Map-Reduce Framework

- Basics of functional programming
  - Fundamentals of functional programming
  - Real world problems modeling in functional style
- Map reduce fundamentals
- Data flow (Architecture)
- Real world problems
- Scalability goal
- Fault tolerance
- Optimization and data locality
- Parallel Efficiency of Map-Reduce

Functional Programming

- Functional programming (FP) is a way of thinking about software construction by creating pure functions. It avoid concepts of shared state, mutable data observed in Object Oriented Programming.
- Functional langauges empazies on expressions and declarations rather than execution of statements. Therefore, unlike other procedures which depend on a local or global state, value output in FP depends only on the arguments passed to the function.
- For Example: Clojure, Haskell, Lisp, etc.

Example of Functional program in Python

- Sample code to showcase functional program in Python:

```
def sum(a, b):
    return (a + b)
print(sum(3,5))
funcAssignment = sum
print(funcAssignment(3,5))
```

- The given example shows the implementation of functional programming in Python to print a list with first 10 Fibonacci number.

```
Fibonacci = (lambda n, first = 0, second = 1: [ ] if n == 0 else [first]
    + Fibonacci (n-1, second, firs t + second) )
print (Fibonacci (10))
```

Characteristics of Functional Programming

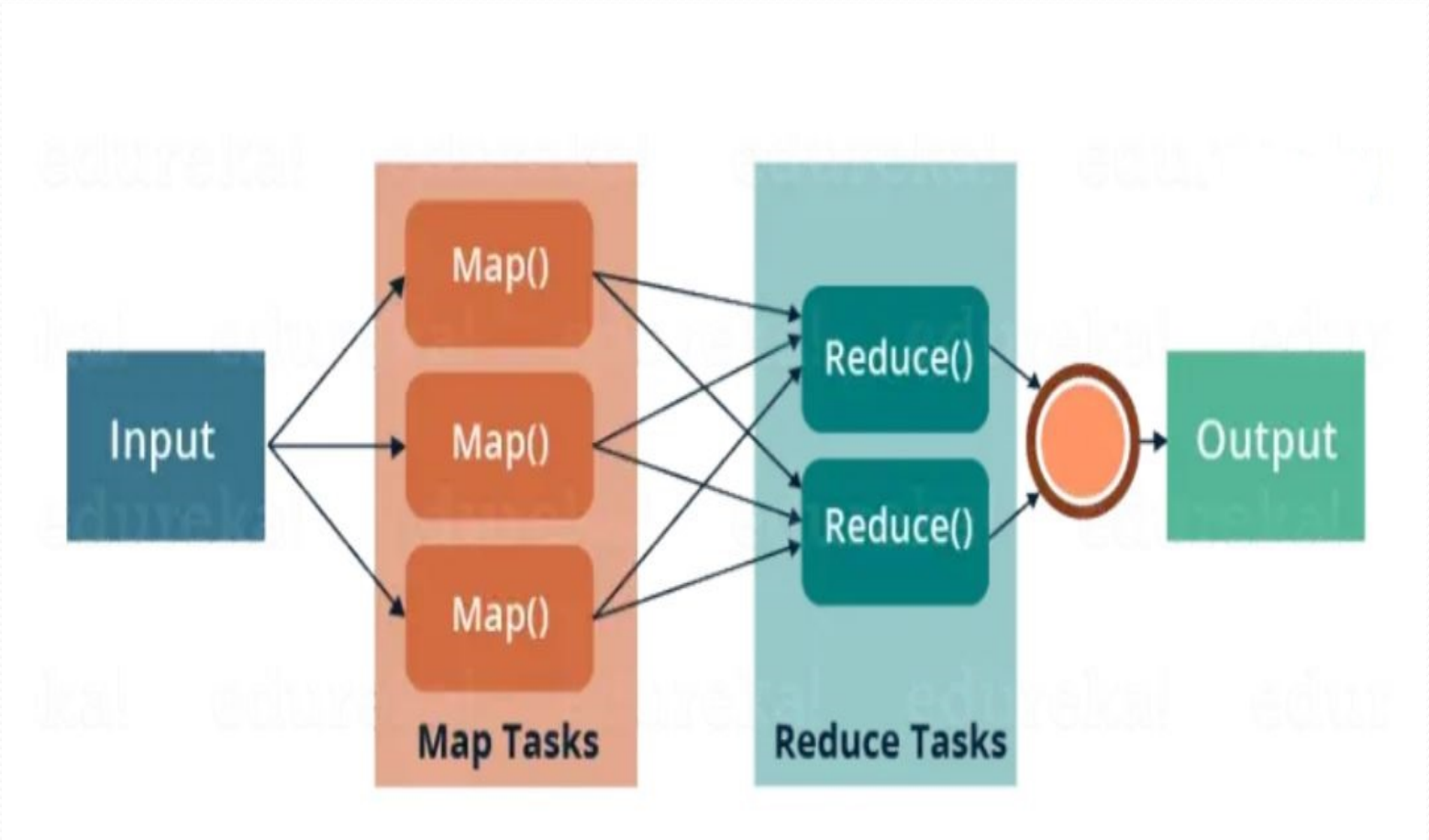
- Functional programming method focuses on results, not the process
- Emphasis is on what is to be computed
- Data is immutable
- Functional programming Decompose the problem into functions
- It is built on the concept of mathematical functions which uses conditional expressions and recursion to perform the calculation
- It does not support iteration like loop statements and conditional statements like If-Else

Functional programming Vs. Imperative programming

- In imperative programming, the programmer focuses on how to perform task and how to track changes. In functional programming, the programmer focuses on what information is desired and what transformations are required.
- State changes in imperative programming. State change do not exist in functional programming.
- Order of execution is important in imperative programming but not important in functional programming.
- Flow control is managed using loops, conditions in imperative programming while using function call and recursion in functional programming.

MapReduce

- MapReduce is a programming framework that allows us to perform distributed and parallel processing on large data sets in a distributed environment.
  - MapReduce consists of two distinct tasks — Map and Reduce.
  - As the name MapReduce suggests, reducer phase takes place after the mapper phase has been completed.
  - So, the first is the map job, where a block of data is read and processed to produce key-value pairs as intermediate outputs.
  - The output of a Mapper or map job (key-value pairs) is input to the Reducer.
  - The reducer receives the key-value pair from multiple map jobs.
  - Then, the reducer aggregates those intermediate data tuples (intermediate key-value pair) into a smaller set of tuples or key-value pairs which is the final output.


Input
Map()
Map()
Map()
Map Tasks
Reduce()
Reduce()
Reduce Tasks
Output

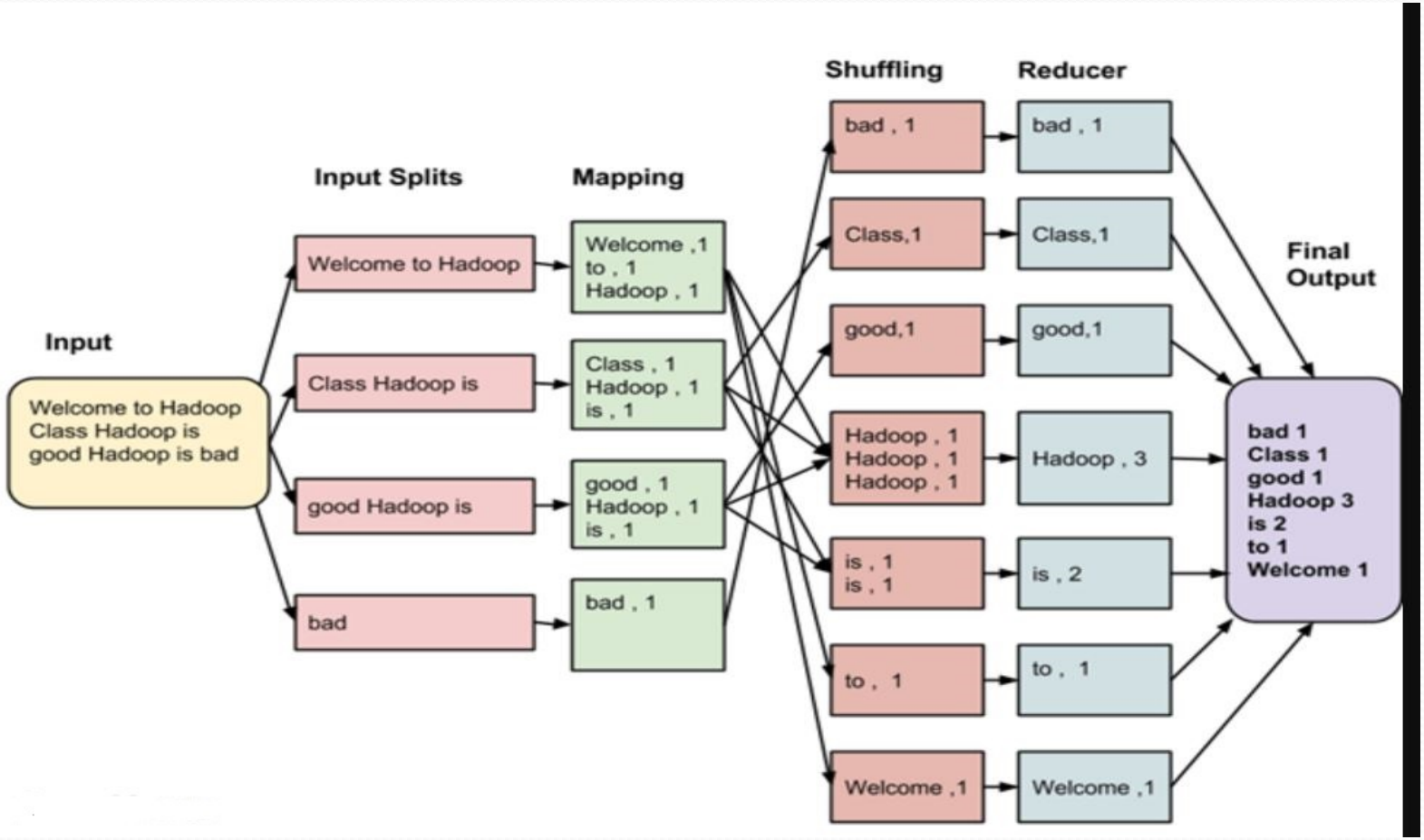
Basic Components of Map Reduce

- MapReduce is composed of two components. They are Mapper and Reducer.
- Mapper is the component that execute map () function. The real input to the system is the input for the mapper. The map () function when executed, the given input is converted to a key/value pair to generate intermediate key/value pairs.
- Reducer is the component that execute reduce () function. The intermediate key/value pair generated from mapper is the input to reduce () function. It merges all the intermediate values associated to the same key.

Usage of MapReduce

- Distributed sort
- Web link-graph reversal
- Web access log stats
- Inverted index construction
- Document clustering
- Machine learning
- Statistical machine translation

MapReduce Architecture

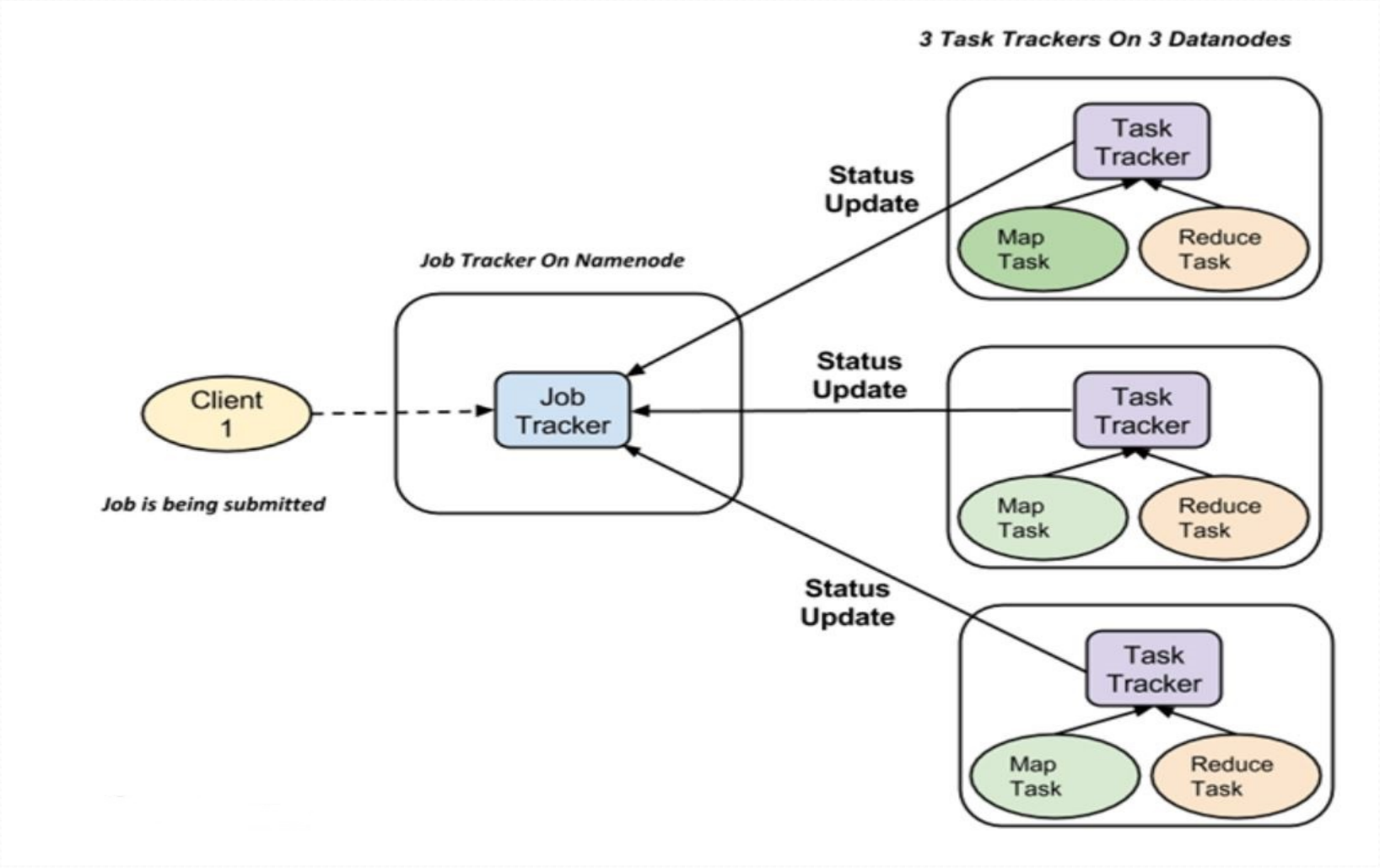


- The whole process goes through four phases of execution namely, splitting, mapping, shuffling, and reducing.
  - **Input Splits:** An input to a MapReduce in Big Data job is divided into fixed-size pieces called **input splits** Input split is a chunk of the input that is consumed by a single map
  - **Mapping :** This is the very first phase in the execution of map-reduce program. In this phase data in each split is passed to a mapping function to produce output values. In our example, a job of mapping phase is to count a number of occurrences of each word from input splits (more details about input-split is given below) and prepare a list in the form of <word, frequency>
  - **Shuffling :** This phase consumes the output of Mapping phase. Its task is to consolidate the relevant records from Mapping phase output. In our example, the same words are clubed together along with their respective frequency.
  - **Reducing :** In this phase, output values from the Shuffling phase are aggregated. This phase combines values from Shuffling phase and returns a single output value. In short, this phase summarizes the complete dataset.

How MapReduce Organizes Work?

- Hadoop divides the job into tasks. There are two types of tasks:
  - **Map tasks** (Splits & Mapping)
  - **Reduce tasks** (Shuffling, Reducing)
- The complete execution process (execution of Map and Reduce tasks, both) is controlled by two types of entities called a
- **Jobtracker**: Acts like a **master** (responsible for complete execution of submitted job)
- **Multiple Task Trackers**: Acts like **slaves,** each of them performing the job
- For every job submitted for execution in the system, there is one **Jobtracker** that resides on **Namenode** and there are **multiple tasktrackers** which reside on **Datanode**.

How MapReduce Organizes Work?



How MapReduce Organizes Work?

- A job is divided into multiple tasks which are then run onto multiple data nodes in a cluster.
- It is the responsibility of job tracker to coordinate the activity by scheduling tasks to run on different data nodes.
- Execution of individual task is then to look after by task tracker, which resides on every data node executing part of the job.
- Task tracker's responsibility is to send the progress report to the job tracker.
- In addition, task tracker periodically sends 'heartbeat' signal to the Jobtracker so as to notify him of the current state of the system.
- Thus job tracker keeps track of the overall progress of each job. In the event of task failure, the job tracker can reschedule it on a different task tracker.

Word Count Example

- Word count is a typical example where Hadoop map reduce developers start their hands on with. This sample map reduce is intended to count the number of occurrences of each word in provided input files.
- **Minimum requirements**
  - Input text file
  - Test VM
  - The mapper, reducer and driver classes to process the input file

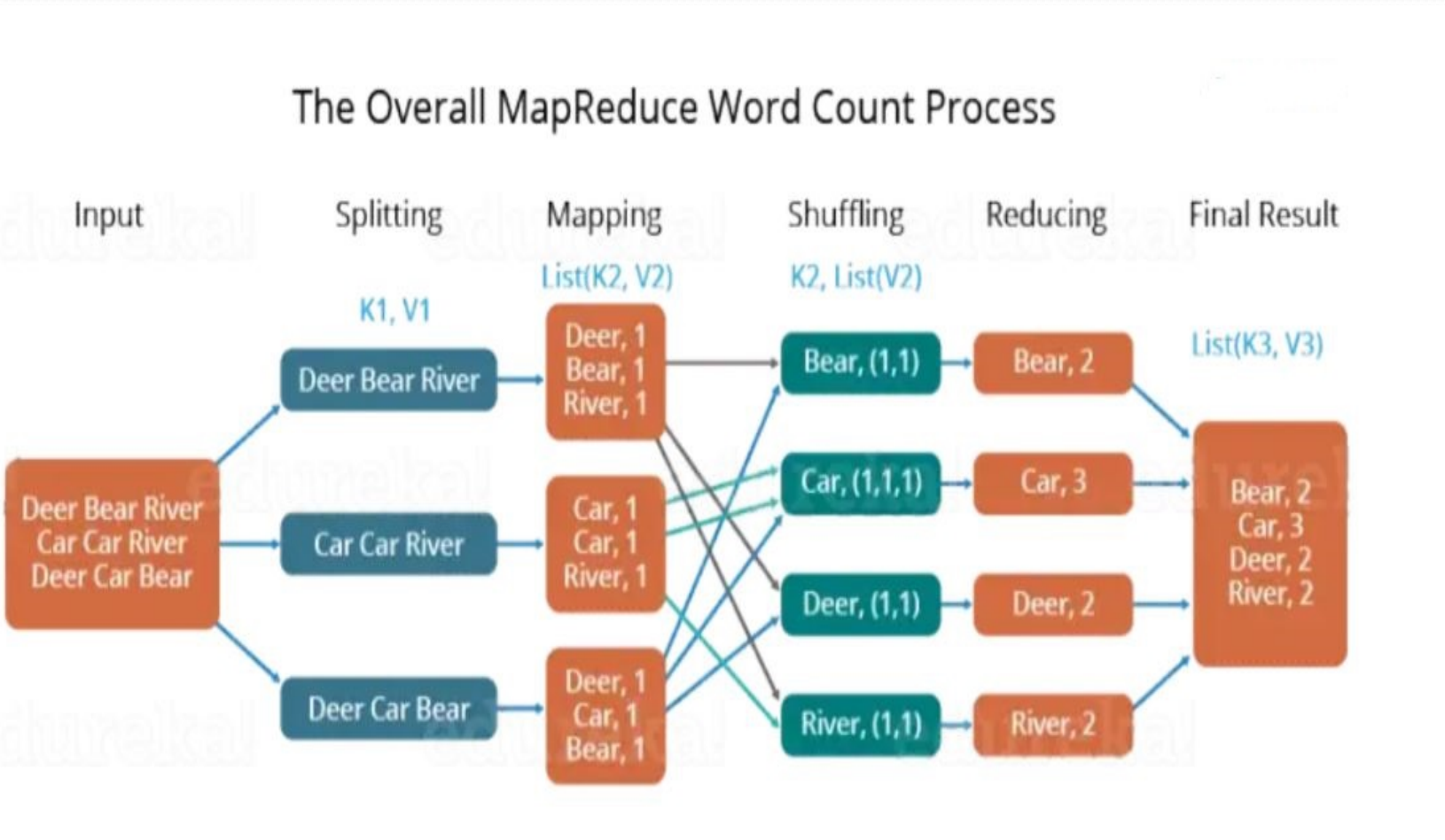
Word Count Example: How it Works

- The work count operation takes place in two stages: a mapper phase and a reducer phase. In mapper phase, first the text is taken into words and we form a key value pair with these words where the key being the word itself and value as its occurrence.
- For example consider the sentence:
  - "tring tring the phone rings"
- In map phase, the sentence would be split as words and form the initial value pair as:
  - <tring, 1>
  - <tring, 1>
  - <the, 1>
  - <phone, 1>
  - <rings, 1>

Word Count Example: How it Works

- In reduce phase, the keys are grouped together and the values of similar keys are added. So there are only one pair of similar keys 'tring', the values of these keys would be added so the output key value pairs would be:
  - <tring, 2>
  - <the, 1>
  - <phone, 1>
  - <rings, 1>
- This would give the number of occurrence of each word in the input. Thus reduce forms an aggregation phase for keys.

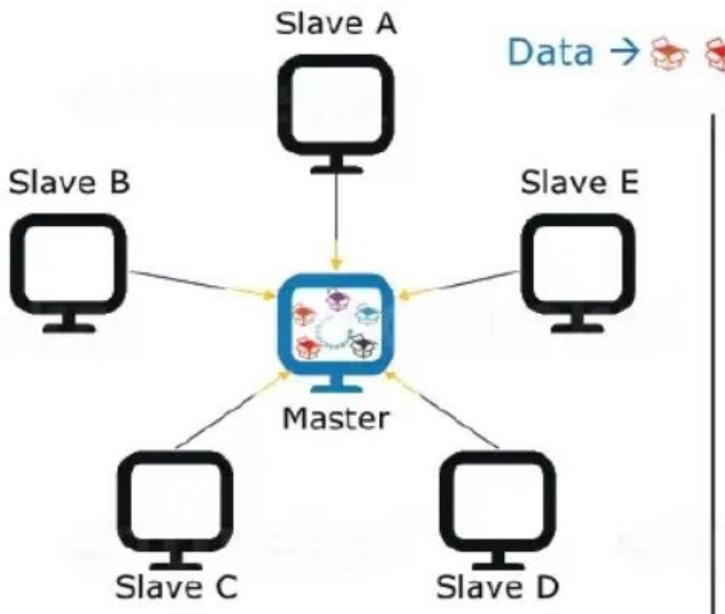
A Word Count Example of MapReduce



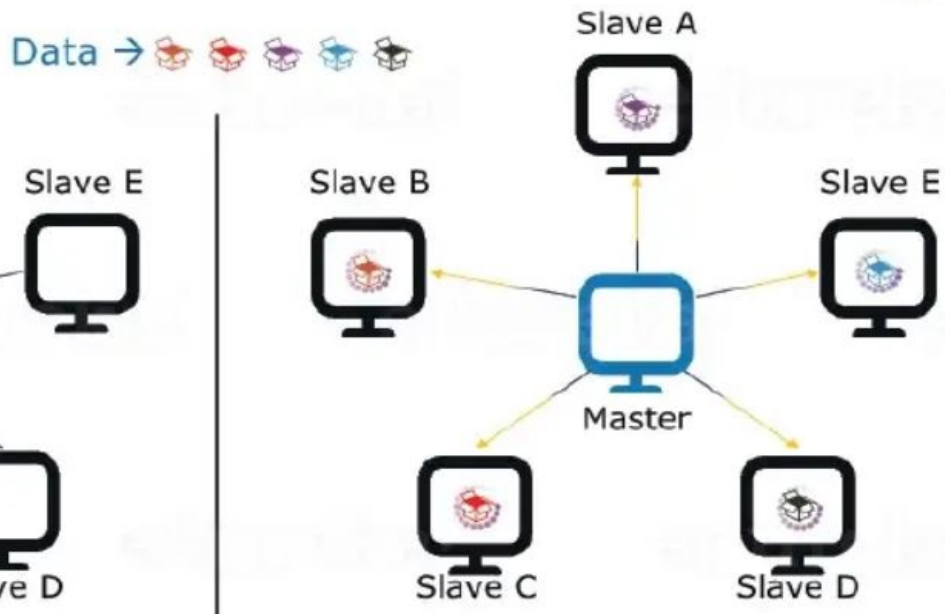
Pseudo Code For Word Count Problem

```
map(String key, String value):
// key: document name
// value: document contents
for each word w in value:
EmitIntermediate(w, "1");
reduce(String key, Iterator values):
// key: a word
// values: a list of counts
int result = 0;
for each v in values:
result += ParseInt(v);
Emit(AsString(result));
```

Traditional Way Vs. MapReduce Way



1. Moving data to the Processing Unit (Traditional Approach)



2. Moving Processing Unit to the data (MapReduce Approach)

Advantages of MapReduce

- **Parallel Processing:** In MapReduce, we are dividing the job among multiple nodes and each node works with a part of the job simultaneously. So, MapReduce is based on Divide and Conquer paradigm which helps us to process the data using different machines. As the data is processed by multiple machines instead of a single machine in parallel, the time taken to process the data gets reduced by a tremendous amount .

Advantages of MapReduce

- **Scalability :** Hadoop is a highly scalable platform and is largely because of its ability that it stores and distributes large data sets across lots of servers. The servers used here are quite inexpensive and can operate in parallel. The processing power of the system can be improved with the addition of more servers. The traditional relational database management systems or RDBMS were not able to scale to process huge data sets.
- **Flexibility :**Hadoop MapReduce programming model offers flexibility to process structure or unstructured data by various business organizations who can use the data and operate on different types of data. Thus, they can generate a business value out of those meaningful and useful data for the business organizations for analysis. Irrespective of the data source, whether it be social media, clickstream, email, etc. Hadoop offers support for a lot of languages used for data processing. Along with all this, Hadoop MapReduce programming allows many applications such as marketing analysis, recommendation system, data warehouse, and fraud detection.

Advantages of MapReduce

- **Fast :** Hadoop distributed file system HDFS is a key feature used in Hadoop, which is basically implementing a mapping system to locate data in a cluster. MapReduce programming is the tool used for data processing, and it is also located in the same server allowing faster processing of data. Hadoop MapReduce processes large volumes of data that is unstructured or semi-structured in less time.
- **Simple Model of Programming :** MapReduce programming is based on a very simple programming model, which basically allows the programmers to develop a MapReduce program that can handle many more tasks with more ease and efficiency. MapReduce programming model is written using Java language is very popular and very easy to learn. It is easy for people to learn Java programming and design a data processing model that meets their business needs.
- **Availability and Resilient Nature :** Hadoop MapReduce programming model processes the data by sending the data to an individual node as well as forward the same set of data to the other nodes residing in the network. As a result, in case of failure in a particular node, the same data copy is still available on the other nodes, which can be used whenever it is required ensuring the availability of data.
  In this way, Hadoop is fault-tolerant. This is a unique functionality offered in Hadoop MapReduce that it is able to quickly recognize the fault and apply a quick fix for an automatic recovery solution.

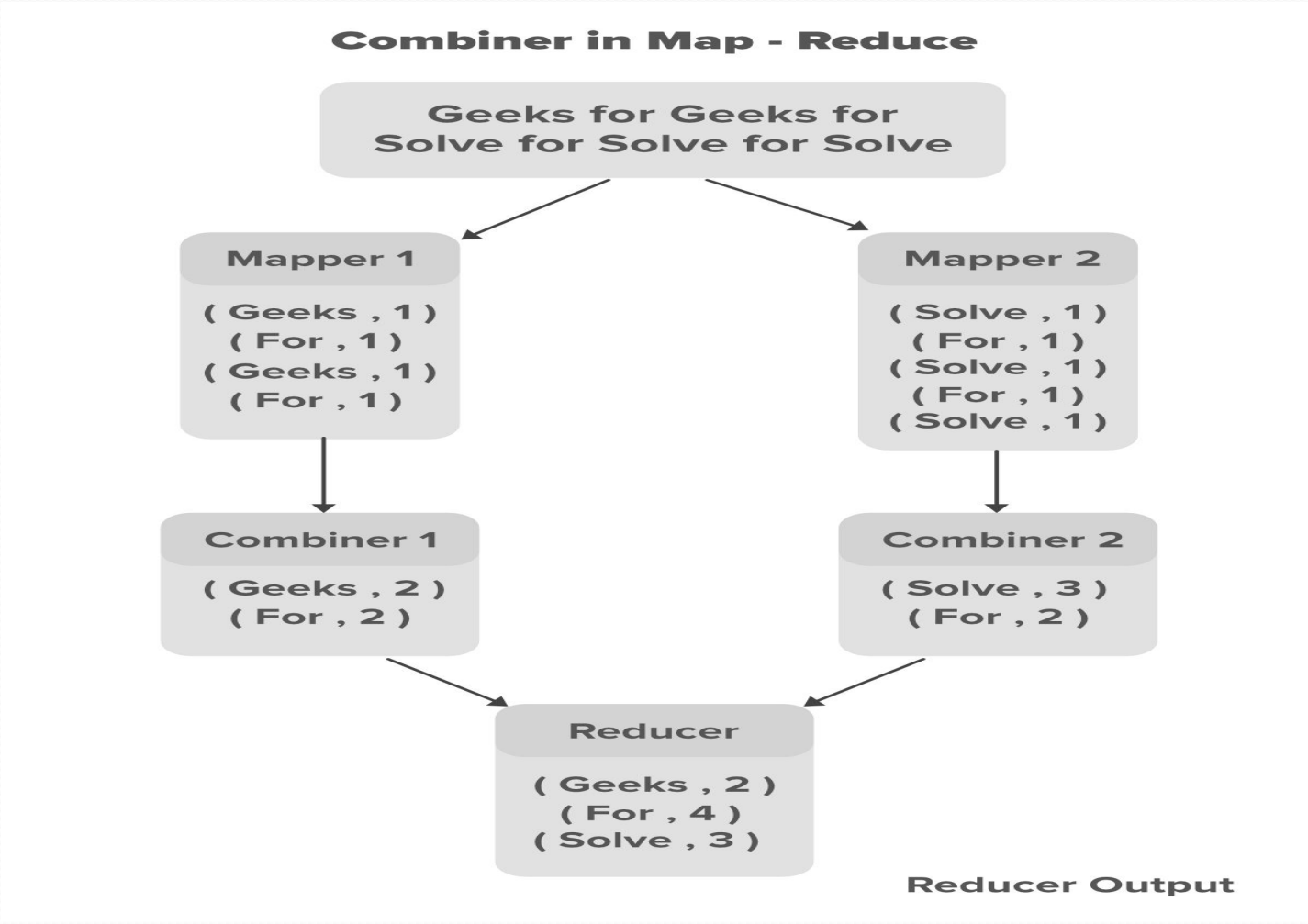
Advantages of MapReduce

- **Security and Authentication :** If any outsider person gets access to all the data of the organization and can manipulate multiple petabytes of the data, it can do much harm in terms of business dealing in operation to the business organization. The MapReduce programming model addresses this risk by working with hdfs and Hbase that allows high security allowing only the approved user to operate on the stored data in the system.
- **Cost-effective Solution :** Such a system is highly scalable and is a very cost-effective solution for a business model that needs to store data growing exponentially in line with current-day requirements. In the case of old traditional relational database management systems, it was not so easy to process the data as with the Hadoop system in terms of scalability. In such cases, the business was forced to downsize the data and further implement classification based on assumptions of how certain data could be valuable to the organization and hence removing the raw data. Here the Hadoop scaleout architecture with MapReduce programming comes to the rescue.

Combiner

- Combiner always works in between Mapper and Reducer. The output produced by the Mapper is the intermediate output in terms of key-value pairs which is massive in size.
- If we directly feed this huge output to the Reducer, then that will result in increasing the Network Congestion. So to minimize this Network congestion we have to put combiner in between Mapper and Reducer.
- These combiners are also known as semi-reducer. It is not necessary to add a combiner to your Map-Reduce program, it is optional.
- Combiner is also a class in our java program like Map and Reduce class that is used in between this Map and Reduce classes.
- Combiner helps us to produce abstract details or a summary of very large datasets. When we process or deal with very large datasets using Hadoop Combiner is very much necessary, resulting in the enhancement of overall performance.

How does combiner work?



- In the above example we can see that two Mappers are containing different data. the main text file is divided into two different Mappers. Each mapper is assigned to process a different line of our data. in our above example, we have two lines of data so we have two Mappers to handle each line. Mappers are producing the intermediate key-value pairs, where the name of the particular word is key and its count is its value.
- The key-value pairs generated by the Mapper are known as the intermediate key-value pairs or intermediate output of the Mapper. Now we can minimize the number of these key-value pairs by introducing a combiner for each Mapper in our program. In our case, we have 4 key-value pairs generated by each of the Mapper. since these intermediate key-value pairs are not ready to directly feed to Reducer because that can increase Network congestion so Combiner will combine these intermediate key-value pairs before sending them to Reducer. The combiner combines these intermediate key-value pairs as per their key. For the above example for data Geeks For Geeks For the combiner will partially reduce them by merging the same pairs according to their key value and generate new key-value pairs .
- With the help of Combiner, the Mapper output got partially reduced in terms of size(key-value pairs) which now can be made available to the Reducer for better performance. Now the Reducer will again Reduce the output obtained from combiners and produces the final output that is stored on HDFS(Hadoop Distributed File System)

Combiners improve the performance of the framework

- Combiners, also known as "local reducers," are a feature in the MapReduce programming model that can improve the performance of the framework by reducing the amount of data that needs to be transferred over the network between the map and reduce tasks.
- In a MapReduce job, the mapper task processes input data and generates a large number of intermediate key-value pairs. These pairs are then grouped by key and sent to the reducer task for further processing. However, if the intermediate data is very large, it can take a significant amount of time and network bandwidth to transfer it from the mapper to the reducer.
- A combiner function is similar to the reduce function and is used to locally aggregate the output of the mapper task before it is sent to the reducer task. This can greatly reduce the amount of data that needs to be transferred over the network, resulting in improved performance and faster job completion times.
- It's important to note that the combiner function must be commutative and associative, which means that the order in which the intermediate data is processed should not affect the final outcome.

MapReduce : Data Locality

- Instead of moving data to the processing unit, we are moving the processing unit to the data in the MapReduce Framework. In the traditional system, we used to bring data to the processing unit and process it. But, as the data grew and became very huge, bringing this huge amount of data to the processing unit posed the following issues:
  - Moving huge data to processing is costly and deteriorates the network performance.
  - Processing takes time as the data is processed by a single unit which becomes the bottleneck.
  - Master node can get over-burdened and may fail.

MapReduce : Locality

- Now, MapReduce allows us to overcome the above issues by bringing the processing unit to the data. So, as you can see in the above image that the data is distributed among multiple nodes where each node processes the part of the data residing on it. This allows us to have the following advantages:
  - It is very cost effective to move the processing unit to the data.
  - The processing time is reduced as all the nodes are working with their part of the data in parallel.
  - Every node gets a part of the data to process and therefore, there is no chance of a node getting overburdened.

MapReduce : Locality

- In MapReduce, data locality refers to the ability of the system to schedule tasks on the same node where the data is stored, in order to minimize the amount of data that needs to be transferred over the network.
- In Hadoop MapReduce, data locality is achieved through the use of data blocks and the Hadoop Distributed File System (HDFS). HDFS stores data in large blocks (typically 128MB) and distributes these blocks across the nodes in the cluster. When a MapReduce job is executed, the JobTracker schedules tasks on the same node where the data blocks are stored, whenever possible. This reduces the amount of data that needs to be transferred over the network, and improves the performance of the job.

MapReduce : Locality

- To achieve data locality, the JobTracker uses a feature called "rack-awareness" which makes the JobTracker aware of the topology of the cluster, including the racks and switches that interconnect the nodes. This allows the JobTracker to schedule tasks on the same rack as the data, whenever possible, which reduces network traffic and improves performance.
- Additionally, to optimize the data locality, the Hadoop scheduler uses a feature called “Data Locality Cost” that can be customized to assign different weights to data-local, rack-local and off-rack tasks, based on the user's cluster topology, network infrastructure, and job requirements.
- Data locality is an important concept in MapReduce, and it is achieved through the use of data blocks, HDFS and the JobTracker, which helps to minimize the amount of data transferred over the network, and improves the performance and efficiency of the job.

Fault Tolerance in Map Reduce

- **Machine Failure**
  - The master pings workers regularly to detect the failures.
  - If no response is returned by the worker, the worker is considered to be faulty.
  - When the map worker failure is encountered, the map task to that worker is reset to idle and resheduled to another map worker. The reduce workers are notified about this resheduling.
  - When the reduce worker failure is encountered, the task that are not completed i.e. in progress task are reset to idle and rescheduled to another reduce worker.
  - In case of failure of the master, the complete MapReduce task is aborted and is notified to the client.

Fault Tolerance in Map Reduce

- Fault tolerance in MapReduce refers to the ability of the system to continue processing and producing correct results even in the presence of failures or errors.
- In Hadoop MapReduce, fault tolerance is achieved through a combination of techniques, such as:
  - Data replication: Input data is replicated across multiple nodes in the cluster to ensure that if a node fails, the data is still available on other nodes.
  - Task tracking: The JobTracker keeps track of the status of tasks and can reschedule a task on a different node if the original node fails.
  - Speculative execution: If a task is running slower than expected, the JobTracker may start a second instance of the task on a different node. The results from the faster task are used, and the slower task is discarded.

Fault Tolerance in Map Reduce

- Checkpointing: The JobTracker periodically saves the state of the job, so that if the JobTracker fails, the job can be resumed from the last checkpoint.
- Backup Tasktracker: The JobTracker maintains a backup TaskTracker for each TaskTracker. The backup is used to restart the original TaskTracker if it fails.
- Re-execution of failed tasks: If a task fails, it is re-executed on a different node.
- Data Integrity: The data is checksummed to ensure that it remains consistent and accurate even in the presence of faults.

- Fault tolerance in MapReduce is important to ensure that large-scale data processing jobs can continue to operate even in the presence of failures, which is common in distributed systems. By using these techniques, Hadoop MapReduce can continue to process and produce correct results, even in the presence of hardware or software failures.

MapReduce: scalability

- Scalability in MapReduce refers to the ability of the system to handle increasing amounts of data and processing power by adding more resources to the cluster.
- Hadoop MapReduce is designed to be highly scalable, allowing it to process and analyze large amounts of data distributed across multiple nodes in a cluster. This is achieved by breaking the data into smaller, manageable chunks and distributing the processing tasks across multiple nodes in the cluster, in parallel.
- One of the key features of Hadoop MapReduce that enables scalability is its use of data replication and distributed computing. Data is replicated across multiple nodes in the cluster, which allows for the processing of the data in parallel, and reduces the risk of data loss in case of a node failure.

MapReduce: Scalability

- Another feature that enables scalability in Hadoop MapReduce is the ability to add more nodes to the cluster as needed. As the volume of data grows, more nodes can be added to the cluster to handle the increased processing power required. This allows Hadoop MapReduce to handle large data sets and increasing amounts of processing power without the need to make significant changes to the system.
- Hadoop ecosystem provides tools like Apache YARN and Apache Mesos, that can be used to schedule and manage the resources of the cluster, allowing to handle multiple concurrent data processing jobs and enables scalability to handle big data use cases.
- Scalability in MapReduce is the ability of the system to handle increasing amounts of data and processing power by adding more resources to the cluster, and it's achieved through the use of data replication and distributed computing, the ability to add more nodes to the cluster as needed, and the use of tools like YARN and Mesos to manage resources of the cluster.

MapReduce: Worker Failure

- In Hadoop MapReduce, worker failure refers to the failure of a node or task tracker that is running a task as part of a MapReduce job.
- When a worker failure occurs, the JobTracker, which is responsible for managing and coordinating the tasks of a MapReduce job, takes several actions to ensure that the job can continue to run and produce correct results. These actions include:
  - Task reassignment: The JobTracker will reassign the failed task to another node in the cluster, and the task will be re-executed on the new node.
  - Speculative execution: In some cases, the JobTracker may start a second instance of a task on another node, even if the original task is still running. The results from the faster task are used, and the slower task is discarded.
  - Backup Tasktracker: The JobTracker maintains a backup TaskTracker for each TaskTracker. The backup is used to restart the original TaskTracker if it fails.
  - Checkpointing: The JobTracker periodically saves the state of the job, so that if the JobTracker or a task tracker fails, the job can be resumed from the last checkpoint.

MapReduce: Worker Failure

- These actions help to ensure that the job can continue to run, and that the data is processed correctly, even in the event of a worker failure. Additionally, the data replication of HDFS also helps to minimize the data loss, in case of a node failure.
- Worker failure in MapReduce refers to the failure of a node or task tracker that is running a task as part of a MapReduce job. The JobTracker takes several actions to ensure that the job can continue to run and produce correct results, such as task reassignment, speculative execution, backup task tracker, and checkpointing, which helps to minimize the impact of a worker failure on the job's completion.

MapReduce: Master Failure

- In Hadoop MapReduce, master failure refers to the failure of the JobTracker, which is the master node that coordinates and manages the tasks of a MapReduce job.
- When a master failure occurs, the following actions are taken to ensure that the job can continue to run and produce correct results:
  - Automatic failover: The Hadoop cluster has a built-in mechanism that detects the failure of the JobTracker and automatically triggers a failover to a backup JobTracker. This ensures that the job can continue to run without interruption.
  - Check pointing: The JobTracker periodically saves the state of the job, so that if the JobTracker fails, the job can be resumed from the last checkpoint.

MapReduce: Master Failure

- Task reassignment: The backup JobTracker reassigns the tasks that were being managed by the failed JobTracker to other nodes in the cluster, and the tasks are re-executed.
- Data replication: Input data is replicated across multiple nodes in the cluster to ensure that if a node fails, the data is still available on other nodes.

- These actions help to ensure that the job can continue to run, and that the data is processed correctly, even in the event of a master failure. Additionally, the data replication of HDFS also helps to minimize the data loss, in case of a node failure.
- Master failure in MapReduce refers to the failure of the JobTracker, which is the master node that coordinates and manages the tasks of a MapReduce job. The Hadoop cluster has a built

Straggler

- In Hadoop MapReduce, a straggler refers to a task that is running slower than expected, compared to the other tasks in the same job. A straggler task can cause a bottleneck in the job's performance, as it may slow down the overall completion time of the job.
- There are several reasons why a task may become a straggler, such as:
  - Data skew: When a task is assigned a disproportionate amount of data to process, it may take longer to complete than other tasks.
  - Machine failure: A task running on a node with hardware issues or high resource utilization may run slower than expected.
  - Network issues: A task running on a node that is far away from the data it needs to process, may experience high network latency and run slower.

Straggler

- To mitigate the impact of stragglers, Hadoop MapReduce uses several techniques such as:
  - Speculative execution: The JobTracker may start a second instance of the task on a different node. The results from the faster task are used, and the slower task is discarded.
  - Task reassignment: The JobTracker can reassign the task to a different node in the cluster, and the task will be re-executed on the new node.
  - Data locality: The JobTracker schedules tasks on local disk

Parallel Efficiency of Map-Reduce

- The parallel efficiency of MapReduce refers to how well the system is able to utilize the available resources to process data in parallel. A high parallel efficiency means that the system is able to effectively use all of the resources to process data quickly and efficiently, while a low parallel efficiency means that the system is not utilizing resources effectively, leading to slower processing times.
- There are several factors that can affect the parallel efficiency of a MapReduce job:
  - Data skew: If the data is not evenly distributed among the tasks, some tasks may have to process much more data than others, leading to uneven task completion times and poor parallel efficiency.
  - Network congestion: If the network is congested, data may not be able to be transferred quickly between nodes, slowing down the processing of tasks and reducing parallel efficiency.

Parallel Efficiency of Map-Reduce

- Resource contention: If multiple tasks are competing for the same resources, such as CPU or memory, this can lead to slower task completion times and reduced parallel efficiency.
- Disk I/O bottleneck: If the disk I/O is slow, it can take longer for tasks to read or write data, leading to slower task completion times and reduced parallel efficiency.
- To improve parallel efficiency, it's important to monitor the job progress and identify the bottlenecks that are causing slow performance, and then take steps to address them. For example, data skew can be addressed by redistributing the data more evenly among the tasks, and network congestion

Skipping bad records

- In Hadoop MapReduce, skipping bad records refers to the ability of the system to continue processing a job even if it encounters invalid or malformed data. This can be achieved by implementing custom input and output formats, and by using the provided methods in the org.apache.hadoop.mapreduce package that allow you to skip bad records and continue processing the job.

References

- https://medium.com/@shaistha24/functional-programming-vs-object-oriented-programming-oop-which-is-better-82172e53a526
- https://romain-b.medium.com/pros-and-cons-of-imperative-and-functional-programming-paradigms-to-solve-the-same-technical-1511ac2f654c
- https://medium.com/front-end-weekly/imperative-versus-declarative-code-whats-the-difference-adc7dd6c8380
- https://medium.com/edureka/mapreduce-tutorial-3d9535ddbe7c
- https://www.edureka.co/blog/mapreduce-tutorial/#:~:text=MapReduce%20is%20a%20programming%20framework,mapper%20phase%20has%20been%20completed
- https://www.geeksforgeeks.org/mapreduce-combiners/
- https://www.educba.com/what-is-mapreduce/
- https://www.spiceworks.com/tech/big-data/articles/what-is-map-reduce/