Reading Report - Week 7 **Varad Meru**
CS 221 - Information Retrieval Student # 26648958
Prof. Cristina Lopes Due Date: 02/21/2015

MapReduce: simplified data processing on large clusters* - Summary

By Jeffrey Dean, Sanjay Ghemawat. (Google).

Summary

The authors start with a preamble of *map* and *reduce* from the functional programming paradigm and present a use case where the functionality can be extended for programming in a very large-scale system with huge data storage and processing requirements. The discussion then heads to the programming model where the *input*, *intermediate* and *output* key/value pairs are explained. The user-defined map function takes the *input* key/value pairs and produces *intermediate* key/value pair. The MapReduce library groups together all intermediate values associated with the same intermediate key I and passes them to the reduce function. The user-defined reduce function accepts an intermediate key I and a set of values for that key. The values are supplied to the reduce function via an iterator, which allows for very large lists of values to fit into the memory.

The input keys and values are drawn from a different domain than the output keys and values. Furthermore, the intermediate keys and values are from the same domain as the output keys and values. This notion is explained with the relation between the keys in (1).

$$\begin{aligned} map(k_1, v_1) &\rightarrow list(k_2, v_2) \\ reduce(k_2, list(v_2)) &\rightarrow list(v_2) \end{aligned} \tag{1}$$

The discussion then turns to the implementation details of the MapReduce system at Google. It was a large clusters of commodity PCs connected together with switched Gigabit Ethernet, with dual-processor x86 processors running Linux, and 4-8GB of memory per machine. Machine failures are common in a cluster containing thousands of machines. Storage is provided by inexpensive IDE disks attached directly to individual machines. GFS† is responsible for storage and used to manage the data stored on these disks. The file system uses replication to provide availability and reliability on top of unreliable hardware.

Discussion the heads to the execution of any task. The input data is partitioned into M *splits*, which then can be processed independently and in parallel. The reduce invocations are distributed by partitioning the intermediate key space into R pieces using the partitioning function. One of the examples of the partitioning functions is *hash(key)* **mod** R. The number of partitions, R, is specified by the user. The execution overview can be seen in Figure 1. The steps are as follows: (1) Split input into pieces of typically 16-64 MB and then start many copies of MapReduce program on a cluster of machines, (2) `master` gives away task to `workers`; there are M map tasks and R reduce tasks, (3) Map task parser input key/value through the user defined map function, (4) Intermediate task is periodically written in local disk, and partitioned into R partitions by the partitioner, (5) Reduce task is notified of the available intermediate outputs by the `master` and it starts reading all the intermediate data. Once is has read all the data, it starts either in-memory or externally sorting the data, (6) Once sorted, it iterated over the intermediate data for each unique key encountered. (7) When map and reduce tasks are completed, the master wakes up. The `MapReduce` call in the user program returns back to the user code.

The master monitors the map states. The worker failures are handled gracefully and with speculative execution. A pessimistic view of failures helps the system to the fault tolerant. Locality sensitivity helps in reducing the data transfer and place tasks as close to data as possible. The master must make *O(M+R)* scheduling decisions and keep *O(M*R)* state in memory. Some statistics are presented for examples such as `grep`, Sorting and Large Scale Indexing.

Commentary: It is a seminal work in the space of distributed systems, distributed data management and information storage and retrieval. The paper describes the state of the art in 2008 and how Google's implementation of MapReduce is build, which enabled the next generation of systems to prop up, such as *Hadoop*, *Phoenix*, etc. and spawned the whole ecosystem of softwares in Google, and in the open source community. This paper was instrumental in bringing the BigData revolution. I specially thank Jeff Dean and Sanjay Ghemawat for such a profound gift to computer science and humanity. It is easily one of my all-time favorite papers to read.

*Dean, Jeffrey, and Sanjay Ghemawat. *MapReduce: simplified data processing on large clusters.* Communications of the ACM 51.1 (2008): 107-113.

†Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. *The Google file system.* ACM SIGOPS operating systems review. 37.5. (2003): 29-43.

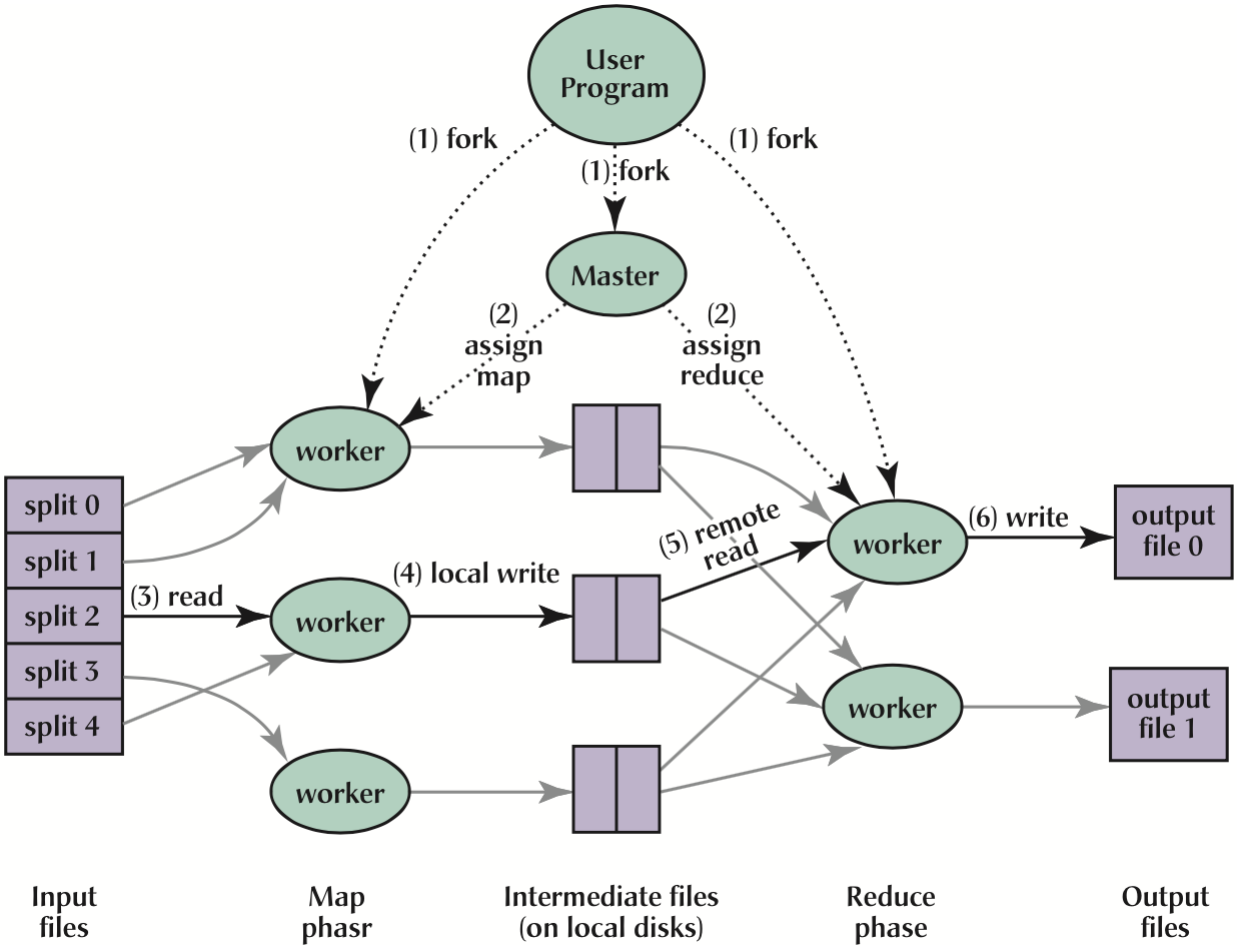

Figure 1: MapReduce execution overview.