Technical Perspective - Implicit Parallelism through Deep Language Embedding

Zachary G. Ives
University of Pennsylvania
zives@cis.upenn.edu

Modern "big data" analysis was motivated by the needs of the large Internet players, but it was enabled by two main technical developments: parallel data processing technologies that support reliable and scalable computation over *unreliable* shared-nothing clusters of computers, and continued advances in machine learning algorithms and techniques. Initial work on these two areas happened largely independently: MapReduce was developed for aggregate computations over large multitudes of records, with minimal control flow and no evident goal of supporting machine learning. Conversely, many of the advances in machine learning research targeted a single machine.

However, many subsequent developments in parallel data processing have indeed been motivated by machine learning tasks. Additionally, many machine learning algorithms have been ported to parallel data processing libraries, and certain machine learning techniques, such as deep learning, almost entirely owe their success to parallel processing techniques. Tremendous activity, in both the distributed systems and database communities, and in both industry and academia, continues in improving the runtime architectures and algorithms. Key points of focus have included more effective data partitioning and load balancing strategies, as well as mechanisms for making reliable computation more efficient, such as checkpointing and recomputation schemes at different levels of granularity. Such work has led to many parallel data processing platforms [1, 2, 3].

Systems-level issues are not the only concern. The database community continues to search for better programming abstractions and interfaces. Efficient parallel dataflow programs adopt control and dataflow patterns where processing is done independently on different compute nodes, then messages are exchanged, and the task iterates until some condition is reached. Parallel data processing researchers have extensively investigated how to express distributed data and control flow. We now understand much more about the trade-offs between iterative computation with synchronization (where all nodes are in the same interation step) versus fully asynchronous execution (which can be more efficient but makes it harder to port algorithms developed in a centralized setting); how to abstract working state within the program; and whether the actual dataflow across nodes should be modeled as values, deltas, or generic messages.

A much less-studied, but at least equally important, issue revolves around the fact that programmers want to write code in familiar languages and take advantage of existing libraries, so there needs to be a strong coupling between the parallel data processing runtime system and an existing *host language* like Java, Scala, or Python. Ideally, the runtime platform could still predict and optimize for the workload. Most existing approaches to host language integration use variations of Microsoft's Language-Integrated Querying (LINQ), which incorporates some SQL constructs into the host language; or user-defined `map` and `reduce` functions, applied to collections of objects, along with second-order functions like `join` and `cogroup`. However, increasingly there are applications where the data or the computation has some innate structure requiring navigation and assembly of results, where such abstractions do little to help.

The "Implicit Parallelism through Deep Language Embedding" paper makes an astute observation that parallel data processing is the first setting that couples structured navigation operations with host programming languages: in the 1990s, the fields of object-oriented databases and persistent programming languages focused on these concerns. While object-oriented databases perhaps never found their "killer application," many innovative ideas were developed.

Alexandrov and collaborators propose Emma, an extension to Scala for parallel data processing, which operates on bags of structured objects. Emma leverages *monad comprehensions*, first developed as primitive query operations for object-oriented databases, to traverse and assemble these structured objects. These monad comprehensions form a very natural and powerful mechanism for declaratively specifying structuring and navigation operations, as illustrated in this paper. The authors also highlight some of the sophisticated query optimization techniques that can be applied, and further techniques and an evaluation appear in the full conference version of the paper. This paper does a great job of taking ideas that may have been "ahead of their time," and using them to cleanly tackle some of the open problems in the parallel data processing space.

1. REFERENCES

[1] R. Chaiken, B. Jenkins, P.-Å. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: easy and efficient parallel processing of massive data sets. *Proc VLDB*, 1(2):1265–1276, 2008.

[2] V. Markl. Breaking the chains: On declarative data analysis and data independence in the big data era. *Proc VLDB*, 7(13):1730–1733, 2014.

[3] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. *HotCloud*, 2010.