Big Data Summary

Map Reduce: Simplified Data Processing on Large Clusters[1]

vs

A Comparison of Approaches to Large-Scale Data Analysis[2]

By Cassandra Graves

[1]Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (January 2008), 107-113. DOI=10.1145/1327452.1327492 http://doi.acm.org/10.1145/1327452.1327492

[2] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker. 2009. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data* (SIGMOD '09), Carsten Binnig and Benoit Dageville (Eds.). ACM, New York, NY, USA, 165-178. DOI=10.1145/1559845.1559865 http://doi.acm.org/10.1145/1559845.1559865

Main Idea About MapReduce

- MapReduce is a programming model that is easy to use
  - Programmers without experience with parallel and distributed systems can easily use MapReduce
    - Hides parallelization, fault-tolerance, locality optimization and load balancing from the programmer
- Users must specify two functions
  - *Map*, function written by user, takes an input pair and produces a set of intermediate key/value pairs. All intermediate values associated with the same intermediate key I are grouped together
  - *Reduce*, function written by user, accepts an intermediate key I and a set of values for that key. Values are passed via iterator, which allows us to handle lists of values that are too large to fit in memory
- MapReduce automatically parallelizes and executes on a large cluster of commodity machines
- MapReduce uses redundant execution to reduce the impact of slow machines as well as to handle machine failures and data loss

Implementing MapReduce

- Implementation varies - based on environment being used upon
- Example implementation:
  - Dual-processor x86 machines running Linux, with 2-4GB of memory per machine
  - Commodity networking hardware
  - Cluster has hundreds or thousands of machines
    - Machine failures expected
  - Storage using IDE disks on each machine and a distributed file system to manage data
  - Submitted jobs pass through a scheduling system
    - Each job’s set of tasks is assigned to available machines within the cluster

- 7 Step Execution
  1. The MapReduce library splits the input files and starts copies of the program on a cluster of machines
  2. One copy becomes the master and it assigns work to the remaining
  3. A worker reads the input, parses key/value pairs to pass into the Map function and are buffered in memory
  4. The buffered pairs are written to local disk, partitioned into R regions and forwarded to the reduce workers
  5. Reduce workers read the buffered data from the local disks, sorts the data by key to get all occurrences of the same key together to do same reduce tasks
  6. Reduce worker passes the key and corresponding values to the Reduce function and the output gets stored in a final output file for this reduce partition
  7. Once all map and reduce tasks are done, control is given back to the user code

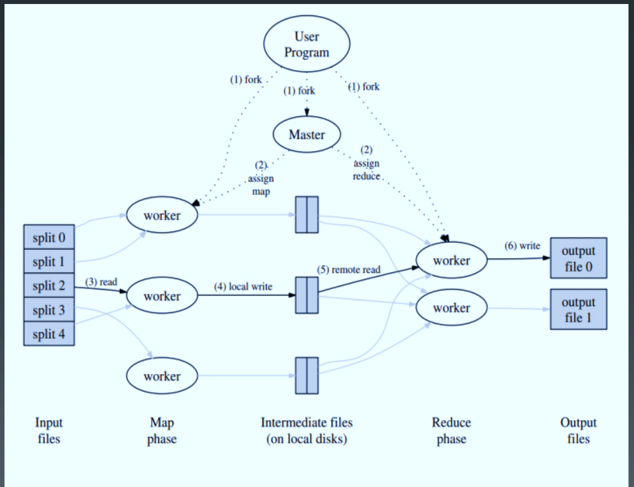

Figure 1: Execution overview

Analysis

- MapReduce appears to be an easy tool to use when looking to distribute work over a cluster of machines
- Since it automatically takes care of coordination between machines in the cluster, that would improve productivity of the programmer
  - More time to add to the tasks versus taking up time coordinating
- Graceful fault-tolerance in MapReduce is a motivating feature, as the program won’t crash if one machine fails
  - As this takes time, it would be wasteful to have to restart the entire program if one machine were to break down
- Use of local storage on machines after *Map* seems helpful, but if a machine only finished *Map* when it fails, then *Map* must be done again
  - Seems redundant and counterproductive if/when many machines fail
- Redundant execution is used towards the end to ensure completion time isn’t halted by a slow machine
  - Seems like a good idea, but a lot of overhead is being added into the program when in most cases, it might not need to
- Google uses it – so it must be good.

MapReduce vs Comparison Paper

- Comparison paper supports claim of MapReduce's simple model to utilize a cluster of machines
  - Mentions specifically as a learning tool
- Parallel DBMS also provide a high-level programming environment
- Parallel DBMS need a well-defined schema for data, instead of a schema, MR needs a custom parser to gather data
- Unlike Parallel DBMS' use of hash or B-tree indexes, MR does not provide indexing, which is complicated to implement and maintain
- DMBS use straight forward requests for information versus MR needing to provide an algorithm to request data
- Basic database tasks take significantly longer to load and execute for MR then it does for Parallel DBMS
- MR performs complete table scans, while DBMS takes advantage of clustered indexes – MR deals with more overhead
- MR has a slow-to-begin "cold start" versus the quick-and-ready "warm start" that DBMS has
- Both are easy to use, but MR is difficult to maintain once beyond beginner concepts

Advantages/Disadvantages

- Advantages
  - With independence of each machine in the cluster and no schema required, MapReduce is very flexible
  - Creation of high-level languages Pig and Hive to alleviate implementation of repetitive tasks done with MapReduce
  - Best implementation of fault tolerance
  - Less challenging to install and configure properly MapReduce(Hadoop)
  - Minimizes work lost when a machine in the cluster fails
- Disadvantages
  - With each machine being independent in the cluster, and not having a defined schema, MapReduce could easily be corrupt by bad data
  - Forced to write algorithms in a low-level language in order to perform record-level manipulation in MapReduce
  - Reduce creates multiple output files, so another instance of Reduce would be necessary to combine them all into one file
  - Load and execution time of basic tasks take significantly longer time for MapReduce(Hadoop) versus DBMS (Vertica & DBMS-X)
  - Extensive amount of overhead