The Michael Jordan of Data Analysis: A Review of MapReduce

Oliver Tipton

April 14, 2024

Novelty

Parallelized data processes require a high-level understanding of concurrency and parallel computation. However, a model which uses concurrency and parallelism "under-the-hood" requires no understanding of concurrency at all. This is what Jeffrey Dean and Sanjay Ghemawat successfully created with their popular data processing model, MapReduce. Simply put, MapReduce makes processing large scale data sets infinitely easier, particularly with repeated keys and applying a function to aggregate all values for repeated keys. In 2003, Dean and Ghemawat introduced the novel model called MapReduce, as a way to hide the ugly back-bone of parallelized data integration and make harnessing the power of a system of large distributed computers much simpler.

Main Results

MapReduce has been widely adopted and accepted in the Computer Science world. In particular, Google has found a handful of use cases where MapReduce has solved major problems. A couple of these include machine learning problems, large graph computations, web-page optimization, and Google News products. In addition to large companies, users and small-scale programmers have found MapReduce to be incredibly useful. The model relies in

large part on local storage, as it splits large datasets into 16-64 Megabyte size files. While there are thousands of machines used for running MapReduce, bandwidth usage is low and processing time is optimal. Dean and Ghemawat have found that MapReduce is faster, computationally friendly, and ultimately far simpler to use than traditional distributed systems. People with no knowledge of concurrency can use MapReduce to concatenate data sets, and companies like Google can find use cases which solve large-scale machine learning problems.

Impact

Following the release of MapReduce in 2003, the model saw immediate success with over 900 instances of its usage by the end of 2004. As said by MapReduce's developers, it has been so successful because, "it makes it possible to write a simple program and run it efficiently on a thousand machines in the course of half an hour "(146). In addition to time complexity and runtime improvements with MapReduce, there is a large impact as a result of its usability. While understanding complex programs is pivotal in Computer Science, part of what a Computer Scientist should always be aiming for is the continual simplification of complex programs. While Computer Scientists continue to find the cutting edge in computer technologies, the public can continue following down the road of technological innovation and do computations which previously required a high-level understanding of complex topics like concurrency.

In addition to making complex data reductions simpler, MapReduce has become integral at Google. Large Scale Indexing was made computationally less expensive and faster through the use of five to ten MapReduce functions. This made it so that all of the code involved with parallelization and distribution is hidden in the MapReduce functions, cutting the amount of lines of code by over half. In addition, many of the hiccups created by machine failures and slow

runtime are fixed by the MapReduce model with aspects like skipping bad records and backup tasks. Overall, MapReduce has made concurrent computations significantly easier, faster, and more achievable as a result of its "under-the-hood" implementation, error handling, and local data storage.

Evidence

Dean and Ghemawat presented concrete evidence of MapReduce's computational efficiency through performance testing. Within their paper, they completed a performance test which involved a cluster of ~1800 machines. The two computations which were tested individually searched through 1 terabyte of data: the first computation searching for patterns and the second computation sorting the data. These computations were chosen as they "are representative of a large sub-set of the real programs written by users of MapReduce"(144). The pattern recognition computation, also known as *grep*, peaked at over 30GB/s and completed in 150 seconds. The sorting computation completed in around 891 seconds, beating out the top recorded result of 1057 seconds using TeraSort. Additionally, the sorting computation was tested with and without backup tasks (an integral part of MapReduce's efficiency), ultimately showing how having no backup tasks added over 3 minutes to the total computation time! This performance testing with large-scale data models and the most commonly used computations by MapReduce users shows the efficiency and overall success of the model.

Prior Work

The map and reduce functions are an integral part of functional programming languages like Lisp. MapReduce builds upon these functions, making a map reduction seamless and

simple. In addition to building upon Lisp, MapReduce also utilizes the active disks technique. This method pushes computation into processing elements which are close to local disks. This reduces the necessity for network usage through local disk storage, trivially reducing the bandwidth needed during computation. Another work which MapReduce draws from is the Charlotte System. The Charlotte System uses something called the eager scheduling mechanism, which tries to schedule tasks whenever there is an available worker. While MapReduce deploys something similar, it also builds upon computation failures in the Charlotte System as a result of bad nodes; MapReduce simply disregards bad records when they take place.

Reproducibility

Aspects of Dean and Ghemawat's findings would be difficult to produce, specifically section 5 which tested two large clusters of machines (~1800). Since Dean and Ghemawat had access to a large number of computers, they were able to compute different runs using backup tasks, no backup tasks, and measuring data transfer rate down to the minute detail. While I do not have access to this many machines or resources, aspects of this type of test could be run in a significantly smaller test case. For example, trying to sort a large data set with a normal map sorting function and comparing that with runtimes of MapReduce could be a way to measure efficiency. That being said, Dean and Ghemawat's findings would be very difficult to reproduce without access to the number of machines and resources which they had.

Critical Question

As the artificial intelligence sector continues to develop, how could AI be integrated into MapReduce? Is there a way to further utilize MapReduce's capabilities by having AI collect data and perform better reductions based on user input?

Criticism

While MapReduce is successful and efficient for the tasks which it can complete, its scope is somewhat limited. As seen in the evidence section above, the vast majority of MapReduce computations involve either sorting or pattern recognition. These are extremely necessary and powerful computations for data analysis and machine learning. However, they are limited to data sets which have patterns. Computer technologies will continue to grow in size and capabilities and unfortunately outgrow MapReduce and its impact on cutting-edge technologies as they are developed. While MapReduce will always be useful for machine learning and large-scale data, this begs the question: how can MapReduce "level-up" to keep up with the cutting edge instead of getting left behind?

Ideas for further work

Within financial analysis, quantitative researchers find the most optimal investment algorithms to generate high returns. However, I would be curious how MapReduce could be used to aggregate historical financial events and current market conditions. For example, it would be interesting to input a series of financial crises (2008 recession, covid dump, etc.), major upticks in the market (bull markets), and the current market conditions and see if current market conditions were grouped together with the crisis conditions or bull market conditions. Could

MapReduce be used to pick up on less clear similarities between market trends and current conditions, in turn giving investors better tools to pattern recognize investment opportunities?