MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function merges all intermediate values associated with the same intermediate key.

The major contribution of this work is a simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with implementations of this interface that achieves high performance on large clusters of commodity PCs.

MapReduce computations can be expressed by: 1)Distributed Grep, 2)Count of URL Access Frequency, 3)Reverse Web-Link Graph, 4)Term-Vector per Host, 5)Inverted Index, 6)Distributed Sort.
Performance: Measurement of MapReduce on two computations running on a large cluster of machines. 1)One computation searches through approximately one terabyte of data looking for a particular pattern. 2)The other computation sorts approximately one terabyte of data.

Machine Failures/Problems: 1)Large scale machine learning problems, 2)Clustering Problems, 3)Extraction of data used to produce reports of popular query, 4)Extraction of properties of web pages for new experiments and products, 5)Large scale graph computations.

Advantages: 1)Indexing code is simpler, smaller, and easier to understand, 2)The performance of the MapReduce library is good enough that we can conceptually unrelated computations separate, instead of mixing them to avoid extra passes over data, 3)The Indexing process has become much easier to operate.
Disadvantages: 1)Master pings every worker periodically, if no response is received from the worker, the master marks worker as failed, 2)Completed tasks are re-executed on a failure, because their input is stored on the local disks, 3)MapReduce is resilient to large-scale worker failures. 4)MapReduce does not provide support for atomic two-phase commits of multiple output files produced by a single task.

Conclusion: MapReduce programming model has been successfully used at Google. Its success is attributed to several reasons: 1)First, restricting the programming model makes it easy to parallelize and distribute computations and make it fault-tolerant. 2)Second, network bandwidth is a scarce resource. 3)Third, redundant execution can be used to reduce the impact of slow machines.