WWJMRD 2018; 4(12): 80-84
www.wwjmrd.com
International Journal
Peer Reviewed Journal
Refereed Journal
Indexed Journal
Impact Factor MJIF: 4.25
E-ISSN: 2454-6615

H. Santhi
School of Computer Science and Engineering, Vellore Institute of Technology (VIT), Vellore, Tamil Nadu, India

G. Gopichand,
School of Computer Science and Engineering, Vellore Institute of Technology (VIT), Vellore, Tamil Nadu, India

Gayathri P
School of Computer Science and Engineering, Vellore Institute of Technology (VIT), Vellore, Tamil Nadu, India

Correspondence:
Gayathri P
School of Computer Science and Engineering, Vellore Institute of Technology (VIT), Vellore, Tamil Nadu, India

Case Study on Graph Processing Using Graph Engine

H. Santhi, G. Gopichand, Gayathri. P

Abstract
We face various challenges in computing for managing and mining large amounts of data. Each day, data on the World Wide Web keeps on increasing. Maintaining the data and making it accessible to users is a complex problem. In this paper, we have focused on a particular database management system, called the Graph Engine, developed by Microsoft. This organizes data in a graph format on multiple machines, making it a distributed database. Graph Engine is applied on large graph databases such as the World Wide Web or a large social network. We have highlighted various modules in the Graph Engine such as partitioning of a billion node graph, subgraph matching operation, query processing and online community search. We have discussed in detail our observations on each module and gaps identified. A comparative analysis has been present to state the uniqueness of our paper.

Keywords: Graph Engine, distributed memory cloud, billion node, subgraph operation, trinity architecture, query processing

1. Introduction
In the present generation of computing, a large amount of research has been inclined towards displaying and managing communities in various forms on networking systems. Data processing in a computing device is the result of efficient query processing and optimization of data. Data can be present is different types and one of the types is an Abstract Data Type. This type of data consists of trees, hashes, heaps, graphs and so on. Each of these has its own particular focal points and detriments. When using real life applications of using an abstract data type, a graph is the preferred due to its ability to handle large amounts of interconnected data, to provide a sustainable relationship amongst data and to navigate between nodes in a constant amount of time [11]. To manage data networks such as the World Wide Web (WWW) or any social network, a data handling database system was built by Microsoft called Trinity, later renamed as Graph Engine [7]. This paper focuses on the outline of this database handling system and the modules it consists such as distributed Graph Engine over a memory cloud, Trinity file system, partitioning of a billion-node graph and so on.
The rest of the paper is organized in the following way. Section 2 describes literature review. Section 3 consists of the gaps identified in modules of the Graph Engine, suggestions for improvement and observations made. Section 4 entails the conclusions that we have identified.

2. Literature Review
The architecture of the Graph Engine was built upon a cluster of interconnected machines. For the storage infrastructure, the Graph Engine systematizes the memory into various available machines. This is called a memory cloud or a distributed memory address space to maintain large graph systems. The Graph Engine is inclined towards offline analytical applications and online query processing. It consists of three components that communicate with one another. They are: clients and libraries, proxies and slaves. The client application enables the user to interact with the Graph Engine cluster machines. The slaves store the data and perform specific computations such as message processing. The proxy servers are different from the slaves in such a way that they only handle message passing to clients from slaves and vice versa. They are known as the message aggregators. The libraries help the

clients to communicate with the slaves and proxies through APIs. The Graph Engine modules consist of a memory cloud. This enables the data to be hosted on a number of machines. A key-value pair is created that forms the main data structure of the whole graph system. This is supported by a consistent hashing mechanism to locate a particular key-value pair. An addressing table is linked to the hashing mechanism to store the given address and locate it on a machine. Thus, the key-value pair is located [8].

Applications of the real life scenarios and comparative case studies were explained using complex algorithms. The problem of overlapping of communities in social networks was taken and described as a NP-hard problem and appropriate algorithms for the overlapping community search by taking certain parameters into account such as the community density, overlapping awareness and type of relationship and consistency of the search. Ambiguous search result problem was solved by taking the two factors into account to get the most approximate search result. They were: the ambiguous names degree and the overlapping communities [1].

The Trinity graph system is an infrastructure which is spread over a number of machines and proposes a cohesive memory space for user programs. In this paper, the authors propose an approach to partition a billion node graph on a general distributed memory system created on the RAM of a computing device. They used a multilevel propagation approach to recursively make the web scale graph smaller and smaller to create a final segregation of the graph. The advantages of using this technique are highlighted along with their complex algorithms. This method is efficient and effective over large graph systems. The authors conducted various experiments on real life billion node graphs and applied their algorithms to partition the graphs. The results were successful with effective time and space efficiencies [11].

Managing and mining large graph data has proved of essential importance in the present day situation. This paper highlights the challenges posed by a large graph system, the specific architectural design requirements for different types of data and its application needs in today's world along with various programming models and an insight on developing patterns and algorithms to curb this problem. The authors have taken real world examples of the Facebook Social Graph, US Road Graph and a Web Graph system by showing the complexity of the data space and advantages of a distributed memory system on the Graph Engine built by Microsoft. They have compared a number of graph systems such as Google's Pregel and Neo4j to look for differences and disadvantages [9].

In real life web scale graph database applications, accessing the stored data is an important aspect. Therefore, path reachability indexing scheme is an essential part of obtaining data. The authors have specified certain parameters to update delete and search a graph database in an efficient manner to reduce the complexity and get the desired result. They have provided a detailed description of the various models of path tree reachability and extraction of the particular key-value pair. Insights into the given module using diagrams and algorithms have made it easier to comprehend. As we obtain an efficient mechanism for searching for the data, the query processing within the large graph becomes simpler and resourceful [5].

3. Gaps Identified, Suggestions and Observations made

The gaps identified are as follows:

- While partitioning the graph, mappings of the graph and the partitions are stored on different locations. This causes a delay as any access to a particular vertex of the graph need to first retrieve the structure and then access the data. This problem is hard to fix and is present in almost all distributed memory databases.
- Sometimes partitioning causes duplication of data and imbalance of distribution onto multiple machines. Certain machines get a large partition whereas some get a smaller partition.
- Subgraph matching operation can be very costly if the graph is stored in an RDBMS or a key-value store as it would require join operations also. There are also some exceptions in which certain queries do require join operations. In such a case, we have to also add the join algorithm code. Hence, making the source code larger and harder increasing the computation work by each machine.

Taking this in notice, many database systems tend to use the indexing scheme instead of the proposed one.

- Subgraph matching is also based upon a 2-hop index mechanism [6]. Its complexity is O (n^4), where *n* is the number of vertices. This is a large complexity and thus not very efficient for large graph databases.

To manage sensitive and important information is extremely necessary. We have noticed that there was no reference paper regarding the security systems in Graph Engine. As this is a distributed system containing sensitive information, there needs to be a structured security mechanism to protect the data. Thus, our group suggests some measures to be taken to protect this system.

- Authentication and authorization to control the number of valid users.
- Data encryption should be in the following manner: the user interface should encrypt any inputs by a valid user and then store it in the graph database. This is to make sure that information is kept safe and reliable.
- Each system should have a security measure that checks each input to update the database. If not validated in a proper format, it may result in cross site scripting problems or a buffer overrun.

The Graph Engine should have the ability to recover the information in case of a failure. Thus, we suggest a data inspecting operation within the Graph System to recover to a valid state when a system failure occurs. This should be present for each machine within the network to maintain data integrity.

Table 1 presents our suggestions for improvement of existing works.

Table 1: Suggestions for improvement

Modules	Reference Papers	Suggestions
Billion node partitioning	Graph exploration method [8, 11]	A combination of join operations and graph exploration methodology
Query processing	Using Trinity Specification Language [8]	Adding a spin lock mechanism for consistency of data
Online overlapping community search	Hybrid of online community search (OCS) and online community detection (OCD) [1]	Only the online community search technique
Distributed memory storage mechanism	Distributed Graph Engine for web scale RDF data [12]	Simple partitioning of data and storing on machines connected through efficient message and query processing
Recovery and Backup mechanism	-	Data inspecting operation
Security methods	-	Data encryption, authorization and authentication

Upon proper insight into the Graph Engine, we have come across various models and implementations of the same. There have been some observations we would like to place in through this paper. The Graph Engine is made for the biggest databases in this world such as the World Wide Web. Such data cannot be stored in a single machine because of two reasons. First, it would take a long time to access data independent of the algorithm used. Second, there will be a high chance of the machine getting corrupted as continuous query processing would take place. An alternative to such disadvantage would be to use multiple machines and distribute the memory over a cloud. In this method, the data would be partitioned into various machines and have a minimal chance of getting degraded.

Our observations on analysis of a billion node graph**,** matching sub graphs from the data**,** memory storage in a cloud**,** query processing and online search of overlapping community are as follows:

3.1 Analysis of billion node graphs

To store the billion node graph into multiple machines, we will first have to partition it. This requires complex algorithms. Before Graph Engine, there had been no accurate and standard way of partitioning billion node graphs. Converting such graphs in one format to another for different systems was an extremely tedious and expensive process. Also, there was no method of using sub graph matching or efficient query processing technique to split the graph.
Thus, Multi-Level Propagation technique has been used along with certain algorithms to implement partitioning of a graph on a distributed network. This is a Label Propagation technique. It runs as follows:

- Each vertex is assigned a unique label id by iteration.
- Upon updating, each vertex is assigned the prevalent label in the locality and this process keeps continuing.
- Process stops when no more changes occur.
- Vertices that have the same label name belong to a particular partition.

The advantage of this mechanism is that it does not contain any intermediate step, unlike the indexing scheme which makes it a lightweight code and results in a more feasible partitioning algorithm.

However, there are a few disadvantages that come along with this technique. It is not the most efficient algorithm used as there would be a number of progressive iterations could also result in an imbalance of memory storage on some machines, that is, some machine might contain a very large partition of the graph whereas the rest would contain a tiny portion of the graph. As a number of machines are involved in this process, there tends to be a communication lag between each step if continuous switching occurs [9].

3.2. Subgraph matching operation

After the partitioning is performed, we have to match the subgraph to any of the machines to search for any data. This has been done by the in-memory graph explorations on a memory cloud [8]. This method does not use any indexing scheme for the subgraph mechanism to save space and time. Though it results in loss of performance, it does become lesser expensive in terms of the join conditions used with the indexing scheme.
Looking into some details, these are major differences of the join and graph exploration method. In the join operation, a number of intermediate steps results in an increased memory space and further applies to major complexities and finally joins all intermediary steps to traverse the graph. On the other hand, the graph exploration method uses the label ids that were assigned to each vertex while partitioning the graph. It forms a link from one node to another to traverse through the graph and produces much lesser intermediary steps than the join operation [10].
Why do we use a subgraph matching operation in Graph Engine? The reason is that it uses computations that apply to the specified vertex, that is, vertex-centric computations [11]. This is a restrictive model and communication between vertices is within a fixed set and thus, it uses a predictable iteration to optimize this operation and hence, the Graph Engine overcomes the performance loss.

3.3. Memory Storage on a cloud

We know that the data is stored after partitioning on a number of machines. But what forms to memory cloud? Each partition is further divided using subgraph operation and a part of this is stored on a section of the machine's primary memory, that is, the RAM [8]. All these are connected and any message passed or query executed is transferred through and efficient communication system. These results in easier accessing of large data stored in bits and pieces.
Advantages of this system are:

- Since all the machines are interconnected, any update in data can be initiated easily from any machine.
- Lesser corruption of data and integrity is maintained
- Any data can be accessed by the user even if it is located on different machines. The Graph Engine user-interface displays the data as if it is present locally.

3.4. Query Processing

Message passing and query processing is a very essential part of a database model. An efficient model will lead to better communication and result in high performance of the system. Thus, for the Graph Engine, a high level language was built for data and network communication called the Trinity Specification Language (TSL) [8].

In a graph database, the communication pattern is dissimilar to other types of databases, thus, TSL makes this easier. Here are some of the following characteristics of TSL:

- It gives a structured interface between the internal and external models in the DBMS, that is, it gives a representation how nodes in a graph system are linked to a relational table.
- It gives an object-oriented data manipulation for the data present in the memory cloud.

TSL also reproduces fast network communication. As graph databases are large and it becomes tedious for user to pass different queries each time for synchronous and asynchronous protocols, TSL initializes an intuitive way of passing message programs for any graph computation.

Our observation is that there needs to be a spin lock mechanism for the same. Spin locks is a mechanism used for concurrency control, that is, to govern the ability of the number of processes to access the same key-value pair on the memory cloud. Therefore, a spin lock mechanism must guarantee that a particular key-value pair is locked to a fixed memory position in order to protect the data that the threads are accessing.

3.5. Online Search of Overlapping Community

In computing terminology, a community is a cluster of vertices well connected to each other. When we take a real life social network, it would consist of various inconsistencies which would make it hard to implement. Such is an example of the online search of overlapping community problem. Current algorithms have some drawbacks:

- They have predetermined criteria to search for a particular data on a graph. But in a real life application each node or vertex of a graph might a different characteristic.
- They cannot search on an evolving framework of data, that is, regular updating and continuously growing databases cannot be searched.
- They are not efficient and costly. [12]

Generally some database networks use the overlapping community detection system just to find out the number of overlapping communities in a network. Upon research we have listed out some of the drawbacks of this system:

- It is a tedious process to figure out all the overlapping communities on a large graph containing billions of nodes.
- It is hard to support dynamically evolving graph databases.

To overcome such drawbacks, researchers used the overlapping community search method. This takes an input as value of the vertex and outputs the overlapping communities containing the particular vertex. This provides a suitable outcome to the user.

Our observation is that this is a unique and efficient method of searching through a large database for a specific category, that is, an overlapping community. It is useful and extremely essential for a social networking database.

Conclusion

Microsoft's Graph Engine is a very vast topic to research on and we as a group have made is effort to cover as many modules as possible. We have listed advantages and disadvantages of the same and have developed a deep insight on how to manage and display such large amount of information. We have done a complete overview on certain modules such as partitioning of billion node graphs, subgraph matching operations, memory storage on a cloud, query processing and online community search. We have highlighted the gaps identified in the reference papers we used and some suggestions related to the security measures and recovery options. Overall, this paper gives an important insight to an upcoming topic in research, that is, distributed and parallel computing.

References

1. Cui, W., Xiao, Y., Wang, H., Lu, Y., & Wang, W. (2013, June). Online search of overlapping communities. In Proceedings of the 2013 ACM SIGMOD international conference on Management of data (pp. 277-288). ACM.
2. J. Baumes, M. Goldberg, and M. Magdon-ismail, (2008, April). "Effecient identification of overlapping communities," in In IEEE International Conference on Intelligence and Security Informatics (ISI, 2005, pp. 27–36.
3. J. Dean and S. Ghemawat. (2010, March). Mapreduce: Simplified data processing on large clusters. OSDI '04, pages 137–150.
4. Jin, R., Liu, L., Ding, B., & Wang, H. (2011). Distance-constraint reachability computation in uncertain graphs. Proceedings of the VLDB Endowment, 4(9), 551-562.
5. Jin, R., Ruan, N., Xiang, Y., & Wang, H. (2011). Path-tree: An efficient reachability indexing scheme for large directed graphs. ACM Transactions on Database Systems (TODS), 36(1), 7.
6. R. Bramandia, B. Choi, and W. K. Ng. Incremental maintenance of 2-hop labelling of large graphs. TKDE, 22(5):682–698, 2010.
7. Shao, B., Wang, H., & Li, Y. (2012). The trinity graph engine. Microsoft Research, 54.
8. Shao, B., Wang, H., & Li, Y. (2013, June). Trinity: A distributed graph engine on a memory cloud. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (pp. 505-516). ACM.
9. Shao, B., Wang, H., & Xiao, Y. (2012, May). Managing and mining large graphs: systems and implementations. In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (pp. 589-592). ACM.
10. Sun, Z., Wang, H., Wang, H., Shao, B., & Li, J. (2012). Efficient subgraph matching on billion node graphs. Proceedings of the VLDB Endowment, 5(9), 788-799.

11. Wang, L., Xiao, Y., Shao, B., & Wang, H. (2014, March). How to partition a billion-node graph. In Data Engineering (ICDE), 2014 IEEE 30th International Conference on (pp. 568-579). IEEE.
12. Zeng, K., Yang, J., Wang, H., Shao, B., & Wang, Z. (2013, February). A distributed graph engine for web scale RDF data. In *Proceedings of the VLDB Endowment* (Vol. 6, No. 4, pp. 265-276). VLDB Endowment.