R202 Survey Report: Large-Scale Parallel Data Processing

Robert Hoff

February 21, 2012

1 Introduction

Computationally intensive tasks on large data are not always feasible as single-processor problems because of extended latencies that are involved. In particularly, during the early 2000s, web-data and search indexing introduced the need, on a regular basis, to perform processing on the global collection of webpages. Google most prominently tackled this problem, and having unique and pragmatic processing needs that was previously not encountered, innovated the software framework MapReduce[3]. It, and its open-source equivalent Hadoop, is the most widely used large-scale data processing frameworks in use today, they are key instruments to exploring large datasets originating from the Internet. Later more general purpose frameworks have been described and implemented, such as Dryad and CIEL that provide a richer programming model. The emphasis on these frameworks are on processing large data-inputs by dividing the computational requirements between independent machines (organised in a cluster). This sets these technologies slightly apart from general-purpose parallel computing. In terms of general parallel computing, apart from covering a wider range of implementation techniques such as those considering intra-machine architectures, the emphasis is on compiling general-purpose code, and doesn't make assumptions on the size of the input. Therefore there is a gap between large-scale execution engines and general parallel computing. The most important difference is the restrictive programming model that is provided by these frameworks, which aids implementation. A key research topic is to make the models as permissive as possible, if they are fully relaxed the problem reduces to one of general purpose compilation.

2 Applications

It is useful to consider MapReduce in the context of distributed execution engines because it has many general characteristics. These include the division of a high-level and restrictive programming interface, which separates the client-problem from the implementation. The implementation includes breaking up the problem and dividing it between many machines, from a client-perspective is often given the illusion of working with a single machine. Different restrictions in the programming interfaces make the parallelising implementation manageable, but limits the potential problem sets. Later generation execution engines provide richer models, in slightly different capacities, these frameworks include Dryad[8] and CIEL[2], there is still work to be done between these pragmatic software frameworks and the fully parallelised automation of a general purpose programming language.

At the time MapReduce was invented in 2004 Google was already the leading search engine. Google's success is attributed to their innovating processing techniques, including the page-rank mechanism[7]. Providing up-to date results necessitated regular processing of the global pool of websites, with slightly different processing requirements depending on the task. One of the common characteristics between tasks were that the data lent itself to parallelising, it could be split up, each processed data-chunk not exhibiting side-effects on others. A large class of problems, such as word indexes and frequency of incoming links, could be expressed by two basic constructs analogous to the constructs in functional programming called map and reduce, hence the name MapReduce.

Although MapReduce is just based on two functions, it has been employed to solve a significant number of large-scale data processing problems. Any problem that involves scanning data (without side-effects) and then concatenating this data is dealt with by MapReduce. This model lends itself well to many statistical tasks, such as searching for string frequencies, or edge frequencies of large graphs. The web itself is an example of a large graph, more recently large social graphs such as Facebook have been analysed for a variety of purposes. For example for determining properties such as connectedness of a node beyond just the closest neighbour.

Dividing any kind of problem between many machines or processing threads has been a long-standing problem in computer science. For example [5]. But Parallel computing has not had as much focus as one might expect, mainly because of rapid advances in single machine processing capability would in many cases have grown at a comparable rate to computational requirements, and not necessitated advances in parallel computing. With the proliferation of the World Wide Web, its growth and corresponding data generation, together with continuous processing requirements outpaced the processing capabilities of any single machine (as computational power has grown approximately by Moore's law). Before the Internet, the aggregation of data was more limited, for example within large cooperations, and time-constraints in general were more relaxed.

The CIEL and Dryad frameworks provide a richer programming model. In the case of Dryad the problem can broken into a graph representation where the nodes represent data, and the edges represent side-effect free processing steps. This is a useful way of thinking about a problem and lends itself well to datasets that have several processing steps. Applications include simulations comparing many independent scenarios, such as climate models, or event simulation and reconstruction in particle physics [4]. Both the CIEL and Dryad provide their APIs by a scripting language, embedded into traditional high-level programming languages.

3 Characteristics of Implementation

Google's MapReduce system uses a single Master, which therefore also is a single point of failure, but otherwise the fault-tolerance is good. Google innovated the use of commodity hardware to run their large enterprise systems. Cheap imported commodity hardware, compared to higher-end products from US domestic companies like IBM, have higher cost-effectiveness with respect to their performance, but they also have a higher failure rate. The high rate of failure can be offset by acknowledging it as the normative behaviour and integrating the fault-tolerance as fundamental to the implementation.

Heterogeneiety, and variations in computation speed, should also be built in. There may be different processing speeds, and some CPUs may be overloaded or lagging due to random behaviour. This imposes the additional requirement that the system should provide good average performance across variations in underlying hardware

Optimisations, in general, may take place with respect to reducing processing requirements, reducing network bandwidth, determining better load-balancing or improving fault-tolerant behaviour. Dryad and CIEL implementations also have single-master architectures, but the master may be replaced within an execution job should it fail. It is important that such a protection is invisible to the client. This fault tolerance becomes important if a job extends for a long duration, or in other cases we may be dealing with continuous queries that execute indefinitely.

4 Future Research

Research on distributed execution engines lie very close to general parallising problems. Phoenix is an MapReduce adaptation to multi-core computers.[1] It was showed that the system lead to scalable performance for both multi-core chips and conventional symmetric multiprocessors.

Since CPU frequency scaling has stalled, predictions suggests that personal computers may soon have thousands of cores. [6] Given this scenario the methods used in any distributed execution engine that was applicable to a data-cluster, are shown in the Phoenix implementation, are directly transferable as an integral part of the operating system of a many-core machine.

There are differences of the average sizes of the datasets that a PC has to contend with than that of a specialised data-cluster. One type of application is computer graphics ray tracing each pixel may be rendered independently which is today solved via GPUs. The principle burden for the host processor at this point centers on modelling the physical properties of the graphical elements that comprise the game or the user interface. Realistic physics requires computational modelling of physical processes that are essentially the same as those required for scientific computing applications.

The processing models that we use for large-scale data demonstrate an early need to more generalised parallisation, and came ahead of the increased focus on single-machine parallel computing. Much of the research so far has been application-orientated, rather than producing general models. This pragmatic approach, which is a symptom of industry led initiatives, has forwarded our understanding into specific domains of computing. The later models, with more generality solves a wider breath of theoretical problems, and reduces the gap between compilers that fully automate the parallisation of general purpose programs.

References

[1] Ranger C., R. Raghuraman, A. Penmetsa, and G. Bradski. Evaluating mapreduce for multi-core and multiprocessor systems. *IEEE 13th International Symposium on High Performance Computer Architecture*, 2007.

[2] Murray D.G., Schwarzkopf M., and et al. Ciel: a univeral execution engine for distributed data-flow computing. *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, 2011.

[3] Dean J. and Ghemawat S. Mapreduce: Simplified data processing on large clusters. *OSDI*, 2004.

[4] Asanovic K. Landscape of parallel computing research. *Technical Report*, 2006.

[5] Valiant L.G. A bridging model for parallel computation. *Communications of the ACM*, 33, Aug 1990.

[6] Borkar S. Thousand core chipsa technology perspective. *Proceedings of the 44th annual Design Automation Conference*, 2007.

[7] Brin S. and Page L. The anatomy of a large-scale hypertextual web search engine. *Proceedings of the Seventh International World Wide Web Conference*, 1998.

[8] Yu Y., Isard M., Fetterly D., and et al. Dryadlinq: A system for general-purpose distributed data-parallel computing using a high-level language. *OSDI'08*, 2008.