Phase2: SQL Implementation

Yangqingwei Shi
`yangqin1@andrew.cmu.edu`

Hao Jin
`haoj@andrew.cmu.edu`

April 25, 2017

1 Plan for activites

- In T3, Hao will be mainly working on the optimizations through database-related techniques such as indexing and query optimization while Yangqingwei will look into possible modifications to the algorithms and metadata caching to further improve the overall performance.
- In T4, Yangqingwei will be working on plotting and analysis of data retrieved from application of our implementation on various datasets, and Hao will be working on adding portability support to the code so that the cost is minimum when we want to move to a different dataset with different number of dimensions.
- In T5, we will be working together on this task as this will be the heaviest one. Each of us will work on some of the datasets and try out different settings. Finally, we will be working together on any possible further optimizations and work on documentations of our findings.

2 Survey

2.1 Papers read by Hao Jin

The first paper was “Graph Analytics using the Vertica Relational Database”.

- *Problem Definition*: Many vertex-centric analyzers of graphs have been developed with the growing interest in graph processing in recent years. However, most of these systems were developed based on the assumption that new type of storage is used to store the data, which conflicts with the fact that most of graph datasets are still stored in traditional relational databases.
- *Main idea*: To avoid the overhead of moving data around from traditional database to latest type of storage to perform graph analyses, this paper proposes that vertex-centric graph processing can be completed by database queries with proper change of the database system and the algorithm. With the switch to traditional databases, graph queries can also make use of the existing optimization techniques in current database systems. The paper analyzed the SQL implementation of Single Source Shortest Path(SSSP), CC(Connected Components), and PageRank together with corresponding query optimizations and system extensions. The optimizations range from optimizations on the queries themselves, to existing execution optimizations in Vertica system, to specialized extensions to Vertica for the graph tasks. The authors performed experiments on sample datasets to compare the performance of the same graph queries in different systems and showed that using Vertica can greatly reduce the memory footprint of typical vertex-centric queries while still achieve competitive performance on sample datasets compared with GraphLab and Giraph. Beyond these kinds of queries, Vertica also performed well on advanced graph analyses such as Strong Overlap and Weak Ties, while Giraph ran out of memory on those two test. In conclusion, the authors show that vertex-centric graph analyses, even the ones that can not be handled by existing graph analytics systems, can be implemented in SQL to take advantage of existing optimizations in DBMS and achieve competitive performance.
- *Use for our project*: This paper is very useful for the project since it provides several approaches for rewriting traditional vertex-centric graph analyses to database SQL queries and also provides the necessary approaches for optimizing the queries for faster queries.
- *Shortcomings*: This papers experiment was performed on Vertica, which is a commercial DBMS product that provide many extra features and many optimizations than Postgres. Since we will be using Postgres for our project, we may not have access to some optimizations mentioned in this paper and will have to find alternative optimizations for the graph algorithms. On the other hand, the paper put emphasis on only 3 typical graph analysis workloads and mentioned 2 other types of graph analysis(Strong Overlap and Weak Ties) briefly. Since we may want to do comprehensive analysis on the datasets in our project, well have to think about how to transform other graph analysis workloads to efficient database queries.

The second paper was “GBASE: an efficient analysis platform for large graphs”

- *Problem Definition*: With the increasing interest in data mining of graphs, theres more and more effort put into efficient processing of computations on graphs. The sizes of target graphs can easily reach billion nodes and edges, thus occupying great amount of disk space and consuming much memory during computation. As a result, efficient storage, computation, and optimization become big problems for mining such graphs and need to be addressed effectively.

- *Main idea*: The paper focuses on the implementation of a general-purpose graph analysis system, GBASE, to address the problems with processing large-scale graphs. The system improves the performance of a bunch of graph analysis algorithms on large-scale graphs through efficient storage methods to reduce cost of storage and cost of query, various fast algorithms for a wide range of graph queries for faster response to queries, and graph processing optimization techniques for efficient execution of queries. The proposed formulation, compression and placement methods of blocks are proven to compress the graph data greatly, thus reducing the storage consumption. For several typical graph queries, the paper gave the matrix-vector multiplication form of the queries together with detailed SQL implementations. Since matrix-vector multiplications can be parallelized for faster performance, the paper also provided the Hadoop algorithm and applicable optimizations for execution. The space efficiency achieved by the storage improvements and the boost in query processing resulted from better algorithm and optimized parallel execution are proven via experiments on different datasets. In conclusion, GBASE is a graph processing platform that provides the users with compression of graph data, efficient algorithms for typical graph queries and effective optimizations of executions.

- *Use for our project*: This paper should be useful for our project. Firstly the detailed SQL implementations provided in the paper can serve as the building blocks of our final project. On the other hand, the reduction of graph query problems to matrix multiplications provide a very good idea of handling other types of graph queries, which may be useful when we get deeper into the project. Finally, the use of open-source parallel programming frameworks provide a good hint for optimization of computation components of the final project.

- *Shortcomings*: The compression method is based on the assumption that the graph data is not updated because the formulation of blocks is based on partition of graphs, if we want to constantly update our graph for latest version of the data, then we need to repeat the formulation, compression, and placement again and again. On the other hand, more transformation from graph queries to SQL queries can be developed for advanced graph queries.

The third paper was "PEGASUS: A Peta-Scale Graph Mining System - Implementation and Observations"

- *Problem Definition*: Sizes of real graphs of interest are ever-growing. With the vast growth of web services, more and more graphs are reaching tera- or peta-byte in size. As a result, we need various analyses of those graph data to be quick enough for those analyses to be feasible for wider range of users. Since all those different kinds of queries serve different purposes, its hard to find a common approach for improving the performance for each one of them.

- *Main idea*: This paper is about the generalization of graph queries into matrix-vector multiplication and the corresponding implementations and optimizations of such generalization. The paper proposed the Generalized Iterative Matrix-Vector multiplication(GIM-V) model as the underlying primary computation for various kinds of graph queries. This model unifies many kinds of computations on graphs to the simpler computations of matrix-vector multiplications. Reductions from PageRank, Random Walk with Restart and Diameter Estimation to GIM-V model are shown in the paper as examples, and are proven to produce the correct results. Then the authors moves one to efficient implementation and execution of GIM-V model. The paper also analyzed the performance of this generalization and compared the effectivenesses of different optimizations in the paper on different types of queries through applying the model and different optimizations on datasets collected from real network services on M45 supercomputing cluster provided by Yahoo!. In conclusion, PEGASUS is a system for efficient graph mining of extreme-scale graphs that adopts GIM-V model as the common underlying primitive for a variety of graph queries. With proper combinations of optimization techniques, the system can perform fast analysis on peta-scale real graph data. With the help of HADOOP framework, the system can achieve portable parallelization of computation.

- *Use for our project*: This paper is definitely useful for our project. Its a further step from the GBASE paper on generalizing useful graph algorithms to matrix-vector multiplications, and it shows very good performance gains from this generalization. Moreover, suitable optimizations with proven performance for the model are also given in the paper, which will be useful for the large datasets we work on for this final project. The paper also mentioned that PEGASUS may be useful for tensor analysis on HADOOP, which fits the topic of the final project very well, so we may want to dig deeper in that direction.

- *Shortcomings*: Only 3 kinds of graph queries are mentioned in the paper. However, there are many other graph queries that are necessary for our project, thus further effort should be invested in thinking about how to transform other types of graph queries to GIM-V model to take advantage of the features of the model.

2.2 Papers read by Yangqingwei Shi

The fourth paper was "A General Suspiciousness Metric for Dense Blocks in Multimodal Data"

- *Problem Definition*: A lot of data in our real life can be represented by multimodal data, we call it a tensor. In most cases, we want to find the suspicious dense block from a given tensor. However, until now there is even no specific criteria of the suspiciousness of a dense block.

- *Main idea*: The main contribution of the paper can be devided into two parts. In the first part, the authors proposed a detailed mathematical model of multimodal dataset and dense blocks. In specific, it gives the definition of tensor χ, subtensor γ, the mass m and density ρ.The author also gives metric criteria of suspiciousness based on a list of five axioms. The five axioms are based on comparison of suspiciousness of two dense blocks. Derived from the five axioms, the author gave out a mathematical expression of suspiciousness of a dense block that is:

  $$f(n, c, N, C) = -log[Pr(Y_n = c)]$$

  It is based on an Erdös-Rényi-Poisson model, and Y_n is the number of entries.
  In the second part of the paper the author gave out a simple algorithms CrossSpot for finding suspicious blocks in a tensor and rank them in order. The algorithm start from a random seed and repeatedly adjust the dataset until it converges. The complexity of the algorithm is $O(T \times K \times (E + NlogN))$, which is quasi-linear in N and linear in the number of entries.

- *Use for our project*: The main use of this paper is its first part, the model of dense blocks in multimodal data as well as the criteria of the suspiciousness of a dense block. The most useful part of this paper is the suspiciousness function. With suspiciousness function , we can quantify the suspiciousness into value so that we can easily evaluate the performance of our project.

- *Shortcomings*: The CrossSpot algorithm is the weakest algorithm among all three algorithms, in comparison with M-Zoom and D-Cube. The local search may take a long time before it converges. Also, the algorithm start from a seed subtensor randomly, which I think might cause error. Although the authors confirmed there is no problem, the scale of the evaluation in the paper was not so sufficient.

The fifth paper was "M-Zoom: Fast Dense-Block Detection in Tensors with Quality Guarantees"

- *Problem Definition*: Finding dense blocks in a multimodal tensor has many applications in real world. In the former paper "CrossSpot", the definition and criteria of data mining in tensor had been clear provided. However, all recent algorithms on dense block detection suffers from problems of flexibility, scalability, effectiveness. There is no algorithm having accuracy guarantees.

- *Main idea*: The main contribution of the paper can be concluded to a better algortihm for dense block detection in tensors, called M-Zoom. In contrast to to other existing algorithms, M-Zoom has better performance, scalaibilty, effectiveness and accuracy. It slight changes the definition of traditional dense block terms by including the word relation R which is a set of attributes and a measure attribute. The algorithm of M-Zoom repeatedly finding a single dense block on the remain relation, moving the block out of relation, and adding the block of the original relation including the exactly same attributes to list of dense blocks. The detection of single dense block is attribute-based. The attribute of the blocks are maintained by a priority queue or min-heap and moved out in order. A snapshot will record the density of the result after each movement and select the block of largest density. The effectiveness and the performance of M-Zoom is promised by this implementation. In each iteration of detecting single dense blocks, the algortihm provides a size bound and filter out all dense blocks that are not in the size bound. The accuracy is guaranteed by the size bound that the accuracy of the dense blocks within size bound is proved to be more accurate than those outside the bounds. The total complxity of the problem is near linear.

- *Use for our project*: M-Zoom is a great algorithm with high performance and accuracy. I think it will be the algorithm we mainly based on in this project because it is effective and also simple. D-Cube algorithm has much higher performance in a distributed system. However, I think the data in our project is not very huge scale. As a result, we do not need a distributed system and M-Zoom will be the best option.

- *Shortcomings or Suggestions*: This algorithm filter the block by size bounds. Actually, I think filtering out all results outside the bound is not practical. Sometimes, small size bound or large size bound data bay also represent a type of data. It might be helpful if we get them at the same time we doing data mining and evaluate them. On the other hand, the paper said the algorithm is scalable, but as the dimension of the if we maintain the same size bound, the result may have less accuracy promise.

The sixth paper was "D-Cube: Dense-Block Detection in Terabyte-Scale Tensors"

- *Problem Definition*: While M-Zoom is a good algorithm in dense block detecting in tensors with accuracy guarantee and good performance, it has limitation that it cannot perform well on big data. In real world, the scale of data is increasingly huge. In most cases, these data are recorded in distributed systems. As a result, we need to optimize the M-Zoom algorithm to a new algorithm that make good use of distributed system.

- *Main idea*: The main contribution of the paper is an optimized M-Zoom algorithm on distributed system called D-Cube. Same as M-Zoom, D-Cube also have good scalabiility and accuracy guarantees. In contrast to to M-Zoom, D-Cube has an enhanced performance on distributed systems. In this paper, the author assumed the tuple data of tensor (single points) are distributed in a distributed file system. Briefly speaking, D-cube optimized M-Zoom by changing the detection of single dense blocks from attribute-based to tuple-based. By such change, we can get more accuracy dense blocks that has a subset on each attribute. Furthermore, tuple-based methods have better performance on distributed system. To prove this point, the paper also gave out an implementation of MapReduce method. In specific, the calculation of attribute now can be performed on distributed file systems as well as the filtering of tuples. As a result, D-Cube is a memory efficient algorithm on distributed system.

- *Use for our project*: D-Cube has higher peroformance than M-Zoom on distributed systems. I think it will be an optional algorithm for our project. Since for small data, M-Zoom algorithm is sufficient. If we found our algorithm do not have good performance, we might change our algorithm to D-Cube and use it on a cloud platform. While on the other hand, the disk-based consideration of this paper is useful since we need implement our algorithm on another particular platform SQL. The consideration tells us that we need to understand SQL well before we simply copy the implementation onto it.

- *Shortcomings*: I think *M-Zoom* can also be implemented on distributed system because we can still distribute tuple data on a distributed file system. If this can be done, the contribution of D-Cube and the improvement will be not significant.

3 Results on DARPA

We test our D-Cube[1] code on DARPA using different density measurements and dimension selectiing principles. The results of the first dense block detected are shown as the following tables (mass, size and density). The file darpa.csv has been bucketized by day with binarized cnt.

	arithmetic	geometric	suspicious
by density	22728	39868	99768
by cardinality	19217	19127	69548

Table 1: Results of Mass

	arithmetic	geometric	suspicious
by density	$34 \times 55 \times 44$	$22 \times 471 \times 45$	$37 \times 23398 \times 57$
by cardinality	$35 \times 36 \times 45$	$35 \times 36 \times 45$	$81 \times 2542 \times 57$

Table 2: Results of Dimension Sizes

	arithmetic	geometric	suspicious
by density	512.66	514.12	420285.97
by cardinality	496.99	500.21	367516.83

Table 3: Results of Density

4 Upcoming Experiments

The upcoming experiments for phase 3 will be focused on how to optimize our implementation of D-Cube[1] for faster execution on big input data. The plan for our experiments is as follows:

1. Try out common optimizations such as creating indexes on tables.
2. Reduce unnecessary expensive queries.
3. Optimize queries for faster execution.
4. Use memory to cache necessary metadata that are guaranteed to fit in memory for faster retrieval and less disk accesses.

4.1 Indexes

Currently most of the tables used during the computation does not have primary keys and indexes, which prevents us from accelerating our implementation from taking advantage of indexes. During phase 3, we will first use the EXPLAIN and ANALYSE queries of Postgres to look for possible bottlenecks, then try adding indexes to the tables and compare the results of EXPLAIN and ANALYSE after the index creation to see if indexes would help.

4.2 Reduction of Expensive Queries

Currently for ensuring correctness, our implementation made quite a few fresh table creations, which tears down the existing table and create a totally fresh one with new data. This may be very expensive since a fresh creation involves deletion of the old table and insertion to the new version of the table. On the other hand, some fresh creation can be substituted with proper updates to current table, then fresh creations become unnecessary. Thus reducing such expensive queries would help to reduce the time wasted on fresh creations during the execution.

4.3 Optimize Queries

As we all know, we should always try to optimize the queries ourselves instead of totally relying on the DBMS' optimizers. We plan to manually go through all queries used in our implementation and think about better ones based on the output from EXPLAIN and ANALYSE functions of Postgres.

4.4 Metadata Caching

Although we cannot assume that the input data fits in the memory, we can always cache some metadata that are used often during our implementation in memory so that we do not have to use SQL queries to get them when we need them. We plan to look for useful metadata during phase 3 and try out caching on them and benchmark the performance improvement.

Appendix A: Revised Plan of Activities

Original:

- In T2, we will work together to combine what we got from all the readings to complete the SQL implementation of D-Cube.
- In T3, Hao will be leading on the optimizations by database indexing while Yangqingwei will look into possible modifications to the algorithms to take full advantage of indexing.
- In T4, Yangqingwei will be working on plotting and analysis of data retrieved from application of the algorithm on datasets, and Hao will be working on applying the SQL codes to the datasets.
- In T5, we will be working together on this task as this will be the heaviest one. Each of us will work on part of the datasets and report interesting findings. Finally, we will working on combining our findings.

Revised:

- In T3, Hao will be mainly working on the optimizations through database-related techniques such as indexing and query optimization while Yangqingwei will look into possible modifications to the algorithms and metadata caching to further improve the overall performance.
- In T4, Yangqingwei will be working on plotting and analysis of data retrieved from application of our implementation on various datasets, and Hao will be working on adding portability support to the code so that the cost is minimum when we want to move to a different dataset with different number of dimensions.
- In T5, we will be working together on this task as this will be the heaviest one. Each of us will work on some of the datasets and try out different settings. Finally, we will be working together on any possible further optimizations and work on documentations of our findings.

Appendix B: List of Unit Tests

We designed unit tests for the basic helper functions. Here's a list of unit test in test.py file:

1. test_get_mass: The unit test for testing the helper function to get the total mass of a table.

2. test_tuple_counts: The unit test for testing the helper function to get the total number of tuples in a table.

3. test_tuple_count_distinct: The unit test for testing the helper function to get the cardinality of a certain dimension of a table.

4. test_rho: The unit test for testing the help function to calculate different density measurements.

We performed unit tests on those function since they are very important building blocks of the whole D-Cube implementation.

References

[1] Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. D-cube: Dense-block detection in terabyte-scale tensors. In *Proceedings of the 10th ACM International Conference on Web Search and Data Mining*, WSDM '17. ACM, 2017.