Big Data
MapReduce intro

Dr. Wenceslao PALMA
wenceslao.palma@ucv.cl

PONTIFICIA UNIVERSIDAD CATOLICA DE VALPARAISO
Escuela de Ingeniería Informática

What is MapReduce?

MapReduce is a programming model for data processing introduced by Google (2004) to support parallel and fault-tolerant computations over large data sets on clusters of computers. It provides an abstraction that hides many system-level details from the programmer.

Big ideas behind MapReduce

1. Scale “out”, not “up”.
2. Assume failures are common.
3. Move processing to the data.
4. Process data sequentially and avoid random access.
5. Hide system-level details from the applicaion developer.
6. Seamless scalability.

(1) **Scale "out", not "up"**. There is evidence to conclude that a cluster of low-end servers approaches the performance of the equivalent cluster of high-end servers. The small performance gap is insufficient to justify the price premium of the high-end servers.

(1) **Scale “out”, not “up”**. There is evidence to conclude that a cluster of low-end servers approaches the performance of the equivalent cluster of high-end servers. The small performance gap is insufficient to justify the price premium of the high-end servers.

MapReduce is an scale out approach that provides *equitable distribution* and *independence*

(2) **Assume failures are common**. Large-scale services distributed across a large cluster must cope with failures as an **intrinsic aspect of its operation**.

(2) **Assume failures are common**. Large-scale services distributed across a large cluster must cope with failures as an **intrinsic aspect of its operation**.

MapReduce implementations cope with failures through automatic restart and replication.

(3) **Move processing to the data**. In high-performance computing processing nodes and storage nodes are linked togheter by a high-capacity interconnect. However, a bottleneck in the network is created when data-intensive workloads are not very processor-demanding.

(3) **Move processing to the data**. In high-performance computing processing nodes and storage nodes are linked togheter by a high-capacity interconnect. However, a bottleneck in the network is created when data-intensive workloads are not very processor-demanding.

MapReduce takes advantage of data locality by running code on the processor where the block of data we need resides.

(4) **Process data sequentially and avoid random access**. Data-intensive processing is desirable to avoid random data access and instead organiz coputations so that data is processed sequentially.

(4) **Process data sequentially and avoid random access**. Data-intensive processing is desirable to avoid random data access and instead organiz coputations so that data is processed sequentially.

In MapReduce all the computations are organized into long streaming operations that take advantage of the aggregated bandwidth of many disks in cluster. Mapreduce trades latency for throughput.

(5) **Hide system-level details from the application developer**. Programming distributed applications leads to the application developer to deal with several threads, processes, or machines.

(5) **Hide system-level details from the application developer**. Programming distributed applications leads to the application developer to deal with several threads, processes, or machines.

MapReduce addresses the challenges of distributed programming by providing an abstraction that isolates the developer from system-level details. MapReduce maintains a separation of **what** computations are to be performed and **how** those computations are actually carried out on a cluster of machines.

(6) **Seamless scalability**. If running an algorithm on a particular dataset takes 100 machine hours, then we should be able to finish in an hour on a cluster of 100 machines, or use a cluster of 10 machines to complete the same task in ten hours.

(6) **Seamless scalability**. If running an algorithm on a particular dataset takes 100 machine hours, then we should be able to finish in an hour on a cluster of 100 machines, or use a cluster of 10 machines to complete the same task in ten hours.

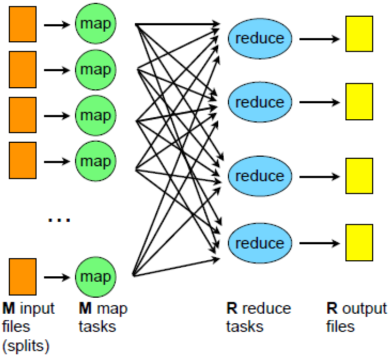
With MapReduce, this isn't so far from the truth, at least for some applications.

MapRaduce is not the first model of parallel computation. However:

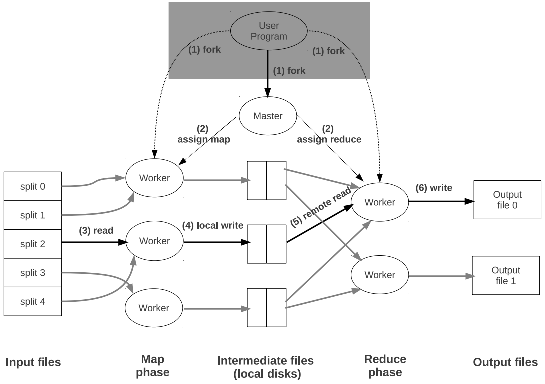
- it has changed the way we organize computations at a massive scale.
- it maked certain large-data problems easier, but suffers from limitations as well.

MapReduce: logical view

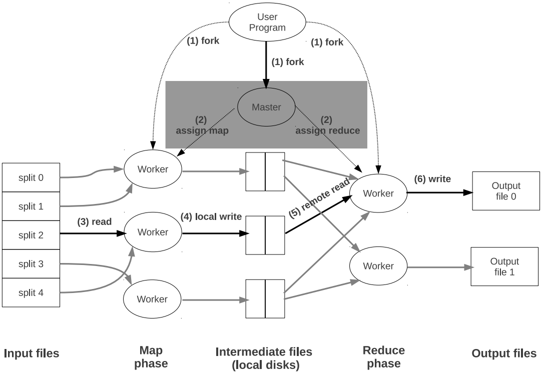
- The input to a MapReduce job is divided into fixed-sized pieces called **splits**.
- A recommended split size is the size of an GDFS/HDFS block (64MB by defautl). However, this can be changed when each file is created.
- Split are processed in parallel by different machines.
- The output ends up in R files on the distributed file system, where R is the number of reducers.



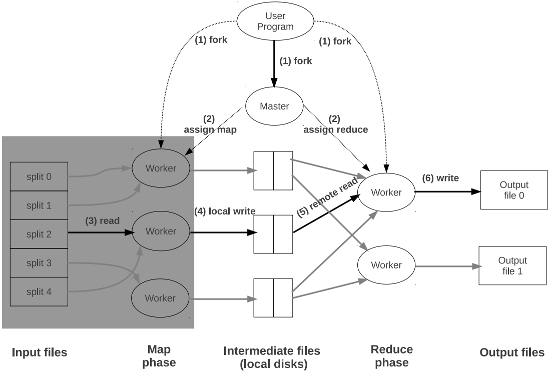
MapReduce: execution overview



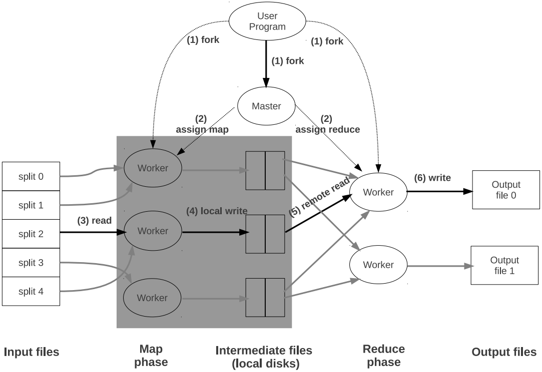
- Mapreduce splits input files into M pieces
- Many copies of the user program are started on the cluster



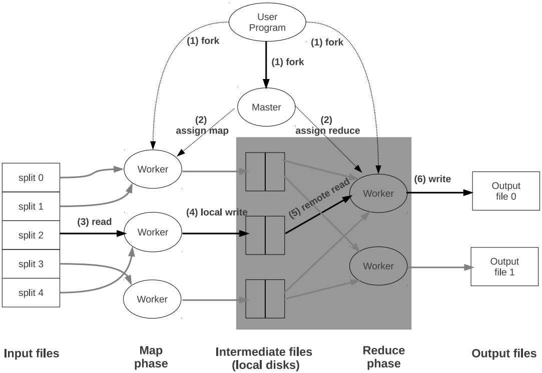
- Master node assigns map or reduce tasks to idle workers.



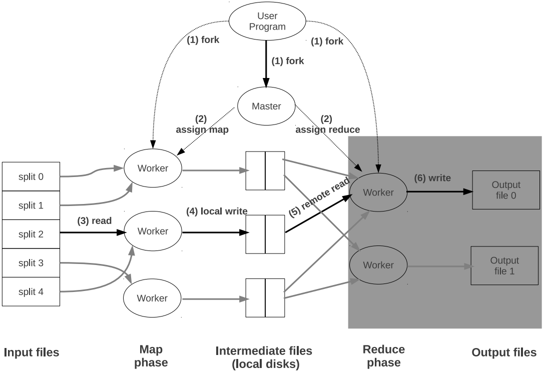
- Workers doing map tasks read a corresponding split
- Intermediate results are buffered in memory



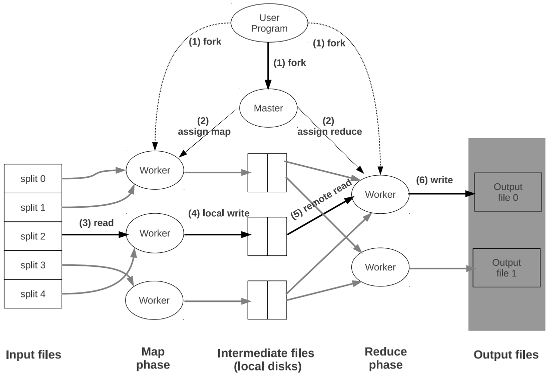
- Periodically, intermediate results are written to local disk.
- These results are partitionated in R regions.
- Locations of these partitions are published to master node.



- Reducers read all input data.
- When reducer has read all input data, it sorts data by intermediate keys.



- Each reducer iterates over sorted intermediate data.
- Output of the reduce function is appended to a final output file.



- The output is available in R output files.
- Typically these files are not combined. They could be kept for an application that is able to handle partitioned data.

References

- *Data-Intensive Text Processing with MapReduce*. Jimmy Lin and Chris Dyer. Pre-production manuscript book, April 2011.