**Understanding MapReduce**

MapReduce programming splits jobs (applications) into two main tasks:

1. **Map tasks** – Responsible for processing small subsets of the data.
2. **Reduce tasks** – Aggregate and generate the final output from intermediate results.

These tasks are executed in parallel across a Hadoop cluster to improve efficiency and scalability.

**Map Task Phases**

A map task involves:

1. **Record Reader**: Reads input data from the Hadoop Distributed File System (HDFS) and converts it into key-value pairs for processing.
2. **Mapper**: Processes the key-value pairs, transforming the data and generating intermediate key-value pairs.
3. **Combiner (optional)**: An optimization step that performs local aggregation on the mapper output to reduce the data size sent to the reducer.
4. **Partitioner**: Determines which reducer will process each intermediate key-value pair.

The output from the map task is referred to as **intermediate keys and values**.

**Reduce Task Phases**

The reduce task takes intermediate key-value pairs and processes them through the following phases:

1. **Shuffle**: Transfers the intermediate data from mappers to reducers.
2. **Sort**: Sorts the intermediate data by keys to prepare for reduction.
3. **Reducer**: Aggregates or processes the sorted data to produce the final output.
4. **Output Format**: Writes the final output back to HDFS in the required format.

**MAPPER**

**1. RecordReader**

* **Function:** Converts a byte-oriented view of the input into a record-oriented view.
* **Input Split:** Data is divided into smaller chunks (input splits) before being passed to the mapper.
* **Output:** Presents data as key-value pairs to the mapper.
  + The key typically represents positional information (e.g., an offset in the file).
  + The value represents a chunk of data (e.g., a line in a text file).

**2. Map**

* **Core Function:** The mapper function processes the input key-value pairs produced by RecordReader and generates zero or more intermediate key-value pairs.
* **Logic:** The transformation logic is user-defined and varies depending on the problem.
  + For example, in word count applications, the mapper generates (word, 1) for each word found.

**3. Combiner (Optional)**

* **Purpose:** Acts as a local reducer to aggregate mapper output before sending it to the reducer.
* **Performance Benefit:** Reduces the amount of data transferred over the network, saving bandwidth and disk space.
* **Functionality:** Combines multiple intermediate key-value pairs (e.g., summing counts for words) before sending them to the reducer.

**4. Partitioner**

* **Function:** Divides intermediate key-value pairs into partitions (shards) and assigns each partition to a reducer.
* **Key Assignment:** Ensures that keys with the same value are sent to the same reducer.
* **Data Storage:** The partitioned data is written to the local disk and pulled by the corresponding reducer for further processing.

**Reducer**

**1. Shuffle and Sort**

* **Function:** The shuffle phase takes the output from all partitioners and downloads it to the reducer’s local machine.
* **Sorting:** Data is sorted by keys to group similar keys together. This grouping is necessary so the reducer can process all values associated with a key in a single pass.
* **Purpose:** Ensures that all key-value pairs for a particular key are processed together, facilitating efficient reduction.

**2. Reduce**

* **Core Task:** The reducer iterates through the sorted data, applies user-defined logic, and processes one key-value group at a time.
* **Operations:** It can perform operations like aggregation, filtering, and combining. For example, in a word count problem, it aggregates word counts from all mappers.
* **Output:** The output can be zero or more key-value pairs, depending on the logic applied in the reduce function.

**3. Output Format**

* **Writing the Output:** The default format separates the key-value pairs with a tab and writes the final results to a file in Hadoop Distributed File System (HDFS).
* **Custom Formatting:** Users can customize the output format as needed.

