1.1 Paper surveyed by Ning Dong

1.1.1 GBASE: an efficient analysis platform for large graphs

Problem definition

This paper proposes a graph processing system called GBASE. It aims to solve 3 problems in graph analysis using their framework. First, represent and store graphs in distributed settings efficiently. Second, define a set of core algorithms. Third, optimize user queries on the graph.

Main idea

For storage, this paper represents graphs in homogeneous blocks(either very sparse or very dense) and compresses them using zip compression or Gap Elias- γ encoding. According to their estimation, Information theoretic minimum cost is achieving asymptotically. Using grid placement, their system guarantees both in-neighbor and out-neighbor queries are handled efficiently, compared to vertical and horizontal placement which has O(K) file accesses where K is number of files.

For defining core algorithms, this paper gives matrix-vector version and SQL version for graph mining operations including k-step neighbors, induced subgraph and single-source shortest distance. In practice, SQL queries could be highly optimized in both single machine and distributed clusters. Users do not need to care about lower level operations when system scales up.

For query optimization, this paper introduces their grid selection strategy which selects only relevant grids for targeted queries. And they answer incidence matrix queries with adjacency matrix in hadoop mapreduce scheme.

This paper presents results of experiments on graphs of different scale, in different clustering and compression settings. GBASE is space and time efficient in indexing and storage. It also gives better performance in both global and targeted query time.

Use for our project

When building GBASE, a set of core algorithms in graph analysis is implemented in SQL, which would be extreamely useful resource to reference, when we implement D-CUBE in SQL.

Shortcomings

Hadoop mapreduce is a heavy I/O task where a lot of read/write operations to disks are involved. In tasks where user wants to use induced graph repetitively, I/O cost would be huge. Today computing frameworks like Spark may offer better in memory solution to this problem and could be plugged into the system.

1.1.2 Graph Analytics using the Vertica Relational Database

Problem definition

This paper aims to do graph analytics on Vertica relational database using SQL queries and column stores. This way, they can avoid the problem in moving data from relational database to other advanced graph processing systems, meanwhile reaching competitive performance. They provide a framework in executing graph analysis algorithms in Vertica efficiently and introduces some best practices.

Main idea

This paper first justifies their choice using relational database and column stores. Graph data in relational database costs much to move. And table joins and aggregates operations benefit from column store.

This paper then introduces Vertex-centric model. Users only need to provide UDFs, which are used to update vertex state and messaging neighbors. The limit of graph processing engines like Giraph is that it requires a fixed query plan for the execution pipeline.

This paper proposes a graph analysis framework in Vertica. First, they translate query plan to SQL by using relational expression as intermediate step. Then 3 kinds of query optimizations could be performed: (1) Replace vertex or edge table with new copy instead of updating them. (2) Perform incremental evaluation that only explores the subgraph we are interested in every iteration. (3) Eliminate redundant joins. Lastly, they describe four key advantages for Vertica to support query analysis: (1) Physical design which manipulates graph data efficiently. (2) Join optimizations which. (3) Query pipelining which avoid materializes intermediate results. (4) Intra-query parallelism which fully utilizes the capability of multiple cores. The paper also covers methods of running UDFs without translating it into SQLs and in-memory execution.

According to experiments in the paper, overall performance of Vertica is comparable to Giraph and Graphlab. The most important reason is that there’s no load/store phase needed for vertica, which dominates Graphlab. To achieve this, users may have to heavily hand craft SQL implementation, as stated in the paper. Vertica gives less memory footprint and read I/O operations however suffers from intermediate result write and thus incurs more write I/O. When performing in memory UDF mentioned above, they reduce write I/O and the tradeoff here is more memory footprint.

Vertica handles advanced graph analytics like strong overlap queries and weak ties better than graph processing engines with heavy in-memory computation. First these tasks are difficult to implement in vertex-centric graphs, second systems like Giraph may run out of memory.

Use for our project

This paper tells us that the performance of using SQL on relational database to do data mining is comparable to those state-of-art graph processing framework. It also discusses tradeoffs (memory and I/O) between two choices. In practice, it gives a lot of examples in how to implement graph analysis algorithms using SQL and how to optimize them.

Shortcomings

Heavy hand craft work on SQL is still needed if we want to achieve expected performance on Vertica. In the future maybe more advanced SQL optimization could be integrated in the system. To handle the risk of users not getting correct results because of aggressive optimization, maybe different levels of optimizations could be a choice, as what many compilers are doing nowadays.

1.1.3 D-Cube: Dense-Block Detection in Terabyte-Scale Tensors

Problem definition

Dense blocks in tensors usually indicate something unusual which is widely used in tasks like fraudulent or anomaly detection. This paper offers a method to detect dense-block detection in large scale data that can not fit into memory or even disk. To overcome the shortcomings in other methods like M-Zoom and CrossSpot, it is designed in distributed and out-of-core setting. This paper discusses their algorithms thoroughly and also gives a mapreduce implementation.

Main idea

This paper first introduces density measures. All three measures, arithmetic average mass, geometric average mass and suspiciousness could be plugged into their D-Cube algorithm.

The algorithm works by finding single blocks one by one. Here we should use updated relation (which subtracts a block from it each iteration) as parameter for find single block function, whereas use original relation to calculate the block to merge into result. To find single block, following steps are operated before all attribute values are removed. First, select dimension by cardinality or density. Second, find all values whose mass on this dimension is less or equal than “average” (defined as total mass divided by set size of this dimension). Then, sort elements in set to be removed, update max density and order of attribute values with it. At the end of the function, reconstruct the block by picking values that have large order.

Two dimension selection algorithms both have its advantages. Maximum cardinality policy is simple to implement and gives accuracy guarantee and maximum density policy works well in real world data.

The paper talks about two optimizations, combining disk-accessing steps in one or near iteration and caching tensor entries, both of which improve performance in practice.

Time complexity of D-CUBE is $O(kN^2|R|L)$ and space is dominated by sum of distinct

value set size in each dimension. The paper also gives proof in accuracy guarantee in D-CUBE. Mapreduce implementation of their algorithm is also given.
D-CUBE is memory efficient and achieves both speed and accuracy. It shows improvements over M-Zoom, CrossSpot and CPD in experiments. The paper also gives results about its scalability and effectiveness.

Use for our project

In phase 2 of our project, we will investigate this D-CUBE algorithm and implement it in SQL.

Shortcomings

There are still some heuristics in the algorithms, including the threshold of mass to remove certain values in find_single_block function and density measures. Although in practice we can try different settings out and find a best one, it’s still helpful if more theoretical research on them could give optimal solutions in principle.