MapReduce Notes

February 6, 2019

The big Data revolution could not be overcome by traditional relational databases. Moreover, parallel and distributed database systems faced challenges in achieving linear scalability.

In 2003/2004 Google published a paper on Google File System(GFS) and MapReduce which started the craze!! Google was facing issues in scalability due to the ever growing raw data generated by Web 2.0, social media (facebook, twitter, etc.), logs, user profiles (searches, click, transaction patterns, etc.). Specifically, Google wanted a scalable, self-managing, distributed processing system to regularly compute PageRank of the crawled web. Their goal was to build such an infrastructure on a large collection of inexpensive commodity hardware connected by Ethernet cables or inexpensive switches to perform processing of raw data....i.e. move away from expensive bulky compute servers and distributed databases.

Google published two papers based on their internal infrastructure:

- **Distributed File System (DFS):** larger file units that provide replication of data or redundancy to protect against the frequent media failures that occur when data is distributed over thousands of low-cost compute nodes.
  Ghemawat, S., Gobioff, H. Leung S. The Google File System. 19th ACM Symposium on Operating Systems Principles,Lake George, NY, October, 2003.

- **MapReduce :** framework that allows users to run applications on computing clusters efficiently and in a way that is tolerant of hardware failures during the computation.
  Dean, J. and Ghemawat, S. MapReduce: simplified data processing on large clusters. Communication of ACM 51, 1 (Jan. 2008), 107-113.

0.1 MapReduce

MapReduce is a programming model Google has used successfully to process its big-data sets (20000 peta bytes per day). Users specify the computation in terms of a map and a reduce function, and the underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, and handles machine failures, efficient communications, and performance issues.

MapReduce is designed for one-off processing tasks, such as Extract, Transform and Load(ETL) and read-once applications. It is not designed for real-time processing or transaction-based applications. MapReduce provides an abstraction that allows engineers to perform simple computations while hiding the details of :

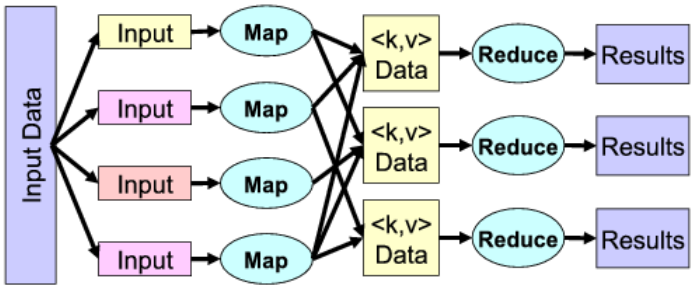
- Automatic parallelization of jobs
- I/O scheduling
- Data distribution Load balancing Fault tolerance

The canonical use of MR is characterized by the following template of five operations:

- Read logs of information from several different sources;
- Parse and clean the log data;
- Perform complex transformations on the data;

- Decide what attribute data to store;
- Load the information into a DBMS or other storage engine; and
- Complex analysis on various data from different sources

Google's MapReduce infrastructure was also made into a open-source platform with Yahoo's efforts. Yahoo called this platform 'Hadoop'. Hadoop's initial design was based on Google's MapReduce paper but many improvements and policy changes have been applied to this platform since the Google paper was published.



MapReduce contacts HDFS to locate the input file's relevant chunks and determines the 'input-splits' (essentially the records) that should be processed. Each Mapper job receives a set of input-splits which it processes to generate a key-value pair. Once all Mappers complete processing, the key-value pairs are then sorted and grouped by key. Each Reducer will receive all values for a given key, which it then will aggregate to generate a result.

The MapReduce flow can also include a 'Combiner' and 'Partitioner' for optimization. The Combiner class is used in between the Map class and the Reduce class to reduce the volume of data transfer between Map and Reduce. Usually, the output of the map task is large and the data transferred to the reduce task is high.

Combiners are an optimization in MapReduce that allow for local aggregation before the shuffle and sort phase. When the map operation outputs its pairs they are already available in memory. When a combiner is used then the map key-value pairs are not immediately written to the output, but instead grouped by key. Hence, doing a reduce-type function, but the operation must be associative and commutative.

The Partitioner class is used in between the Map class and the Reduce class (after the Combiner... if a combiner is used). Partitioners are responsible for diving up the intermediate space and assigning intermediate key-value pairs to Reducers. Partitioners are used to ensure that approximately the same number of keys are assigned to each reducer (dependent on the quality of the hash function).

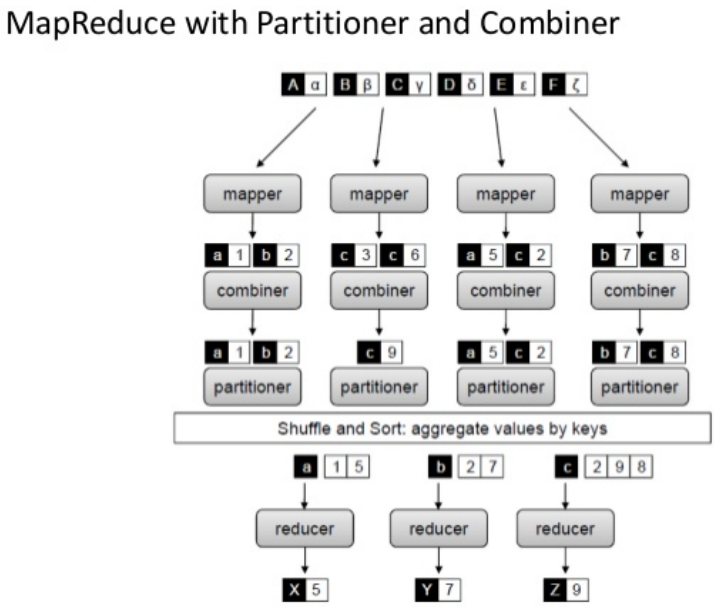
0.2 Google Distributed File System (GFS)

The Google Distributed File System (GFS) is a file system for large units of data that provide replication and redundancy to protect against the frequent media failures that occur when data is distributed over thousands of low-cost compute nodes.

A large file is split into chunks (64MB blocks) which is distributed across **'chunk servers'.** The chunks are replicated 2-3x for fault tolerance in case any chunk server goes down. The location of a given file's chunks is stored at the **'master node'** which maintains the meta data about the file and the chunk servers and manages the chunk servers. Client library contacts the master node to find the relevant chunk servers that contain the various chunks of a given file. Once the client knows all the relevant chunk servers, it will then contact the chunk servers directly to access the data.

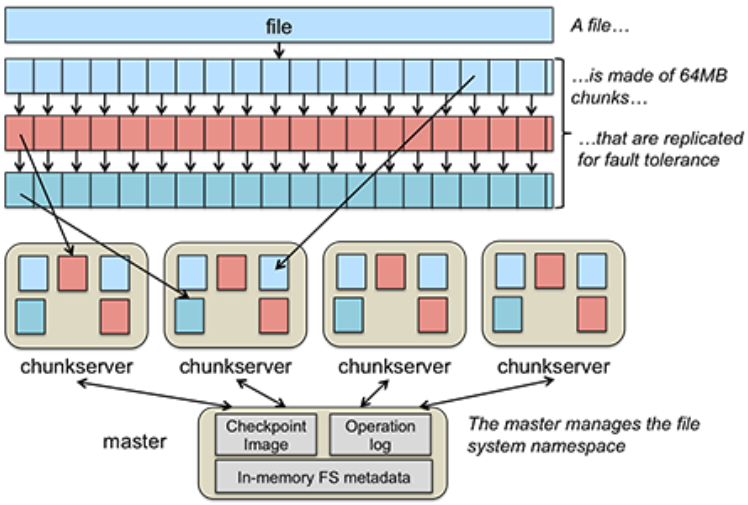
Summary:

- Chunk servers
  - A large file is split into contiguous chunks (typically 64MB)



  - Each chunk is replicated (usually 2x or 3x) and stored on chuck server
  - Replication policy makes sures chunk servers are on two different racks.. in case of rack failures.

- Master node
  - Stores metadata about where files are stored



Once Google's GFS paper was published, Yahoo wanted to utilize a similar file systems architecture for storing its large files. Hence, Yahoo started working on 'Hadoop Distributed File System (HDFS)' and made the project open source. HDFS is basically modeled after GFS with a few variations on the design, replication policy, etc. Here a mapping of the terminology between GFS and HDFS:

- Chunk servers are called 'Data Node' in HDFS
- Maser node is called the 'Name Node' in HDFS

1 Link-Analysis via MapReduce (following week's discussion)

Early search engines crawled the Web and extracted terms to build an inverted index, which maps a term to all relevant Web pages. When a search query is issued, the query is decomposed into search terms, and the pages with those terms are look-up from the inverted index and ranked based on importance (using TF-IDF, or cosine similarity function, etc.)

In 1996, it became content similarity (term-frequency) alone was no longer sufficient because of several factors:

- Number of pages grew rapidly in the mid-late 1990s
- A search query can result in 10M relevant pages, how to rank results suitably?
- Content similarity can be easily spammed by repeating term and add many relevant terms to boost ranking of the page, where in-fact the page may not be related to the search query.

During 1997-1998 two most influential link-based page ranking approaches were proposed. Both proposed algorithms exploit the link-structure of the Web to rank pages according to authority or importance:

1. **HITS**: Jon Kleinberg (Cornel University), at Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, January 1998.
2. **PageRank**: Sergey Brin and Larry Page, PhD students from Stanford University, at Seventh International World Wide Web Conference (WWW7) in April, 1998.

Google realized that with the ever growing inter-connectivity of the web, it can be modeled as a directed graph. A link from on page to another is a directed edge in the graph. With this model, we can make the assumption that nodes with higher number of incoming links, are more likely to be important. Thus, Google decided to rank the pages by the link structure. Page Rank (PR) was used to simulate where a Web surfers, starting at a random pages, could congregate by following links from one page to another. PR is essentially a vote by all other pages on the Web about the important another page.

- A page is more important if its has more in-coming links (those are the ones the page cannot control).
- Are all incoming links equal? Nah links from more important pages count more

1.1 Page Rank Formulation

Suppose that a page P_j has L_j links. If one of those links is to page P_i, then P_j will pass on $\frac{1}{L_j}$ of its importance to P_i.

The importance ranking of P_i is then the sum of all the contributions made by the pages linking to it. That is, if we denote the set of pages linking to P_i by B_i, then :

$$PR(P_i) = \sum_{P_j \in B_i} \frac{PR(P_j)}{L_j}$$

The PR is usually computed iteratively (vs algebraically). The transition matrix M (n by n matrix, where n is the number of pages) represents the probability distribution of the location of a random surfer step j. The vector V represents the principle eigenvector of M.

The probability x_i that a random surfer will be at node i at the next step:

$$x_i = \sum M_{ij} V_j$$

V_j is the probability that the surfer was at node j at the previous step

M_{ij} is the probability that a surfer at node j will move to node i at the next step.

If we surf any of the n pages of the Web with equal probability

1. The initial vector v_0 will have $\frac{1}{n}$ for each component
2. After one step, the distribution of the surfer will be Mv_0
3. After two steps, $M(Mv_0) = M_2v_0$ and so on
4. Multiplying the initial vector v_0 by M a total of i times gives the distribution of the surfer after the i^{th} steps

The distribution of the surfer approaches a limiting distribution v that satisfies v = Mv provided two conditions are met:

- The graph is strongly connected : It is possible to get from any node to any other node
- There are no dead ends : Dead ends = nodes that have no outgoing links

To avoid dealing with dead ends and non-strongly connected graphs, we modify the calculation of PageRank to allow each random surfer a small probability of teleporting to a random page rather than following an out-link from their current page. The iterative step, where we compute a new vector estimate of PageRanks v' from the current PageRank estimate v and the transition matrix M is

$$v' = \beta Mv + \frac{e(1\beta)}{n}$$

Where,

β is a chosen constant (usually in the range 0.8 to 0.85)

e is a vector with 1s for the appropriate number of components

n is the number of nodes in the Web graph

The term βMv represents the case where the random surfer decides to follow an out-link from their present page (with probability β). The term $(1-\beta)/n$ is a vector representing the introduction, with probability 1-, of a new random surfer at a random page.

2 Review Questions

1. The Hadoop Distributed File System (HDFS) stores the chunks (or blocks) of a large file across the DataNodes in the cluster. Which of the following is true? **(select all that apply)**
   (a) HDFS's replication policy is to store two copies on one rack, and a third copy on a remote rack.
   (b) HDFS is a master-slave architecture composed of a NameNode and DataNodes.
   (c) HDFS stores large files as blocks in a distributed manner.
   (d) HDFS is a peer-to-peer architecture where is there is no master node, only DataNodes.
   ANSWER: A, B, C

2. The scheduler works with HDFS and MapReduce to schedule jobs such that it exploits data locality. What did we mean by data locality? Consider how the jobs are assigned to the machines.
   ANSWER: it means it schedules the job so that the processing is done on the machine that has the data (bring computation to the data).

3. How is the number of Reducers determined?
   (a) Use cluster configuration or value set by the user during Job creation
   (b) Is determined based on the number of key-value pairs generated by the Mapper
   (c) Same way the number of Mappers are determined, based on the size of the input file.
   (d) Randomly generated number

Answer: A

4. (MR) Describe the flow of a MapReduce job. What does the Mapper receive as input, what does it generate? What happens in-between the Map and Reduce?

5. (MR) In a MapReduce framework, besides a Mapper and Reducer we can also define a Combiner and Partitioner. Provide a description of the Combiner and Partitioner, and their use cases.

6. (MR) Explain how the map-reduce framework can be used to join two tables **Student(studentId, name, adviserId)** and **Adviser(adviserId, adviserName)**. The two tables Student and Adviser can be joined on the key 'adviserId'. You can assume that you have two different files, one containing the 'records' of the table Student as a CSV file and another contain the records of the table 'Adviser' in the same format. Basically, describe the actions performed in the Mapper, what is being outputted by the Mapper. The actions performed by the Reducer, and the input and output of the Reducer.

   Answer: Basically, MapReduce receives a line of input of either the student record or the advisor record. The Map function must parse that line (string) and determine if its a record from student or advisor. if its a line/record from student then it output a key value pair of (advisorId, [studentId,name]) where the advisorId is the key and the other items are the value of the output. If its a line/record from adviser, then it outputs (adviserId, advisorName). The reducer will then receive tuples with the same key and be able to perform the join.

```
map(key , value )
   record = str(value)
   if split(record, ",") is of length =3 then
      // student record
      output (adviserId, (studentId,name))
   else:
      output (adviserId, adviserName)

reduce( key, values )
    for each v in values:
       perform join between pairs of advisor tuples and student tuples
       output resulting join
```

(PageRank) Answer the following questions about Web link-analysis (Page Rank). Consider that you are calculating PageRank values for web pages. There are 10 Billion web pages and you have created a 10 Billion x 10 Billion transition matrix M. As a part of iterative computations, you use the MapReduce computing framework without Taxation. The k^{th} iteration of the MapReduce job will create a vector v^k with 10 Billion items. The j^{th} item in v^k is calculated using the following formula:

$$v_j^{(k+1)} = \sum_j m_{ij} v_j^k$$

7. What does the values m_{ij} in the transition matrix M represent?

   (A) The total number of times that web page i has been visited

   (B) The page i's page rank value after j^{th} iteration

   (C) Random number generated by server

(D) The probability that page i is to be visited from the j^{th} page

8. What are the values v_j for the k^{th} iteration?

(A) The average PageRank value of the page j after the k^{th} step

(B) The probability that the surfer was at the node j at the $(k-1)^{th}$ step

(C) The highest PageRank value of the page j after the k^{th} step

(D) The lowest PageRank value of the page j after the k^{th} step