Summary of ”Evaluation of Graph Databases Performance Through Indexing Techniques”

Adney Cardoza - ac79873

Possible Taxonomy Hint: Index comparison of graph databases, Benchmarking, Limited comparison between commercially available open source databases

1. Overview. This paper [1] gives the reader a brief overview of the database trends over the years, and discusses the shift from relational databases to graph databases as the use of such databases has changed and also the input data to these databases has increased in magnitude. Along with this high influx of input data, these databases need to retrieve data and relations fast enough to provide satisfactory user experience. This paper then goes on to evaluate two of the many available graph databases (Neo4j and OrientDB) and compares their indexing methods to finally state that OrientDB performs better than Neo4j when indexes are used over large social-network like workload.

2. Graph Databases for highly relational data. Relational Databases Management Systems (RDBMS) fall short when the input data is high in volume and dense in relations. To elaborate, RDBMS, even NOSQL databases, experience performance degradation in scalability and complexity. Most current world RDBMS make concessions on the Consistency part of the CAP theorem [3], when scaling out - which the author dubs such systems as BASE (Basically Available, Soft-state, Eventually) as high availability of data and partition tolerance is preferred when there is a high use demand, whereas data consistency constraints can be relaxed.

Graph databases on the other hand, are inherently built to handle such high number of relations between high number of nodes, in order to efficiently facilitate data retrieval as well as efficient data storage. Where RDBMS are reaching limits of the CAP problem [3] even with optimized schemas, and suffer query execution performance drop when a query involves a lot of table joins. Along with this when using RDBMS, we also face problems of scalability, schema evolution over time, modeling of tree structures, semi-structured data, hierarchies and networks, etc. Graph Database Management Systems (GDBMS) don’t have to do multiple table joins when operating on a relationship in a network, and this problem shifts from using efficient join algorithms, to finding efficient ways to traverse the graph structure for query results.

The authors discuss some non-standard but widely used categorisation for GDBMS into native and non-native GDBMS. Native GDBMS use optimized native graph storage and are designed for storing and managing graphs, whereas non-native GDBMS do not use graph specific storage ideologies, and in turn serialize their data into an object-oriented database, a relational database or some other general-purpose data store [1]. The authors also mention some GDBMS use index-free adjacency, that is, nodes store physical pointers to their neighboring nodes - facilitating efficient graph traversal and data retrieval - but they further mention that they consider any database that exposes a graph data model through a CRUD API as a GDBMS. Furthermore, this variety in how graph databases are implemented comes down to what trade offs the user wants to prioritize over the others. Native graph processing benefit graph traversal performance, however at the expense of making some non-traversal operations more memory intensive or complex[2; 4].

3. Indexes make things faster. In any kind of database system, indexes over the data make data retrieval much faster than without indexes. In RDBMSone would have to create such an index using a BTree or maybe a HashMap, whereas in GDBMSthe natural structure of how data is stored in a graph provides a natural adjacency index technique. As mentioned before, traversing the data graph is much faster than joining data over global index.

4. Comparison of Neo4j and OrientDB. The authors of this paper compare these popular open source graph database systems available today - Neo4j and OrientDB, each using a different implementation of how they store data underneath, and how data is retrieved when a query is made. They conduct two distinct experiments, one to test the raw performance times of executing a query on a 5,500 node Social Network graph model, once without an index, and then with an index, for a specific key, and then for all instances of a specific key without and with an index. They also run a query to find the shortest path between two individuals on the two databases. It is seen that OrientDB is superior in it's performance gain when an index is introduced having reduced query times from 105ms to as low as 5ms, whereas the same comparison when done on Neo4j, leads to very minimal performance gain, from 48 ms query execution time without an index to only 40 ms with an index.

The authors also conduct a breadth-first search traversal experiment with a fixed depth of 5 on the social network graph mentioned above, and then count the number of distinct friend of friends on two sets with 10k people and 100k people. It is seen that, Neo4j and OrientDB both provide different results - but the authors have not explained this result in detail. Assuming a higher number is better as it provides more relations at a particular depth, we see that OrientDB performs much better than Neo4j in retaining deep relations for dense networks.

5. Conclusion. Hence, it is seen from the results that, indexing is required for optimal performance when there is a large input size. To conclude the authors state that when there is a large number of nodes, and where indexing techniques are necessary for optimum performances, OrientDB is the way to go over Neo4j.

References.

[1] Steve Ataky T. M, Lucas Ferreira, Marcela Ribeiro, and Marilde Prado Santos. Evaluation of graph databases performance through indexing techniques. *International Journal of Artificial Intelligence Applications (IJAIA)*, 06:87–98, 09 2015.

[2] Marek Ciglan, Alex Averbuch, and Ladialav Hluchy. Benchmarking traversal operations over graph databases. In *2012 IEEE 28th International Conference on Data Engineering Workshops*, pages 186–189, 2012.

[3] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, jun 2002.

[4] Peter Macko, Daniel Margo, and Margo Seltzer. Performance introspection of graph databases. In *Proceedings of the 6th International Systems and Storage Conference*, SYSTOR '13, New York, NY, USA, 2013. Association for Computing Machinery.