Review - MapReduce: Simplified Data Processing on Large Clusters

Kunal Baweja - kb2896
COMS E6998 Cloud Computing and Big Data

Introduction
MapReduce is a programming model based on the *map* and *reduce* primitives of the functional programming languages, like *Lisp*. MapReduce allows large computations to consume as well produce large amounts of data easily by providing data parallelization via two major functions in it's library: *Map* and *Reduce*.

Design and Working
A programmer using MapReduce to execute large computations needs to specify the input as a key/value pair in a *Map* function, which further generates intermediate key/value pairs and passes on to the *Reduce* job which then distributes the jobs to each of the machines and schedules their execution, combines results and sends back to the programmer as key/value pair.

MapReduce is aimed at providing data parallelization at low computational costs and hence runs on top of commodity hardware, working in close synchronization with the *Google File System(GFS)* designed specifically to store large amounts of data over commodity hardwares and provide fault tolerance. So with respect to data, GFS provides the fault tolerance and high availability of data. In case a component failure also causes loss of computed data then MapReduce simply recomputes that corresponding part before combining the final output.

In brief, a MapReduce job includes following steps:

1. The MapReduce splits the input files into smaller chunks of about 16-64 MB each and the copies of program (computational logic) are stored on each machines of the MapReduce cluster.
2. One machine is designated as the master node and the rest as workers, wherein the master drives the execution of program and each worker handles a map or reduce job.
3. Intermediate key/values generated by the *map* function are stored in a memory buffer and periodically written to disks and this disk address is passed on to the master node.
4. The reduce worker reads the intermediate values from disk performs computation, generates results and writes back to files.

The master node handles all the metadata maintenance and fault tolerance by periodically pinging the worker nodes and if it detects that a worker is not available, it reassigns that worker’s task to another worker node and notifies all other workers of this re-execution of partial computation. This mechanism makes MapReduce highly tolerant towards component failures. Since, there is only one master node, a possibility of it’s failure also needs to be handled and MapReduce does so by marking checkpoints of data stored and allowing rollbacks to previous checkpoints in case of a master node failure.