‘MapReduce: Simplified Data Processing on Large Clusters’

By Jeffery Dean and Sanjay Ghemawat

‘A Comparison of Approaches to Large-Scale Data Analysis’

By Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel Abadi, David DeWitt, Samuel Madden, and Michael Stonebraker

Marcus Anton Zimmermann

10/16/2016

MapReduce: Simplified Data Processing and Large Clusters

The Main Idea

This paper assesses the MapReduce programming model and its implementation for processing and generating large data sets distributed across massive machine clusters. It is a solution to many large computational problems, attracting a large user community for the following reasons:

1. MapReduce is simple, managing to hide the confusing details of parallelization, fault tolerance, data distribution, and load balancing.
2. There is a huge variety of computational problems that can be expressed in the MapReduce format. This includes sorting, data mining, machine learning, and many other systems.
3. MapReduce is capable of scaling to massive machine clusters comprised of thousands of machines, making incredibly efficient use of machine resources.

How MapReduce is implemented

As the name suggests, MapReduce utilizes two essential functions, map and reduce. Both of them are user defined. Input data is split up and distributed across many machines called workers. When the workers are assigned a map task, their job is to read the contents of the input data and parse key/value pairs that are then passed to the map function. The intermediate key/value pairs produced by this function are then used as input data for the reduce workers. Reducers sort all of the intermediate data by the intermediate keys, ensuring all instances of the same key are grouped together. Each intermediate key and its corresponding set of intermediate data is passed to the reducer function. The output of this function is then written to an output file.

Google has implemented MapReduce for large-scale machine learning problems, clustering problems for the Google News and Froogle products, large scale graph computations, and other computational problems.

Analysis of MapReduce and its implementation

MapReduce is certainly an appealing approach to the processing and generating of large data sets. Conceptually, this programming model is simple, hiding many of the complexities often associated with distributed programming. It's ability to handle thousands of terabytes of data across thousands of nodes make it a suitable approach for many computational problems.

With all of that being said, many may argue that the MapReduce framework is too restrictive, limiting what can be done with data. Frequent machine failures, while properly handled in most cases by simply killing processes or by allowing process redundancy, are also a concern. This means that MapReduce, while a powerful approach to big data, is still not perfect and should only be implemented for computational problems in which it is best suited.

A Comparison of Approaches to Large-Scale Data Analysis

The Main Idea

This paper compares two paradigms in large-scale data analysis, MapReduce and parallel SQL database management systems.

MapReduce is relatively new and is drawing a lot of attention as a solution to the processing and generating of large data sets. Parallel DBMSs are equally capable of large-scale data analysis but have been around for more than two decades. This means that they possess a number of significant performance advantages as a result of there many years of development.

Testing results documented within this paper allow us to reach a general consensus regarding the strengths and weaknesses of each framework and a better understanding of the implementations for which they are best suited.

How these ideas are implemented

The MapReduce and parallel DBMS models were tested by completing a series of tasks under three different frameworks. The Hadoop system was used for MapReduce, and DBMS-X and Vertica were used for parallel DBMS. The following tasks were used:

1. Grep Task (includes data loading and task execution)
2. Analytical Tasks (includes data loading, selections task, aggregation task, join task, and UDF aggregation task)

Analysis of these ideas and their implementations

This paper concludes by saying that the DBMS-X and Vertica parallel database systems both significantly outperformed Hadoop's MapReduce system.

This is because of numerous technologies, such as B-tree indices and aggressive compression techniques, having been developed over the past two decades. MapReduce's "Schema Later" or "Schema Never" commitement is also responsible for some of the performance difference.

With all of that being said, it is understandable why systems like Hadoop have become so popular. It is very easy to set up in comparison to parallel DBMSs, and there is also a significant upfront cost advantage.

The final paragraph states that there is a lot to learn from both systems and that they are ultimately moving toward each other.

Comparison of the ideas and implementations of the two papers

The first paper seems to present MapReduce as the best option for large-scale data analysis, providing a number of examples of how Google is effectively using it for data mining, machine learning, and other systems. It hides the complexities of distributive programming, manages to solve many computational problems, and efficiently uses machine resources.

The second paper is certainly different. It makes an objective analysis of MapReduce and parallel DBMS models, concluding that parallel DBMS systems (DBMS-X and Vertica) almost always outperform MapReduce systems (Hadoop). However, the low upfront cost, simple configuration, and massive scalability potential of MapReduce should not be ignored. Ultimately, the two systems will become closely integrated, taking advantage of each other’s strengths.

One Size Fits All – An Idea Whose Time Has Come and Gone

Main Idea

One size does NOT fit all. Stonebreaker goes through a list of markets and discusses the highly specialized forms of relational DBMSs used in each one. Row-based relational DBMSs are soon to have a place in none of them. This is almost already the case. There is now a huge diversity of engines, each one oriented towards a specific vertical or application. Row store is simply not suitable for any of them. Because of this, Stonebraker expects to see many new implementations in the near future.

Advantages and Disadvantages of the main idea of the chosen paper in the context of the comparison paper and the Stonebreaker talk

The second paper and the Stonebraker talk seem to both help and hurt the case for using MapReduce.

On the one side, it’s not row based. As mentioned in the Stonebraker talk, row stores will soon have little to no space in the market. New implementations like MapReduce are specialized forms that are capable of meeting the new demands of big data.

On the other side, MapReduce is just one of many implementations soon to come. This point is made in both the comparison paper and the Stonebraker talk. You would be jumping the gun if you said it’s the best implementation for data analysis.