Big Data Paper Summary

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

A Comparison of Approaches to Large-Scale Analysis

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

Michael Stonebraker at ICDE 2015

Christopher Lee
5/7/2015

MapReduce Overview

- Programming model and implementation
- Generates and processes large data sets with a parallel, distributed algorithm
- Input key/value pairs, output key/value pairs
- Computation comprised of two functions: *Map* and *Reduce*
- Abstraction inspired by primitives of the same name in Lisp and other functional languages
- Simple and powerful interface that enables automatic parallelization and distribution

MapReduce Implementation

- *Map* produces intermediate key/value pairs
- All key/value pairs with the same intermediate key are grouped for *Reduce*
- *Reduce* merges all values of the same intermediate key for a set of smaller values
- Potential Uses
  - URL Access frequency count (map logs web page requests, reduce adds all the requests for a count)
  - Inverted Index (map produces a <word, doc> sequence, reduce produces a <word, list(doc)>
- Rewrote the indexing system for Google search

Personal Analysis of MapReduce

- MapReduce separates base code from parallelization and distribution code
  - Makes code easier to read and debug
- Great deal of practical applications to split big data into *map* and *reduce*
- Potential system slowdowns such as machine failures do not require operator intervention anymore
  - MapReduce's parallelization and distribution code takes care of all of that automatically
- I like the concept and implementation

Big Data Approaches Comparison

- Basic control flow of parallel SQL DBMS has existed for a while; cluster computing
  - Harnessing a large amount of low-end processors in parallel to solve big data problems
- MapReduce (MR) – a new computing model
- DBMS requires data to conform to a schema
- MR allows data to be in any arbitrary format
- DBMS performance is better than MR
- MR loads data and tunes parallel computing much faster than DBMS

Comparison Implementation

- Hadoop: MapReduce implementation by Yahoo! and Apache Software, DBMS-X, and Vertica: column-store rather than row-store
- Data Loading: Hadoop was the most efficient, copying from local disk to HDFS to cluster node
- “Grep Task”: Hadoop was the least efficient; very little data being processed so Hadoop's expensive startup cost becomes a problem
- Various other tasks indicate parallel database systems have a significant advantage over Hadoop MR

Personal Analysis of Comparison

- Apparently Google's MR is much faster than Hadoop's MR, but this does not matter
  - MR always starts with a full scan of input files
  - This puts a lot of processing power on startup
- MR's advantage comes when a lot of data is being processed over time
- Once MR starts up, additional data processing tasks require less processing power than the parallel databases

Stonebraker Talk Overview

- Attempts in 1970-2000 to make RDBMS the “answer”, or “one size fits all”
- Six markets are mentioned, all of which do not benefit from RDBMS over other databases
- Column stores, Key-value stores, Record stores, Big Table stores, JSON stores
  - All seeing use over RDBMS
- Main point: Many database markets are designing and implementing unique database systems for specific needs
- The mistake of the 90's: One size cannot fit all

Compare, Contrast & Analyze

- I chose MapReduce, whose main benefits seems to be the following:
  - Readability and ease of use for programmers
    - Functionality comprised of *Map* and *Reduce*
  - Separates base code from parallization and distribution code
  - Efficient at processing big data
- Drawbacks are:
  - Slow “startup” time, scans entire input file
  - When processing low quantities of data, this becomes the biggest drawback, requiring extra time to start up over other parallel database implementations