locality in Hadoop – MapReduce*



https://data-flair.training/blogs/data-locality-in-hadoop-mapreduce/#google_vignette

## CATEGORIES OF DATA LOCALITY IN HADOOP

### Data local data locality in Hadoop

- When the data is located on the same node as the mapper working on the data it is known as data local data locality.
- In this case, the proximity of data is very near to computation. This is the most preferred scenario.

### Intra-Rack data locality in Hadoop

- It is not always possible to execute the **mapper** on the same datanode due to resource constraints.
- In such case, it is preferred to run the mapper on the different node but on the same rack.

### Inter-Rack data locality in Hadoop

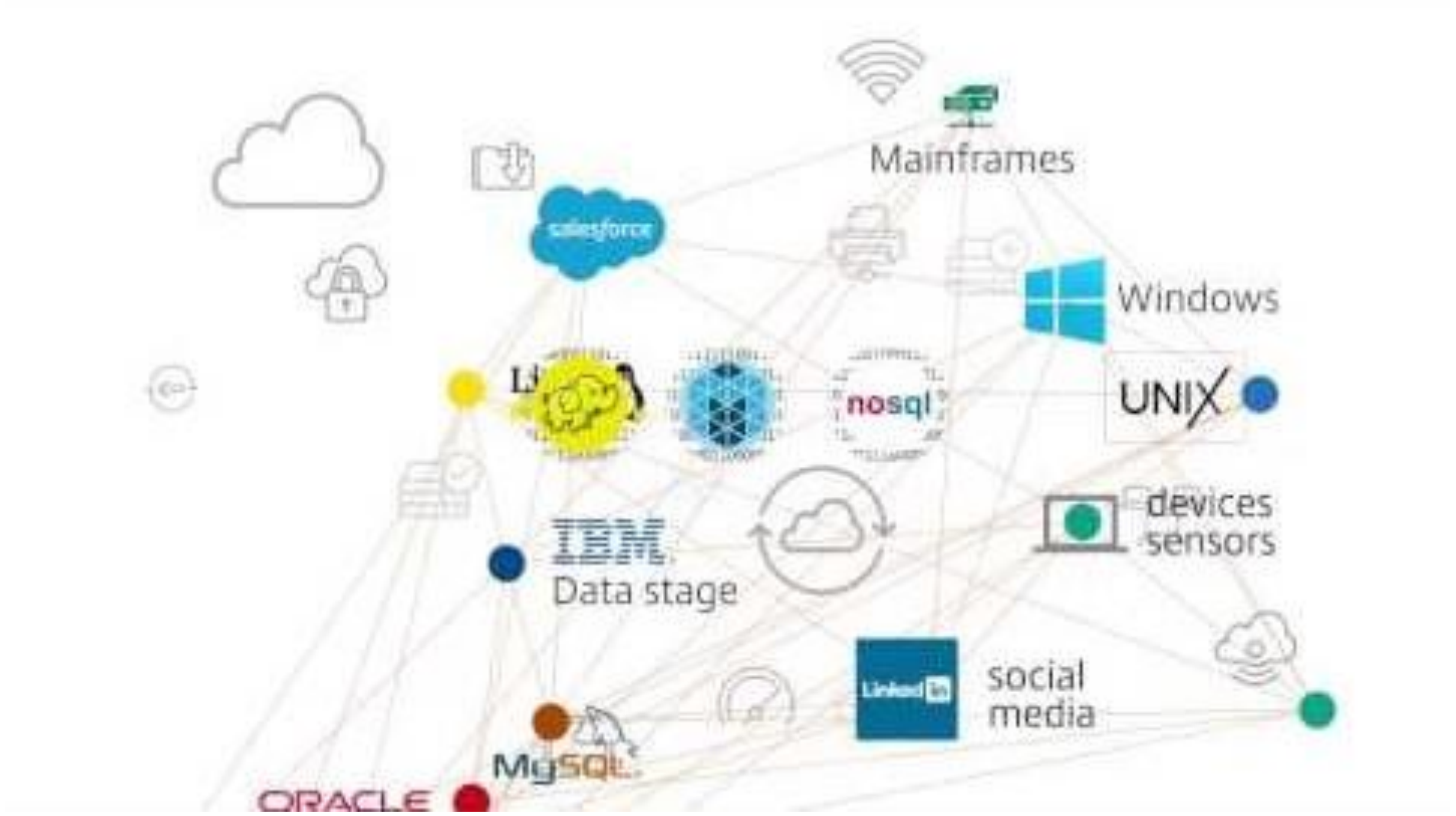
- Sometimes it is not possible to execute mapper on a different node in the same rack due to resource constraints.
- In such a case, we will execute the mapper on the nodes on different racks. This is the least preferred scenario.

## REAL WORLD CASES

- Financial services companies use analytics to assess risk, build investment models, and create trading algorithms; Hadoop has been used to help build and run those applications.
- Retailers use it to help analyze structured and unstructured data to better understand and serve their customers.
- In the asset-intensive energy industry Hadoop-powered analytics are used for predictive maintenance, with input from Internet of Things (IoT) devices feeding data into big data programs.
- Telecommunications companies can adapt all the aforementioned use cases. For example, they can use Hadoop-powered analytics to execute predictive maintenance on their infrastructure. Big data analytics can also plan efficient network paths and recommend optimal locations for new cell towers or other network expansion. To support customer-facing operations telcos can analyze customer behavior and billing statements to inform new service offerings.
- There are numerous public sector programs, ranging from anticipating and preventing disease outbreaks to crunching numbers to catch tax cheats.

https://www.bmc.com/blogs/hadoop-examples/

## THE SECRET SOURCE OF BIG DATA SUCCESS



https://youtu.be/SF4572r-63c