Paper Review Form (1000 words maximum) *cs940*: MapReduce: Simplified Data Processing on Large Clusters [1]

February 19, 2018

Paper Summary

This paper reports the design and implementation of Google's MapReduce programming model, which was one of the first generalised task models facilitating highly-parallelised batch data processing. Abstracting common data processing tasks into combinations of *map* and *reduce* operations – along with additional functionalities and optimisations – the implemented computation system allowed programmers to trivially create parallelised tasks scaling for a few thousand machines. The performance of the system was demonstrated through representative live benchmarks. Extensive redundancies and fault handling features built into the implementation were also tested during evaluations.

Pros and Cons

I believe that the paper has the following positive features:

- The relatively simple *map* and *reduce* model bears such great descriptive power that some operations only require half of the model to implement, such as distributed grep.
- The use of backup executions close to the completion of an operation substantially reduces expected execution time through preemptively addressing straggling distributed tasks, despite adding to the overall workload. This resolved a long-standing difficulty in distributed computing.
- The benchmarks performed adequately demonstrated the effects of additional optimisations such as data locality, without having to perform specific and complex operations.

I believe that the paper has the following negative features:

- Despite providing ordering guarantee within a partition, rows of input data must still be uncorrelated overall, as any correlation information is not guaranteed to be preserved during partitioning.

- Unlike the successor Borg [2], MapReduce implementation's master processes are not redundant, and in practice require workers to fully abort on encountering master failures. The assumption that single masters are unlikely to fail became untenable in Borg, and led to the introduction of consensus-backed replications.
- During evaluations on fault-tolerance, only machine task processes were killed, with no hardware failures simulated – which in fact are a common occurrence in large scale deployments.

The Problem/Motivation

Even in 2004, commercial operations often required data processing at a large scale, which could only be achieved through extensive parallelisation of tasks. Implementing explicit parallelism in individual programs would involve large amounts of repetitive and non-generalisable work, therefore a common parallel programming model or framework was required to abstract away low level details. Existing solutions included parallel prefix computations [3] which lacked flexibility and fault-tolerance at scale; Bulk Synchronous Programming [4] which lacked sufficient parallelisation automation due to a less restricted model; and eager scheduling systems like Charlotte [5] which lacked automated failure avoidance mechanisms. Therefore, MapReduce was built and customised to Google's needs.

The Solution/Approach

The design goal of MapReduce was to abstract away implementations of parallelisation, fault-tolerance, partitioning, and loading balancing, allowing the user to easily transform their existing programs into simple computation processes with inputs and outputs matching the key-value structures of *map* (performs intermediate pairing) and *reduce* (merges pairs). Input data can then be partitioned and executed by individual task instances on clustered machines, which are managed in a master-slave configuration.

The master performs bookkeeping and information passing for workers on individual machines, and is not redundant. Failures of workers on individual machines are handled through automatic rescheduling, which works comparatively better for operations with deterministic functions. Data locality is used to improve performance and reduce network congestion, along with fine task granularity and the use of backup tasks to improve scheduling performance.

Refinements such as partitioning functions, customisable types, local executions and human-readable real time information reduce the complexities of programming and debugging MapReduce tasks, improving usability. MapReduce was able to carry the vast majority of Google's data processing tasks at the time.

Evaluation

Unlike the offline evaluation for the successor system Borg [2], the authors were able to conduct live evaluations of a full-scale deployment of 1800 machines during system idle times on a weekend. Two types of operations were performed: *grep* of an uncommon pattern in large quantities of text, and sorting of large, ordered data. The authors claimed that the operations were representative of most tasks designed to be performed through MapReduce.

Both operations were completed by the cluster within very short amount of time, demonstrating limited parallel overhead. Effects of optimisations such as partitioning, data shuffling, and backup tasks were analysed and showed to have a positive impact on performance. Fault-tolerance was tested through injecting process failures on machines, showing minimal overhead; while effects of hardware failures were not tested. The implementation's success in commercial operation was also illustrated with quantitative figures.

Your Opinion

Many features of MapReduce were very innovative at the time of the publication, such as a common abstraction of data processing functionalities, automatic avoidance of bug-related faults in distributed computing, and preemptive straggler handling. High performance parallel computing achieved through these features allowed technology companies to process an ever greater amount of data, which has become one of the most valuable assets in this industry today. Fault-tolerance of master processes in distributed computing were of less concern then than they are now, which have been achieved through consensus systems and master-replication, such as in Yahoo's ZooKeeper [6] and Google's own Chubby [7]. With hardware improvements slowing down, how data processing can be further upscaled against demand in the future remains to be seen.

Questions for the Authors

- MapReduce no doubt laid many foundations for the design of Borg, which provides a more generalised distributed computing scheduling system. What lessons learned from implementing MapReduce went into the design of Borg?
- With distributed computing systems increasingly specialised for individual organisations' needs, it becomes harder and harder to find idle times to evaluate a system in full scale as conducted for MapReduce's implementation. What alternative methods do you think would be effective in evaluating distributed computing systems from a limited perspective?

References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 2015, p. 18.

[3] S. Gorlatch, "Systematic efficient parallelization of scan and other list homomorphisms," in *European Conference on Parallel Processing*. Springer, 1996, pp. 401–408.

[4] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[5] A. Baratloo, M. Karaul, Z. M. Kedem, and P. Wijckoff, "Charlotte: Metacomputing on the web," *Future Generation Computer Systems*, vol. 15, no. 5-6, pp. 559–570, 1999.

[6] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems." in *USENIX annual technical conference*, vol. 8, no. 9. Boston, MA, USA, 2010.

[7] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," in *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 2006, pp. 335–350.