**MapReduce is Good Enough Summary**

In this paper the author is trying to make a transmission from the cliché, “If you have a hammer, then everything looks like a nail” to “If all you have is a hammer, then throw away everything that’s not a nail”. This analogy is applied to the fact that a large class of algorithms are not amenable to MapReduce implementations, there exist alternate solution to the same underlying problem that can be easily implemented in MapReduce. The author suggests the alternate technique of avoiding such algorithms which are amenable to MapReduce.

MapReduce lies at the core of an emerging stack for data analytics. There are three large classes of problems that serve as the poster children for MapReduce bashing: iterative graph algorithms (e.g., PageRank), gradient descent (e.g., for training logistic regression classifiers), and expectation maximization (e.g., for training hidden Markov models, k-means).

**Iterative Graph Algorithm**: PageRank is defined as the stationary distribution over vertices by a random walk over the graph. MapReduce jobs have high startup cost, to cope with this shortcoming, a number of extensions to MapReduce or alternative programming models have been proposed like Pregel, HaLoop, Twister and PrIter. But they all have one drawback: They are not Hadoop!

**Gradient Descent**: Each iteration of gradient descent corresponds to a map reduce job. This leads to a few drawbacks like the speed of iteration is bound by the slowest mapper and the combination of stragglers and using only a single reducer potentially causes poor cluster utilization.

**Expectation Maximization:** EM is an iterative algorithm that finds a successive series of parameter estimates that improve the marginal likelihood of the training data, used in cases where there are incomplete (or unobservable) data. Similar to iterative graph algorithms and gradient descent, each EM iteration is typically implemented as a Hadoop job. Hence, it has all the shortcomings of iterative graph algorithm and gradient descent.

The author believes that we must fix all the things we have a good idea how to fix in Hadoop and revisit the issue of whether “MapReduce is Good Enough”. The author believes that this approach of incrementally refining Hadoop has a greater chance of making impact (at least by my definition of impact in terms of adoption) than a strategy that abandons Hadoop. This leads to the conclusion that while we are hammering, nothing should prevent us from developing a jack-hammer.

**Big Data Processing with Hadoop – MapReduce in Cloud Systems Summary**

|  |
| --- |
| There is a massive growth of data to cutting-edge businesses such as Google, Facebook, Amazon, Microsoft, Twitter etc. Hadoop-MapReduce has become a powerful tool which as a computation model to address this problem. Hadoop HDFS has become a more popular amongst Big Data tools as it is open source with flexibility, scalability, less cost of ownership and allows data store of any type. Hadoop MapReduce is a programming model and software framework for writing applications that rapidly process vast amounts of data in parallel on large clusters of compute nodes. |

When a company needed to store and access more data they had one of two choices. One option would be to buy a bigger machine with more CPU, RAM, disk space, etc. This is known as scaling vertically. Scale horizontally means contacting some database vendor to buy a bigger solution. These solutions do not come cheap and therefore required a significant investment. Hadoop-MapReduce programming model consists of data processing functions: *Map and Reduce*. The Map task runs in parallel and produce a collection of <key, value> pair. The processing is managed by a *Task Tracker* and *Job Tracker.*

*Hadoop* is a batch processing system for a cluster of nodes that provides the underpinnings of most BigData analytic activities because it bundle two sets of functionality most needed to deal with large unstructured datasets namely, Distributed file system and MapReduce processing. It has the following components:

* **Core**: Set of components and interfaces that provide access to the HDFS and I/O
* **MapReduce**: A programming model and software framework to rapidly process vast amount of data in parallel on a cluster
* **HDFS**: The primary storage system for Hadoop
* **HBase**: A column-oriented distributed database that maps HDFS data into a database-like structure.
* **Pig**: Platform for analyzing large set of data
* **ZooKeeper**: Cluster configuration tool and distribution serializing manager useful to build large cluster of data
* **Hive**: A data warehousing infrastructure built on top of Hadoop
* **Chukwa**: Used for monitoring large distributed cluster of servers
* **HCatalog**: A storage management layer for Hadoop that enables users with different data processing tools
* **Name Node**: Manages file system metadata and access control. There is only one Name Node in each cluster
* **Secondary Name Node:** Downloads periodic checkpoints from the name Node for fault-tolerance. There is exactly one Secondary Name Node in each cluster.
* **Job Tracker**: Hands out tasks to the slave nodes
* **Data Node**: Holds file system data
* **Task Tracker**: Slaves that carry out map and reduce tasks