Comparison of Big Data Strategies

Bigtable: A Distributed Storage System for Structured Data

vs.

A Comparison of Approaches to Large-Scale Data Analysis

By John Paul Welsh

8 May 2015

Chang, Fay, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. "Bigtable: A Distributed Storage System for Structured Data." *ACM Transactions on Computer Systems* 26.2 (2008): 1-26. Web. 6 May 2015.
A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," in *SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data*, New York, NY, USA, 2009, pp. 165-178.

Bigtable: Main Idea

- Multidimensional sorted map indexed by string keys
  - (row, column, timestamp) → string_value
  - Sorted lexographically by row key
- Row range = “tablet”
- Update to row key is atomic, regardless of # of columns involved
- Cell can contain multiple versions of same data, arranged by decreasing timestamp order
  - Old versions removed by garbage collection on a customizable basis
- Chubby (distributed lock service)
  - Holds onto a set of directories and files which are used as locks

Bigtable: Details

- Component 1) Library that is linked to every client
- Component 2) Tablet servers (many) that manage tablets
  - One tablet in one server at a time
- Component 3) Master server (one) that manages tablet servers
  - No direct interaction from clients
- Chubby keeps track of who has which locks, and manages granting and revoking then as necessary
- METADATA table keeps tabs on a lot of things
- Memtable holds recently committed updates, sequence of Google File System SSTables holds older ones
- Major/minor compactions help keep SSTables/memtable efficient and small

Bigtable: My Analysis

- Lexographical sort keeps related elements near each other, increasing query speed and simplicity
- Clients interact directly with tablet servers, don’t have to go through master servers
  - Master server has easier workload
- Chubby is very cognizant of which locks are needed where, and for how long they are to be kept

Stonebreaker Paper: Main Idea

- In light of MapReduce technologies, is it really all that much better than a distributed relational database?
- MapReduce = Map (apply a transformation to each "record" from file, split into "buckets") + Reduce (combine bucket contents into one output file)
- Parallel DBMS = SQL query is split across parallel clusters, joined back at the end

Stonebreaker Paper: Details

- MR is fine for small teams but a hassle for bigger teams
- Flexibility
  - No inherent structure to MR, developers need to agree on one for themselves
- Usability
  - DBMS narrows down working set more efficiently than MR, deals with less data at a time
- Reliability
  - No inherent schema means the DB can't enforce consistent rules by default
  - MR has better fault tolerance because the failed task can be localized and restarted without messing up the rest. DBMSs need to restart everything if there is a failure

Stonebreaker Paper: My Analysis

- In the experiments run by the team, MR was better than the DBMSs in terms of ease of setup, extensibility, and minimizing work done in the event of a failure
- On the other hand, DBMS was better in performance overall, while also being more energy-efficient and having the benefit of 25+ years of improvements and enhancement tools at its disposal

Comparison of Two Papers

- Bigtable is a very abstract concept that has a rather large learning curve to maneuver, while relational DBs have nearly household-level recognition
- Many of the shortcomings of MR strategies could have been fixed since the publication of *Comparisons* paper
  - Most of the criticisms are common for when comparing new technologies to old ones

Bigtable in Light of *Comparisons*

- Advantages
  - Can perform many intense processes without needing to force SQL to fit around the specifications of the task
  - Ease of use
  - Error handling much more straightforward and time-efficient than for DBMSs
- Disadvantages
  - Seems suited for some problems but not others
  - Potentially more wasteful of energy
  - Requires developers to do a lot of the legwork themselves, rather than use DBMSs built-in functionality to simplify operations