MapReduce: Simplified Data Processing on Large Clusters

Robert Hoff

November 15, 2011

1 Summary

MapReduce is a programming model and implementation used at Google for processing large dataset that are used for their search engine and other applications. MapReduce uses principles of functional programming to enable problems to be distributed between many machines, in Google's case potentially thousands. A relatively simple interface is separated from the implementation, so clients can use MapReduce without technical knowledge of parallel computing. The interface is restrictive though and only lends itself to a limited domain, but is still able to address a range of interesting problems. In performance the system is able to compute large volume of data at speeds that are fast enough for Google's processing requirements.

2 Problem

Google requires the processing of very large datasets (e.g. $>$ 1TB) for several of their applications. Consider for example their search-engine; its performance relies on a mapping between words and the frequency of their occurrence for over a billion websites. Since the web is continuously evolving and changing the calculation needs to be performed on a regular basis, but processing a dataset like this would take way too long for a single computer. MapReduce is a method splitting a problem-set into manageable chunks that can be divided between thousands of computers. It uses principles of functional programming, because in this language paradigm functions don't have side-effects and computational loads can safely be divided. The words *map* and *reduce* are primitives taken from Lisp, and their meaning in this paper are the same.

3 Implementation

Processing a dataset with MapReduce involves two functional steps: *map* and *reduce*, both steps are suitable in a distributed context. *map* is a function that takes a <key,value> list as the input and outputs an intermediate <key, value> list. The intermediate output is generally not the same type, but it is an n to n mapping. The *reduce* step performs an iteration over all of the intermediate output and produces either a single value or a list; it's an n to 1 mapping. With this basic interface a number of useful problems can be solved, such as search, word-count, sorting, and other application specific conditioning of the data (such as the word-frequency computation). The interface enables users to write simple scripts to process the data without having to know about the technicalities of the parallelism, they are hidden in the MapReduce library. The hardware and run-time implementation consists of a single master that distributes the workload in a network of worker machines. The system is fault tolerant because the master checks if worker machines are responsive, if one worker fails a work-chunk can be easily reassigned. The chance of the master failing is small because it is only one machine, but if it does fail, the MapReduce task would have to be assigned to a different system.

4 Evaluation

Performance is measured on a network of 1800 commodity machines. In a test using *grep*, we see that the system can scan large data at a rate up to 30 GB/s. In another example, a sort task on about 1TB of data, the system is able to take care of the problem in around 600 seconds. Performance is resistant to killing off a large proportion of worker machines into the computation, and we see that performance may be slightly enhanced by utilising an optimisation.

5 My Opinion

MapReduce is a restricted programming model for enabling parallel computing. This is an active research area, but the contribution of the paper seems to be in isolation. The MapReduce is rather an implementation of well-known concepts, more than advancing any specific knowledge of parallel computing. Just looking at the results in the paper leaves a lot of unexplored territory. The performance of the system was tested on large 'Google style' systems. But perhaps it would have been more useful to use smaller systems, and to compare performance with existing projects. It may have been interesting to explore how the performance would scale with number of computers for a given computation. Overall the paper is Google-

centric, it solves Google's problems well enough. A priority wasn't to make it 100% optimal or prove that it is the best. On the positive, they do present a piece of engineering that solves big problems at impressive speeds. The description of the system is also highly replicable.

6 Questions

1. There are many distributed computer projects, how do these techniques compare to MapReduce? Why wasn't it possible to use any existing techniques for Google's purposes?

2. In terms of application, MapReduce seems specific and may wonder what limitations for example the <word, frequency> mappings have on quality of search results. Is the data produced my MapReduce a sustainable way of organising data? Do we need to come up with something better in the future? Will enhancing MapReduce be good enough, or do we need new systems?