The origin of the modern Map Reduce algorithm lies in earlier implementations in functional languages. Specifically the original implementation, that also provided the names map & reduce, was a part of the Lisp language, which was released in 1958.

This basic implementation has two independent components: The map function applies another function on each element of an input list separately and creates an output list of equal length, that contains the individual results.

The reduce function on the other hand also takes another function and an input list, but it applies this function differently. It executes the function on the first two elements and then subsequently on the next element and the previous result. It therefore returns only a single result unlike the map function.

In practical use the functions were often use together. First a map executed a function, that does not depend on relations between the data, on a list, its results will subsequently be "reduced" to a single result.

Although originally purely sequential, newer functional programming languages often implement the map function using implicit parallelism. This is possible, because the map function applies its parameter function on each element of the input list separately and because the functional paradigm demands that a function has input and returns output and that's it. It must not change data, there are no global variables, … . Thus there is no need for manual control over parallel access on shared data using locks, … .

Map Reduce therefore splits the processing into data parallel code and data interdependent code. But Map Reduce is not restricted to one call. The results of an entire Map Reduce iteration can again be the input of a different Map Reduce run, and again, and again.

When Google introduced the modern form of Map Reduce in 2004, there were two main additions to this base concept.

Firstly the simple input and output lists were replaced with key value pairs. Each parallel executed function call within the map phase can therefore return more than just on value that will be entered into a simple output list. It rather emits key value pairs. The values (of all map functions) for each key will be collected into a list. Each key value(-list) pair will be the input for a reduce function.

Secondly the extend of parallelism was massively increased. The modern Map Reduce concept is distributed, not just parallel. Since a mapping function is independent of all data, except for its input data, the mapping functions can be distributed over many different machines. The scalability is only limited by the size of the input data. This has changed the face of Map Reduce from being a simple programming tool to being an entire cluster computing framework.