Big Data Processing: Conceptual Analysis

MapReduce: Simplified Data Processing on Large Clusters – Jeffrey Dean and Sanjay Ghemawat

A Comparison of Approaches to Large-Scale Data Analysis – Daniel J. Abadi et al.

Analysis/Summary by Michael Figueiredo

December 8, 2014

Database Management

MapReduce: A Broad Overview

- MapReduce is a method of handling large amounts of input data through the application of a function or program upon divisions of the data on a network of computers and regrouping that divided workload back into a problem solution that could not have been feasibly or efficiently reached through conventional single-system processing.
- Abstraction of aspects of the MapReduce program implementation allows MapReduce to be made more efficient and self-reliant without the need for users to be very experienced with multi-processing or distributed systems.
- Many real-world problems with big data handling issues can be addressed with MapReduce methodologies and the power of a well-managed parallel system synchronized to handle the problem via data and workload distribution.

MapReduce: Implementation

- A large data set is divided up into numerous splits, each split assigned to a worker computer on the network which processes the data with a specified program/function (map), returning intermediate key-value pairs which are then compiled together by key grouping into a result set (reduce) that matches what would happen in a sequential system, but in a vastly shorter amount of time.
- A master node in the network handles assigning and management of data partitions and load-balancing of work between the worker computers.
- Abstraction of concepts such as hardware fault-handling, result data partitioning/sorting and worker inefficiencies can be self-managed by the MapReduce code along with communication between the worker computers and the master computer so that users of the MapReduce implementation do not need intimate knowledge of the system and distributed processing in general.

MapReduce: My Thoughts

- Having worked with functional languages and the impressive capabilities of the map function, I feel that it is an extremely scalable method of handling data sets of sizes that are becoming commonplace in the technological setting we live within today.
- While it is very useful in many cases as described by the paper, there are still a wide variety of problems that can't be solved using simple functions called upon divided input data on distributed systems. Big data problems involving intrinsic sequential dependencies lack the logical ability to be divided on a distributed system reliably in many cases unlike data division which is often much easier to implement.
- I'm still curious about how much complete re-execution of failed map/reduce split steps impacts overall execution time of the MapReduce function, as the paper was somewhat vague and lacking in statistics on this point. Though the logical reasoning for this decision, forced by the inaccessibility of data or results on failed systems, makes sense to me, I wonder if there is some way to use data replication and step tracking to continue execution on a nearby system in the network to improve execution time.

Is MapReduce the Best Option?

- In *A Comparison of Approaches to Large-Scale Data Analysis,* parallel database management systems (DBMS) are proposed as a long-standing and reliable alternative to MapReduce.
- Parallel DBMS and MapReduce both use similar ideas in terms of data division and parallel execution on smaller subsets of the data, but rigidity of data format and access methods vary between the methodologies.
- Tests of both system types on 100-node networks yielded positive arguments for MapReduce on some tests for much faster data loading times, but program execution times were almost exclusively faster on parallel DBMS as data compression and minimal overhead of system start-up times yielded much faster processing times on most tests.

Pros/Cons of MapReduce

- Pros:
  - Easy to use by end users not experienced with distributed systems/parallel programming
  - Low-cost worker nodes allows large distributed systems to be set up for relatively low cost
  - No need for rigidly structured data sets or formatted access methods for data retrieval
  - Restart node execution allows for recovery from hardware failure faster than DBMS full query restarts
- Cons:
  - Lack of structured data formatting leads to varied program/data structure and less reusable code between MapReduce use cases
  - Increased communication overhead between nodes for implementation of node synchronization
  - Addition of less expensive worker nodes and, in turn, increase in hardware failures adds to execution times
  - Compression methods are less effective than on parallel DBMS, negatively affecting performance times