**Big Data Processing with Hadoop-MapReduce in Cloud Systems**

In recent years, there has been an exponential growth in data generation by resources like google, Facebook, twitter, other web logs etc. This growth has demanded more advanced and powerful data storage and analyzing tools. One way could be to increase disk space, size of CPU, RAM which is called as vertical scaling. But there is always a limit to these factors. Hadoop and MapReduce provides an easy, efficient way to process large scale datasets. The data generated today from various sources is mainly unstructured and very large. It is not possible to process such a huge data in traditional relational database management systems.

Hadoop is a framework for storing and processing for large data sets on clusters of commodity hardware. It provides two major functionalities- the distributed file system and MapReduce processing. Along with these two, the other well-known sub-projects include HBase, Pig and Hive. HBase hosts very large tables running on top of Hadoop in a distributed and column oriented structure. Hive facilitates faster and ad-hoc querying on HDFS files using Hive SQL.

Hadoop architecture consists of one master node on java Job tracker (a java process) runs and it is responsible for access control. Master node assigns map or reduce task to one of the idle data nodes and stores metadata about all the data nodes. Secondary name node stores all the metadata periodically. But secondary name node is not a direct replacement of name node in case of failure. Several data nodes contain file system data in their own local storage systems. On data nodes, the task tracker processes run which carry our map and reduce tasks.

MapReduce is a programming model for parallel and distributed processing of large data sets introduced by Google. In map phase the inputs files are split into block and each data node performs the user defined map function on part of input file received by it. Output of map phase is set of intermediate key-value pairs which are stored in context. The output of map phase is shuffled and grouped together based on same keys. The master node stores the location of intermediate files and it lets data nodes who perform the reduce task know these locations. Reducers use RPCs (Remote Procedure Calls) in order to read data from mappers and perform some user defined reduce task; usually an aggregation to reduce the key pairs with same key. The final output is written into HDFS. MapReduce model is fault tolerant because it is assumed data, while dealing with huge number of nodes, failure of node is a norm rather than an exception. Failure can be of two types. In case of worker failure, the entire task is performed again. In case of master failure, which is a rare phenomenon, entire MapReduce job is repeated again. The MapReduce model is implemented in following three features: Google MapReduce, Apache Hadoop and Stanford Phoenix.

**MapReduce Is Good Enough?**

In recent times, the Hadoop and MapReduce framework have become the norm for big data storage and analysis. However there are certain algorithms that do not confront with the MapReduce programming model. The general solution to this issue is that, propose an extension and alternative to this problem and develop a new model or framework. The algorithms which do not comply with the Hadoop framework earlier now work fine.

The paper takes a very different approach. Considering Hadoop as “hammer” and all those who are not amenable with Hadoop model as “not nails”. The author suggest that removing all that are not nails rather than creating an altogether new model. Also, finding alternative solutions to problems so that they can easily fit into the MapReduce model. According to author, there are two ways of addressing this issue. One is, improving the same hammer so that it can do some stuff. Second is, to develop entirely new model so that it will perform tasks which fundamentally Hadoop and MapReduce cannot do. But, author suggest that improving for classes on which Hadoop is actually good at does not make sense.

Some of the well-known algorithms for which Hadoop performs poorly are Iterative Graph Algorithms (e.g. PageRank), Gradient Descent, and Expected Maximization. There are some limitations on these algorithms running in MapReduce. So some extensions like HaLoop is proposed which supports iterative graph algorithms. These newer frameworks like HaLoop elegantly solve the drawbacks of MapReduce. But they are no Hadoop. Considering the fact that Hadoop has become trivial in large scale data processing, it becomes difficult to adopt entirely new framework just for iterative graph algorithms.

While describing the Good Enough fundamental, author has made some assumptions: Hadoop has become quite trivial platform for large scale data processing. There is no single framework that can excel in every single aspect. The decision criterion 1 is to develop end-to-end solution by adopting frameworks like HaLoop. At the same time it should be considered that how fruitful the integration of new framework is; with respect to cost incurred. If it does not satisfy this consideration then the Hadoop stack can be considered good enough. While it is better to use a best tool for the job at hand, there are always very high amount of costs involved in designing and joining new framework and new programming models to the existing ones. The author recommends to use the trivial platform that is widely used in the industry, even if it doesn’t solve the issue completely.