# Chapter 10
# MapReduce and Hadoop

**Luis Rodero-Merino**
*École Supérieure de Lyon, France & Polytechnic University of Madrid, Spain*

**Gilles Fedak**
*École Supérieure de Lyon, France*

## ABSTRACT

*This chapter introduces the MapReduce solution for distributed computation. It explains the fundamentals of MapReduce and describes in which scenarios it can be applied (basically, processing of massive data by easily parallelizable algorithms). Also, this chapter gives an overview of the open source project Hadoop, an implementation of MapReduce. Its architecture is depicted, and an easy step-by-step guide to install Hadoop is included, along with programming examples of how to use Hadoop.*

## WHAT IS MAPREDUCE?

MapReduce is a solution to address the analysis of huge amounts of data, even in the order of petabytes. The analysis is performed in two steps which are denoted (not surprisingly) *Map* and *Reduce*. The power of MapReduce comes from the fact that each step can be split in tasks to be assigned to different nodes which will run independently and in parallel. MapReduce implementations are usually fault tolerant: if any node fails, the task can be reassigned to some other node. Also, MapReduce shows very good scalability, MapReduce executions can be run on several thousand nodes with the corresponding gain in speed. This makes MapReduce an ideal solution to run distributed tasks on commodity hardware.

MapReduce was introduced in (Dean & Ghemawat, 2008). There, the authors explained how MapReduce emerged as a common solution to address different computation problems, all of them involving the analysis of huge amounts of data by an easily parallelizable algorithm. The solution provides a framework to simplify the programming of those tasks, that at the same time takes care of the addition and removal of hosts, data transfer (in optimal ways), gathering of results, coordination of task executors, execution planning, status reports, etc. As the authors themselves state, this work was inspired by the map and reduce primitives present in some functional languages.

DOI: 10.4018/978-1-4666-0098-0.ch010

## How MapReduce Works

As mentioned before the algorithm has two basic steps: Map and Reduce. At each step a special function programmed by the user is run: one function for the Map step, or Map function; and another for the Reduce step, or Reduce function.

Figure 1 shows an overview of MapReduce running on different nodes. The input data is split into *M* parts (*M* is defined by the user), and each part is sent to one of the nodes running the Map function (Map nodes). The Map function input is a key *K* and a value *V*, the user must define how these keys and values must be extracted from the data. The Map function "emits" several new keys and values associated, possibly in domains different from those of *K* and *V*. As can be seen in the figure, the values associated to each key by the Map function can be different in different nodes. The framework needs to temporarily store those associations (keys and values) as they are the input for the Reduce processing step.

The Reduce step starts once the Map function has finished. As in the case of the Map function, the Reduce function is run in several nodes. The Reduce function has as input the keys and values generated by the Map function. This set is split into *R* parts (as *M*, *R* is defined by the user). The keys are distributed among the Reduce nodes, so if for example key *k1* is assigned to some Reduce node, that node will retrieve all the key-value pairs where the key is *k1* from all the intermediate results created by the Map nodes. See for example how in Figure 1 keys *k1* and *k2* are assigned to the Reduce node on the left, and how all Map nodes send the values associated to those keys to that node (and only to it). When all values from all keys assigned to that node have been received, the Reduce node sorts them, creating a list of ordered values. This sorting process can be regarded as a sub-stage of the MapReduce algorithm, and can demand a high amount of processing power. Then, the Reduce function is called for each key in that reduce node, passing as input the key and its list of sorted values. The output of the Reduce function will be another list of values associated to that key (from the same domain of the intermediate values). Finally, all keys and values are gathered from the Reduce nodes and given to the user.

Both Map and Reduce nodes can be run in the same machine. Also, depending on the amount of Map and Reduce nodes available, each node can run one or more Map or Reduce tasks. Finally, there is a Master node that coordinates the transfer of data, the call to the Map and Reduce function on the corresponding nodes, etc.

## MapReduce as a Cloud Solution

MapReduce defines an approach to deal with certain computational jobs that demand high processing power and operate over big sets of data. It can be implemented in a variety of ways, and depending on its features each implementation can be considered as a cloud solution or not.

To be considered as a proper cloud system, a MapReduce implementation must free users from infrastructure (software and hardware) management concerns, allowing them to focus on their own problem. Such MapReduce implementations can be deemed as a *Platform-as-a-Service* (PaaS) system. They provide a framework and runtime where developers just upload their code (and input data). The MapReduce implementation will take care of handling that code execution (spawning of map and reduce tasks, recovery from failures, load balancing among nodes, etc.) and retrieving the results. In this regard, it follows the same philosophy of other PaaS solutions, freeing developers from the tedious and repetitive tasks related with the platform management (software stack and hosting machines). Hadoop[1], the framework for MapReduce tasks programming that is introduced in this chapter, can be considered as a PaaS cloud system: once it is properly installed in a production environment, developers can use it as the runtime platform for their data processing