Pregel: A system for large-scale graph processing

Author(s): Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski

This paper differs from the previous texts we have read, which focus on individual algorithms optimized for parallel or distributed frameworks, in describing a computational model that can efficiently run various algorithms on distributed systems. The authors' motivation is Google's cluster architecture, where geographically interconnected racks of individual PCs have high intra-rack bandwidth but still need to be able to communicate to the other racks to perform large-scale computations on web graphs (sites, social networks etc.)

The authors address the limitations that, in 2010, existed around large graph processing. Users would have to choose between developing/adopting a custom infrastructure for the algorithm or graph topology of choice, or rely in an off-the shelf distributed computing platform unsuitable for graph processing, limiting themselves ot a single-computer graph algorithm library, or using a parallel graph system that was not fault tolerant. Their vision was to create an API that would be scalable, fault-tolerant (i.e. incorporated useful checkpointing), and could run any arbitrary graph algorithm.

A defining feature of the Pregel model is that vertices are the primary agents ("first-class citizens") in performing calculations, voting to terminate the algorithms, changing graph topology, and sending messages to other vertices. Indeed, this agency is so pronounced that the authors note that explicitly listed edges may not be necessary in cases where there is a register of all vertices and the partition of the graph they are responsible for. In addition to local operations performed by individual vertices, the authors introduce the concept of a "superstep", during which vertices can send messages (to be received in the next superstep), modify their state, or modify the topology of the graph. This message-based system was chosen over emulating a shared memory system because it is faster and also compatible with many graph algorithms.

The message-based system does, however, come with some issues. How does one indicate to the process that it should terminate? How to resolve message conflicts (e.g. creating a vertex with conflicting starting values), how to minimize message overhead, and how to keep global statistics up-to-date? The authors address these problems with a range of solutions that naturally emerge from the proposed architecture of Pregel. Terminating the process, for example, is accomplished by having vertices "vote" to terminate (e.g. if performing a max-value operation, once a vertex receives a message from a vertex with a value lower than its own, it votes to terminate. Once all vertices have voted to terminate, they must share the maximum value.) Additionally, Pregel uses constructs called combiners (joining multiple messages into one through some function) and aggregators (global variables that can be updated by vertices at each superstep) to optimize and manage coordination over the distributed system.

The paper is relatively concise but expressive. After laying out the overall framework for the Pregel API, the authors discuss its application to relevant graph algorithms (PageRank and shortest path are naturally important to Google's business). Some of the more interesting comments in this section address work efficiency (the wavefront-based approach to shortest path visits more vertices than sequential versions like Dijkstra, but is massively scaleable in a way that Dijkstra is not.) and the new approaches to standard algorithms made possible by Pregel (e.g. using votes for termination to coordinate bipartite matching.)

The greatest weakness of the paper is maybe the fact that its experiments demonstrate limited comparison to other solutions. Perhaps the argument is that the qualitative statements about the shortcomings of contemporary alternatives suffices to convince the reader, but it would have been nice to see the claim that MapReduce does not perform well on graph algorithms demonstrated as a contrast to Pregel, or even Parallel Boost Graph and CCMgraph, which are discussed in the following section.

Since this is an industry paper, I imagine most of the improvements building on Pregel will be internal to Google, which means that in addition to improving efficiency and building algorithms on top of the framework, the authors will be concerned about longevity of the system, documentation, and usability (e.g. the HTTP server that produces a user interface for the master.) The authors' note at the end is telling. They are no longer "at liberty ot change the API without considering compatibility", indicating that the process of developing this framework had to be thorough enough that an inability to update the underlying system would not be overly detrimental.