* The MapReduce computation takes a set of input key/value pairs and produces a set of output key/value pairs.
* The Map function written by the user, takes the input pairs and produces a set of intermediate key/value pairs, where the Reduce function takes the intermediate pairs I via an iterator and produces the output pairs. This process allows us to handle lists of values that are too large to fit in memory.
* MapReduce is resilient to large-scale worker failures. Any map task or reduce task in progress on a failed worker is also reset to idle and becomes eligible for rescheduling.
* Since Network bandwidth is a relatively scarce resource in the computing environment, when running large MapReduce operations on a significant fraction of the workers in a cluster, most input data is read locally.
* A general mechanism to alleviate the problem of stragglers is when a MapReduce operation is close to completion, the master schedules backup executions of the remaining in-progress tasks. The task is marked as completed whenever either the primary or the backup execution completes.
* MapReduce is being evaluated objectively in the article *MapReduce – a major step backward.* The author reasoned from the following aspects, that the MapReduce framework does not have the advantages that Modern DBMS has. It’s also a bad implementation since there is no indexing, so only can be used as a processing option. On the other hand, is not novel, missing features, and is not compatible with other DBMS tools.
* Hive structures data into the well-understood database concepts, so that users can use SQL like language, and it’s also compatible with MapReduce. It still has its limitation, but it’s overall a great tool.
* The three papers all talk about MapReduce to some extent, the first one talks about the mechanism, the second talks about its drawbacks, yet the third one introduced another framework Hive that connects the MapReduce with modern DBMS.