Big Data Presentation

Katharine Craven

05/08/15

Building a High-Level Dataflow System on top of Map-Reduce: The Pig Experience

Alan F. Gates, Olga Natkovich, Shubham Chopra, Pradeep Kamath, Shravan M. Narayanamurthy, Christopher Olston, Benjamin Reed, Santhosh Srinivasan, Utkarsh Srivastava

A Comparison of Approaches to Large-Scale Data Analysis

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

One Size Fits All - An Ideas Whose Time Has Come and Gone

Michael Stonebraker, Ugur Cetintemel

Pig - Main Idea

Pig is a high level dataflow system that strives to allow for high level data manipulation while keeping the expressive generality of Map-Reduce. Map-Reduce has several weaknesses that Pig attempts to address, such as a lack of support for complex N-step dataflows and combined processing of multiple data sets, and the need to code data manipulation primitives by hand. By using Pig Latin, the language Pig compiles, a programmer can program at a higher level. Pig takes a Pig Latin program and compiles it into Map-Reduce jobs, then executes those jobs using Hadoop.

Pig handles data types, Map-Reduce optimizations, and user-defined functions written in Java. Pig struggles with memory management in a lot of situations. It also lacks certain optimizations, such as indexing and column groups. Pig also lacks support for languages other than Java at this time. Many features will hopefully be added to Pig in the future. In the hopes that Pig will continue to improve, it has been a goal since the beginning to make it open source. It is expected that Pig will continue to gain popularity.

Pig -Implementation

The data types Pig supports are int, long, double, chararray, bytearray, map, tuple, and bag(a collection of tuples). Bytearray is used for unknown data types and lazy evaluation. Type can be explicitly defined by the programmer as well. Pig programs go through three steps before being executed. First is parsing, which performs syntax and type checking, as well as schema inference. Next, the parses passes through a logical optimizer, where logical optimizations are carried out and compiled into Map-Reduce jobs. Finally, the jobs are sorted and submitted to Hadoop for execution in their sorted order.

In the Logicial-to-Map-Reduce compilation, the logical plan is turned into a physical plan, which is then turned into a Map-Reduce plan. If a physical plan contains more than one STORE command, the generated physical file plan contains a SPLIT physical operator. The MULTIPLEX operator sends tuples to their correct sub-plan. These operators were added so that a data set could be processed in multiple ways without having to read it several times. The downside to this could cause memory problems. Pig leaves the decision of whether to multiplex or not to the user. Once the Map-Reduce plan has been generated, distributive and algebraic aggregate functions broken into three steps, which are assigned to map, combine, and reduce stages. The final compilation step is to convert all Map-Reduce combinations into a Hadoop job description and to send it off for execution.

Pig is implemented in Java, and as a result had difficulties resulting from the fact that Java does not allow direct control over memory allocation. There is commonly trouble with memory overflow dealing with large bags of tuples. User defined functions must be in Java. Streaming allows data to be pushed through external executables.

Pig Analysis

I think what the Pig system is trying to accomplish is very useful. Certainly having code that is easier to read and more familiar is a useful quality of this system. It retains the fault tolerance of MapReduce as well as other qualities many people find attractive. The memory problems in Pig should be addressed in the future, as it appears to me that there are a lot of ways you could currently run into memory overflow. Perhaps Pig could look into utilizing languages other than Java that would allow for more control over memory.

Pig feels somewhat as though it is a large project rather than a finished product. The paper mentioned various optimizations it was looking forward to, but which hadn't been implemented in either Hadoop or Pig yet. For example, the essay claims that once a Map-Reduce plan has been generated, there may be room for many optimizations, but that only one has been implemented. It also mentions that it currently misses out on various optimized storage structures.

Comparison - Main Idea

The paper attempts to compare the performance of parallel database systems versus that of MapReduce. MapReduce is loved for its simplicity, containing two functions. Map reads a set of records from an input file, performs filtering/transformations, and outputs a set of records with new key/value pairs. Reduce combines records assigned to it and and writes to an output file. MapReduce system is not required to follow a relational schema. However, to make up for this, a lot of additional coding must be done. Additionally, MapReduce has a better fault tolerance than parallel systems.

They run various tests on two parallel systems (DBMS-X and Vertica) and on one MapReduce system (Hadoop). These tests included comparisons on data loading time, task execution time, task startup, failure model, compression, and ease of use. There were different tradeoff in different categories, and the paper suggested that one could learn from both types of systems.

Comparison - Implementation

When it comes to ease of use, MapReduce suffers in how it can be shared. Because there is no necessary schema, programmers must all agree on and individually code the desired structure. However, MapReduce is more expressive than SQL in general. In the ease of use test, it was found that it was easier to setup Hadoop than the other systems, but finding optimal configurations was trial and error due to lack of tools. MapReduce also has a superior failure model compared to parallel systems as well. If a single node fails during a query on a parallel system, the full query has to be restarted. When it came to the load times test, Hadoop clearly won in the Grep test, but in analytical task where data needed to be modified, it performed much slower than the other systems.

In terms of task execution and startup, Hadoop's startup time limits its performance, causing it to perform the worst of the three. MapReduce takes some time before all nodes run at full capacity. Parallel databases, however, are started at boot time and thus are always "warm", or waiting for query. Parallel databases can also compress data, which can be optimal if the executor can operate on compressed data directly. However, in Hadoop it was shown that compression added no benefit and sometimes resulted in slower execution.

Comparison - Analysis

In general, I feel like parallel databases came out better out of the systems tested. The best quality of MapReduce appears to be its failure handling. The expressiveness of the language becomes a hindrance if it has to be changed for different tasks and difficult to share. I'd imagine the system will definitely need to be changed and maintained by various people, so I feel as though that is a very big flaw.

MapReduce performed a lot slower in general, save for Grep load times, so based on the examples I was given, I feel that performance-wise, there is not a lot of use for MapReduce except under specific conditions.

Pig/Comparison Analysis

Pig handles the code problem that was brought up in the Comparison paper. I feel like Pig makes MapReduce much easier to share. Pig also seems to better handle data processing, something that cost MapReduce a lot of performance time when going up against parallel systems.

If Pig could be further developed to optimize the computations in MapReduce, it could possibly be enough to offset MapReduce's performance issues while retaining more generality and better failure handling than parallel systems.

Stonebreaker Main Ideas

From approximately 1970-2000 it was believed that the relational database model was the one solution, or “one size fits all”. As time went on, it was discovered that not only did this data model not cater to every market, but that it may in fact be obsolete. Column stores are two orders of magnitude faster than rows stores, and are being used in data warehouses. OLTP is moving towards main memory deployments. The Complex Analytics Market, Streaming Market, and Graph Analytic Markert all are a poor fit for the traditional database model. There is even a NoSQL market, which has very little standards implemented at all, and many ideas. Overall, the idea that relational databases were the one answer was incorrect, and in fact, the old model may be good for nothing.

Pig Advantages/Disadvantages

Advantages:

- Not forced to comply with a specific schema (no need to comply with the relational model, which is possibly outdated, and could use whatever model best fit the situation).
- Good Failure Handling
- High-level language for ease of programming

Disadvantages:

- Memory Problems
- Currently lacks optimizations such as column store and indexing
- Only supports Java