1. Hadoop MapReduce

This **Hadoop MapReduce** describes all the concepts of **Hadoop** MapReduce in great details. In this, we will understand **what is MapReduce** and how it works, what is Mapper, Reducer, shuffling, and sorting, etc. This Hadoop MapReduce also covers internals of MapReduce, **Dataflow**, architecture, and Data locality as well. So, let's get started with the Hadoop MapReduce.

2. What is MapReduce?

MapReduce is the processing layer of **Hadoop**. MapReduce programming model is designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks. You need to put business logic in the way MapReduce works and rest things will be taken care by the framework. Work (complete job) which is submitted by the user to master is divided into small works (tasks) and assigned to slaves.

MapReduce programs are written in a particular style influenced by functional programming constructs, specifical idioms for processing lists of data. Here in MapReduce, we get inputs from a list and it converts it into output which is again a list. It is the heart of Hadoop. Hadoop is so much powerful and efficient due to MapReduce as here parallel processing is done.

This is what MapReduce is in Big Data. In the next step of MapReduce we have MapReduce Process, MapReduce dataflow how MapReduce divides the work into sub-work, why MapReduce is one of the best paradigms to process data:

learn Big data Technologies and Hadoop concepts.

i. High-level Understanding of Hadoop MapReduce

Now in this Hadoop MapReduce let's understand the MapReduce basics, at a high level how MapReduce looks like, what, why and how MapReduce works?

Map-Reduce divides the work into small parts, each of which can be done in parallel on the cluster of servers. A problem is divided into a large number of smaller problems each of which is processed to give individual outputs. These individual outputs are further processed to give final output.

Hadoop Map-Reduce is scalable and can also be used across many computers. Many small machines can be used to process jobs that could not be processed by a large

machine. Next in the MapReduce we will see some important MapReduce Terminologies'.

ii. Apache MapReduce Terminologies

Let's now understand different terminologies and concepts of MapReduce, what is Map and Reduce, what is a job, task, task attempt, etc.
Map-Reduce is the data processing component of Hadoop. Map-Reduce programs transform lists of input data elements into lists of output data elements. A Map-Reduce program will do this twice, using two different list processing idioms-

- Map
- Reduce

In between Map and Reduce, there is small phase called **Shuffle** and **Sort** in MapReduce.
Let's understand basic terminologies used in Map Reduce.

- **What is a MapReduce Job**?

MapReduce Job or a A "full program" is an execution of
a **Mapper** and **Reducer** across a data set. It is an execution of 2 processing layers i.e mapper and reducer. A MapReduce job is a work that the client wants to be performed. It consists of the input data, the MapReduce Program, and configuration info. So client needs to submit input data, he needs to write Map Reduce program and set the configuration info (These were provided during **Hadoop setup** in the configuration file and also we specify some configurations in our program itself which will be specific to our **map reduce job)**.

iii. Map Abstraction

Let us understand the abstract form of **Map** in MapReduce, the first phase of MapReduce paradigm, what is a map/mapper, what is the input to the mapper, how it processes the data, what is output from the mapper?
The **map** takes **key/value pair** as input. Whether data is in structured or unstructured format, framework converts the incoming data into key and value.

- Key is a reference to the input value.
- Value is the data set on which to operate.

Map Processing:

- A function defined by user – user can write custom business logic according to his need to process the data.
- Applies to every value in value input.

Map produces a new list of key/value pairs:

- An output of Map is called intermediate output.
- Can be the different type from input pair.
- An output of map is stored on the local disk from where it is shuffled to reduce nodes.

Next in Hadoop MapReduce is the Hadoop Abstraction

iv. Reduce Abstraction

Now let's discuss the second phase of MapReduce – **Reducer** in this MapReduce, what is the input to the reducer, what work reducer does, where reducer writes output? **Reduce** takes intermediate Key / Value pairs as input and processes the output of the mapper. Usually, in the reducer, we do aggregation or summation sort of computation.

- Input given to reducer is generated by Map (intermediate output)
- Key / Value pairs provided to reduce are sorted by key

Reduce processing:

- A function defined by user – Here also user can write custom business logic and get the final output.
- Iterator supplies the values for a given key to the Reduce function.

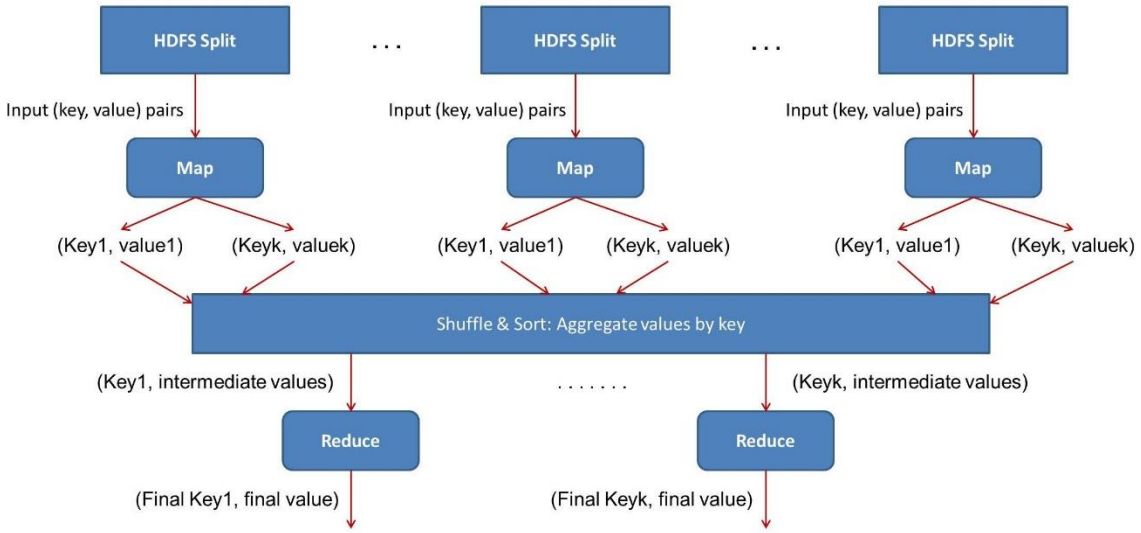
Reduce produces a final list of key/value pairs:

- An output of Reduce is called Final output.
- It can be a different type from input pair.
- An output of Reduce is stored in **HDFS**.

Let us understand in this Hadoop MapReduce How Map and Reduce work together.

v. How Map and Reduce work Together?

Let us understand how Hadoop Map and Reduce work together?

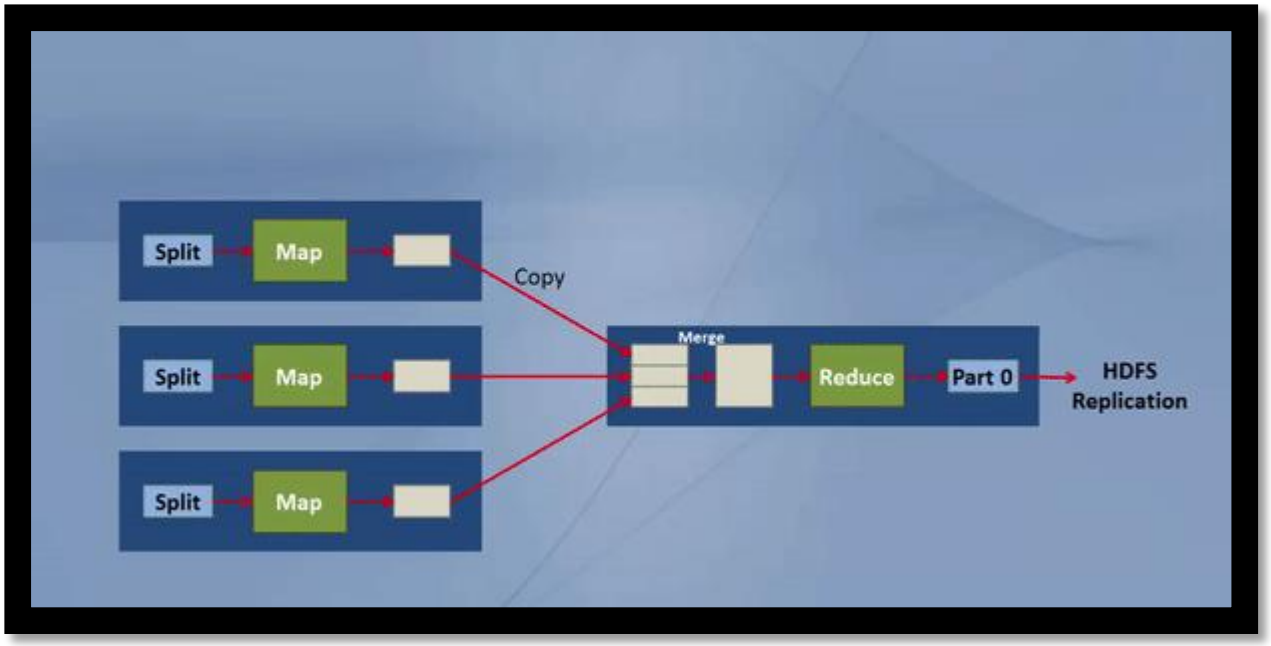


Hadoop MapReduce: Combined working of Map and Reduce

Input data given to **mapper** is processed through user defined function written at mapper. All the required complex business logic is implemented at the mapper level so that heavy processing is done by the mapper in parallel as the number of mappers is much more than the number of **reducers**. Mapper generates an output which is intermediate data and this output goes as input to reducer.
This intermediate result is then processed by user defined function written at reducer and final output is generated. Usually, in reducer very light processing is done. This final output is stored in **HDFS** and replication is done as usual.

vi. MapReduce DataFlow

Now let's understand in this Hadoop MapReduce complete end to end data flow of MapReduce, how input is given to the mapper, how mappers process data, where mappers write the data, how data is shuffled from mapper to reducer nodes, where reducers run, what type of processing should be done in the reducers?



Hadoop MapReduce: Hadoop MapReduce Dataflow Process

As seen from the diagram of MapReduce workflow in Hadoop, the square block is a **slave**. There are 3 slaves in the figure. On all 3 slaves mappers will run, and then a reducer will run on any 1 of the slaves. For simplicity of the figure, the reducer is shown on a different machine but it will run on mapper node only.
Let us now discuss the map phase:

An input to a mapper is 1 **block** at a time. (Split = block by default) An output of mapper is written to a local disk of the machine on which mapper is running. Once the map finishes, this intermediate output travels to reducer nodes (node where reducer will run).

Reducer is the second phase of processing where the user can again write his custom business logic. Hence, an output of reducer is the final output written to **HDFS**.

By default, on a slave, 2 mappers run at a time which can also be increased as per the requirements. It depends again on factors like DataNode hardware, block size, machine configuration etc. We should not increase the number of mappers beyond the certain limit because it will decrease the **performance**.

Mapper in Hadoop MapReduce writes the output to the local disk of the machine it is working. This is the temporary data. An output of mapper is also called intermediate output. All mappers are writing the output to the local disk. As First mapper finishes,

data (output of the mapper) is traveling from mapper node to reducer node. Hence, this movement of output from mapper node to reducer node is called **shuffle**.

Reducer is also deployed on any one of the DataNode only. An output from all the mappers goes to the reducer. All these outputs from different mappers are merged to form input for the reducer. This input is also on local disk. Reducer is another processor where you can write custom business logic. It is the second stage of the processing. Usually to reducer we write aggregation, summation etc. type of functionalities. Hence, Reducer gives the final output which it writes on HDFS.

Map and reduce are the stages of processing. They run one after other. After all, mappers complete the processing, then only reducer starts processing.

Though 1 block is present at 3 different locations by default, but framework allows only 1 mapper to process 1 block. So only 1 mapper will be processing 1 particular block out of 3 replicas. The output of every mapper goes to every reducer in the cluster i.e every reducer receives input from all the mappers. Hence, framework indicates reducer that whole data has processed by the mapper and now reducer can process the data.

An output from mapper is partitioned and filtered to many partitions by the partitioner. Each of this partition goes to a reducer based on some conditions. Hadoop works with key value principle i.e mapper and reducer gets the input in the form of key and value and write output also in the same form. Follow this link to **learn How Hadoop works internally?** MapReduce Dataflow is the most important topic in this MapReduce. If you have any query reading this topic or ant topic in the MapReduce, just drop a comment and we will get back to you. Now, let us move ahead in this MapReduce with the Data Locality principle.

viii. Data Locality in MapReduce

Let's understand **what is data locality**, how it **optimizes Map Reduce jobs**, how data locality improves **job performance?**

"*Move computation close to the data rather than data to computation*". A computation requested by an application is much more efficient if it is executed near the data it operates on. This is especially true when the size of the data is very huge. This minimizes network congestion and increases the throughput of the system. The assumption is that it is often better to move the computation closer to where the data is present rather than moving the data to where the application is running. Hence,

HDFS provides interfaces for applications to move themselves closer to where the data is present.

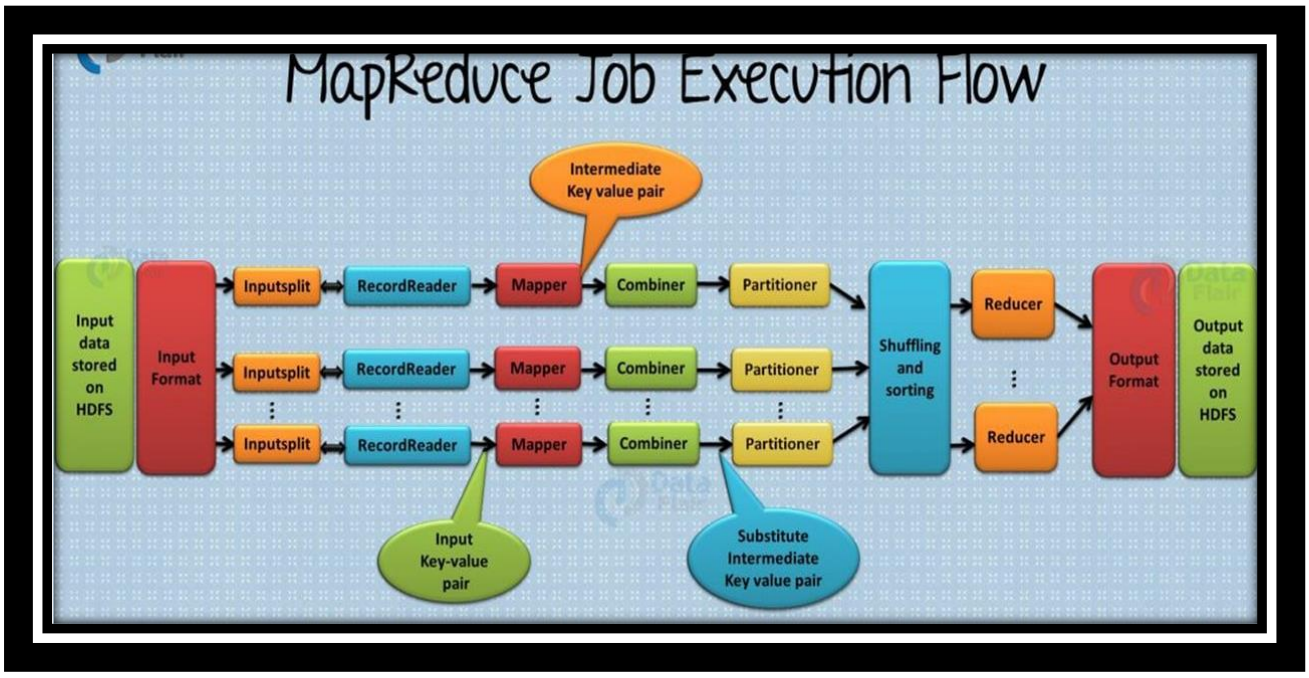
Since Hadoop works on huge volume of data and it is not workable to move such volume over the network. Hence it has come up with the most innovative principle of moving algorithm to data rather than data to algorithm. This is called **data locality**.

How Hadoop MapReduce Works – MapReduce

1. Objective

MapReduce is the core component of **Hadoop** that process huge amount of data in parallel by dividing the work into a set of independent tasks. In MapReduce data flow in step by step from Mapper to Reducer. In this , we are going to cover how Hadoop MapReduce works internally?

This blog on Hadoop MapReduce data flow will provide you the complete MapReduce data flow chart in Hadoop. The covers various phases of MapReduce job execution such as **Input Files**, **InputFormat in Hadoop**, **InputSplits**, **RecordReader**, **Mapper**, **Combiner**, **Partitioner**, **Shuffling and Sorting**, **Reducer**, **RecordWriter and OutputFormat** in detail. We will also learn How Hadoop MapReduce works with the help of all these phases.



How Hadoop MapReduce Works – MapReduce

2. What is MapReduce?

MapReduce is the data processing layer of Hadoop. It is a software framework for easily writing applications that process the vast amount of structured and unstructured data stored in the **Hadoop Distributed Filesystem (HDFS)**. It processes the huge amount of data in parallel by dividing the job (submitted job) into a set of independent tasks (sub-job). By this parallel processing speed and reliability of cluster is improved. We just need to put the custom code (business logic) in the way map reduce works and rest things will be taken care by the engine.

3. How Hadoop MapReduce Works?

In Hadoop, MapReduce works by breaking the data processing into two phases: Map phase and Reduce phase. The map is the first phase of processing, where we specify all the complex logic/business rules/costly code. Reduce is the second phase of processing, where we specify light-weight processing like aggregation/summation.

4. MapReduce Flow Chart

Now let us see How Hadoop MapReduce works by understanding the end to end Hadoop MapReduce job execution flow with components in detail:

4.1. Input Files

The data for a MapReduce task is stored in **input files**, and input files typically lives in **HDFS**. The format of these files is arbitrary, while line-based log files and binary format can also be used.

4.2. InputFormat

Now, **InputFormat** defines how these input files are split and read. It selects the files or other objects that are used for input. InputFormat creates InputSplit.

4.3. InputSplits

It is created by InputFormat, logically represent the data which will be processed by an individual **Mapper** (We will understand mapper below). One map task is created for each split; thus the number of map tasks will be equal to the number of InputSplits. The split is divided into records and each record will be processed by the mapper.

4.4. RecordReader

It communicates with the **InputSplit** in Hadoop MapReduce and converts the data into key-value pairs suitable for reading by the mapper. By default, it uses TextInputFormat for converting data into a key-value pair. RecordReader communicates with the InputSplit until the file reading is not completed. It assigns byte offset (unique number) to each line present in the file. Further, these key-value pairs are sent to the mapper for further processing.

4.5. Mapper

It processes each input record (from RecordReader) and generates new key-value pair, and this key-value pair generated by Mapper is completely different from the input pair. The output of Mapper is also known as intermediate output which is written to the local disk. The output of the Mapper is not stored on HDFS as this is temporary data and writing on HDFS will create unnecessary copies (also HDFS is a high latency system). Mappers output is passed to the combiner for further process

4.6. Combiner

The combiner is also known as 'Mini-reducer'. Hadoop MapReduce Combiner performs local aggregation on the mappers' output, which helps to minimize the data transfer between mapper and **reducer** (we will see reducer below). Once the combiner functionality is executed, the output is then passed to the partitioner for further work.

4.7. Partitioner

Hadoop MapReduce, **Partitioner** comes into the picture if we are working on more than one reducer (for one reducer partitioner is not used).

Partitioner takes the output from combiners and performs partitioning. Partitioning of output takes place on the basis of the key and then sorted. By hash function, key (or a subset of the key) is used to derive the partition.

According to the key value in MapReduce, each combiner output is partitioned, and a record having the same key value goes into the same partition, and then each partition

is sent to a reducer. Partitioning allows even distribution of the map output over the reducer.

4.8. Shuffling and Sorting

Now, the output is Shuffled to the reduce node (which is a normal slave node but reduce phase will run here hence called as reducer node). The shuffling is the physical movement of the data which is done over the network. Once all the mappers are finished and their output is shuffled on the reducer nodes, then this intermediate output is merged and sorted, which is then provided as input to reduce phase.

4.9. Reducer

It takes the set of intermediate key-value pairs produced by the mappers as the input and then runs a reducer function on each of them to generate the output. The output of the reducer is the final output, which is stored in HDFS.

4.10. RecordWriter

It writes these output key-value pair from the Reducer phase to the output files.

4.11. OutputFormat

The way these output key-value pairs are written in output files by RecordWriter is determined by the OutputFormat. OutputFormat instances provided by the Hadoop are used to write files in HDFS or on the local disk. Thus the final output of reducer is written on HDFS by OutputFormat instances.

Hence, in this manner, a Hadoop MapReduce works over the cluster.

Hadoop Mapper – 4 Steps Learning to MapReduce Mapper

1. Hadoop Mapper – Objective

Mapper task is the first phase of processing that processes each input record (from Record Reader) and generates an intermediate key-value pair. **Hadoop** Mapper store intermediate-output on the local disk. In this Hadoop mapper , we will try to answer

what is a **MapReduce** Mapper how to generate **key-value pair** in Hadoop, what is InputSplit and RecordReader in Hadoop, how mapper works in Hadoop.

We will also discuss the number of mappers in Hadoop MapReduce for running any program and how to calculate the number of Hadoop mappers required for a given data. After this you can refer our on **MapReduce Reducer** to gain complete insights on both mapper and reducer in Hadoop. It is really fun to understand mapper and reducer in Hadoop.

We will also discuss the number of mappers in Hadoop MapReduce for running any program and how to calculate the number of mappers required for a given data.

2. What is Hadoop Mapper?

Hadoop Mapper task processes each input record and it generates a new <key, value> pairs. The <key, value> pairs can be completely different from the input pair. In mapper task, the output is the full collection of all these <key, value> pairs. Before writing the output for each mapper task, **partitioning** of output take place on the basis of the key and then **sorting** is done. This partitioning specifies that all the values for each key are grouped together.
MapReduce frame generates one map task for each InputSplit (we will discuss it below.) generated by the **InputFormat** for the job.

Mapper only understands <key, value> pairs of data, so before passing data to the mapper, data should be first converted into <key, value> pairs.

3. How is key value pair generated in Hadoop?

Let us now discuss the key-value pair generation in Hadoop.

- **InputSplit –** It is the logical representation of data. It describes a unit of work that contains a single map task in a MapReduce program. Learn about InputSplit in detail.
- **RecordReader –** It communicates with the InputSplit and it converts the data into key-value pairs suitable for reading by the Mapper. By default, it uses TextInputFormat for converting data into the key-value pair. RecordReader communicates with the Inputsplit until the file reading is not completed.

4. How does Hadoop Mapper work?

Let us now see the mapper process in Hadoop.
InputSplits converts the physical representation of the block into logical for the Hadoop mapper. To read the 100MB file, two InputSlits are required. One InputSplit is created for each block and one RecordReader and one mapper are created for each InputSplit.

InputSpits do not always depend on the number of blocks, we can customize the number of splits for a particular file by setting *mapred.max.split.size* property during job execution.

RecordReader's responsibility is to keep reading/converting data into key-value pairs until the end of the file. Byte offset (unique number) is assigned to each line present in the file by RecordReader. Further, this key-value pair is sent to the mapper. The output of the mapper program is called as intermediate data (key-value pairs which are understandable to reduce).

5. How many map tasks in Hadoop?

In this section of this Hadoop mapper , we are going to discuss the number of mapper in Hadoop MapReduce for running any program and how to calculate the number of mappers required for a given data?
The total number of **blocks** of the input files handles the number of map tasks in a program. For maps, the right level of parallelism is around 10-100 maps/node, although for CPU-light map tasks it has been set up to 300 maps. Since task setup takes some time, so it's better if the maps take at least a minute to execute.

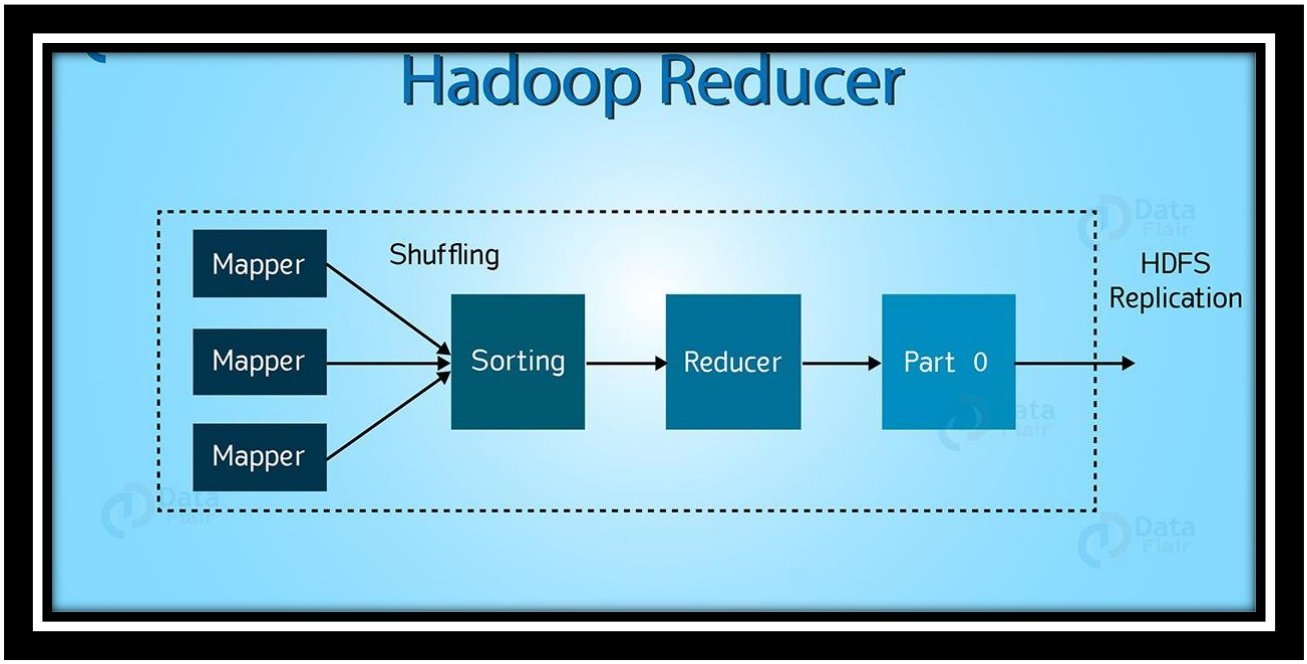
For example, if we have a block size of 128 MB and we expect 10TB of input data, we will have 82,000 maps. Thus, the InputFormat determines the number of maps.
Hence, **No. of Mapper= {(total data size)/ (input split size)}**
For example, if data size is 1 TB and InputSplit size is 100 MB then,
No. of Mapper= (1000*1000)/100= 10,000

Hadoop Reducer – 3 Steps learning for MapReduce Reducer

1. Hadoop Reducer – Objective

In **Hadoop**, **Reducer** takes the output of the **Mapper** (intermediate key-value pair) process each of them to generate the output. The output of the reducer is the final output, which is stored in HDFS. Usually, in the Hadoop Reducer, we do aggregation or summation sort of computation.

In this Hadoop Reducer, we will answer what is Reducer in Hadoop MapReduce, what are the different phases of **Hadoop MapReduce** Reducer, shuffling and sorting in Hadoop, Hadoop reduce phase, functioning of Hadoop reducer class. We will also discuss how many reducers are required in Hadoop and how to change the number of reducers in Hadoop MapReduce.



Hadoop Reducer – 3 Steps learning for MapReduce Reducer

2. What is Hadoop Reducer?

Let's now discuss what is Reducer in MapReduce first.

The Reducer process the output of the mapper. After processing the data, it produces a new set of output. At last **HDFS** stores this output data.

Hadoop Reducer takes a set of an intermediate **key-value pair** produced by the mapper as the input and runs a Reducer function on each of them. One can

aggregate, filter, and combine this data (key, value) in a number of ways for a wide range of processing. Reducer first processes the intermediate values for particular key generated by the map function and then generates the output (zero or more key-value pair).

One-one mapping takes place between keys and reducers. Reducers run in parallel since they are independent of one another. The user decides the number of reducers. By default number of reducers is 1.

3. Phases of MapReduce Reducer

As you can see in the diagram at the top, there are 3 phases of Reducer in Hadoop MapReduce. Let's discuss each of them one by one-

3.1. Shuffle Phase of MapReduce Reducer

In this phase, the sorted output from the mapper is the input to the Reducer. In Shuffle phase, with the help of HTTP, the framework fetches the relevant partition of the output of all the mappers.

3.2. Sort Phase of MapReduce Reducer

In this phase, the input from different mappers is again sorted based on the similar keys in different Mappers. The shuffle and sort phases occur concurrently.

3.3. Reduce Phase

In this phase, after shuffling and sorting, reduce task aggregates the key-value pairs. The *OutputCollector.collect()* method, writes the output of the reduce task to the Filesystem. Reducer output is not sorted.

4. MapReduce Number of Reducers

In this section of Hadoop Reducer, we will discuss how many number of Mapreduce reducers are required in MapReduce and how to change the Hadoop reducer number in MapReduce?
With the help of *Job.setNumreduceTasks(int)* the user set the number of reducers for the job. The right number of reducers are 0.95 or 1.75 multiplied by (<no. of nodes> * <no. of the maximum container per node>).

With 0.95, all reducers immediately launch and start transferring map outputs as the maps finish. With 1.75, the first round of reducers is finished by the faster nodes and second wave of reducers is launched doing a much better job of load balancing.

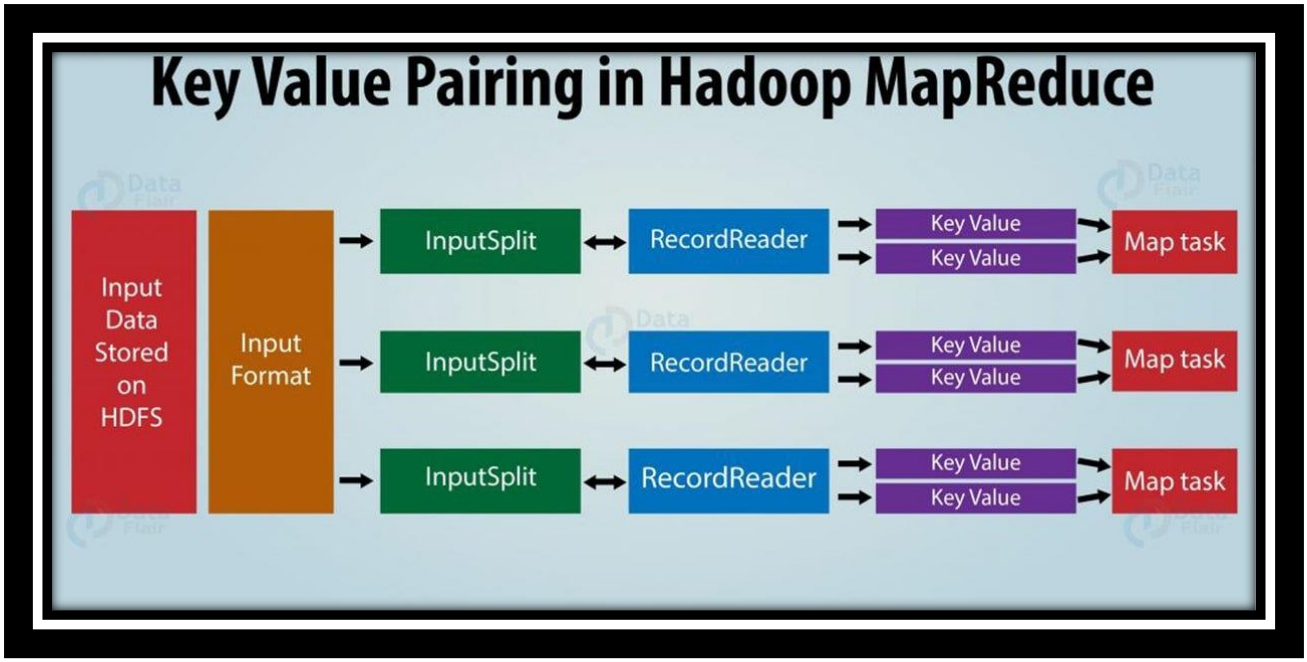
Increasing the number of MapReduce reducers:

- Increases the Framework overhead.
- Increases load balancing.
- Lowers the cost of failures.

Learn the Concept of Key-Value Pair in Hadoop MapReduce

1. Objective

In this **MapReduce,** we are going to learn the concept of a key-value pair in **Hadoop**. The key Value pair is the record entity that MapReduce job receives for execution. By default, Record Reader uses TextInputFormat for converting data into a key-value pair. Here we will learn what is a key-value pair in MapReduce, how key-value pairs are generated in Hadoop using InputSplit and RecordReader and on what basis generation of key-value pairs in Hadoop MapReduce takes place? We will also see Hadoop key-value pair example in this .



Learn the Concept of Key-Value Pair in Hadoop MapReduce

2. What is a key-value pair in Hadoop?

Apache Hadoop is used mainly for **Data Analysis**. We look at statistical and logical techniques in data Analysis to describe, illustrate and evaluate data. Hadoop deals with structured, unstructured and semi-structured data. In Hadoop, when the schema is static, we can directly work on the column instead of keys and values, but, when the schema is not static, then we will work on keys and values. Keys and values are not the intrinsic properties of the data, but they are chosen by user analyzing the data.

MapReduce is the core component of Hadoop which provides data processing. Hadoop MapReduce is a software framework for easily writing an application that processes the vast amount of structured and unstructured data stored in the **Hadoop Distributed Filesystem (HDFS).** MapReduce works by breaking the processing into two phases: Map phase and Reduce phase. Each phase has key-value as input and output.

3. Key-value pair Generation in MapReduce

Let us now learn how key-value pair is generated in Hadoop MapReduce?In MapReduce process, before passing the data to the **mapper**, data should be first

converted into key-value pairs as mapper only understands key-value pairs of data. key-value pairs in Hadoop MapReduce is generated as follows:

- **InputSplit –** It is the logical representation of data. The data to be processed by an individual Mapper is presented by the InputSplit.
- **RecordReader –** It communicates with the InputSplit and it converts the Split into records which are in form of key-value pairs that are suitable for reading by the mapper. By default, RecordReader uses TextInputFormat for converting data into a key-value pair. RecordReader communicates with the InputSplit until the file reading is not completed.
- In MapReduce, map function processes a certain key-value pair and emits a certain number of key-value pairs and the Reduce function processes values grouped by the same key and emits another set of key-value pairs as output. The output types of the Map should match the input types of the Reduce as shown below:
- **Map:** (K1, V1) -> list (K2, V2)
- **Reduce:** {(K2, list (V2 }) -> list (K3, V3)

4. On what basis is a key-value pair generated in Hadoop?

Generation of a key-value pair in Hadoop depends on the data set and the required output. In general, the key-value pair is specified in 4 places: Map input, Map output, reduce input and Reduce output.

a. Map Input

Map-input by default will take the line offset as the key and the content of the line will be the value as Text. By using custom **InputFormat** we can modify them.

b. Map Output

Map basic responsibility is to filter the data and provide the environment for grouping of data based on the key.

- **Key** – It will be the field/ text/ object on which the data has to be grouped and aggregated on the **reducer** side.
- **Value** – It will be the field/ text/ object which is to be handled by each individual reduce method.

c. Reduce Input

The output of Map is the input for reduce, so it is same as Map-Output.

d. Reduce Output

It depends on the required output.

5. MapReduce key-value pair Example

Suppose, the content of the file which is stored in HDFS is **John is Mark Joey is John**. Using Input Format, we will define how this file will split and read. By default, Record Reader uses TextInputFormat to convert this file into a key-value pair.

- **Key –** It is offset of the beginning of the line within the file.
- **Value –** It is the content of the line, excluding line terminators.

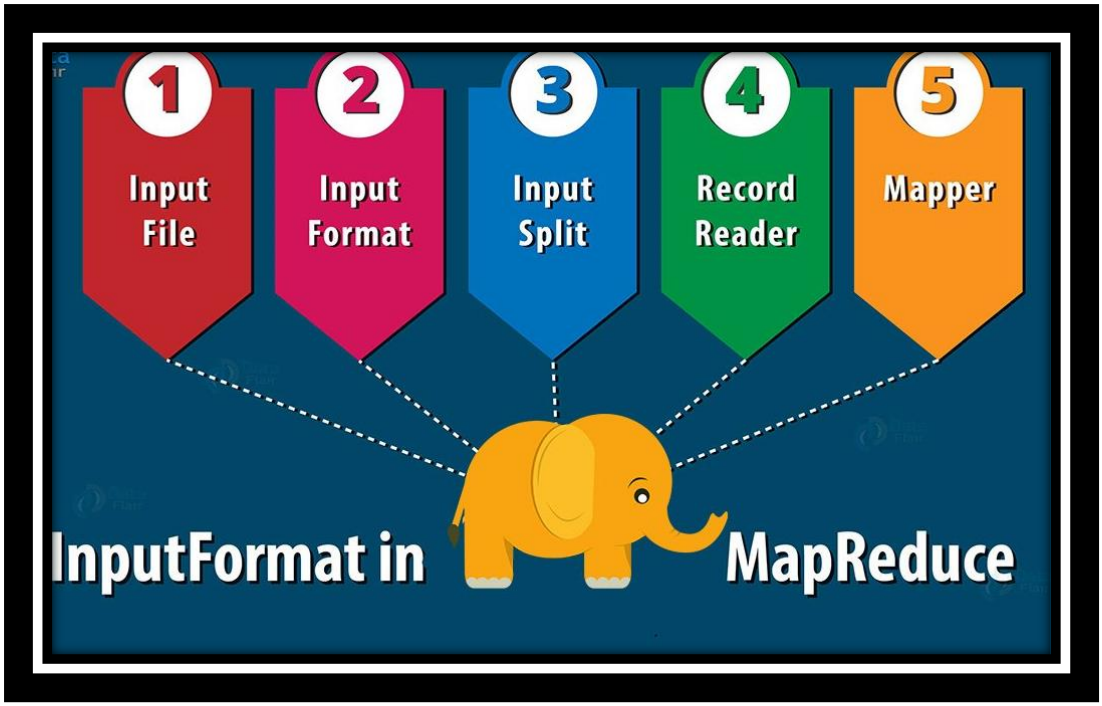
From the above content of the file-

- **Key** is 0
- **Value** is John is Mark Joey is John.

Hadoop InputFormat, Types of InputFormat in MapReduce

1. Objective

Hadoop InputFormat checks the Input-Specification of the job. InputFormat split the Input file into InputSplit and assign to individual Mapper. In this Hadoop InputFormat , we will learn what is InputFormat in Hadoop **MapReduce**, different methods to get the data to the mapper and different types of InputFormat in Hadoop like FileInputFormat in Hadoop, TextInputFormat, KeyValueTextInputFormat, etc. We will also see what is the default InputFormat in Hadoop?



Hadoop InputFormat, Types of InputFormat in MapReduce

2. What is Hadoop InputFormat?

How the input files are split up and read in Hadoop is defined by the InputFormat. An Hadoop InputFormat is the first component in Map-Reduce, it is responsible for creating the input splits and dividing them into records. If you are not familiar with MapReduce Job Flow, so follow our **Hadoop MapReduce Data flow** for more understanding.

Initially, the data for a MapReduce task is stored in input files, and input files typically reside in **HDFS**. Although these files format is arbitrary, line-based log files and binary format can be used. Using InputFormat we define how these input files are split and read. The InputFormat class is one of the fundamental classes in the Hadoop MapReduce framework which provides the following functionality:

- The files or other objects that should be used for input is selected by the InputFormat.
- InputFormat defines the Data splits, which defines both the size of individual **Map tasks** and its potential execution server.
- InputFormat defines the **RecordReader**, which is responsible for reading actual records from the input files.

3. How we get the data to mapper?

We have 2 methods to get the data to **mapper** in MapReduce: getsplits() and createRecordReader() as shown below:

[php]public abstract class InputFormat<K, V>
{
public abstract List<InputSplit> getSplits(JobContext context)
throws IOException, InterruptedException;
public abstract RecordReader<K, V>
createRecordReader(InputSplit split,
TaskAttemptContext context) throws IOException,
InterruptedException;
}[/php]

4. Types of InputFormat in MapReduce

Types of InputFormat

Let us now see what are the types of InputFormat in Hadoop?

4.1. FileInputFormat in Hadoop

It is the base class for all file-based InputFormats. Hadoop FileInputFormat specifies input directory where data files are located. When we start a Hadoop job, FileInputFormat is provided with a path containing files to read. FileInputFormat will read all files and divides these files into one or more InputSplits.

4.2. TextInputFormat

It is the default InputFormat of MapReduce. TextInputFormat treats each line of each input file as a separate record and performs no parsing. This is useful for unformatted data or line-based records like log files.

- **Key –** It is the byte offset of the beginning of the line within the file (not whole file just one split), so it will be unique if combined with the file name.
- **Value –** It is the contents of the line, excluding line terminators.

4.3. KeyValueTextInputFormat

It is similar to TextInputFormat as it also treats each line of input as a separate record. While TextInputFormat treats entire line as the value, but the KeyValueTextInputFormat breaks the line itself into key and value by a tab character ('/t'). Here Key is everything up to the tab character while the value is the remaining part of the line after tab character.

4.4. SequenceFileInputFormat

Hadoop **SequenceFileInputForma**t is an InputFormat which reads sequence files. Sequence files are binary files that stores sequences of binary **key-value pairs**. Sequence files block-compress and provide direct serialization and deserialization of several arbitrary data types (not just text). Here Key & Value both are user-defined.

4.5. SequenceFileAsTextInputFormat

Hadoop **SequenceFileAsTextInputFormat** is another form of SequenceFileInputFormat which converts the sequence file key values to Text objects. By calling **'tostring()'** conversion is performed on the keys and values. This InputFormat makes sequence files suitable input for streaming.

4.6. SequenceFileAsBinaryInputFormat

Hadoop **SequenceFileAsBinaryInputFormat** is a SequenceFileInputFormat using which we can extract the sequence file's keys and values as an opaque binary object.

4.7. NLineInputFormat

Hadoop **NLineInputFormat** is another form of TextInputFormat where the keys are byte offset of the line and values are contents of the line. Each mapper receives a variable number of lines of input with TextInputFormat and KeyValueTextInputFormat and the number depends on the size of the split and the length of the lines. And if we want our mapper to receive a fixed number of lines of input, then we use NLineInputFormat.
N is the number of lines of input that each mapper receives. By default (N=1), each mapper receives exactly one line of input. If N=2, then each split contains two lines. One mapper will receive the first two Key-Value pairs and another mapper will receive the second two key-value pairs.

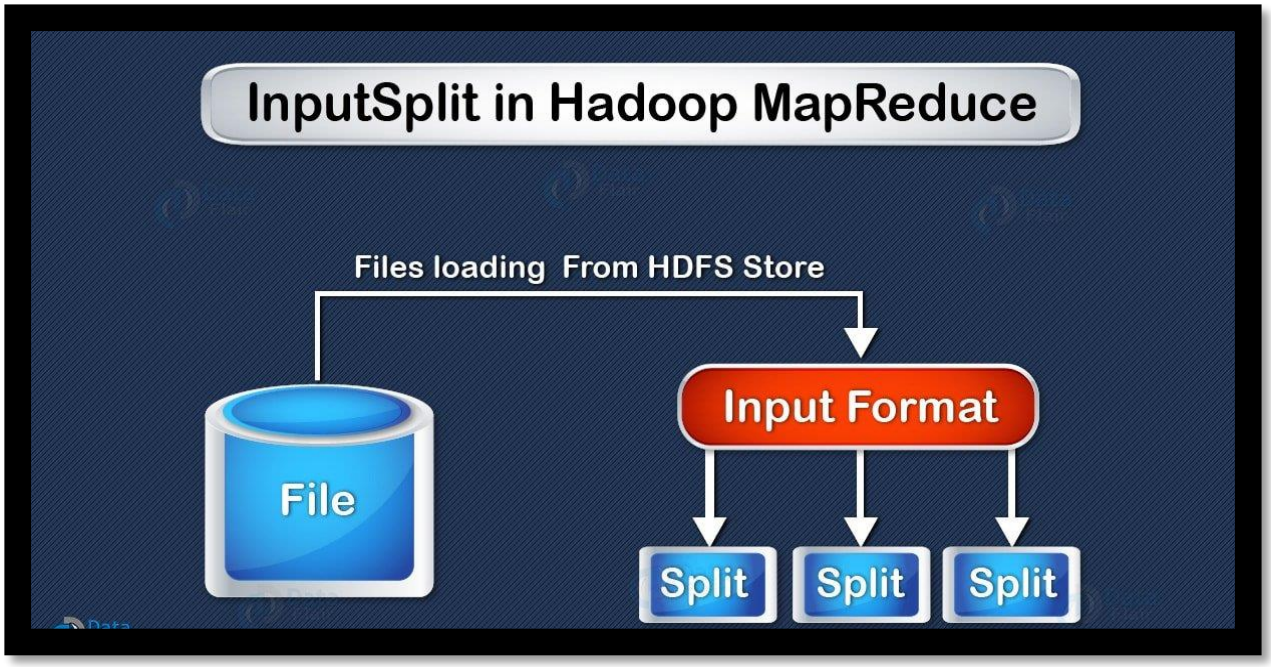
4.8. DBInputFormat

Hadoop **DBInputFormat** is an InputFormat that reads data from a relational database, using JDBC. As it doesn't have portioning capabilities, so we need to careful not to swamp the database from which we are reading too many mappers. So it is best for loading relatively small datasets, perhaps for joining with large datasets from HDFS using MultipleInputs. Here Key is LongWritables while Value is DBWritables.

InputSplit in Hadoop MapReduce – Hadoop MapReduce

1. Objective

In this **Hadoop** MapReduce, we will provide you the detailed description of InputSplit in Hadoop. In this blog, we will try to answer What is Hadoop InputSplit, what is the need of inputSplit in MapReduce and how Hadoop performs InputSplit, How to change split size in Hadoop. We will also learn the difference between InputSplit vs Blocks in **HDFS.**



InputSplit in Hadoop MapReduce – Hadoop MapReduce

2. What is InputSplit in Hadoop?

InputSplit in Hadoop **MapReduce** is the logical representation of data. It describes a unit of work that contains a single map task in a MapReduce program. Hadoop InputSplit represents the data which is processed by an individual **Mapper**. The split is divided into records. Hence, the mapper process each record (which is a **key-value pair**).

MapReduce InputSplit length is measured in bytes and every InputSplit has storage locations (hostname strings). MapReduce system use storage locations to place map tasks as close to split's data as possible. Map tasks are processed in the order of the size of the splits so that the largest one gets processed first (greedy approximation algorithm) and this is done to minimize the job runtime (Learn MapReduce job optimization techniques)The important thing to notice is that Inputsplit does not contain the input data; it is just a reference to the data.

As a user, we don't need to deal with InputSplit directly, because they are created by an **InputFormat** (InputFormat creates the Inputsplit and divide into records). FileInputFormat, by default, breaks a file into **128MB** chunks (same as blocks in HDFS) and by setting *mapred.min.split.size* parameter in *mapred-site.xml* we can

control this value or by overriding the parameter in the Job object used to submit a particular **MapReduce job**. We can also control how the file is broken up into splits, by writing a custom InputFormat.

3. How to change split size in Hadoop?

InputSplit in Hadoop is user defined. User can control split size according to the size of data in MapReduce program. Thus the number of map tasks is equal to the number of InputSplits.

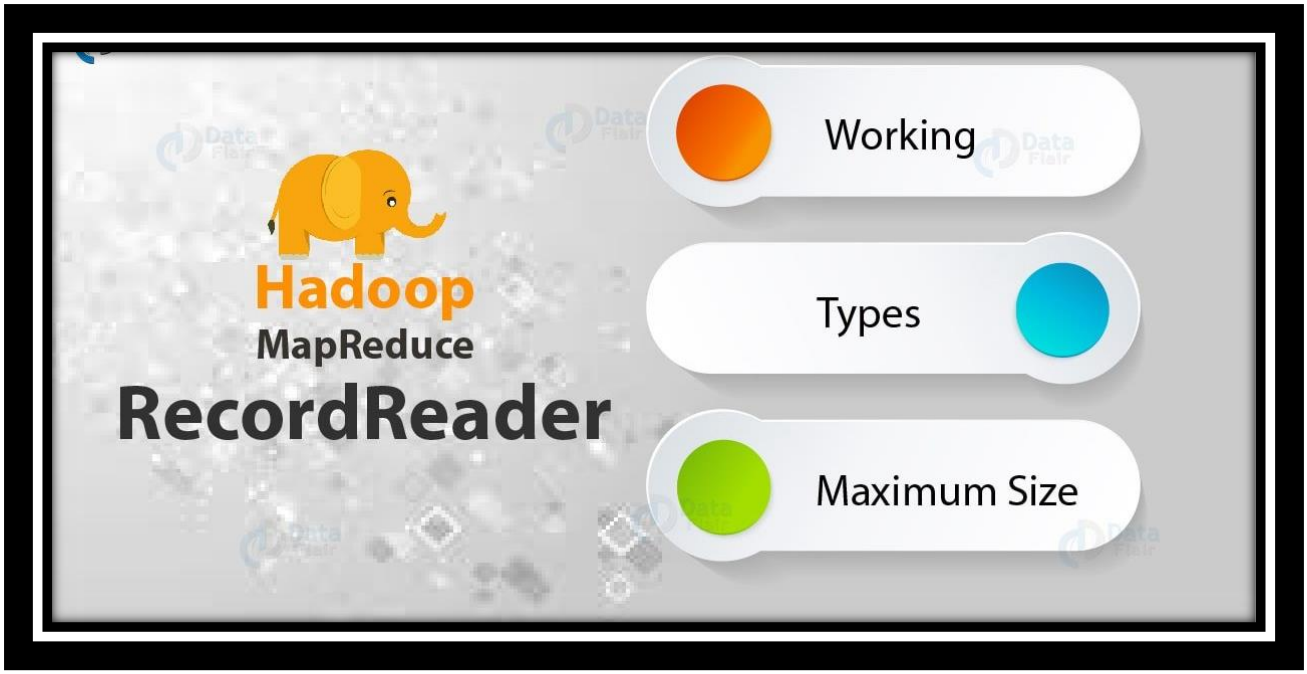
The client (running the job) can calculate the splits for a job by calling '**getSplit()**', and then sent to the application master, which uses their storage locations to schedule map tasks that will process them on the cluster. Then, map task passes the split to the *createRecordReader()* method on InputFormat to get **RecordReader** for the split and Record Reader generate record (key-value pair), which it passes to the map function.

Hadoop RecordReader – How Record Reader Works in Hadoop?

1. Hadoop RecordReader– Objective

In this Hadoop RecordReader , We are going to discuss the important concept of **Hadoop** MapReduce i.e. **RecordReader**. The MapReduce RecordReader in Hadoop takes the byte-oriented view of input, provided by the InputSplit and presents as a record-oriented view for Mapper. It uses the data within the boundaries that were created by the InputSplit and creates Key-value pair.

This blog will answer what is RecordReader in Hadoop, how Hadoop RecordReader works and types of Hadoop RecordReader – SequenceFileRecordReader and Line RecordReader, the maximum size of a record in Hadoop.

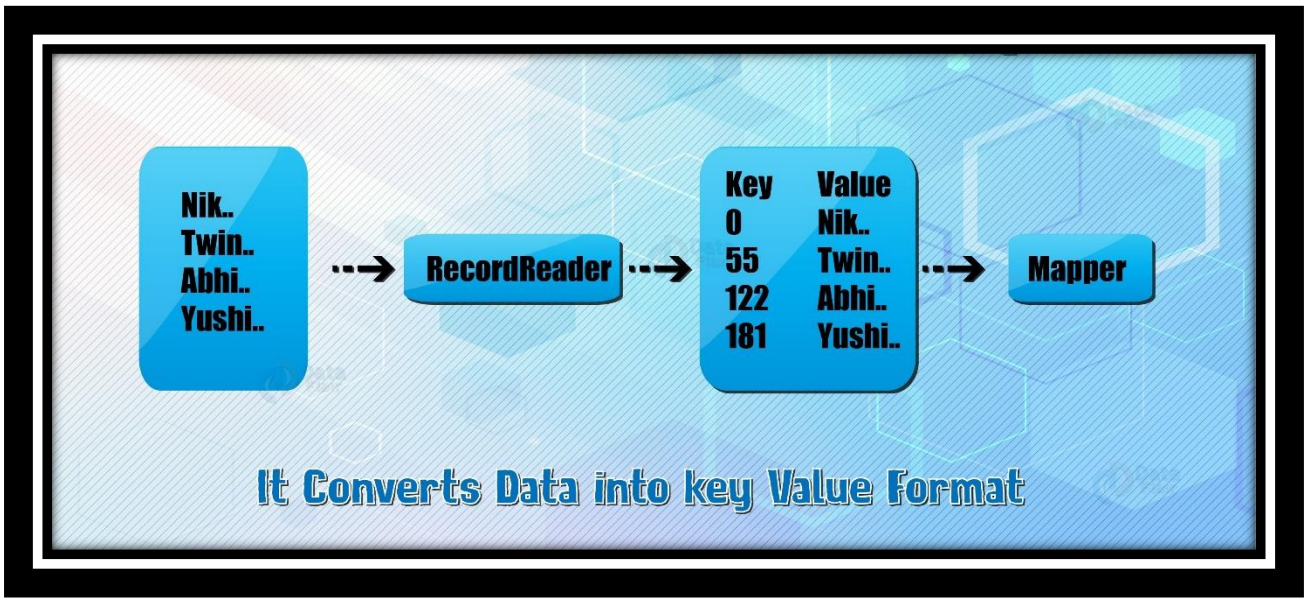


Hadoop MapReduce RecordReader

2. What is Hadoop RecordReader?

To understand recordreader in Hadoop, we need to understand the Hadoop data flow. So, Let's see how the data flow in Hadoop?

MapReduce has a simple model of data processing. Inputs and Outputs for the map and reduce functions are **key-value pairs.** The map and reduce functions in Hadoop MapReduce have the following general form:



Hadoop Record Reader and its types.

- **map:** (K1, V1) → list(K2, V2)
- **reduce:** (K2, list(V2)) → list(K3, V3)

Now before processing, it needs to know on which data to process, this is achieved with the **InputFormat** class. InputFormat is the class which selects the file from **HDFS** that should be input to the map function. An InputFormat is also responsible for creating the **InputSplits** and dividing them into records. The data is divided into the number of splits(typically 64/128mb) in HDFS. This is called as inputsplit which is the input that is processed by a single map.

InputFormat class calls the **getSplits()** function and computes splits for each file and then sends them to the **JobTracker**, which uses their storage locations to schedule map tasks to process them on the **TaskTrackers**. Map task then passes the split to the **createRecordReader()** method on InputFormat in task tracker to obtain a RecordReader for that split. The RecordReader load's data from its source and converts into key-value pairs suitable for reading by the **mapper.**

Hadoop RecordReader uses the data within the boundaries that are being created by the inputsplit and creates Key-value pairs for the mapper. The "start" is the byte position in the file where the RecordReader should start generating key/value pairs and the "end" is where it should stop reading records. In Hadoop RecordReader, the data is loaded from its source and then the data is converted into key-value pairs suitable for reading by the Mapper. It communicates with the inputsplit until the file reading is not completed.
Read: Mapper in Mapreduce

3. How Hadoop RecordReader works?

Let us now see the working of RecordReader in Hadoop.
A RecordReader is more than iterator over records, and map task uses one record to generate key-value pair which is passed to the map function. We can see this by using mapper's run function:

[php]public void run(Context context) throws IOException, InterruptedException{
<pre>setup(context);
while(context.nextKeyValue())
{

map(context.setCurrentKey(),context.getCurrentValue(),context)
}
cleanup(context);
}[/php]

After running **setup()**, the **nextKeyValue()** will repeat on the context, to populate the key and value objects for the mapper. The key and value is retrieved from the record reader by way of context and passed to the **map()** method to do its work. An input to the map function, which is a key-value pair(K, V), gets processed as per the logic mentioned in the map code. When the record gets to the end of the record, the **nextKeyValue()** method returns false.

A RecordReader usually stays in between the boundaries created by the inputsplit to generate key-value pairs but this is not mandatory. A custom implementation can even read more data outside of the inputsplit, but it is not encouraged a lot.
Read: Reducer in MapReduce

4. Types of Hadoop RecordReader in MapReduce

The RecordReader instance is defined by the InputFormat. By default, it uses TextInputFormat for converting data into a key-value pair. TextInputFormat provides 2 types of RecordReaders:

i. LineRecordReader

Line RecordReader in Hadoop is the default RecordReader that textInputFormat provides and it treats each line of the input file as the new value and associated key is byte offset. LineRecordReader always skips the first line in the split (or part of it), if it is not the first split. It read one line after the boundary of the split in the end (if data is available, so it is not the last split).

ii. SequenceFileRecordReader

It reads data specified by the header of a sequence file.
Read: Partitioner in MapReduce

5. Maximum size for a Single Record

There is a maximum size allowed for a single record to be processed. This value can be set using below parameter.

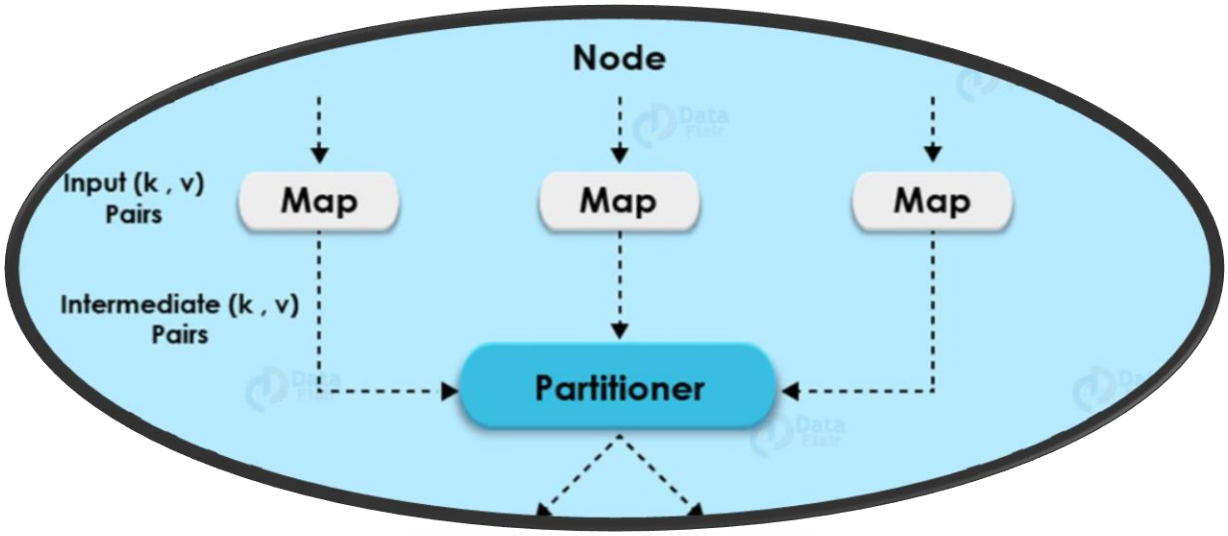
[php]conf.setInt("mapred.linerecordreader.maxlength", Integer.MAX_VALUE);[/php]
A line with a size greater than this maximum value (default is 2,147,483,647) will be ignored.

MapReduce Partitioner

1. Hadoop Partitioner / MapReduce Partitioner

In this **MapReduce** , our objective is to discuss what is **Hadoop** Partitioner. The **Partitioner** in MapReduce controls the partitioning of the key of the intermediate mapper output. By hash function, key (or a subset of the key) is used to derive the partition. A total number of partitions depends on the number of reduce task. Here we will also learn what is the need of Hadoop partitioner, what is the default Hadoop partitioner, how many practitioners are required in Hadoop and what do you mean by poor partitioning in Hadoop along with ways to overcome MapReduce poor partitioning.

Hadoop Partitioner – Internals of MapReduce Partitioner



2. What is Hadoop Partitioner?

Before we start with MapReduce partitioner, let us understand what is Hadoop **mapper**, Hadoop **Reducer**, and **combiner in Hadoop**?

Partitioning of the keys of the intermediate map output is controlled by the Partitioner. By hash function, key (or a subset of the key) is used to derive the partition. According to the **key-value** each mapper output is partitioned and records having the same key value go into the same partition (within each mapper), and then each partition is sent to a reducer. Partition class determines which partition a given (key, value) pair will go. Partition phase takes place after map phase and before reduce phase. Lets move ahead with need of Hadoop Partitioner and if you face any difficulty anywhere in Hadoop MapReduce , you can ask us in comments.

Read: A Guide on Big Data Hadoop for beginners

3. Need of Hadoop MapReduce Partitioner?

Let's now discuss what is the need of Mapreduce Partitioner in Hadoop?

MapReduce job takes an input data set and produces the list of the key-value pair which is the result of map phase in which input data is split and each task processes the split and each map, output the list of key-value pairs. Then, the output from the map phase is sent to reduce task which processes the user-defined reduce function on map outputs. But before reduce phase, partitioning of the map output take place on the basis of the key and sorted.
This partitioning specifies that all the values for each key are grouped together and make sure that all the values of a single key go to the same reducer, thus allows even distribution of the map output over the reducer.

Partitioner in Hadoop MapReduce redirects the mapper output to the reducer by determining which reducer is responsible for the particular key.
Read: Hadoop MapReduce

4. Default MapReduce Partitioner

The Default Hadoop partitioner in Hadoop MapReduce is Hash Partitioner which computes a hash value for the key and assigns the partition based on this result.

5. How many Partitioners are there in Hadoop?

The total number of Partitioners that run in Hadoop is equal to the number of reducers i.e. Partitioner will divide the data according to the number of reducers which is set by *JobConf.setNumReduceTasks()* method. Thus, the data from single partitioner is processed by a single reducer. And partitioner is created only when there are multiple reducers.

6. Poor Partitioning in Hadoop MapReduce

If in data input one key appears more than any other key. In such case, we use two mechanisms to send data to partitions.

- The key appearing more will be sent to one partition.
- All the other key will be sent to partitions according to their hashCode().

But if **hashCode()** method does not uniformly distribute other keys data over partition range, then data will not be evenly sent to reducers. Poor partitioning of data means that some reducers will have more data input than other i.e. they will have more work to do than other reducers. So, the entire job will wait for one reducer to finish its extra-large share of the load.

How to overcome poor partitioning in MapReduce?

To overcome poor partitioner in Hadoop MapReduce, we can create Custom partitioner, which allows sharing workload uniformly across different reducers.

Explanation to MapReduce Combiner

1. Hadoop Combiner / MapReduce Combiner

Hadoop Combiner is also known as "**Mini-Reducer**" that summarizes the Mapper output record with the same Key before passing to the Reducer. In this on **MapReduce** combiner we are going to answer what is a Hadoop combiner, MapReduce program with and without combiner, advantages of Hadoop combiner and disadvantages of the combiner in Hadoop.

Hadoop Combiner – Best Explanation to MapReduce Combiner

2. What is Hadoop Combiner?

On a large dataset when we run **MapReduce job**, large chunks of intermediate data is generated by the Mapper and this intermediate data is passed on the Reducer for further processing, which leads to enormous network congestion. MapReduce framework provides a function known as **Hadoop Combiner** that plays a key role in reducing network congestion.

We have already seen earlier **what is mapper** and what is **reducer** in Hadoop MapReduce. Now we in the next step to learn Hadoop MapReduce Combiner.

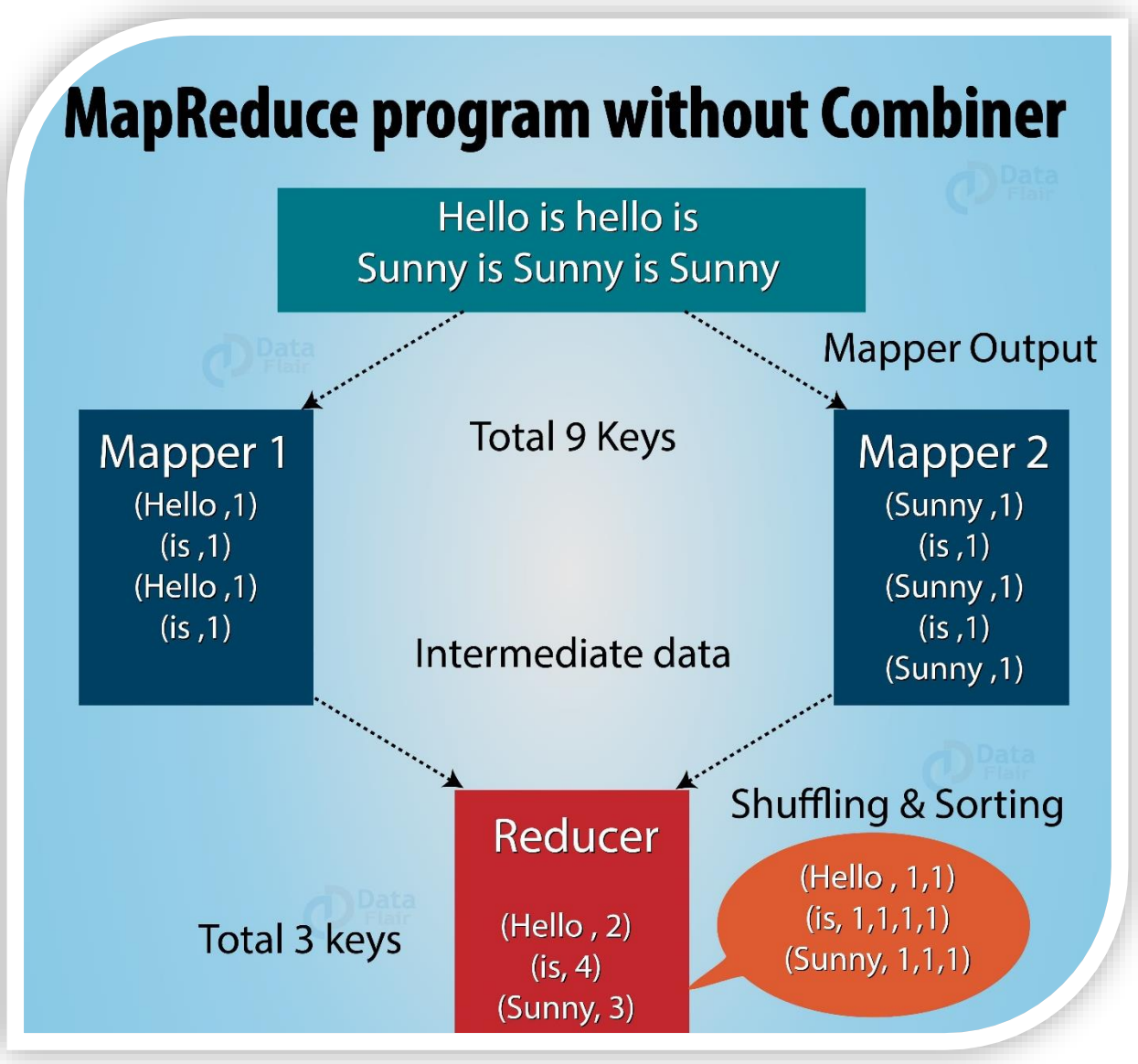
The combiner in MapReduce is also known as 'Mini-reducer'. The primary job of Combiner is to process the output data from the Mapper, before passing it to Reducer. It runs after the mapper and before the Reducer and its use is optional.

Read: Key-value Pairs in MapReduce

3. How does MapReduce Combiner work?

Let us now see the working of the Hadoop combiner in MapReduce and how things change when combiner is used as compared to when combiner is not used in MapReduce?

3.1. MapReduce program without Combiner



MapReduce Combiner : MapReduce program without combiner

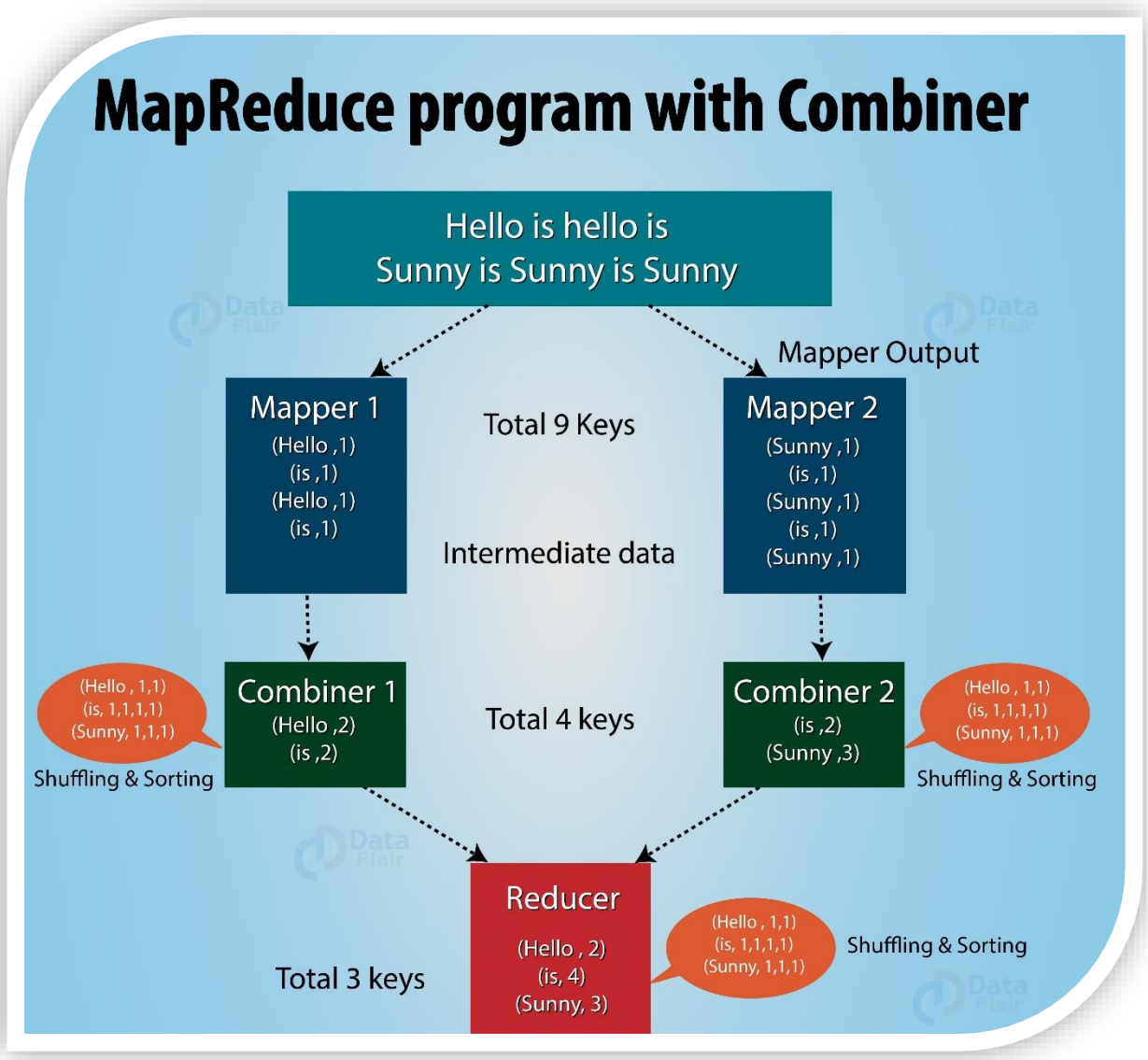
In the above diagram, no combiner is used. Input is split into two mappers and 9 keys are generated from the mappers. Now we have (9 **key/value**) intermediate data, the further mapper will send directly this data to reducer and while sending

data to the reducer, it consumes some network bandwidth (bandwidth means time taken to transfer data between 2 machines). It will take more time to transfer data to reducer if the size of data is big.

Now in between mapper and reducer if we use a hadoop combiner, then combiner shuffles intermediate data (9 key/value) before sending it to the reducer and generates 4 key/value pair as an output.

Read: Data Locality in MapReduce

3.2. MapReduce program with Combiner in between Mapper and Reducer



MapReduce Combiner: MapReduce program with combiner

Reducer now needs to process only 4 key/value pair data which is generated from 2 combiners. Thus reducer gets executed only 4 times to produce final output, which increases the overall performance.

4. Advantages of MapReduce Combiner

As we have discussed what is Hadoop MapReduce Combiner in detail, now we will discuss some advantages of Mapreduce Combiner.

- Hadoop Combiner reduces the time taken for data transfer between mapper and reducer.
- It decreases the amount of data that needed to be processed by the reducer.
- The Combiner improves the overall performance of the reducer.

5. Disadvantages of Hadoop combiner in MapReduce

There are also some disadvantages of hadoop Combiner. Let's discuss them one by one-

- MapReduce jobs cannot depend on the Hadoop combiner execution because there is no guarantee in its execution.
- In the local filesystem, the key-value pairs are stored in the Hadoop and run the combiner later which will cause expensive disk IO.

Shuffling and Sorting in Hadoop MapReduce

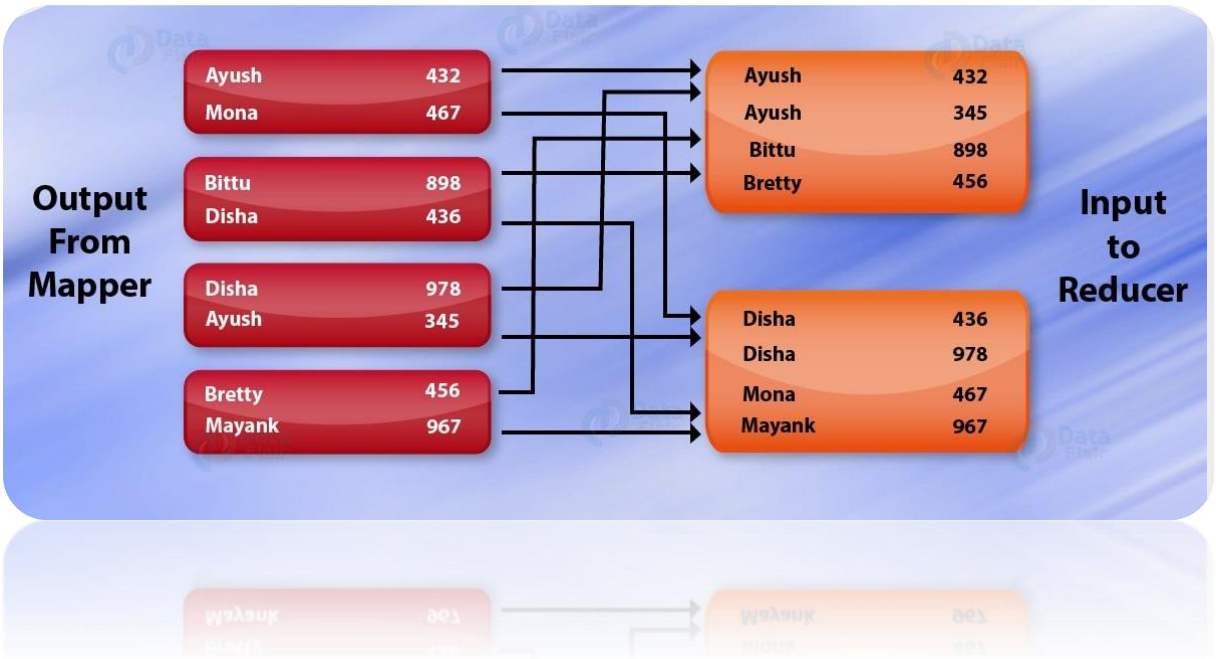
1. Objective

In **Hadoop**, the process by which the intermediate output from **mappers** is transferred to the **reducer** is called Shuffling. Reducer gets 1 or more keys and associated values on the basis of reducers. Intermediated **key-value** generated by mapper is sorted automatically by key. In this blog, we will discuss in detail about shuffling and Sorting in **Hadoop MapReduce**.

Here we will learn what is sorting in Hadoop, what is shuffling in Hadoop, what is the purpose of Shuffling and sorting phase in **MapReduce**, how MapReduce shuffle

works and how MapReduce sort works. We will also learn what is secondary sorting in MapReduce?

Shuffling and Sorting in Hadoop MapReduce



2. What is Shuffling and Sorting in Hadoop MapReduce?

Before we start with Shuffle and Sort in MapReduce, let us revise the other phases of MapReduce like **Mapper**, **reducer** in MapReduce, **Combiner**, **partitioner in MapReduce** and **inputFormat in MapReduce.**

Shuffle phase in Hadoop transfers the map output from Mapper to a Reducer in MapReduce. **Sort phase** in MapReduce covers the merging and sorting of map outputs. Data from the mapper are grouped by the key, split among reducers and sorted by the key. Every reducer obtains all values associated with the same key. Shuffle and sort phase in Hadoop occur simultaneously and are done by the MapReduce framework.

Let us now understand both these processes in details below:

3. Shuffling in MapReduce

The process of transferring data from the mappers to reducers is known as shuffling i.e. the process by which the system performs the sort and transfers the map output to the reducer as input. So, MapReduce shuffle phase is necessary for the reducers, otherwise, they would not have any input (or input from every mapper). As shuffling can start even before the map phase has finished so this saves some time and completes the tasks in lesser time.

4. Sorting in MapReduce

The keys generated by the mapper are automatically sorted by MapReduce Framework, i.e. Before starting of reducer, all intermediate **key-value pairs** in MapReduce that are generated by mapper get sorted by key and not by value. Values passed to each reducer are not sorted; they can be in any order.

Sorting in Hadoop helps reducer to easily distinguish when a new reduce task should start. This saves time for the reducer. Reducer starts a new reduce task when the next key in the sorted input data is different than the previous. Each reduce task takes key-value pairs as input and generates key-value pair as output.

Note that shuffling and sorting in Hadoop MapReduce is not performed at all if you specify zero reducers (setNumReduceTasks(0)). Then, the MapReduce job stops at the map phase, and the map phase does not include any kind of sorting (so even the map phase is faster).

5. Secondary Sorting in MapReduce

If we want to sort reducer's values, then the secondary sorting technique is used as it enables us to sort the values (in ascending or descending order) passed to each reducer.

Hadoop Output Format – Types of Output Format in Mapreduce

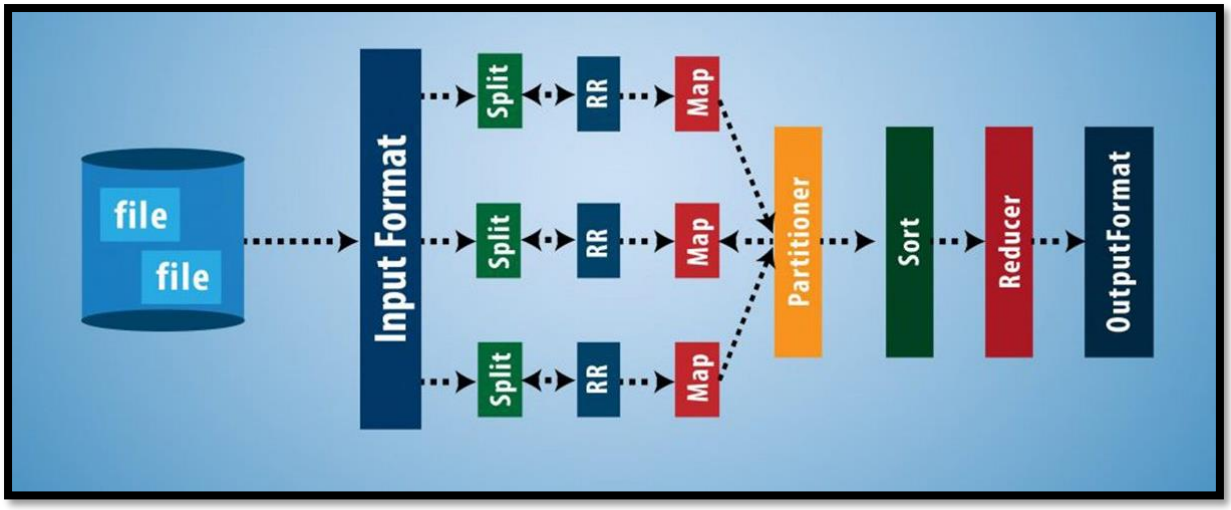
1. Hadoop Output Format – Objective

The Hadoop Output Format checks the Output-Specification of the job. It determines how RecordWriter implementation is used to write output to output files. In this

blog, we are going to see what is Hadoop Output Format, what is Hadoop RecordWriter, how RecordWriter is used in Hadoop?

In this Hadoop Reducer Output Format guide, will also discuss various types of Output Format in Hadoop like textOutputFormat, sequenceFileOutputFormat, mapFileOutputFormat, sequenceFileAsBinaryOutputFormat, DBOutputFormat, LazyOutputForma, and MultipleOutputs.

Learn **How to install Cloudera Hadoop CDH5 on CentOS**.



Hadoop Output Format – Types of Output Format in Mapreduce

2. What is Hadoop Output Format?

Before we start with Hadoop Output Format in **MapReduce**, let us first see what is a RecordWriter in MapReduce and what is its role in MapReduce?

i. Hadoop RecordWriter

As we know, **Reducer** takes as input a set of an intermediate **key-value pair** produced by the **mapper** and runs a reducer function on them to generate output that is again zero or more key-value pairs.

RecordWriter writes these output key-value pairs from the Reducer phase to output files.

ii. Hadoop Output Format

As we saw above, Hadoop RecordWriter takes output data from Reducer and writes this data to output files. The way these output key-value pairs are written in output

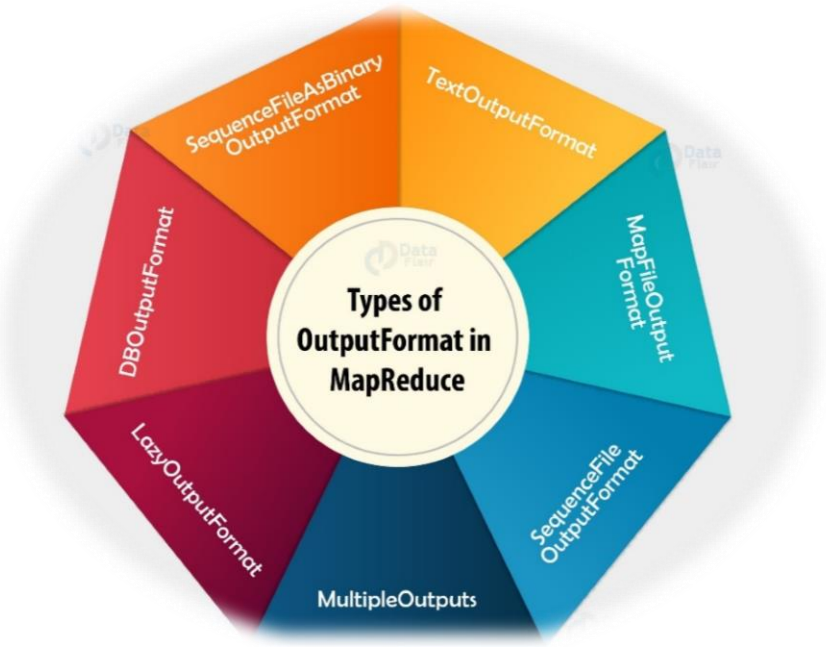
files by RecordWriter is determined by the Output Format. The Output Format and **InputFormat** functions are alike. OutputFormat instances provided by Hadoop are used to write to files on the **HDFS** or local disk. OutputFormat describes the output-specification for a **Map-Reduce job**. On the basis of output specification;

- MapReduce job checks that the output directory does not already exist.
- OutputFormat provides the RecordWriter implementation to be used to write the output files of the job. Output files are stored in a FileSystem.

FileOutputFormat.setOutputPath() method is used to set the output directory. Every Reducer writes a separate file in a common output directory.

3. Types of Hadoop Output Formats

There are various types of Hadoop OutputFormat. Let us see some of them below:



Types of Hadoop Output Formats

i. TextOutputFormat

MapReduce default Hadoop reducer Output Format is **TextOutputFormat**, which writes (key, value) pairs on individual lines of text files and its keys and values can be of any type since TextOutputFormat turns them to string by calling toString() on them. Each key-value pair is separated by a tab character, which can be changed using ***MapReduce.output.textoutputformat.separator*** property.

KeyValueTextOutputFormat is used for reading these output text files since it breaks lines into key-value pairs based on a configurable separator.

ii. SequenceFileOutputFormat

It is an Output Format which writes sequences files for its output and it is intermediate format use between MapReduce jobs, which rapidly serialize arbitrary data types to the file; and the corresponding SequenceFileInputFormat will deserialize the file into the same types and presents the data to the next mapper in the same manner as it was emitted by the previous reducer, since these are compact and readily compressible. Compression is controlled by the static methods on SequenceFileOutputFormat.

iii. SequenceFileAsBinaryOutputFormat

It is another form of SequenceFileInputFormat which writes keys and values to sequence file in binary format.

iv. MapFileOutputFormat

It is another form of FileOutputFormat in Hadoop Output Format, which is used to write output as map files. The key in a MapFile must be added in order, so we need to ensure that reducer emits keys in sorted order.
Any doubt yet in Hadoop Oputput Format? Please Ask.

v. MultipleOutputs

It allows writing data to files whose names are derived from the output keys and values, or in fact from an arbitrary string.

vi. LazyOutputFormat

Sometimes FileOutputFormat will create output files, even if they are empty. LazyOutputFormat is a wrapper OutputFormat which ensures that the output file will be created only when the record is emitted for a given partition.

vii. DBOutputFormat

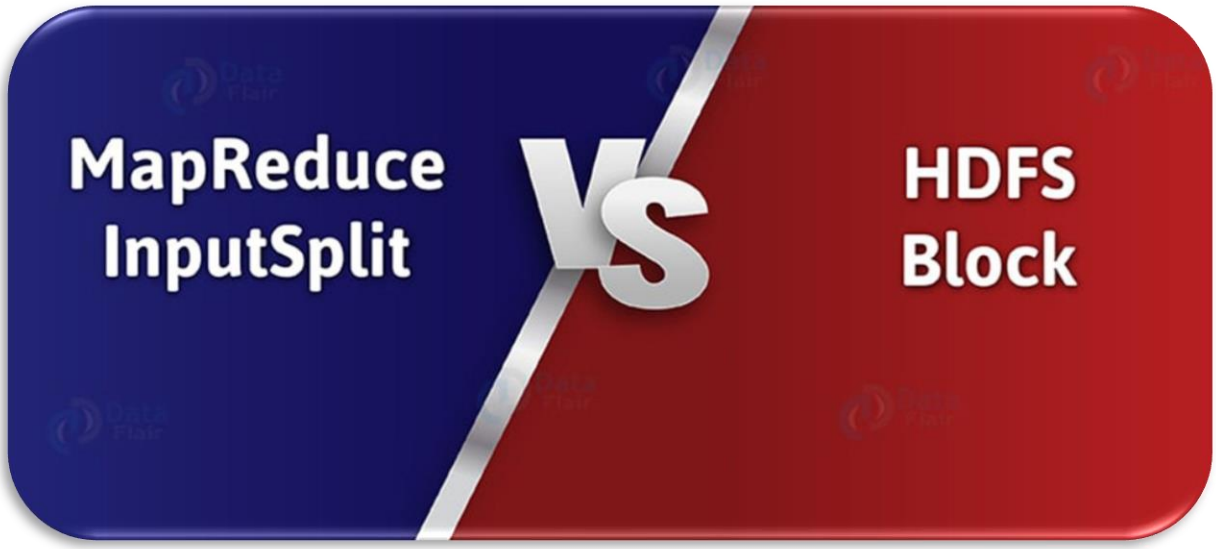
DBOutputFormat in Hadoop is an Output Format for writing to relational databases and **HBase**. It sends the reduce output to a SQL table. It accepts key-value pairs,

where the key has a type extending DBwritable. Returned RecordWriter writes only the key to the database with a batch SQL query.
This was all on Hadoop Output format .

MapReduce InputSplit vs HDFS Block in Hadoop

1. Objective

In this **Hadoop** InputSplit vs Block , we will learn what is a block in **HDFS**, what is **MapReduce** InputSplit and difference between MapReduce InputSplit vs Block size in Hadoop to deep dive into Hadoop fundamentals.



MapReduce InputSplit vs HDFS Block in Hadoop

2. MapReduce InputSplit & HDFS Block – Introduction

Let us start with learning what is a block in Hadoop HDFS and what do you mean by Hadoop InputSplit?

Block in HDFS

Block is a continuous location on the hard drive where data is stored. In general, FileSystem stores data as a collection of blocks. In the same way, HDFS stores each

file as blocks. The Hadoop application is responsible for distributing the data block across multiple nodes. Read more about blocks here.

InputSplit in Hadoop

The data to be processed by an individual **Mapper** is represented by InputSplit. The split is divided into records and each record (which is a **key-value pair**) is processed by the map. The number of map tasks is equal to the number of InputSplits.

Initially, the data for MapReduce task is stored in input files and input files typically reside in HDFS. **InputFormat** is used to define how these input files are split and read. InputFormat is responsible for creating InputSplit.

3. MapReduce InputSplit vs Blocks in Hadoop

Let's discuss feature wise comparison between MapReduce InputSplit vs Blocks-

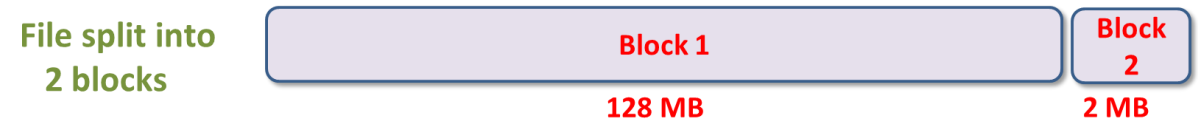
i. InputSplit vs Block Size in Hadoop

- **Block –** The default size of the HDFS block is 128 MB which we can configure as per our requirement. All blocks of the file are of the same size except the last block, which can be of same size or smaller. The files are split into 128 MB blocks and then stored into Hadoop FileSystem.
- **InputSplit –** By default, split size is approximately equal to block size. InputSplit is user defined and the user can control split size based on the size of data in MapReduce program.

ii. Data Representation in Hadoop Blocks vs InputSplit

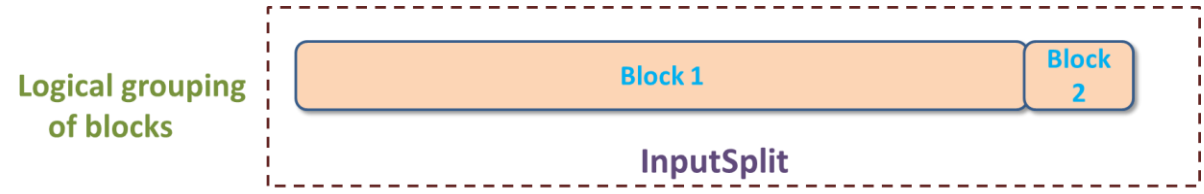
- **Block –** It is the physical representation of data. It contains a minimum amount of data that can be read or write.
- **InputSplit –** It is the logical representation of data present in the block. It is used during data processing in MapReduce program or other processing techniques. InputSplit doesn't contain actual data, but a reference to the data.

iii. Example of Block vs InputSplit in Hadoop



InputSplit vs Block

InputSplit vs Block



inputSplit vs Block

Consider an example, where we need to store the file in HDFS. HDFS stores files as blocks. Block is the smallest unit of data that can be stored or retrieved from the disk and the default size of the block is 128MB. HDFS break files into blocks and stores these blocks on different nodes in the cluster. Suppose we have a file of 130 MB, so HDFS will break this file into 2 blocks.

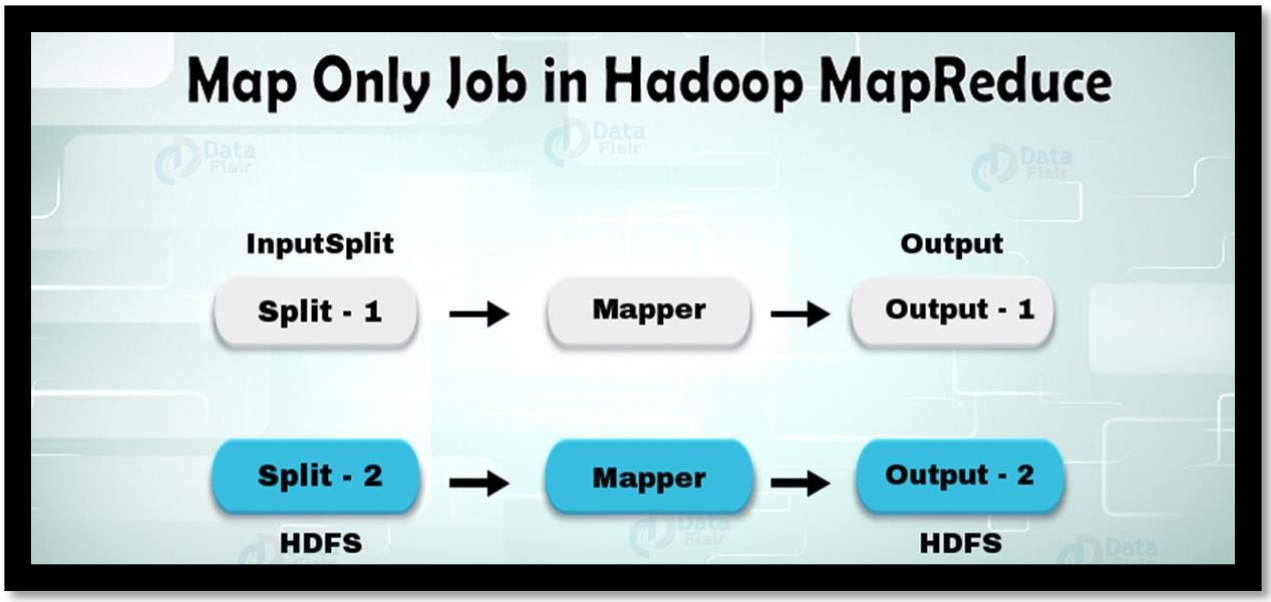
Now, if we want to perform MapReduce operation on the blocks, it will not process, because the 2nd block is incomplete. Thus, this problem is solved by InputSplit. InputSplit will form a logical grouping of blocks as a single block, because the InputSplit include a location for the next block and the byte offset of the data needed to complete the block.

Map Only Job in Hadoop MapReduce with example

1. Objective

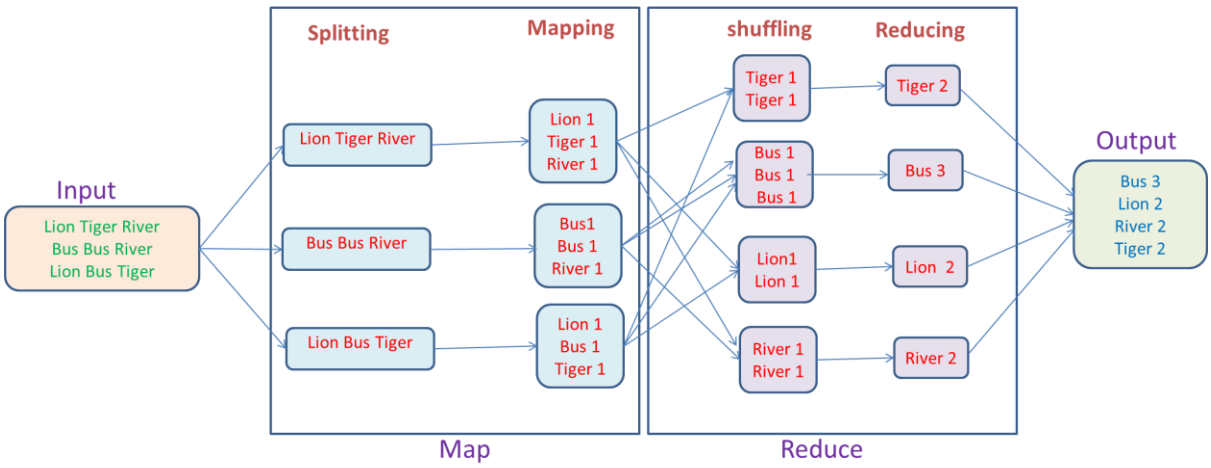
In **Hadoop**, Map-Only job is the process in which mapper does all task, no task is done by the reducer and mapper's output is the final output. In this on Map only job in Hadoop MapReduce, we will learn about MapReduce process, the need of map only job in Hadoop, how to set a number of reducers to 0 for Hadoop map only job.

We will also learn what are the advantages of Map Only job in Hadoop MapReduce, processing in Hadoop without reducer along with MapReduce example with no reducer.



Map Only Job in Hadoop MapReduce with example

2. What is Map Only Job in Hadoop MapReduce?



Hadoop MapReduce – Map Only job

MapReduce is a software framework for easily writing applications that process the vast amount of structured and unstructured data stored in the **Hadoop Distributed Filesystem (HDFS)**. Two important tasks done by MapReduce algorithm are: **Map task** and **Reduce task.** Hadoop Map phase takes a set of data and converts it into another set of data, where individual element are broken down

into tuples (**key/value pairs**). Hadoop Reduce phase takes the output from the map as input and combines those data tuples based on the key and accordingly modifies the value of the key.

From the above word-count example, we can say that there are two sets of parallel process, **map** and **reduce**; in map process, the first input is split to distribute the work among all the map nodes as shown in a figure, and then each word is identified and mapped to the number 1. Thus the pairs called tuples (key-value) pairs.

In the first **mapper** node three words lion, tiger, and river are passed. Thus the output of the node will be three key-value pairs with three different keys and value set to 1 and the same process repeated for all nodes. These tuples are then passed to the **reducer** nodes and **partitioner** comes into action. It carries out shuffling so that all tuples with the same key are sent to the same node. Thus, in reduce process basically what happens is an aggregation of values or rather an operation on values that share the same key.

Now, let us consider a scenario where we just need to perform the operation and no aggregation required, in such case, we will prefer 'Map-Only job' in Hadoop. In Hadoop Map-Only job, the map does all task with its **InputSplit** and no job is done by the reducer. Here map output is the final output.
Refer this guide to learn Hadoop features and design principles.

3. How to avoid Reduce Phase in Hadoop?

We can achieve this by setting *job.setNumreduceTasks(0)* in the configuration in a driver. This will make a number of reducer as 0 and thus the only mapper will be doing the complete task.

4. Advantages of Map only job in Hadoop

In between map and reduces phases there is key, **sort** and **shuffle** phase. Sort and shuffle are responsible for sorting the keys in ascending order and then grouping values based on same keys. This phase is very expensive and if reduce phase is not required we should avoid it, as avoiding reduce phase would eliminate sort and shuffle phase as well. This also saves network congestion as in shuffling, an output of mapper travels to reducer and when data size is huge, large data needs to travel to the reducer.

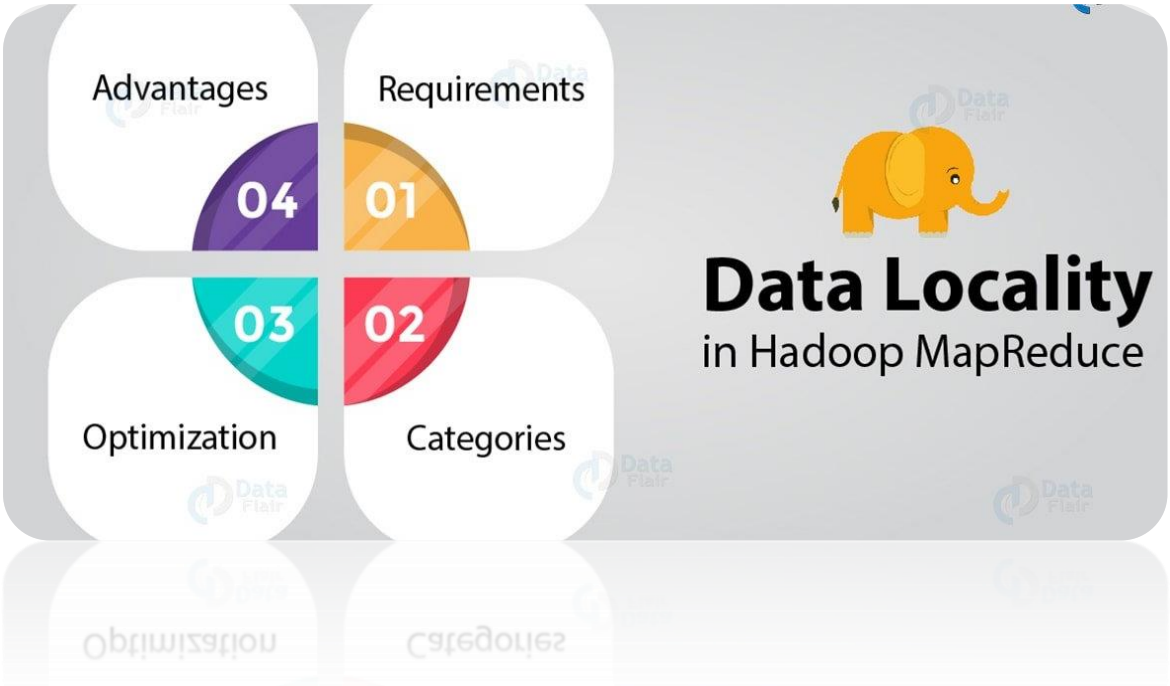
The output of mapper is written to local disk before sending to reducer but in map only job, this output is directly written to HDFS. This further saves time and reduces cost as well.

Also, there is no need of **partitioner** and **combiner** in Hadoop Map Only job that makes the process fast.

Data locality in Hadoop: The Most Comprehensive Guide

1. Data Locality in Hadoop – Objective

In **Hadoop**, Data locality is the process of moving the computation close to where the actual data resides on the node, instead of moving large data to computation. This minimizes network congestion and increases the overall throughput of the system. This feature of Hadoop we will discuss in detail in this . We will learn **what is data locality in Hadoop**, data locality definition, how Hadoop exploits Data Locality, what is the need of Hadoop Data Locality, various types of data locality in Hadoop **MapReduce**, Data locality optimization in Hadoop and various advantages of Hadoop data locality.



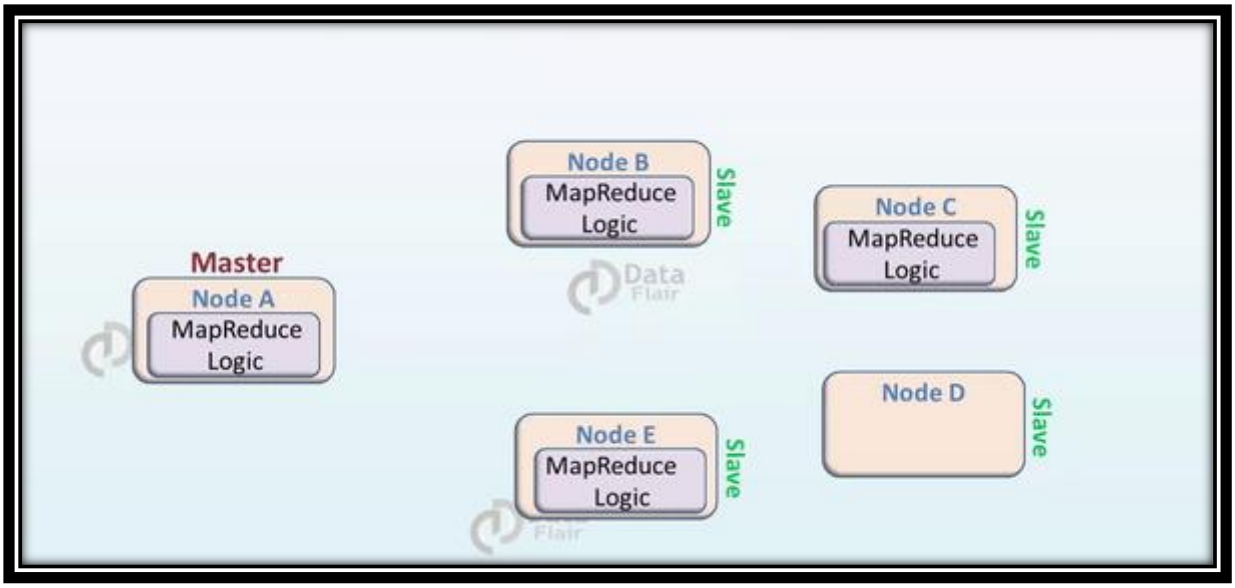
Data locality in Hadoop: The Most Comprehensive Guide

2. The Concept of Data locality in Hadoop

Let us understand **Data Locality concept** and what is Data Locality in MapReduce?

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. This minimizes network congestion and increases the overall throughput of the system.

In Hadoop, datasets are stored in **HDFS**. Datasets are divided into blocks and stored across the datanodes in **Hadoop cluster**. When a user runs the **MapReduce job** then NameNode sent this MapReduce code to the datanodes on which data is available related to MapReduce job.



Data Locality in Hadoop – MapReduce

3. Requirements for Data locality in MapReduce

Our system architecture needs to satisfy the following conditions, in order to get the benefits of all the advantages of data locality:

- First of all the cluster should have the appropriate topology. Hadoop code must have the ability to read data locality.

- Second, Hadoop must be aware of the topology of the nodes where tasks are executed. And Hadoop must know where the data is located.

4. Categories of Data Locality in Hadoop

Below are the various categories in which Data Locality in Hadoop is categorized:

i. 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.

ii. 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.

iii. 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.

5. Hadoop Data Locality Optimization

Although Data locality in Hadoop MapReduce is the main advantage of Hadoop MapReduce as map code is executed on the same data node where data resides. But this is not always true in practice due to various reasons like **speculative execution in Hadoop**, Heterogeneous cluster, Data distribution and placement, and Data Layout and Input Splitter.
Challenges become more prevalent in large clusters, because more the number of data nodes and data, less will be the locality. In larger clusters, some nodes are newer and faster than the other, creating the data to compute ratio out of balance thus, large clusters tend not to be completely homogenous. In speculative execution even though the data might not be local, but it uses the computing power. The root cause also lies in the data layout/placement and the used Input Splitter. Non-local data processing puts a strain on the network which creates problem to scalability. Thus

the network becomes the bottleneck.
We can improve data locality by first detecting which jobs has the data locality problem or degrade over time. Problem-solving is more complex and involves changing the data placement and data layout, using a different scheduler or by simply changing the number of mapper and **reducer** slots for a job. Then we have to verify whether a new execution of the same workload has a better data locality ratio.

6. Advantages of Hadoop Data locality

There are two benefits of data Locality in MapReduce. Let's discuss them one by one-

i. Faster Execution

In data locality, the program is moved to the node where data resides instead of moving large data to the node, this makes Hadoop faster. Because the size of the program is always lesser than the size of data, so moving data is a bottleneck of network transfer.

ii. High Throughput

Data locality increases the overall throughput of the system.

Speculative Execution in Hadoop MapReduce

1. Objective

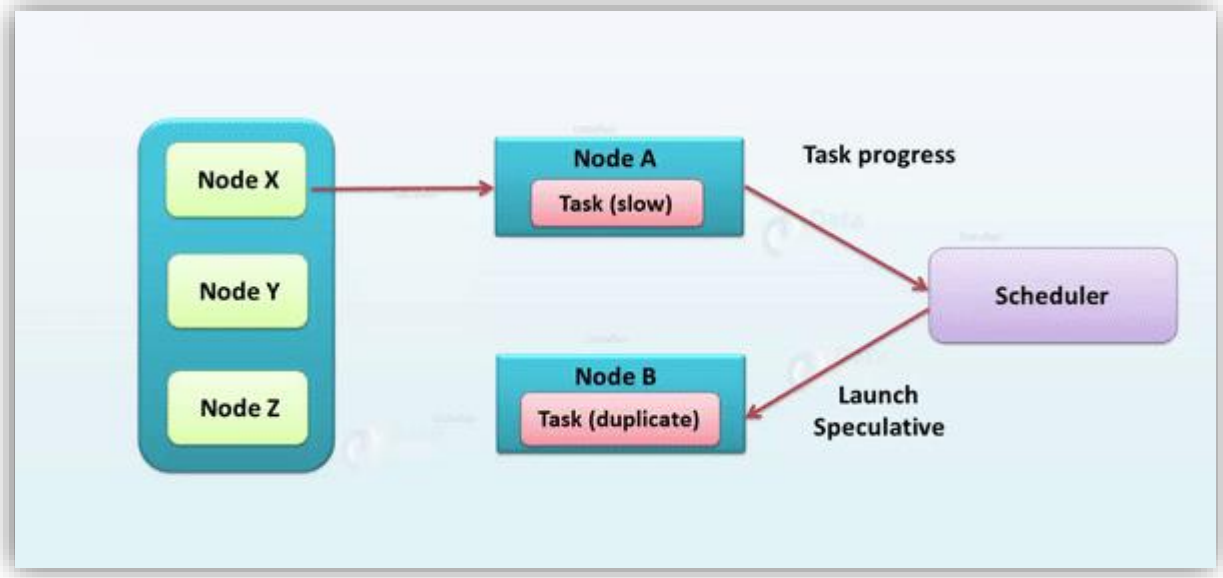
In this **Big data** Hadoop , we are going to learn **Hadoop** speculative execution. Apache Hadoop does not fix or diagnose slow-running tasks. Instead, it tries to detect when a task is running slower than expected and launches another, an equivalent task as a backup (the backup task is called as speculative task). This process is called speculative execution in Hadoop.

In this we will discuss speculative execution – A key feature of Hadoop that improves job efficiency, what is the need of speculative execution in Hadoop, is Speculative execution always helpful or do we need to turn it off and how can we disable speculative execution in Hadoop.

Speculative Execution in Hadoop MapReduce

2. What is Speculative Execution in Hadoop?

Let us first understand what is Hadoop speculative execution?



Speculative Execution in Spark

In Hadoop, **MapReduce** breaks jobs into tasks and these tasks run parallel rather than sequential, thus reduces overall execution time. This model of execution is sensitive to slow tasks (even if they are few in numbers) as they slow down the

overall execution of a job.
There may be various reasons for the slowdown of tasks, including hardware degradation or software misconfiguration, but it may be difficult to detect causes since the tasks still complete successfully, although more time is taken than the expected time. Hadoop doesn't try to diagnose and fix slow running tasks, instead, it tries to detect them and runs backup tasks for them. This is called **speculative execution** in Hadoop. These backup tasks are called Speculative tasks in Hadoop.

3. How Speculative Execution works in Hadoop?

Let us now see Hadoop speculative execution process.
Firstly all the tasks for the job are launched in Hadoop MapReduce. The speculative tasks are launched for those tasks that have been running for some time (at least one minute) and have not made any much progress, on average, as compared with other tasks from the job. The speculative task is killed if the original task completes before the speculative task, on the other hand, the original task is killed if the speculative task finishes before it.

4. Is Speculative Execution Beneficial?

Hadoop MapReduce Speculative execution is beneficial in some cases because in a **Hadoop cluster** with 100s of nodes, problems like hardware failure or network congestion are common and running parallel or duplicate task would be better since we won't be waiting for the task in the problem to complete.
But if two duplicate tasks are launched at about same time, it will be a wastage of cluster resources.

5. How to Enable or Disable Speculative Execution?

Speculative execution is a **MapReduce job optimization technique** in Hadoop that is enabled by default. You can disable speculative execution for **mappers** and **reducers** in *mapred-site.xml* as shown below:
[php]<property>
<name>mapred.map.tasks.speculative.execution</name>
<value>false</value>
</property>
<property>
<name>mapred.reduce.tasks.speculative.execution</name>

<value>false</value>
</property>[/php]

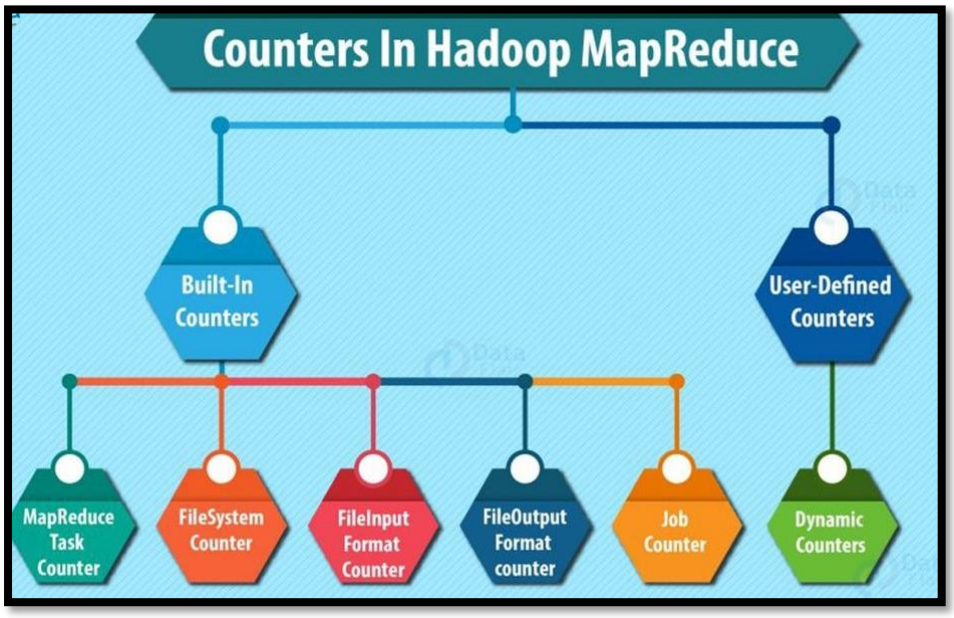
6. What is the need to turn off Speculative Execution?

The main work of speculative execution is to reduce the job execution time; however, the clustering efficiency is affected due to duplicate tasks. Since in speculative execution redundant tasks are being executed, thus this can reduce overall throughput. For this reason, some cluster administrators prefer to turn off the speculative execution in Hadoop.

Hadoop Counters | The Most Complete Guide to MapReduce Counters

1. Hadoop Counters: Objective

In this MapReduce Hadoop Counters , we will provide you the detailed description of MapReduce Counters in **Hadoop**. The covers an introduction to Hadoop **MapReduce** counters, Types of Hadoop Counters such as Built-in Counters and User-defined counters. In this Hadoop counters , we will also discuss the FileInputFormat and FileOutputFormat of Hadoop MapReduce.



Hadoop Counters

2. What is Hadoop MapReduce?

Before we start with Hadoop Counters, let us first see the overview of Hadoop MapReduce.

MapReduce is the core component of Hadoop which provides data processing. MapReduce works by breaking the processing into two phases; **Map** phase and **Reduce** phase. The map is the first phase of processing, where we specify all the complex logic/business rules/costly code, whereas the Reduce phase is the second phase of processing, where we specify light-weight processing like aggregation/ summation.

In Hadoop, MapReduce Framework has certain elements such as **Counters**, **Combiners**, and **Partitioners,** which play a key role in improving the performance of data processing.
Let's now focus on Hadoop MapReduce Counters here.

3. What are Hadoop Counters?

Hadoop Counters Explained: Hadoop Counters provides a way to measure the progress or the number of operations that occur within **map/reduce job**. Counters in Hadoop MapReduce are a useful channel for gathering statistics about the MapReduce job: for quality control or for application-level. They are also useful for problem diagnosis.

Counters represent Hadoop global counters, defined either by the MapReduce framework or applications. Each Hadoop counter is named by an "**Enum**" and has a long for the value. Counters are bunched into groups, each comprising of counters from a particular Enum class.

Hadoop Counters validate that:

- The correct number of bytes was read and written.
- The correct number of tasks was launched and successfully ran.
- The amount of CPU and memory consumed is appropriate for our job and cluster nodes.

4. Types of Hadoop MapReduce Counters

There are basically 2 types of MapReduce Counters:

- Built-In Counters in MapReduce
- User-Defined Counters/Custom counters in MapReduce

Let's discuss these types counters in Hadoop MapReduce:

4.1. Built-In Counters in MapReduce

Hadoop maintains some built-in Hadoop counters for every job and these report various metrics, like, there are counters for the number of bytes and records, which allow us to confirm that the expected amount of input is consumed and the expected amount of output is produced.

Hadoop Counters are divided into groups and there are several groups for the built-in counters. Each group either contains task counters (which are updated as task progress) or job counter (which are updated as a job progress).
There are several groups for the Hadoop built-in Counters:

a. MapReduce Task Counter in Hadoop

Hadoop Task counter collects specific information (like number of records read and written) about tasks during its execution time. For example, the MAP_INPUT_RECORDS counter is the Task Counter which counts the input records read by each map task.

Hadoop Task counters are maintained by each task attempt and periodically sent to the application master so they can be globally aggregated.

b. FileSystem Counters

Hadoop FileSystem Counters in Hadoop MapReduce gather information like a number of bytes read and written by the file system. Below are the name and description of the file system counters:

- **FileSystem bytes read**– The number of bytes read by the filesystem by map and reduce tasks.
- **FileSystem bytes written**– The number of bytes written to the filesystem by map and reduce tasks.

c. FileInputFormat Counters in Hadoop

FileInputFormat Counters in Hadoop MapReduce gather information of a number of bytes read by map tasks via FileInputFormat. Refer this guide to learn about InputFormat in Hadoop MapReduce.

d. FileOutputFormat counters in MapReduce

FileOutputFormat counters in Hadoop MapReduce gathers information of a number of bytes written by map tasks (for map-only jobs) or reduce tasks via FileOutputFormat. refer this guide to learn about OutputFormat in Hadoop MapReduce in detail.

e. MapReduce Job Counters

MapReduce Job counter measures the job-level statistics, not values that change while a task is running. For example, TOTAL_LAUNCHED_MAPS, count the number of map tasks that were launched over the course of a job (including tasks that failed). Application master maintains MapReduce Job counters, so these Hadoop Counters don't need to be sent across the network, unlike all other counters, including user-defined ones.

4.2. User-Defined Counters/Custom Counters in Hadoop MapReduce

In addition to MapReduce built-in counters, MapReduce allows user code to define a set of counters, which are then incremented as desired in the **mapper** or **reducer.** For example, in Java, 'enum' is used to define counters. A job may define an arbitrary number of 'enums', each with an arbitrary number of fields. The name of the enum is the group name, and the enum's fields are the counter names.

a. Dynamic Counters in Hadoop MapReduce

Java enum's fields are defined at compile time, so we cannot create new counters in Hadoop MapReduce at runtime using enums. To do so, we use dynamic counters in Hadoop MapReduce, one that is not defined at compile time using java enum.

Hadoop Optimization | Job Optimization & Performance Tuning

1. Hadoop Optimization : Objective

This on Hadoop Optimization will explain you **Hadoop** cluster optimization or **MapReduce** job optimization techniques that would help you in optimizing MapReduce job performance to ensure the best performance for your Hadoop cluster.

Hadoop Optimization | Job Optimization & Performance Tuning

2. 6 Hadoop Optimization or Job Optimization Techniques

There are various ways to improve the Hadoop optimization. Let's discuss each of them one by one-

i. Proper configuration of your cluster

- Dfs and MapReduce storage have been mounted with -noatime option. This disables access time and can improve I/O performance.
- Avoid **RAID** on **TaskTracker** and datanode machines, it generally reduces performance.
- Make sure you have configured ***mapred.local.dir*** and ***dfs.data.dir*** to point to one directory on each of your disks to ensure that all of your I/O capacity is used.
- Ensure that you have smart monitoring to the health status of your disk drives. This is 1 of the best practice for Hadoop **MapReduce performance tuning**. MapReduce jobs are **fault tolerant,** but dying disks can cause performance to degrade as tasks must be re-executed.
- Monitor the graph of swap usage and network usage with software like ganglia, Hadoop monitoring metrics. If you see swap being used, reduce the amount of RAM allocated to each task in ***mapred.child.java.opts.***

ii. LZO compression usage

This is always a good idea for Intermediate data. Almost every Hadoop job that generates a non-negligible amount of map output will benefit from intermediate data compression with **LZO**. Although LZO adds a little bit of CPU overhead, it saves time by reducing the amount of disk IO during the shuffle.
In order to enable LZO compression set ***mapred.compress.map.output*** to **true**. This is one of the most important Hadoop optimization techniques.

iii. Proper tuning of the number of MapReduce tasks

- If each task takes 30-40 seconds or more, then reduce the number of tasks. The start of **mapper** or **reducer** process involves following things: first, you need to start JVM (JVM loaded into the memory), then you need to initialize JVM and after processing (mapper/reducer) you need to de-initialize JVM. All these JVM tasks are costly. Now consider a case where mapper runs a task just for 20-30 seconds and for this we have to start/initialize/stop JVM, which might take a considerable amount of time. It is recommended to run the task for at least 1 minute.

- If a job has more than 1TB of input, you should consider increasing the block size of the input dataset to 256M or even 512M so that the number of tasks will be smaller. You can change the block size of existing files by using the command **Hadoop distcp –Hdfs.block.size=$[256*1024*1024] /path/to/inputdata /path/to/inputdata-with-largeblocks**
- So long as each task runs for at least 30-40 seconds, you should increase the number of mapper tasks to some multiple of the number of mapper slots in the cluster.
- Don't schedule too many reduce tasks – for most jobs, the number of reduce tasks equal to or a bit less than the number of reduce slots in the cluster.

iv. Combiner between mapper and reducer

If your algorithm involves computing aggregates of any sort, it is suggested to use a **Combiner** to perform some aggregation before the data hits the reducer. The MapReduce framework runs combine intelligently to reduce the amount of data to be written to disk and that has to be transferred between the Map and Reduce stages of computation.

v. Usage of most appropriate and compact writable type for data

Big data new users or users switching from Hadoop Streaming to Java MapReduce often use the Text writable type unnecessarily. Although Text can be convenient, it's inefficient to convert numeric data to and from UTF8 strings and can actually make up a significant portion of CPU time. Whenever dealing with non-textual data, consider using the binary Writables like **IntWritable, FLoatwritable** etc.

vi. Reusage of Writables

One of the common mistakes that many MapReduce users make is to allocate a new Writable object for every output from a mapper or reducer. For example, to implement a **word-count mapper:**

```
[php]public void map(...) {
...
for (String word : words) {
output.collect(new Text(word), new IntWritable(1));
}[/php]
```

This implementation causes allocation of thousands of short-lived objects. While Java garbage collector does a reasonable job at dealing with this, it is more efficient to write:

```
[php]class MyMapper ... {
Text wordText = new Text();
IntWritable one = new IntWritable(1);
public void map(...) {
... for (String word : words)
{
wordText.set(word);
output.collect(word, one); }
}
}[/php]
```

This is also one of the Hadoop job optimizing technique while Data flows in MapReduce.

Hadoop MapReduce Performance Tuning Best Practices

1. MapReduce Performance Tuning

Performance tuning in **Hadoop** will help in optimizing the Hadoop cluster performance. This on Hadoop MapReduce performance tuning will provide you ways for improving your Hadoop cluster performance and get the best result from your programming in Hadoop. It will cover 7 important concepts like Memory Tuning in Hadoop, Map Disk spill in Hadoop, tuning mapper tasks, Speculative execution in **Big data** Hadoop and many other related concepts for Hadoop MapReduce performance tuning. If you face any difficulty in Hadoop MapReduce Performance tuning , please let us know in the comments.

2. Hadoop MapReduce Performance Tuning

Hadoop performance tuning will help you in optimizing your Hadoop cluster performance and make it better to provide best results while doing Hadoop programming in **Big Data** companies. To perform the same, you need to repeat the process given below till desired output is achieved at optimal way.

Run Job –> Identify Bottleneck –> Address Bottleneck.

The first step in hadoop performance tuning is to run Hadoop job, Identify the bottlenecks and address them using below methods to get the highest performance. You need to repeat above step till a level of performance is achieved.

3. Tips for Hadoop MapReduce Performance Tuning

Here we are going to discuss the ways to improve the
Hadoop **MapReduce** performance tuning. We have classified these ways into two categories.

- Hadoop run-time parameters based performance tuning.
- Hadoop application-specific performance tuning.

Let's discuss how to improve the performance of Hadoop cluster on the basis of these two categories.

i. Tuning Hadoop Run-time Parameters

There are many options provided by Hadoop on CPU, memory, disk, and network for performance tuning. Most Hadoop tasks are not CPU bounded, what is most considered is to optimize usage of memory and disk spills. Let us get into the details in this Hadoop performance tuning in Tuning Hadoop Run-time parameters.

a. Memory Tuning

The most general and common rule for memory tuning in MapReduce performance tuning is: use as much memory as you can without triggering swapping. The parameter for task memory is ***mapred.child.java.opts*** that can be put in your configuration file.
You can also monitor memory usage on the server using
Ganglia, **Cloudera** manager, or Nagios for better memory performance.

b. Minimize the Map Disk Spill

Disk IO is usually the performance bottleneck in Hadoop. There are a lot of parameters you can tune for minimizing spilling like:

- Compression of mapper output

- Usage of 70% of heap memory ion mapper for spill buffer

But do you think frequent spilling is a good idea?
It's highly suggested not to spill more than once as if you spill once, you need to re-read and re-write all data: 3x the IO.

c. Tuning Mapper Tasks

The number of **mapper** tasks is set implicitly unlike **reducer** tasks. The most common hadoop performance tuning way for the mapper is controlling the amount of mapper and the size of each job. When dealing with large files, Hadoop split the file into smaller chunks so that mapper can run it in parallel. However, initializing new mapper job usually takes few seconds that is also an overhead to be minimized. Below are the suggestions for the same:

- Reuse jvm task
- Aim for map tasks running 1-3 minutes each. For this if the average mapper running time is lesser than one minute, increase the ***mapred.min.split.size***, to allocate less mappers in slot and thus reduce the mapper initializing overhead.
- Use Combine file input format for bunch of smaller files.

ii. Tuning Application Specific Performance

Let's now discuss the tips to improve the Application specific performance in Hadoop.

a. Minimize your Mapper Output

Minimizing the mapper output can improve the general performance a lot as this is sensitive to disk IO, network IO, and memory sensitivity on shuffle phase.
For achieving this, below are the suggestions:

- Filter the records on mapper side instead of reducer side.
- Use minimal data to form your map output key and map output value in Map Reduce.
- Compress mapper output

b. Balancing Reducer's Loading

Unbalanced reducer tasks create another performance issue. Some reducers take most of the output from mapper and ran extremely long compare to other reducers. Below are the methods to do the same:

- Implement a better hash function in Partitioner class.
- Write a preprocess job to separate keys using MultipleOutputs. Then use another map-reduce job to process the special keys that cause the problem.

c. Reduce Intermediate data with Combiner in Hadoop

Implement a **combiner** to reduce data which enables faster data transfer.

d. Speculative Execution

When tasks take long time to finish the execution, it affects the MapReduce jobs. This problem is being solved by the approach of speculative execution by backing up slow tasks on alternate machines.
You need to set the configuration parameters '***mapreduce.map.tasks.speculative.execution***' and '***mapreduce.reduce.tasks.speculative.execution***' to true for enabling speculative execution. This will reduce the job execution time if the task progress is slow due to memory unavailability.
This was all about the Hadoop Mapreduce Combiner.