**MAPREDUCE IS GOOD ENOUGH?**

**Introduction:**

Mapreduce has become an easy choice programming model for programmers’ building Big Data Systems. This paper focuses towards the algorithms used in the Mapreduce and addresses the issues in using iterative algorithms in MapReduce.

Iterative Graph Algorithms:

**PageRank Algorithm**- in MapReduce, PageRank works in the following way(Process):

1. Graph is serialized as lists with vertex and their current PageRank value.
2. Mapper processes the adjacency lists and emits a destination vertex as “Key” and partial PageRank contribution as “value”.
3. Shuffle stage performs as large group-by on the same destination vertex and sums up the PageRank value.

**Drawbacks:**

1. Each iteration of PageRank requires a MapReduce job. This requires a dozens of iterations to converge and so, a control program to look after the convergence is required.
2. A Crucial factor in this process is that the mapper should emit an adjacency list along with the vertex id as the key. This is extremely important to carry out further iterations.

**Challenges/Shortcomings with Hadoop:**

1. MapReduce jobs have high startup costs.
2. Scale free graphs that follow power laws burden reducer functions by creating stragglers.
3. At each iteration, algorithm must shuffle the graph structure which is adjacency lists from mappers to reducers.
4. PageRank vector is serialized to HDFS at each iteration that provides fault tolerance consuming at the cost of performance.

**Some Alternatives currently are:**

Pregel: Computations are and Algorithms proceed in supersteps with synchronization barriers between each. In this implementation, all state including the graph structure is retained in memory.

HaLoop: An extension of Hadoop that employs various caching mechanisms and with more focus on data locality, schedules tasks for the iterations and thus supporting iterative algorithms.

Twister: Intermediate data is retained in memory, thus reducing the iteration overhead.

PrIter: prioritizes those iterations whose computation converge quickly.

The major issue is that these are not Hadoop and with the growing trend, it is actually expensive to adopt a new data processing framework, just because to compute iterative algorithms.

**Some Points to be noted:**

1. Following schimmy pattern, which involves consistent partitioning and parallel merge join between graph structures solves graph iteration issues in mapreduce.
2. PageRank is not computed from the scratch, the previously updated value in graph is considered for the iterations and so this reduces iterations involved.
3. Graph streaming algorithms for computing PageRank tells that there are non-iterative solutions for iterative algorithms.

**Gradient Descent:**

Gradient descent helps in solving a machine learning problem, supervised classification. This involves setting an input and a desired output for a given set of training samples. MapReduce processes each training example in parallel and provides a partial contribution to gradient which is then emitted to reducer as intermediate key/value pair.

Requirements of MapReduce to perform gradient descent:

1. To arrive at gradient descent, a number of mapreduce programs are to be run and checked for convergence every time, thus requires a driver program to oversee the convergence.
2. Mappers compute the gradient w.r.t training data, which are nothing but updated results from previous iterations. Hence values for mapper computation should be fed every new iteration.

Suggested Alternative: A quasi-Newton approach could be adopted such as L-BFGS.

**Drawbacks currently are:**

1. High startup costs similar to PageRank.
2. Reducer must wait for all mappers to finish and so, stragglers are possible.
3. Using single reducer and taking into account the presence of stragglers causes serious cluster utilization issues.

Alternatives: All the alternatives suggested for PageRank holds good for gradient descent as well, in addition to,

1. Spark uses resilient Distributed datasets, that can be cached from memory or retrieved from durable storage when needed.
2. Bu et al., translates iterative mapreduce and Pregel style programs into recursive queries in Datalog. Using this approach, efficient execution plans are identified by following query optimization techniques in database and these queries executed on Hyracks data parallel processing engine.
3. Online learning technique-stochastic gradient, involve updating the model instead of the parameters in every map reduce job. This way, we get rid of iterative approach and this helps gradient descent achieve accuracy relatively faster.

Stochastic gradient: Involves streaming of training data at single point of time and then updating the model. Even if it has to stream loads of data, the issue could possibly be a bottleneck at disk throughput only. However, this address the iterative approach only. Using an ensemble of classifiers and training each on a data partition provides a better solution. This has proven results at Twitter where, machine learning is integrated with Pig in a scalable fashion.

**Expectation Maximization:**  Involves two steps 1) Expectation-that computes the posterior contribution over the observable data and a set of parameters 2) Maximization – computes the expected log likelihood of joint distribution w.r.t distribution computed in E-step.

Similar to other iterative algorithms, E-step is performed in mapper and R-step in reducer.

Two Examples: Applying Mapreduce to Hidden Markov Models(HMM) and k-means clustering showed improvised results in processing time, however only at the cost of performance. Using a different application framework other than MapReduce might produce a different perspective to this issue altogether.

**Is MapReduce Good Enough?**

Different programming models offer a different perspective to our problems. *The advantages of being able to elegantly formulate a solution in a particular framework must be weighed against the costs of integrating that framework into an end-to-end solution.*

Hadoop YARN is the nextGen resource scheduling abstraction that allows *multiple application framework to coexist on a same physical cluster.*

1. Real time computations on large scale data sets are not suitable for MapReduce application framework.
2. Application frameworks that solve the real time computations at a faster rate are the dire need in this time. However, since they come at the expense of maintenance at production environments, one has to carefully think before making a decision.

**Conclusion:**

In short, MapReduce is not a suitable framework for iterative algorithms however, good enough to work with, when compared to the trade-offs that arrive from other frameworks in place. We need to choose our framework based on the necessity and factors that outweigh the benefits from using MapReduce, if we consider choosing a different framework.

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

**Introduction:**

The growth of Web2.0 companies like Google, Facebook, Twitter, Amazon led to exponential growth in data. Most part of this growth is because of the customer interaction, i.e. data collected from the customer/user and so lead to a necessity to optimally utilize it for their future growth. Thus came the need to store and process data efficiently.

**Big Data and Hadoop:**

Any data that exceeds the minimum storage of a basic hard disk drive available for a retail user at the market right now, (few Terabytes maximum) is to be considered Big Data. Thus the web 2.0 companies that faced the actual issue of processing the data came up with a solution. Google File System is the brainchild of the current Hadoop File System. Yahoo developed HDFS and their open source project was popular in the web arena and well received by other technology giants.

MapReduce is the framework used to program on a Hadoop. There are many other frameworks available in the market both open source and commercial, however, since MapReduce was available since the inception of GFS, it is a popular and widely used Hadoop frameworks.

HBase is a SQL version (Column oriented database) to query on a HDFS. Since the data stored in Hadoop as Files, it required a new environment to accept Java input (Map Reduce programs) and produce database results on the program.

**Architecture of Hadoop:**

1. Name node: Master node that looks after the entire MR program. Assigns mappers and reducers to nodes, and contains metadata about data nodes. Exactly one name node per cluster.
2. Data Node: Replicated data is stored in data node. These involve in processing the data as directed by name node.
3. Secondary Name node: A back up of Name node, that is invoked manually when a name node is down. Contains metadata information present in name node.
4. Job Tracker: Assigns the jobs and monitors through the completion of jobs. Only one for a cluster.
5. Task Tracker: Assigns mapper and reducer tasks to slave nodes (Data nodes). Can be more than one per cluster.

**HDFS and MapReduce:**

Hadoop distributed file system contains a single name node and a bunch of data nodes. The client request is initially sent to the name node and the metadata look up is used to access the data and process it on data nodes. Name node is updated on every activity that is carried out on data nodes. Task trackers keeps track of mapper and reduces tasks performed on each data node. Job tracker keeps track of the tasks performed on data nodes by map reduce programs. The output from data nodes are fed to the client.

**Mapper task:** The data is sent to a mapper where an intermediate key value pair is generated.

**Combiner Task:** Since the intermediate key value pair will be a huge set, it is refined by a combiner, which does a reducer’s task in short and generates lesser key-value pairs.

**Reducer Task**: The data from combiner which is shuffled and sort is sent to reducer, that performs the directed operation on values. The resultant is given as output to the client.

**Fault tolerance:** MapReduce involves fail over recovery. Both name node and data nodes comes with a back-up. If name node is down, a secondary name node is available to take over but the task has to be initiated manually. Data nodes store replicated data in their nodes. A default replication factor of 3 is given when a Hadoop is being set up on a distributed mode. This ensures that even if a data node is down, the data would be available for processing on other nodes.