Review : MapReduce: Simplified Data Processing on Large Clusters

Mili Shah(mashah@cs.umass.edu)

April 18, 2018

1 Summary

Many tasks, which are simple, take a long time and might become unfeasible because they need to be executed on a large amount of data, for example, inverse indexing of a huge amount of unstructured text, transforming representation of huge graph structures, collecting metadata, etc. Such tasks are usually parallelized across a number of machines, but doing so requires a dealing with a lot of complex issues related to parallelization, distributing data and failure handling.

The authors in this work introduce a programming model MapReduce, and also describe a corresponding implementation for such tasks dealing with big data. For any task that can be expressed as a combination of one or multiple map and reduce operations, the model and the corresponding implementation presented in this work can be used to highly parallelize the task easily, without explicitly programming parts related to parallelizing tasks and distributed systems like partitioning of data, scheduling and failure handling.

A map operation takes a list of key-value pairs and produces one or multiple key-value pairs for each input pair. A reduce operation takes all the key-value pairs generated by the map functions, grouped by values having the same key, and performs operations on such groups. The programmers need to write only the map and reduce functions, and they are automatically run in parallel across many machines, without the programmers having to specify details about parallelization.

The implementation deals with parallelization, fault-tolerance, data distribution and load balancing. Map and reduce tasks are parallelized. They are distributed on a large number of machines. There is one master node and many worker nodes. The data is split, with each split to be given to a map or reduce task. The master node assigns tasks to worker nodes nodes and they perform the tasks. Failure handling is chiefly done by re-execution. A distributed file system handles the storage, with replication for availability and reliability.

2 Strengths

Two very big strengths of this approach are scalability and abstraction.

Map reduce tasks are automatically scaled to a huge number of machines. These machines can also be commodity machines - the implementation of mapreduce does not require specialized hardware. Thus, the implementation is easily deployable and easily highly scalable.

Because a lot of details related to parallelization are abstracted, that is, the programmers don't need to code them explicitly - they only need to specify the map and reduce functions, it becomes possible for programmers without experience in parallel computing and distributed systems to be able to program parallel tasks easily. This also reduces a lot of time required to spend in coding such tasks.

3 Weaknesses and Extensions

- A lot of data is shuffled between different worker nodes, with sorting with map and reduce tasks. Further work can be done to optimize the shuffling or to avoid it.
- This model is not relevant to the huge number of real world tasks that are not expressible in map and reduce operations and is not very efficient for dealing with streaming data, and thus has limited applicability.