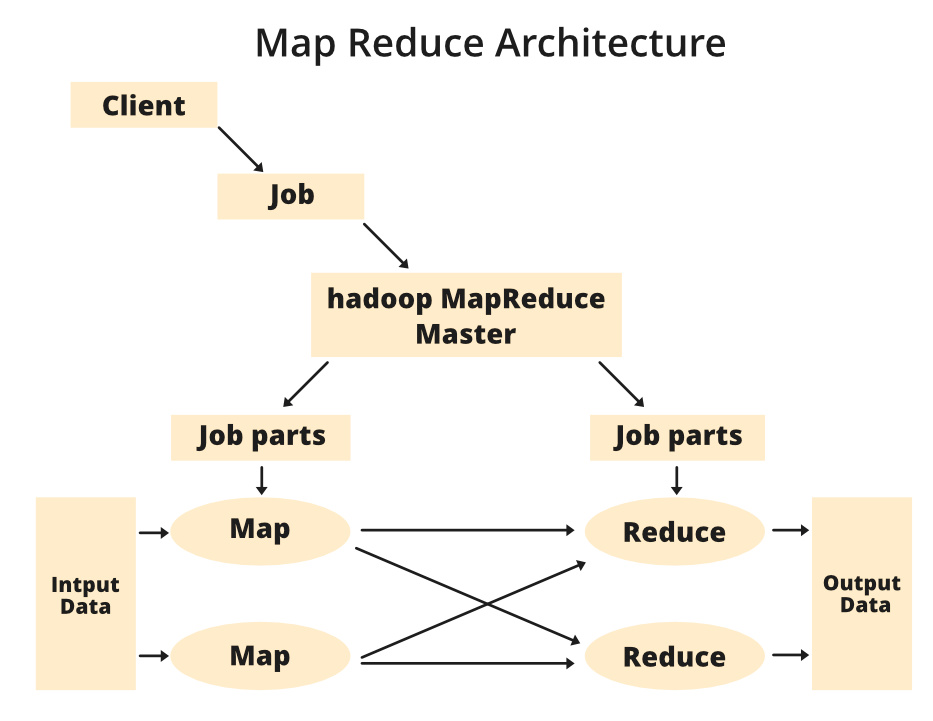
**Map Reduce in Hadoop**

* MapReduce and HDFS are key components of Hadoop, enabling efficient, distributed processing of large datasets.
* MapReduce is a programming model used for efficient processing in parallel over large data-sets in a distributed manner.
* The data is first split and then combined to produce the final result. The libraries for MapReduce is written in so many programming languages.

**MapReduce Architecture:**



**Components of MapReduce Architecture:**

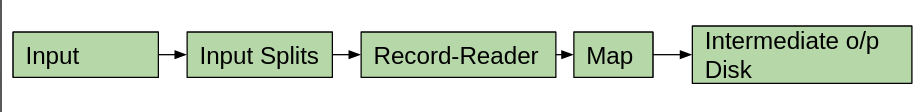
1. **Client:** Submits jobs to the MapReduce system for processing. Multiple clients can send jobs continuously.
2. **Job:** The main work submitted by the client, consisting of smaller tasks.
3. **Hadoop MapReduce Master:** Splits the main job into smaller parts (job-parts) for execution.
4. **Job-Parts:**  Sub-jobs created after dividing the main job; their results are combined to produce the final output.
5. **Input Data:** The dataset provided to MapReduce for processing.
6. **Output Data:** The final result obtained after the processing the input .

The MapReduce task is mainly divided into two phases [Map Phase](https://www.geeksforgeeks.org/hadoop-mapper-in-mapreduce/) and [Reduce Phase](https://www.geeksforgeeks.org/hadoop-reducer-in-map-reduce/).

**Hadoop Mapper :**

* The **Mapper** is a function in Hadoop that processes input records from a file and produces output as **key-value pairs**.
* This output serves as input for the Reducer. The raw input data must first be converted into key-value pairs because the Mapper cannot directly process raw records or tuples.

**Diagram :**



**Components of the Hadoop Mapper**:

**Input**:

* Refers to records or datasets used for analysis.
* Managed by **InputFormat**, which identifies the location of input data stored in **HDFS.**

**Input-Splits**:

* Large input data is divided into smaller **logical splits**.
* Each split is assigned to a separate Mapper.
* Splits don’t contain actual data but point to its location.

**Record-Reader**:

* Processes input-splits and converts them into **key-value pairs** for the Mapper.
* By default, **TextInputFormat** is used to generate key-value pairs from the input, because Mapper can only handle key-value pairs.

**Map**:

* Takes the key-value pairs from the Record-Reader and processes them to produce **intermediate key-value pairs**.

**Intermediate Output Disk**:

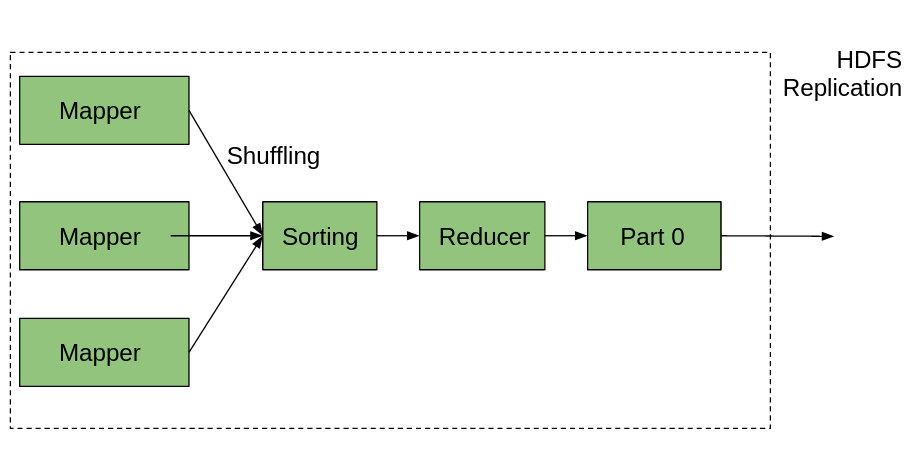
* Intermediate key-value pairs are temporarily stored on the **local disk** (not HDFS) to reduce writing costs and improve efficiency.
* If the job terminates, local intermediate data is easily cleaned up.
* Data is first buffered in memory (default size: **100MB**)
* For **Map-only jobs** (no Reducer), this output is treated as the final result and saved to HDFS.

**In short**: The Mapper splits input data, reads it, processes it into key-value pairs, and stores intermediate results locally for further processing by Reducers or as the final output in Map-only jobs.

**Hadoop – Reducer in Map-Reduce :**

* The **Reducer** is the second phase and processes the output from the **Mapper**, which comes as **key-value pairs**.
* Before reaching the Reducer, the key-value pairs are **shuffled and sorted** based on their keys. This ensures that values with the same key are grouped together.
* The **Reducer** performs computations like **addition**, **filtering**, and **aggregation** on these grouped values. The output generated by the Reducer is the **final result**, stored in **HDFS**.
* By default, only **1 Reducer** is used, but this can be configured based on the user’s needs.

Diagram :



**Phases of Reducer in Map-Reduce**:

1. **Shuffle**:
   * Transfers the output (key-value pairs) from the **Mapper** to the **Reducer** using **HTTP**.
   * Ensures the right partition of data reaches the correct Reducer.
2. **Sort**:
   * Sorts the key-value pairs received from the Mapper based on their **keys**.
   * Groups values with the same key together for processing.
3. **Reduce**:
   * Processes the grouped key-value pairs and performs computations like addition, aggregation, or filtering.
   * Uses **OutputCollector.collect()** to write the final output to **HDFS**.
   * Note: The output of the Reducer is **not sorted**.

**In short**: The Reducer shuffles data from the Mapper, sorts it by key, and processes it to generate the final output.

**Job Tracker and Task Tracker in Hadoop**:

1. **Job Tracker**:
   * Manages **resources** and **jobs** across the Hadoop cluster.
   * Schedules Map and Reduce tasks on **Task Trackers** running on the same DataNode to optimize data locality.
   * Coordinates hundreds of nodes in the cluster.
2. **Task Tracker**:
   * Acts as the **worker/slave** node.
   * Executes the **Map** and **Reduce** tasks as instructed by the **Job Tracker**.
   * Runs on each node in the cluster to perform the actual work.

Example of Map-Reduce: Word Count :

**Input :** cat dog cat bird dog cat

**Map Phase :** The Mapper reads the file **line by line** and converts each word into a **key-value pair**, where the word is the *key* and 1 is the *value*.

**Mapper Output**:

**(cat, 1)**

**(dog, 1)**

**(cat, 1)**

**(bird, 1)**

**(dog, 1)**

**(cat, 1)**

**Shuffle and Sort:** The framework groups the key-value pairs by key (word) and prepares them for the Reducer.

cat → [1, 1, 1]

dog → [1, 1]

bird → [1]

**Reduce Phase:** The Reducer **adds up the values** for each key to get the total count.

**Reducer Output**:

cat → 3

dog → 2

bird → 1

**Final Result :**

The total word count is:

cat: 3

dog: 2

bird: 1

**Advantages:**

1. **Handles Big Data:** Can process very large amounts of data across many computers.
2. **Fault Tolerance:** If one computer fails, others can take over.
3. **Flexible Language Support:** Can be used with different programming languages.
4. **Cost-Effective:** Uses inexpensive hardware.

**Disadvantages:**

1. **Complex Setup:** Setting it up can be complicated.
2. **Slow Processing:** Can be slower due to the need to manage multiple tasks.
3. **Not for Real-Time:** Not suitable for tasks needing instant results.
4. **High Resource Usage:** Needs a lot of computational power and storage.

**Applications:**

1. **Data Processing:** Processes large data sets in various industries.
2. **Log Analysis:** Analyzes server logs for patterns.
3. **Text Analysis:** Handles and examines large amounts of text data.
4. **Machine Learning:** Trains machine learning models with big data.