MapReduce

MapReduce

MapReduce is the key algorithm that the Hadoop MapReduce engine uses to distribute work around a cluster.

The core concepts are described in Dean and Ghemawat.

The Map

A map transform is provided to transform an input data row of key and value to an output key/value:
map(key1,value) -> list<key2,value2>

That is, for an input it returns a list containing zero or more (key,value) pairs:

- The output can be a different key from the input
- The output can have multiple entries with the same key

The Reduce

A reduce transform is provided to take all values for a specific key, and generate a new list of the *reduced* output.
reduce(key2, list<value2>) -> list<value3>

The MapReduce Engine

The key aspect of the MapReduce algorithm is that if every Map and Reduce is independent of all other ongoing Maps and Reduces, then the operation can be run in parallel on different keys and lists of data. On a large cluster of machines, you can go one step further, and run the Map operations on servers where the data lives. Rather than copy the data over the network to the program, you push out the program to the machines. The output list can then be saved to the distributed filesystem, and the reducers run to merge the results. Again, it may be possible to run these in parallel, each reducing different keys.

- A distributed filesystem spreads multiple copies of the data across different machines. This not only offers reliability without the need for RAID-controlled disks, it offers multiple locations to run the mapping. If a machine with one copy of the data is busy or offline, another machine can be used.
- A job scheduler (in Hadoop, the JobTracker), keeps track of which MR jobs are executing, schedules individual Maps, Reduces or intermediate merging operations to specific machines, monitors the success and failures of these individual *Tasks*, and works to complete the entire batch job.
- The filesystem and Job scheduler can somehow be accessed by the people and programs that wish to read and write data, and to submit and monitor MR jobs.

Apache Hadoop is such a MapReduce engine. It provides its own distributed filesystem and runs [HadoopMapReduce] jobs on servers near the data stored on the filesystem -or any other supported filesystem, of which there is more than one.

Limitations

- For maximum parallelism, you need the Maps and Reduces to be stateless, to not depend on any data generated in the same MapReduce job. You cannot control the order in which the maps run, or the reductions.
- It is very inefficient if you are repeating similar searches again and again. A database with an index will always be faster than running an MR job over unindexed data. However, if that index needs to be regenerated whenever data is added, and data is being added continually, MR jobs may have an edge. That inefficiency can be measured in both CPU time and power consumed.
- In the Hadoop implementation Reduce operations do not take place until all the Maps are complete (or have failed and been skipped). As a result, you do not get any data back until the entire mapping has finished.
- There is a general assumption that the output of the reduce is smaller than the input to the Map. That is, you are taking a large datasource and generating smaller final values.

Will MapReduce/Hadoop solve my problems?

If you can rewrite your algorithms as Maps and Reduces, then yes. If not, then no.

It is not a silver bullet to all the problems of scale, just a good technique to work on large sets of data when you can work on small pieces of that dataset in parallel.