Technical Perspective: From Think Parallel to Think Sequential

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

In recent years, the database and distributed systems communities have built a wide variety of runtime systems and programming models for large-scale computing over graphs. Such "big graph processing systems" [1, 2, 4, 5, 7] o support highly scalable parallel execution of graph algorithms — e.g., computing shortest paths, graph centrality, connected components, or perhaps even graph clusters. As described in the excellent survey by Yan et al [6], most big graph processing systems require the programmer to adopt a *vertex-centric* or *block-centric* programming model. For the former, code only "sees" the state at one vertex, receives messages from other vertices, and can send messages to other vertices. Under the latter, code manages a set of vertices within a subgraph ("block") and can communicate with the code managing other blocks.

In "From think Parallel to Think Sequential," Fan and colleagues argue that vertex- and block-centric programming models are not natural for programmers trained to think sequentially. Instead, they argue that a more intuitive programming model can be developed out of several very simple primitives that can be composed to do incremental computation (as has also been studied in more general "big data" systems [4, 3]). The authors propose four elegant building blocks: (1) a *partial evaluation* function, (2) an *incremental update* handling function, (3) mechanisms for updating and sharing parameters in global fashion, and (4) an *aggregate function* for when multiple workers are updating the same parameter. They build the GRAPE GRAPh Engine system, which implements this programming model, and they show that it provides excellent performance for a variety of graph algorithms.

The paper presents a compelling case that, at least for certain classes of algorithms, the simple primitives may be both more natural and more amenable to optimization than standard vertex-centric approaches.

1. REFERENCES

[1] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. Graphlab: A distributed framework for machine learning in the cloud. *CoRR*, abs/1107.0922, 2011.

[2] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD*, pages 135–146, 2010.

[3] Svilen Mihaylov, Zachary G. Ives, and Sudipto Guha. REX: Recursive, delta-based data-centric computation. In *PVLDB*, 2012.

[4] Derek Gordon Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. Naiad: a timely dataflow system. In *SOSP*, pages 439–455, 2013.

[5] Reynold S Xin, Joseph E Gonzalez, Michael J Franklin, and Ion Stoica. Graphx: A resilient distributed graph system on spark. In *First International Workshop on Graph Data Management Experiences and Systems*, page 2. ACM, 2013.

[6] Da Yan, Yingyi Bu, Yuanyuan Tian, and Amol Deshpande. Big graph analytics platforms. *Foundations and Trends® in Databases*, 7(1-2):1–195, 2017.

[7] Da Yan, James Cheng, Yi Lu, and Wilfred Ng. Blogel: A block-centric framework for distributed computation on real-world graphs. *Proceedings of the VLDB Endowment*, 7(14):1981–1992, 2014.