**Brief Summary:**

The large amount of data prompts the computation distributed across many machines to reduce the computation time, which may need more concern such as how to parallelize the computation and distribute the data, etc. By using the definition of map and reduce, MapReduce computes and parallelizes large computations easily. There are many different implementations of the MapReduce and this paper introduces an implementation to the computing environment: large clusters of commodity PCs connected together with switched Ethernet. When the user program calls the MapReduce function, firstly MapReduce library splits the input files into M pieces, and then starts up many copies of the program on a cluster of machines. Among the copies, the master is the special one, which is responsible for assigning a map task or reduce task for each one of the rest (workers), and storing the locations of intermediate key/value pairs on the local disk and the state of workers (*idle*, *in-progress* or *completed*). Map tasks read the contents of input split, parse key/value pairs, pass every pair to *Map* function and create an intermediate key/value pair buffered in memory. A reduce worker uses remote procedure calls to read the buffered data, sorts and groups the data by the intermediate keys, iterates over the sorted data and passes the output to *Reduce* function. The master will wake up the user program if all the tasks have been finished. The MapReduce tolerates machine failures, such like worker failure, master failure and semantics in the presence of failures. Most of MapReduce operations consume no network bandwidth, which is a relatively scarce resource in the environment. The writers often perform MapReduce computations with M= 200,000 and R = 5,000 using 2000 worker machines. To alleviate the stragglers, when a MapReduce operation is close to completion, master will schedules backup executions of remaining *in-progress* tasks, and the task will be marked as completed when the primary or backup execution completes. There are also some useful extensions, such as Partitioning Function, Ordering Guarantees, Combiner Function, Input and Output Types, Side-effects, Skipping Bad Records, Local Execution, Status Information and Counters.