Brief bio and research interests

I obtained my B.Sc. in Computer Science from the Swiss Federal Institute of Technology in Lausanne (EPFL) in July 2009, and I am now a first year Master’s student participating in the UC Berkeley/EPFL graduate exchange program for the 2009/2010 academic year. I am interested in program analysis and verification and programming languages. By taking this class, I would like to acquire a deep knowledge of parallel computing paradigms and the challenges involved.

Application problem: Processing and generating large data sets using MapReduce

Introduction

MapReduce is a programming model which provides a very simplified way of writing parallel applications. Created and initially implemented in Google, and presented in the work by Dean and Ghemawat [2], this restrictive model is impressive in that it hides from the user the complexity of distributing and parallelizing work over multiple computers, with fault-tolerance and load balancing.

MapReduce applications are written in a functional style, with the programmer specifying essentially two functions:

- A map function that takes an input (key, value) pair and produces a set of intermediate (key, value) pairs.
- A reduce function which, when applied to all intermediate (key, value) pairs which have the same key, combines the corresponding values in a desired manner, and producing a –possibly smaller– set of values as output. This set has typically cardinality 0 or 1.

While the MapReduce interface can be implemented on a number of platforms such as a small memory-shared machine or a large NUMA processor [2], the initial targeted platform is a large cluster of commodity PCs connected using switched Ethernet, a computing environment extensively used in Google [1]. This initial implementation was coded in the C++ programming language.

Overview

The execution of MapReduce can be described as following:

- First, the input, which is a set of (key, value) pairs is partitioned into $M$ pieces called splits. These pieces are typically of size 16 megabytes to 64 megabytes, and can be processed in parallel by different worker machines. The user has to specify the number of partitions $R$ for the intermediate key space, along with a partitioning function, such as a hashing function. This number corresponds to the number of reduce tasks that will be assigned.

- Multiple copies of the program are launched in the cluster, with one being special: the master. The master assigns the $M$ map tasks and $R$ reduce tasks to idle workers.

- A worker who is assigned a map task processes the input in the split corresponding to the task, and stores the intermediate results in its stable storage partitioned into $R$ regions, using the
partitioning function mentioned above. It then passes the location of these results to the master, which is responsible for notifying a worker assigned to a reduce task of this location. The reduce worker uses remote procedure calls to retrieve this data into its own storage. It then sorts all intermediate data it received by key, to apply the reduce function to all pairs sharing the same key. The output is appended to a file corresponding to this reduce task.

Applications

Many uses of MapReduce involve processing large amounts of raw data such as documents crawled from the web and logs, to produce various kinds of derived data. Some examples of such problems are distributed grep, counting URL access frequencies, producing reverse web-link graphs, computing inverted indices to map words to documents in which they appear, and distributed sort.

The performance of the distributed sort implementation in MapReduce, which takes about 50 lines of code, is remarkable. On a cluster of approximately 1800 machines each with two 2GHz Intel Xeon processors, 4GB of memory, two 160GB IDE disks and a gigabit Ethernet link, arranged in a two-level tree-shaped switched network, the TetraSort benchmark finishes in about 850 seconds, the best reported result being 1057 seconds at the time of evaluation.

The applications of MapReduce are far from being limited to the ones mentioned above. For instance, Hadoop, which is an open-source implementation of the framework [3], has been used to show promising results in parallelizing bioinformatics applications, such as BLAST (Basic Local Alignment and Search Tool) and GSEA (Gene Set Enrichment Analysis) [4].

Conclusion

The MapReduce framework is a powerful interface to parallel programming which has been successfully used to parallelize a variety of computation tasks since its creation. The challenges rely in describing a task in this restricted functional programming model, but this restrictiveness abstracts away some of the complexities involved in writing parallel applications and provides a very simple tool for achieving parallelism on a variety of platforms.

References


