Graph Analytics on Relational Databases

Alekh Jindal, Samuel Madden, Amol Deshpande, Michael Stonebraker

CSAIL, MIT

Abstract

Graph analytics is getting increasingly popular these days and there is a deluge of new systems for graph analytics. However, it is not clear how good or bad are the relational databases for graph analytics. In this talk, I will share our experiences with graph analytics on relational databases. Contrary to the popular belief, modern relational databases can have very good performance over graph analytics. Furthermore, we can offer better (and efficient) programming interfaces for expressing graph queries in relational databases, thereby not forcing the users to SQL.

1. INTRODUCTION

Graph data management has received a lot of attention in recent times and a number of graph data management systems have been proposed recently. These systems address roughly two kinds of query workloads: (i) low latency online graph query processing, e.g. social network transactions, and (ii) high throughput offline graph analytics, e.g. PageRank computation. Typical examples of systems for online graph processing include RDF stores (such as Jena [8] and AllegeroGraph [1]) and key value stores (such as Neo4j [12] and HypergraphDB [7]). Some recent graph processing systems wrap around relational databases to provide efficient online query processing. These include TAO [2] from Facebook and FlockDB [5] from Twitter, both of which wrap around MySQL to build a distributed and scalable graph processing system.

On the other hand, graph analytics differs from traditional data analytics in three ways: (i) iterative nature of queries, (ii) maintaining state across iterations, (iii) very large graphs. Hadoop has gained a lot of popularity as a platform for large scale data analytics in recent years. And indeed there have been several works to improve upon iterative queries in Hadoop. These include HaLoop [3], Twister [4], and PrItr. Though these works improve the performance of iterative queries, users still need to *think* their analytical graph queries as MapReduce jobs. Alternatively, other approaches allows programmers different interfaces for expressing their analytical graph queries. For instance, Pegasus [9] allows users to express their graph queries as matrix operations and Pregel [11] employs vertex centric model to express graph queries. Both Pegasus and Giraph [6] (open source implementation of Pregel), however, translate the user queries to MapReduce jobs. Trinity [13] and GraphLab [10], also based on vertex centric computation model, bypass the MapReduce query layer and talk directly to the underlying distributed data store (key-value store and HDFS respectively).

2. PROBLEM

This brings us to the question: what happens to the good old relational database systems in the context of graph analytics? do they become obsolete or are they still relevant for large scale graph analytics? do users now need to dump their data from the relational database to a graph database or can they perform graph analytics (with comparable performance) from within the relational engine? Interestingly, relational databases have undergone several path breaking changes in recent years. These include the advent of column-oriented databases, main-memory databases, and array-oriented databases. Could these recent advances be used for efficient graph analytics as well? How good or bad are relational databases for graph analytics anyways? A recent work demonstrated finding shortest paths in a graph on Oracle [15], a row-oriented database system. However, several works in the column store literature have shown the superiority of column stores over analytical workloads. Could columns stores be exploited for graph analytics as well?

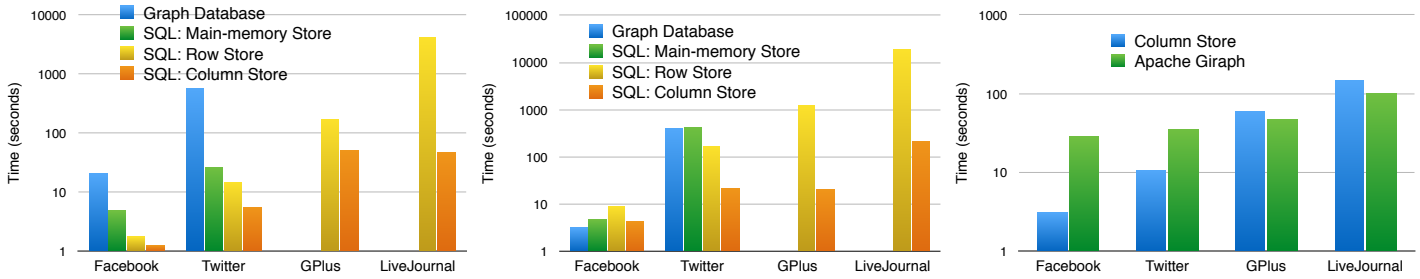
Thus, there is an absence of a comparative study of modern relational databases in the context of graph analytics. In our effort, we are trying to benchmark and understand the performance of relational databases for graph analytics. In this paper, we report our findings so far. Our key observations are: (1) parallel graph exploration can significantly reduce the number of joins, a key bottleneck in relational databases, (2) relational databases can match or even significantly outperform a graph database, and (3) column stores have superior performance over graph analytics as well. Furthermore, apart from expressing graph analytics as SQL queries, we also consider exposing a more natural vertex-centric programming interface on top of a relational database. The vertex-centric interface means that programmers can now actually *think* in terms of a graph while still running their queries in a relational engine.

3. PERFORMANCE

We model a graph in a relational database as a set of edges, i.e. we have an edge table for storing the edges and a node for storing the node ids. However, each edge hop is a self-join over the edge table. This could be fine for queries, such as triangle counting, which involve only a handful of joins [14]. But for queries such as finding shortest path, the number of join could quickly grow very large, and hence very expensive, when exploring a large graph. To address this, we explore graph in parallel along all outgoing edges, instead of each edge at a time. This means that we do more work in each iteration but significantly reduce the number of joins. In fact, the number of joins is bounded by the diameter of the graph, which in many graphs e.g. social networks is pretty small.

We compared the performance of relational databases on graph analytics as follows. We picked a leading graph database system and three relational databases: a row-oriented database, a column-oriented database, and a main-memory database. We implemented two queries, PageRank and Shortest Paths, on each of these systems. We created the appropriate sort orders and indexes for all systems. We used four datasets from Stanford large network dataset collection: (i) Facebook dataset having 4K nodes and 88K edges, (ii) Twitter dataset having 81K nodes and 1.8M edges, (iii) Google+ dataset having 107K nodes and 13M edges, and (iv) LiveJournal dataset having 4.8M nodes and 68M edges.

Figure 1(a) and 1(b) show the result. We can see that relational databases outperform the graph database on PageRank by up to two



(a) PageRank (b) Shortest Paths (c) Shortest Paths on Vertex-centric Interface

Figure 1: Graph Analytics on Relational Databases.

orders of magnitude. This is because PageRank involves full scanning and joining of the nodes and edges table, something which relational databases are very good at doing. The graph database, on the other hand, cannot handle larger graphs. Finding shortest paths involves starting from a source node and successively exploring its outgoing edges, a very different access pattern from PageRank. Still we see from Figure 1(b) that relational databases match or outperform the graph database in most cases. In fact, the column-oriented database is two orders of magnitude faster than the graph database on Twitter dataset. The only exception is the main-memory database over Twitter dataset[1]. The graph database again cannot handle very large graphs.

Note that the column-oriented database system stands out from other database systems in terms of performance over graph analytics. This suggests that column stores could be highly suited for graph analytics, just as they are suited for relational data analytics. We are currently working on extending our benchmark to larger datasets as well as on including more queries.

4. EASE-OF-USE

A key criticism of relational databases could be forcing programers to use SQL, the de facto query interface in relational databases. Indeed, implementing graph queries in SQL is tricky and time consuming. Thus, the question is whether we can have other graph-friendly query interfaces on top of a relational database. Vertex-centric programming interface, e.g. Pregel, is one such popular interface for graph analytics. We developed a similar vertex-centric programming interface on top of the column-oriented database system. The idea is that the users simply write their vertex centric compute functions, same as in Pregel, and our interface automatically maps it to the underlying SQL engine. As a result, users do not have to deal with SQL at all. The system invokes the compute function in each super-step and passes the messages from one super-step to the other. This is very similar to Giraph, the open source implementation of Pregel, which maps vertex centric computations to MapReduce jobs, without having users to deal with them explicitly.

Figure 1(c) shows the performance of running shortest paths on the vertex-centric interface of the column store compared to Giraph. We can see that the column-store significantly outperforms Giraph for smaller graphs. This is because Giraph suffers from the initial startup cost of Hadoop MapReduce, which is not the case for the column-store. Even for larger graphs the vertex-centric interface on column store performs very similar to Giraph. This is in spite of Giraph loading the entire graph in memory and not persisting any intermediate output, whereas the column store needs to store a lot messages to disk. Thus, we see that column stores can provide an efficient vertex-centric interface.

With vertex centric interface users can now easily express several other graph analytic queries such as connected components, random walk with restart, and other message passing algorithms, right *in* their good old relational database.

[1]This is because the main-memory database currently does not support updates with join condition and therefore we need to first select the qualifying rows and then update them using an iterator.

5. CONCLUSION

Graph analytics is emerging as an important application area and several graph data management systems have been proposed recently. However, this requires users to switch to yet another data management system. In this paper, we revisited relational databases in the context of graph analytics. We implemented two popular graph queries, PageRank and Shortest Paths, on different relational database engines. Our results show that relational databases can match or even outperform a graph database. In particular, column-oriented database can perform really well. Furthermore, we showed vertex-centric programming interface, which allows programmers to easily specify their graph queries, on top of a relational database. Our results demonstrate that graph analytics over relational databases is promising and we could do both traditional as well as graph data management within a single system.

6. REFERENCES

[1] AllegroGraph, http://franz.com/agraph/allegrograph.
[2] N. Bronson and al. TAO: Facebooks Distributed Data Store for the Social Graph. *USENIX ATC*, 2013.
[3] Y. Bu and al. The HaLoop Approach to Large-Scale Iterative Data Analysis. *VLDB J.*, 21(2):169–190, 2012.
[4] J. Ekanayake and al. Twister: A Runtime for Iterative MapReduce. *HPDC*, 2010.
[5] FlockDB, http://github.com/twitter/flockdb.
[6] Apache Giraph, http://giraph.apache.org.
[7] HyperGraphDB, http://www.hypergraphdb.org.
[8] Apache Jena, http://jena.apache.org.
[9] U. Kang and al. PEGASUS: A Peta-Scale Graph Mining System - Implementation and Observations. *ICDM*, 2009.
[10] Y. Low and al. Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud. *PVLDB*, 2012.
[11] G. Malewicz and al. Pregel: A System for Large-Scale Graph Processing. *SIGMOD*, 2010.
[12] Neo4j, http://www.neo4j.org.
[13] B. Shao and al. Trinity: A Distributed Graph Engine on a Memory Cloud. *SIGMOD*, 2013.
[14] Counting Triangles with Vertica, http://www.vertica.com/2011/09/21/counting-triangles.
[15] A. Welc and al. Graph Analysis Do We Have to Reinvent the Wheel? *GRADES*, 2013.