Hybrid Transaction and Analytics Processing Systems for Graph Structured Data

Puneet Mehrotra
University of British Columbia

The Problem: A lot of information is naturally expressible as graphs - from proteomics to business applications, large graphs are ubiquitous[1] and there is a great demand for scaling graph processing to trillion-edge datasets[2]. We have come to understand graph processing to require the use of specialized storage and processing engines that work best when distributed across many compute nodes to gain maximum parallelism. This is contrary to several works that obviate the need for specialized graph processing systems[3, 4] and challenge the notion that distributing the analytics tasks always performs better[5]. An unfortunate side effect of this design is the need to have costly preprocessing, partitioning, and ingestion steps that can quickly come to dominate the overall runtime of the system. The preprocessing step can be amortized if the data doesn't evolve, but the ingestion (and transformation to in-memory data structures) must be repeated for every run.

Creating a distributed graph processing system is predicated on an understanding that the parallelism needed to scale computation is not available on a single compute node; this is not quite true since there is a lot of untapped parallelism available on commodity servers with high core counts and ample amounts of RAM. Besides, the average graphs are not large enough to warrant distribution. Additional care must be taken to scale these systems to work for an evolving graph and under conditions where real-time analytics must be performed.

At first glance, databases appear to be the obvious solution to store and process graphs, especially considering that most data to be analyzed is often already present in some relational database. Databases such as Postgres and MySQL have mature storage backends that support high insertion rates and provide strong consistency guarantees, but they remain largely difficult and non-intuitive to use for graph processing. Graph Databases store data natively as a graph but are usually unable to scale to graphs with billions of nodes and suffer from poor ingestion rates. Besides, there are very few real-life use-cases that exclusively rely on graph queries, and therefore graph databases are usually used alongside more traditional databases.

Utopia: This tension between the graph data storage and processing must be addressed and reconciled. An ideal midpoint in the design space would be a hybrid transaction and analytics system that stores the graph-structured data in an on-disk layout that is close to the in-memory structure needed for the analytics tasks that follow. Data layouts can be customized to get maximum performance based on the statistical characteristics of the dataset, access patterns of the analytics algorithms and the read-write ratio, and hardware specifications. If the data is stored using an engine that delivers robust performance on reads while also supporting high insertion rates, we can use the same backend as the transactional store as well as run analytics workloads. Once we optimize for strong single node performance, we can scale out the computation to other nodes [1].

Getting there: Our goal is to use performance-driven design to produce a single-node graph storage and processing system that can be configured to efficiently handle variable workloads and from which a distributed graph processing system can be developed. We begin with selecting a high-performance key-value store as the underlying storage substrate. A key-value store is easier to experiment with than a rigid-schema RDBMS, and using a mature solution such as WiredTiger is likely to let us create a realistic production deployment of a graph-native database.

On top of this backend, we develop a graph API that implements a family of graph storage structures that work well for both transactional and analytics workloads. The API provides the basic primitives that are needed to construct graph queries and traversals and converts these operations into WiredTiger get and set operations. We will then conduct a multi-dimensional performance study that considers individually and together: underlying representation, benchmark, algorithmic implementation of the benchmark, handling of vertices with no edges, different vertex orderings, and partitioning schemes.

Should we find that our prototype is not sufficiently competitive with the state-of-the-art CSR implementations, we will migrate our work to a native mutable CSR representation[6] and incorporate it into the framework developed.

The main idea is to develop an analytic model that can guide the development of storage engines tailored for specific workloads. In the spirit of the Data Calculator[7], we can perform datastructure and workload modeling, which should allow us to develop heuristics for selecting an appropriate representation for data given the workload.

References

[1] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M. Tamer Özsu. The ubiquity of large graphs and surprising challenges of graph processing. *Proc. VLDB Endow.*, 11(4):420–431, December 2017.
[2] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. One trillion edges: Graph processing at facebook-scale. *Proceedings of the VLDB Endowment*, 8(12):1804–1815, 2015.
[3] Jing Fan, Adalbert Gerald Soosai Raj, and Jignesh M Patel. The case against specialized graph analytics engines. In *CIDR*, 2015.
[4] Adam Welc, Raghavan Raman, Zhe Wu, Sungpack Hong, Hassan Chafi, and Jay Banerjee. Graph analysis: do we have to reinvent the wheel? In *First International Workshop on Graph Data Management Experiences and Systems*, pages 1–6, 2013.
[5] Frank McSherry, Michael Isard, and Derek Gordon Murray. Scalability! but at what cost? In *HotOS*, volume 15, pages 14–14. Citeseer, 2015.
[6] P. Macko, V. J. Marathe, D. W. Margo, and M. I. Seltzer. LLAMA: Efficient graph analytics using Large Multiversioned Arrays. In *2015 IEEE 31st International Conference on Data Engineering*, 2015.
[7] Stratos Idreos, Kostas Zoumpatianos, Brian Hentschel, Michael S Kester, and Demi Guo. The data calculator: Data structure design and cost synthesis from first principles and learned cost models. In *Proceedings of the 2018 International Conference on Management of Data*, pages 535–550, 2018.

[1] "You can have a second computer once you've shown you know how to use the first one." – Paul Barham[5]