Map-Reduce for Machine Learning on Multicore

This paper aims at adapting the MapReduce programming paradigm to solve complex machine learning (ML) problems with multicore/multiprocessor machines. The authors tried to express ML algorithms in a special form called summation form that is easily expressed in the MapReduce framework. They developed a multicore version of MapReduce and ran ten example algorithms on dual processors. The experiments showed nearly linear speedups, thus proved the possibility and promise to scale up machine learning, which is of potentially great significance in the big data time.

Merits

S1 The topic of the paper is well-worth discuss in a time when statistics & machine learning (ML) meet distributed and cloud computing. People already began to use clusters of machines to process large-scale datasets and MapReduce is one shining result of these efforts. Machine learning on MapReduce is challenging due to its complexity, but its previous success suggests the need to scale it up. This paper proposes a novel and promising approach to parallelize machine learning and is thought-provoking. The obvious future work is to deploy it on larger clusters. If successful, it would significantly benefit anyone who deals with data (IT companies, governments, banks, scientists, etc).

S2 The proposed solution is straightforward, general, and effective. It is based on an observation that many ML solutions could be expressed in a summation form (as an example, any matrix computation is likely to have an equivalent summation form). This summation form suits well into the MapReduce programming paradigm. It even solves some problems of MapReduce regarding iterative updates in the context of the paper since the reducer does not directly output into the file system, thus avoiding disk I/O's.

S3 The proposed solution is well-supported by effective experiment results. The authors show their results on ten different algorithms and show their improved performance on increasing the number of cores/processors. Though not perfectly linear, it shows us the promise and provokes discussions and future work.

Concerns

W1 Though the authors tried to include as many machine learning areas as they could, it is too hard to cover them all. The examples given in the paper share a common property that the solutions are all easily expressed as matrix computation, thereby as a summation of vectors. However, there are also a wide range of ML tasks that are not in the matrix form, e.g. PageRank. They require iterative updates for each involved entity (e.g. web page). It would thus be beneficial to discuss the limitations of their methods, typical ML tasks they are good at, related workarounds, or possible future improvements.

W2 One thing I'm particularly interested in this paper is that it solves the iterative update problem with MapReduce in a multicore setting. It would be better if the authors can provide more insight into this. For example, the K-means algorithm fits poorly into the MapReduce paradigm since it involves iteration, but achieves a speed up of ten in this paper. My guess is that instead of writing to a file system, the reducer here just keeps all the data in memory for next use. This is an important issue that's worth a stand-alone paper and more optimization (e.g. resilient distributed datasets (RDD) from UC Berkeley), but the author mentions nothing about it.

W3 Another issue with this paper is also related to iterative updates. In Section 4.1, the complexity of iterative algorithms is analyzed for one iteration, but the cost between iterations might be large depending on system architecture (for example, in the experiment results, EM and K-means achieve a speed up of about 10, while others achieve up to 14). The authors should briefly explain this and reflect it in their theoretical complexity analysis.