Trident: An Adaptive Low-Level Database for Very Large Graphs

Jacopo Urbani
Vrije Universiteit Amsterdam

Efficient query answering over very large graphs is a problem that attracts a significant interest in the database research community. There are multiple reasons for that. First, graphs are ubiquitous data structures. For instance, consider the multitude of real-world networks that populate our digital world. Second, some graphs are very large and grow at a high pace [8]. For instance, consider the size of social networks on the Web. Finally, graphs do not always have a rich structure that we can exploit for efficient querying. For instance, consider the content in RDF knowledge bases [6].

The research landscape around graph processing is so large that it is virtually impossible to provide a list of references without missing some important works. Therefore, we will limit ourselves to refer to some recent surveys and the references therein [3, 4, 7, 8]. In general, if we look at the most prominent works in this space, then we can make a couple of important observations. First, many graph engines tend to focus on a particular workload. For instance, Virtuoso [1] is a state-of-the-art centralized engine for SPARQL querying while Apache Giraph [1] targets more distributed analytical workloads. Second, graph engines tend to offer vertical solutions; that is, they provide the entire technological stack from the parsing of the query to the physical execution. An advantage of focusing on a specific workload is that we can optimize the computation more effectively than if we would use a more general-purpose technology. The disadvantage, however, is that the user may have to load the same graph in multiple databases if she needs to to perform different operations.

In my presentation, I will describe **Trident** [9] – a new graph database developed at the Vrije Universiteit Amsterdam with the ambitious goal of providing a "universal" engine that works well in many scenarios. The development of **Trident** was driven by two simple principles. The first is that the engine should be able to support as many workloads as possible. The second is that the engine should be able to interface efficiently with existing libraries to avoid to "reinvent the wheel".

The result of our development is a novel architecture that combines several components to provide both node- and edge-centric computation. These components include a single B+Tree for storing information about the nodes and a variable lists of binary tables to store adjacency lists of edges. A distinctive feature is that the binary tables are stored on disk using different layouts, effectively *adapting* depending on the structure of the input graph. To support different workloads, **Trident** provides a list of low-level primitives that can be used to implement many higher-level operations.

Trident has been built from scratch and works with all major operating systems. It supports updates and has a small memory footprint. Recently, due to the help of Samsung, it also works on Android platforms. The list below summarizes some key observations obtained from our experiments:

- **Trident** supports efficient SPARQL querying, thanks to the coupling with RDF3X [5]. Our comparison shows that our engine is faster than several leading academic and industrial competitors on multiple SPARQL benchmarks.
- **Trident** is also capable to deal with ML-based workloads. We compared its performance on a well-known ML task, namely the learning of graph embeddings [10] and our engine was significantly faster than an alternative implementation on Tensorflow.
- **Trident** can interface with the SNAP library [2]; hence it supports many graph algorithms (e.g., PageRank, triangle counting). Our comparison against the SNAP native storage system shows that **Trident** is much faster and can load much bigger graphs.
- **Trident** has an excellent scalability. In our largest experiment, we managed to load a graph with 100B edges (10^{10}) in a machine that costs less than $5k.

The presentation will focus on describing the novelty behind **Trident**, providing details about its infrastructure and the experiments that we conducted against the state of the art. **Trident** is freely available at `https://github.com/karmaresearch/trident`.

[1] `https://giraph.apache.org/`

References

[1] O. Erling and I. Mikhailov. Rdf support in the virtuoso dbms. In *Networked Knowledge-Networked Media*, pages 7–24. Springer, 2009.

[2] J. Leskovec and R. Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1, 2016.

[3] N. Liu, D.-s. Li, Y.-m. Zhang, and X.-l. Li. Large-scale graph processing systems: a survey. *Frontiers of Information Technology & Electronic Engineering*, 21(3):384–404, 2020.

[4] R. R. McCune, T. Weninger, and G. Madey. Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing. *ACM Computing Surveys (CSUR)*, 48(2):1–39, 2015.

[5] T. Neumann and G. Weikum. RDF-3X: a RISC-style engine for RDF. *Proceedings of the VLDB Endowment*, 1(1):647–659, 2008.

[6] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor. Industry-scale Knowledge Graphs: Lessons and Challenges. *Commun. ACM*, 62(8):36–43, 2019.

[7] S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin, and M. T. Özsu. The ubiquity of large graphs and surprising challenges of graph processing: extended survey. *The VLDB Journal*, 29(2):595–618, 2020.

[8] S. Sakr, A. Bonifati, H. Voigt, A. Iosup, K. Ammar, R. Angles, W. Aref, M. Arenas, M. Besta, P. A. Boncz, K. Daudjee, E. D. Valle, S. Dumbrava, O. Hartig, B. Haslhofer, T. Hegeman, J. Hidders, K. Hose, A. Iamnitchi, V. Kalavri, H. Kapp, W. Martens, M. T. Özsu, E. Peukert, S. Plantikow, M. Ragab, M. R. Ripeanu, S. Salihoglu, C. Schulz, P. Selmer, J. F. Sequeda, J. Shinavier, G. Szárnyas, R. Tommasini, A. Tumeo, A. Uta, A. L. Varbanescu, H.-Y. Wu, N. Yakovets, D. Yan, and E. Yoneki. The future is big graphs: a community view on graph processing systems. *Communications of the ACM*, 64(9):62–71, 2021.

[9] J. Urbani and C. Jacobs. Adaptive Low-level Storage of Very Large Knowledge Graphs. In *Proceedings of The Web Conference 2020*, WWW '20, pages 1761–1772, Taipei, Taiwan, 2020. Association for Computing Machinery.

[10] Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge Graph Embedding: A Survey of Approaches and Applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.