UNIT IV FRAMEWORKS

MapReduce – Hadoop, Hive, MapR – Sharding – NoSQL Databases - S3 - Hadoop Distributed File Systems – Case Study- Preventing Private Information Inference Attacks on Social Networks-Grand Challenge: Applying Regulatory Science and Big Data to Improve Medical Device Innovation

---

MAPREDUCE

MapReduce is a functional programming paradigm that is well suited to handling parallel processing of huge data sets distributed across a large number of computers. MapReduce is the application paradigm supported by Hadoop and the infrastructure

MapReduce works in two steps:

1. **Map**: The map step essentially solves a small problem: Hadoop's partitioner divides the problem into small workable subsets and assigns those to map processes to solve.
2. **Reduce**: The reducer combines the results of the mapping processes and forms the output of the MapReduce operation.

Maps specific keys to specific values. For example, if we were to count the number of times each word appears in a book, our MapReduce application would output each word as a key and the value as the number of times it is seen. Or more specifically, the book would probably be broken up into sentences or paragraphs, and the Map step would return each word mapped either to the number of times it appears in the sentence (or to "1" for each occurrence of every word) and then the reducer would combine the keys by adding their values together. Prior to submitting the job to Hadoop, we have to first load your data into Hadoop. It would then distribute the data, in blocks, to the various slave nodes in its cluster. Then we have to submit the job to Hadoop, it would distribute the code to the slave nodes and have each map and reduce task process data on that slave node. The map task would iterate over every word in the data block passed to it (assuming a sentence in this example), and output the word as the key and the value as "1". The reduced task would then receive all instances of values mapped to a particular key; for example, it may have 1,000 values of "1" mapped to the word "apple", which would mean that there are 1,000 apples in the text. The reduce task sums up all of the values and outputs that as its result. Then the Hadoop job would be set up to handle all of the output from the various reduced tasks.

An example of MapReduce

Example 1:

Assume you have five files, and each file contains two columns (a key and a value in Hadoop terms) that represent a city and the corresponding temperature recorded in that

city for the various measurement days.In this example, city is the key and temperature is the value.

Toronto, 20
Whitby, 25
New York, 22
Rome, 32
Toronto, 4
Rome, 33
New York, 18

Out of all the data we have collected, we want to find the maximum temperature for each city across all of the data files (note that each file might have the same city represented multiple times). Using the MapReduce framework, we can break this down into five map tasks, where each mapper works on one of the five files and the mapper task goes through the data and returns the maximum temperature for each city. For example, the results produced from one mapper task for the data above would look like this: (Toronto, 20) (Whitby, 25) (New York, 22) (Rome, 33)

Let's assume the other four mapper tasks (working on the other four files not shown here)

produced the following intermediate results: (Toronto, 18) (Whitby, 27) (New York, 32) (Rome, 37)(Toronto, 32) (Whitby, 20) (New York, 33) (Rome, 38) (Toronto, 22) (Whitby, 19) (New York, 20) (Rome, 31)(Toronto, 31) (Whitby, 22) (New York, 19) (Rome, 30). All five of these output streams would be fed into the reduce tasks, which combine the input results and output a single value for each city, producing a final result set as follows:

(Toronto, 32) (Whitby, 27) (New York, 33) (Rome, 38)

Example 2:

The typical introductory program or 'Hello World' for Hadoop is a word count program. Word count programs or functions do a few things:

1. look at a file with words in it,
2. determine what words are contained in the file, and
3. count how many times each word shows up and potentially rank or sort the results.

For example, you could run a word count function on a 200 page book about software programming to see how many times the word "code" showed up and what other words were more or less common. A word count program like this is considered to be a simple program.

The word counting problem becomes more complex when we want it to run a word count function on 100,000 books, 100 million web pages, or many terabytes of data