Ahmad Afiq Bin Che Johari

Rachid Chelouah

Initiation to Research

1 Apr 2014

MapReduce:: Simplified Data Processing on Large Clusters Summary

Introduction

The authors, Jeffrey Dean and Sanjay Ghemawat are both employees at Google.
At Google, it's very common to analyse a large input data in order to perform computations.

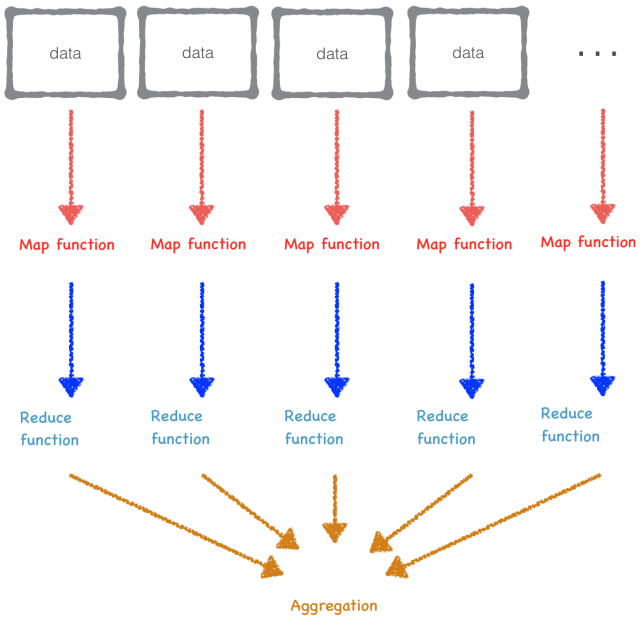
For example, to output the set of most frequent queries against a database on a given day, analysing the web request logs, graph structure of web documents, etc.
Since the input data is large, several computers or machines need to be used to distribute the computations so that we can get our answer faster.

Most of the time the computations are straightforward but the main challenge is with the parallelisation of computation, distributing the data and how to handle failures on the distributed machines.

Therefore, we seek for a programming model that allows the programmers to hide away the complexity of parallelisation on distributed machines while performing the computations.

Method

Below is the schematic representation of the authors MapReduce model.

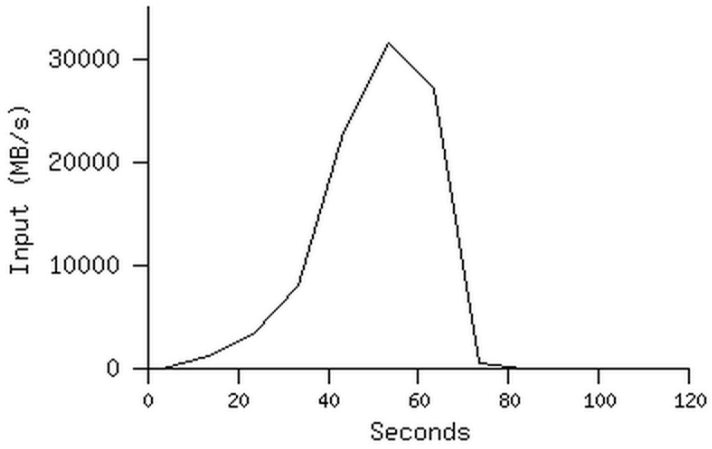


The data is split into a number of partitions and each partition of data will be passed through a map function independently of each other. The map function outputs an intermediary key-value results. And then, the reduce function will receive the intermediary key-value to output a smaller set of key-value, possibly one key-value. Finally, the results from the reduce function of each independent machines are aggregated to produce the final result.

Results

In order to provide a sense of benchmark to MapReduce, the authors measure the performance of its implementation on two computations. The first involves a searching for a pattern through a roughly one terabyte of data and the second, a sort on a roughly one terabyte of data. These two examples should be representative of using MapReduce against large datasets, that is the sort example to show a typical program that needs to shuffle a large data and the search example to represent a program that needs to extract information from a large data. The following is the result for the search implementation of MapReduce.

The Grep program(search)



The figure above is result of the distributed grep program implementing MapReduce model in the authors paper. It shows the progress of computing the search over time.

The input file is 10^10 100-byte records and the grep searches for a three-character pattern which is relatively rare(92,337 records out of 10^10). The input file is then split into 64MB pieces, that is 15,000 * 64MB, and the resulting output is then stored in one file.

The rate gradually increases at the beginning as many machines are assigned during the Mapping phase. It peaked at 30GB/s when around 1764 workers are assigned. As the Mapping phase terminates, the rate decreases quite dramatically

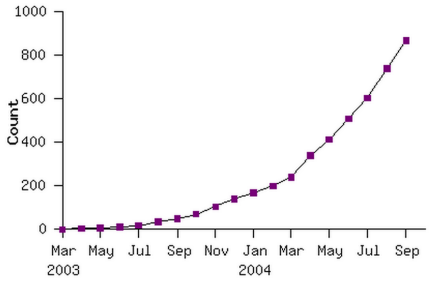
Perspectives

In fact, the first version of the MapReduce library was implemented in February of 2003 and enhancements were being made later in August of 2003 dealing mainly with the more technical parallelisation problems like locality optimisation, dynamic load balancing of task execution across worker machiens, etc.

The authors were also surprised by how broadly the MapReduce paradigm could be implemented at Google. Notable examples are:

- large-scale machine learning problem
- clustering problems for the Google News and Froogle products.
- large-scale graph computations.

MapReduce Programs In Google Source Tree



Example uses:

distributed grep	distributed sort	web link-graph reversal
term-vector per host	web access log stats	inverted index construction
document clustering	machine learning	statistical machine translation

There has been a significant growth of the number of different instances of MapReduce programs since its introduction at Google.

The main factor that contributes to its success is that a program that could be written in MapReduce model is simple and highly scalable, meaning it can be executed on different machines while hiding away the complexity of parallelisation. Therefore, a programmer who has no experience with distributed or parallel systems could easily exploit the re sources of the distributed machines.

Below is some interesting statistics related to MapReduce instances at provided by the authors from Google's code management system.

Usage: MapReduce jobs run in August 2004

Number of jobs	29,423	
Average job completion time	634	secs
Machine days used	79,186	days
Input data read	3,288	TB
Intermediate data produced	758	TB
Output data written	193	TB
Average worker machines per job	157	
Average worker deaths per job	1.2	
Average map tasks per job	3,351	
Average reduce tasks per job	55	
Unique *map* implementations	395	
Unique *reduce* implementations	269	
Unique *map/reduce* combinations	426	

Conclusion

The MapReduce programming model has been widely accepted at Google for many different purposes.

The proposed model hides away the parallelisation details allowing programmers to have a higher abstraction on a given problem. So, we focus more on the problem and not on the parallelisation details.

Though not all, but many problems are expressible in MapReduce computations.

Bibliography

The oldest reference provided by the authors dates back to 1989 but most of the references were from the late nineties i.e 1997, 1996, and early twenties i.e 2001, 2003. Most of the references are related to parallel computation.