Jahong Liu

System Design Basics 20: MapReduce

MapReduce

MapReduce pertains to big data processing, enabling the handling of extensive amounts of data, performing computations on it, and producing the results. Imagine having a billion rows of data containing people's names and their Social Security Numbers (SSNs), and our goal is to process this information such that we only retain the names while redacting the SSNs. Accomplishing this efficiently would be a task for Map-Reduce.

MapReduce is a programming model coupled with a specific implementation designed specifically for processing the generating large datasets. This model is particularly beneficial for distributed computing on vast sets of data spanning terabytes or even petabytes.

Before delving too deeply into MapReduce, it's important to discuss two common methods through which data is typically processed.

Batch Processing

In the batch processing model, data is processed in substantial groups, or batches. Here, all data is accumulated over a specified period and subsequently processed as a unit. A practical example would be counting the frequency of each word occurring in a book, or a series of books.

Batch processing doesn't occur in real-time; instead, it takes place when batch jobs are executed. The frequency of these jobs might range from weekly, like generating weekly reports, to daily for producing daily reports.

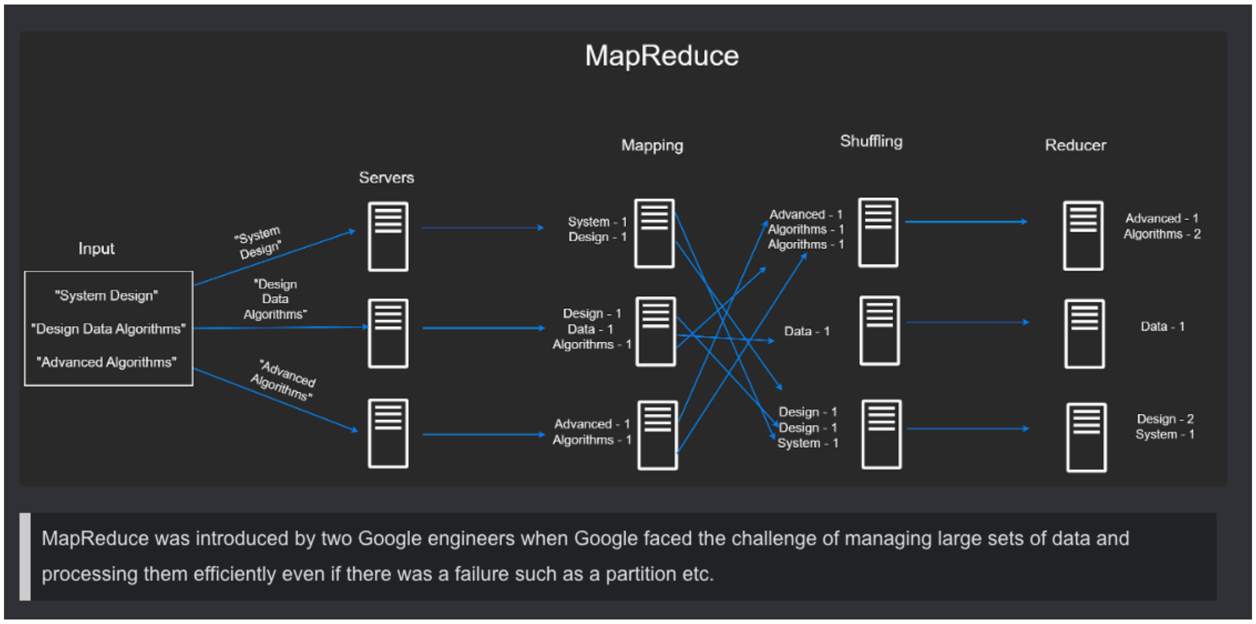
Batch Processing

Stream processing involves processing data in real-time as it is received. Instead of being stored, data is processed individually in its raw, unbatched form. An example of this could be redacting a customer's credit card expiration data or last name upon payment. This task cannot be performed in a batch and must be don't in real-time, since this information needs to be immediately updated for subsequent operations.

MapReduce Function and Implementation

In a MapReduce framework such as Apache Hadoop, the system typically consists of one “master” node and multiple “worker” nodes (also known as “slave” nodes). Here’s how the word count scenario would play out in such a system.

1. Master Node: This node is tasked with managing the distribution of the MapReduce job across the worker nodes. It keeps an eye on the status of each task and re=assigns tasks if any failures occur.
2. Worker Nodes: These nodes are the venues where the actual data processing tasks place. The master node assigns each worker node a portion of the data and a copy of the MapReduce program.
3. Map Phase: Each worker node executes the Map operation on its assigned data portion. In our scenario, this would entail mapping each word to a key-value pair where the key is the word, and the value is frequency of the word.
4. Shuffle and Sort Phase: Following the Map phase, the worker nodes reorganize the key-value pairs to that all values associated with the same key are grouped together. This process is known as the shuffle and sort phase. So, for instance, given the word “The”, if worker 1 processed 3 occurrences, worker 2 processed 7, and worker 3 processed 100, these would be grouped together during this phase.
5. Reduce Phase: The Reduce operation is performed on each group of values, producing a final count for each word. This result is then written to some form of storage or database.



> MapReduce was introduced by two Google engineers when Google faced the challenge of managing large sets of data and processing them efficiently even if there was a failure such as a partition etc.

While MapRedue is a potent technique, it does come with its limitations. A significant one is its restrictive data processing model. It works well with data that fits into the Map and Reduce steps. However, for data processing tasks that don’t align with this model, alternative models may prove to be more suitable.