Improving Query Response Time for Graph Data Using Materialization

Abdul Waheed[1], Dr. Saif ur Rahman[2]

[1,2]*Shaheed Zulfikar Ali Bhutto Institute of Science and Technology (SZABIST) Karachi, Pakistan*

[1]mudassar.iqb@gmail.com
[2]rafi.muhammad@gmail.com

Abstract - Graphs are used in many disciplines, from communication networks, biological, social networks including maths and other fields of science. This is the latest and most important field of computer science today. In this research, the authors have worked on the materialization to improve the query response of graph data. The large graph dataset have been divided into two categories; one contains the topological data and other contains the aggregate data and both are accessed via a PAM (Predicate Aggregate Materialization) engine which plays an intermediary role. PAM engine stores the query results and it checks whether the query is new or already processed every time the query appears. If it is found already processed than it just get the results which are materialized and if it finds a new query than it goes for the extraction of data from required datasets. After completion of process, PAM engine materialize the extracted data for reuse. The technique works and it reduces the processing time and improves response time.

Keywords - graph database, materialization, Neo4j, SQL Server, attributed data, topological data, PAM Engine.

I. INTRODUCTION

Graph processing and optimization is one of the most active areas of research today. Graphs are valuable because they give brief statistics in a way that is easy for every person to understand. Graphs are used all over the world for effective data representation purpose. It is computationally challenging to manage and analyse graphs to support effective decision making graph processing. It is time and resource consuming process. There are many algorithms, models and techniques which are developed for accessing and processing graphs and are still continued developing today.

In recent years, the area of graph has got wide attention and outstanding growth especially in the social applications like facebook, linkedin, twitter, instagram and foursquare. It has increased the usage of graphs. These networks are modelled as large graphs with vertices representing entities and edges as relationship between entities. These applications have got much popularity after the web 2.0. Numerous distributed graph processing systems have been proposed such as graph-lab, and power-graph. These systems are vertex-centric and follow a bulk synchronous parallel model (where vertices send messages to each other through their connections or ids). They are tailored and efficient for graph processing operators that require iterative graph traversal such as page rank, shortest path, bipartite matching and semi-clustering. However, it is costly to support graph computation. In fact, it will incur high overhead cost for message (carrying the attributes values) passing across the graph to find the vertices or edges with the same attribute values.

In this research, the authors have focused on the graph processing and to decrease the graph query response time and have determine a significant technique that optimizes the query response by materializing the intermediate results of graph data.

II. RELATED CONCEPTS

In this section, authors have briefly discussed about the techniques and technologies used in research for improving response time of graph data.

A. Materialization

Materialization is used with large databases. It stores or cashes the processed query results. Materialization is widely used technique mostly with data warehousing or online analytical processing to extract or select query data from large datasets in a minimum time. Materialization stores processed data that reused when required. The data warehouse contains huge datasets of organizational data and querying these large datasets is time consuming task.

One solution used in data warehouses to query these large datasets in a minimum time and to improve performance is to create summaries and indexes on data. Indexes are also useful for improving performance but both are combinely used. Summaries are the aggregates (sum, max, min and count) of key values. These pre-calculated aggregates are stored in a separate database or tables which are summarized data or materialized view. For example, we can create a summary data which contain the sums of salaries and expenses by department or years.

B. Materialization Cost

Materialization increases the response time but the benefit provided by materialization have a cost. As the materialization stores the processed results, when the primary data is changed then the materialized data need to be changed or updated. It is more useful in data warehousing or online analytical processing databases, where the processed results are changed, edited or updated on a scheduled time. As shown in Figure 1, every time the user made a query, it gets data from already processed aggregated joined and summarized database which is faster, reduces the processing and improves the performance. User query does not reach at the primary database. The worst case is, if the summarized data is not updated with primary database a cost of maintenance or updation is charged every time the primary database gets updated. In a data warehouse, a special technical person is employed which takes care of any problem and timely updates in both databases. More than one summarized or materialized views can be crated for achieving better results.

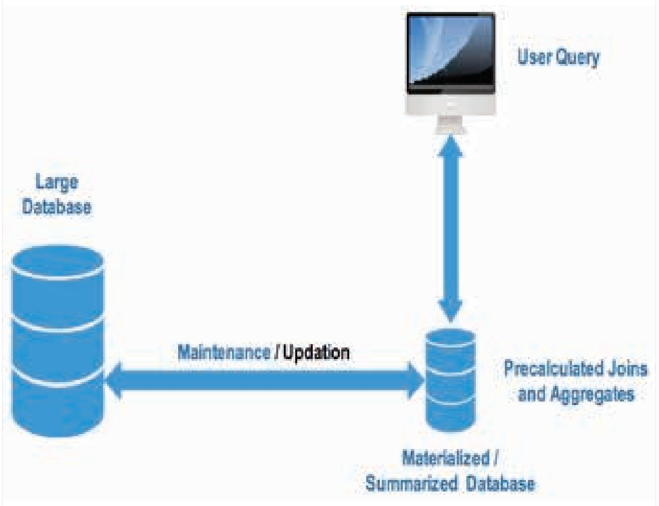

User Query
Large Database
Maintenance / Updation
Precalculated Joins and Aggregates
Materialized / Summarized Database

Fig. (1). Materialized Structure of Database

Considering the Figure 2 which is more suitable for online tractional processing (OLTP), but when it comes to data warehousing, this technique is not efficient because it increases the processing. Having a benefit of no maintenance or upgradation and every time the user gets updated data directly from the primary database. The main problem in this technique is if the dataset is very large than it takes a lot of time.

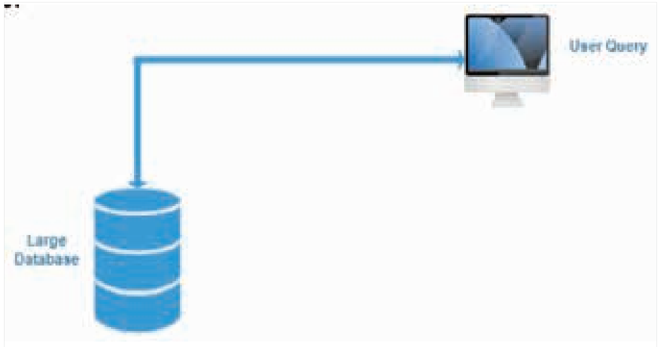
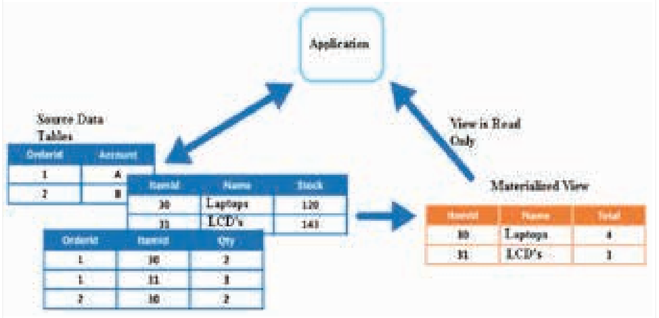


Fig. (2). Non Materialized Structure of Database

In Figure 3, the overall conclusion is shown. What would happens if when we go for the materialized data and what happens if we direct make a query data without using materialization? The Figure 3 shows that there are three tables on the left which are not materialized and if user get data, it pays the every time for aggregation while using the materialized view on the right the user pays the onetime cost and upgradation cost only if the primary data gets changes. We can perform the materialization on a single table by aggregating its values, or can perform materialization on multiple tables by applying joins or combining both using aggregates and joins. Whatever the technique used, it works and it saves time which can be understood from the figure given below:



OrderId	Amount
1	A
2	B

ItemId	Name	Stock
30	Laptops	120
31	LCD's	140

OrderId	ItemId	Qty
1	30	2
1	31	3
2	30	2

ItemId	Name	Total
30	Laptops	4
31	LCD's	3

Fig. (3). Materialization and Non Materialization Comparisons

III. PREDICATED AGGREGATE MATERIALIZATION (PAM)

In this section, the steps and flow of the PAM Engine have been discussed in detail with the help of figures and charts.

A. PAM Engine Characteristics

The characteristics of PAM engine is given below

- *PAM engine is designed using java.*
- *PAM engine is capable of getting data from multiple databases.*
- *PAM engine reduces the execution time of query.*
- *PAM engine can only used for data extraction.*
- *PAM engine can be used in windows and web applications.*

B. PAM Benefits

The PAM engine will be beneficial in many environments where the repeated queries are performed regularly such as online News websites/ search engines e,g. Considering the News scenario, thousands of peoples search for the same entities or objects, if data is fetched from databases every time, it will be more time consuming but if we use the PAM engine, then it will be more beneficial and queries can be responded much faster. Further, it may be considered that the entities and objects are stored in different databases as entities in graph database while the objects in relational database. PAM engine also combines the results accessed from both databases.

C. DBLP Schema

The DBLP data set is available in raw xml format publically. The DBLP schema contains the basic columns entities like author id publication id and publication type. The basic relationship is between publication id, author id and published by. The indexes are built on two columns publication key and author key. The Preprocessing details about the DBLP are given below.

D. Conversion from XML to Graph Database

The DBLP data set is available in raw xml format publically. The whole process of converting the DBLP data into graph data and relational data is given below as;

In order to obtain appropriate format of data, we first pre-processed the data. We produced two different formats of data; one for column oriented database which contained attributed information and other for graph oriented database which contained topological information between nodes.

DBLP is available as a raw xml format, so we used BaseX xml database to load xml database into our xml data. After loading xml database, we extracted the required information by using xquires then we loaded data into relational database. We generated the csv files from an xml database and uploaded the data from those csv files into relational database. It is easy to transform data into any form from relational model.

For instance, we required topological data as well as attributed data to store in two separate databases. Therefore, we have to pre-process data to achieve such a form of the data. In this regard, we first loaded data into a relational database (Microsoft SQL Server). We loaded data into three tables named DBLPaut, DBLPpub and DBLPnet. The DBLPaut table contains id (auto generated incremental) and name of author, DBLPpub table contains id (which is the id of specific publication in DBLP dataset), name of publication, year of publication and type of publication. Finally the table DBLPnet contains the relational information between DBLPaut and DBLPpub.

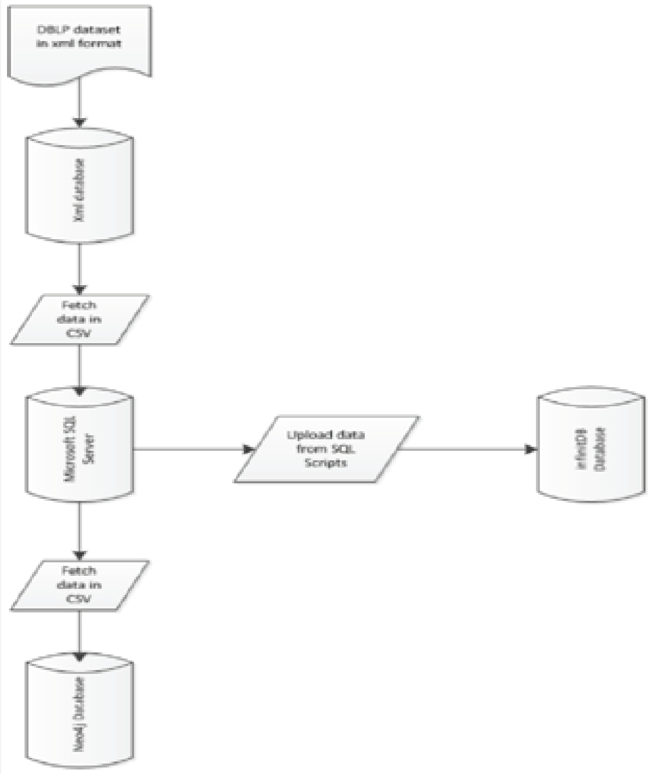


Fig. (4). Conversion from XML to Relational and Graph Databases

Finally, we created two separate instances for Neo4j database; one has all the data including attributed and topological information and other one only contains the topological information as describes in figure 4. Once we got the data loaded into a relational database, for first phase we store all the attributed and topological data into Neo4j and second phase separate the topological data and store it in graph oriented database. We extracted topological data and load it into Neo4j using csv files, so that we upload this data into graph database using cypher query [1].

E. Divide and Conquer

Dividing a large problem into small sub problems become easier to solve. The same approach we have used here in order to process large dataset of DBLP. To process a large graph dataset requires lots of time, storage and processing power, but to divide the large datasets into small datasets may not minimize the storage requirements but definitely have improved the processing time and have required less processing power.

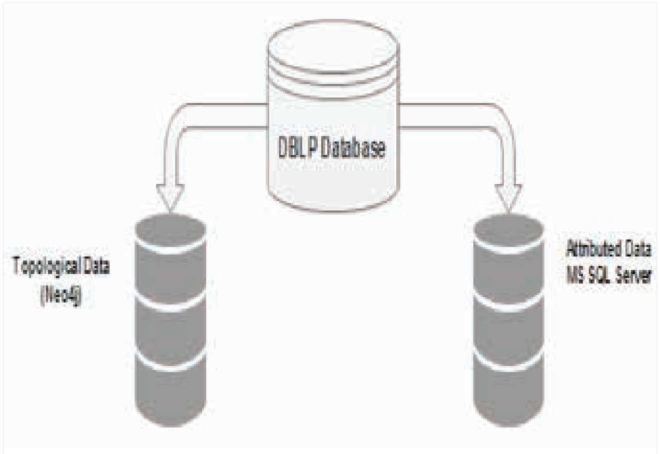


Fig. (5). Non Database Division

Figure 5 shows that we have processed and divided a large dataset of graph into two categories. We have divided the graph data by preprocessing the aggregate values and separate the topological values. The large DBLP datasets is processed and first part is separated which contain the topological information such as (Ids, counts). We can say it contains the aggregated data or summarized data which has benefited us in further processing when have queried this data. The second part which contains the attributed data such as (Ids, Names) is separated and converted to SQL server.

F. Intermediate Results Materialization

We have materialized the intermediate results of query and reuse it when the same query appears again. This has reduced the processing and improves the query response time. As a sample, we have used a DBLP dataset as read-only dataset of graph. DBLP is a standard dataset contains the information of publications and authors we have converted it into graph data in Neo4j and divide it into two parts as discussed above; one is topological data remain in Neo4j and other is attributed data which is in MS SQL Server and write an intermediary program in Java to be used in PAM engine which gets input query and data from both databases as given in the Figure 6. The PAM engine is consisted on three steps which are discussed in detail as below; when it is executed after getting query input, first part of the PAM engine gets the attributed data based on where clause from the MS SQL Server by executing SQL queries, once the data based on the where clause is extracted from MS SQL Server then PAM engine materialize the query and results in a separate table in SQL Server the query with parameters is stored separately in header table and the results are stored in child table with header query reference, which benefits when we go for the selection next time. Every time query appears it checks in header table whether the query is already executed or a new query is appears for execution.

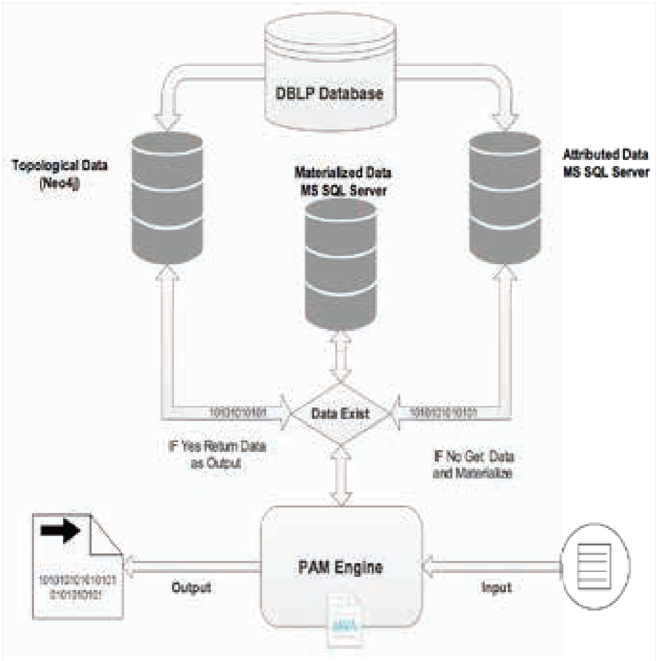


Fig. (6). Predicate Aggregate Materialization Engine Working Diagram

If the query is new than it process query and after processing it stores the query and results, otherwise if the same query appears again, it will not go for the further processing, it only gets the results from child table reference to header table Ids, which are already stored or materialized in header table in database. If query does not contain the where clause than it will directly go on the next step of the PAM engine which gets the data from the Neo4j using cypher query as shown in Figure 7.

The Next step of PAM engine is to get the topological data from Neo4j using cypher query. The Neo4j database contain only topological data which means no Ids, so the Ids are passed as extracted in first step based on where clause data (means the attributed data), depends on if where clause exists in query. After the successful cypher query execution, the results are stored in a CSV file as there may be millions or billions of records based on query so all the data can't be cashed so a CSV file will contains all the extracted data from Neo4j. After that, the same process is repeated here of materi-alization. It stores the query in header table and results in a child table with header reference. In this step, the results are in CSV file. So it is imported to SQL Server by using bulk insertion method in a separate table and each time a query appears, it checks in the materialized header table, if data exist it escape the query execution and directly retrieves the results from child table by using the header query reference. If query does not exist than it will be processed as given above and it will go for the next step. The complete flow chart is given below in Figure 7. The third and final step of the PAM engine is to combine both query results of first step and second step which both are now in SQL server format using join query of SQL server we can easily extract data. After the successful extraction of results, it gives the required output and terminates the PAM engine.

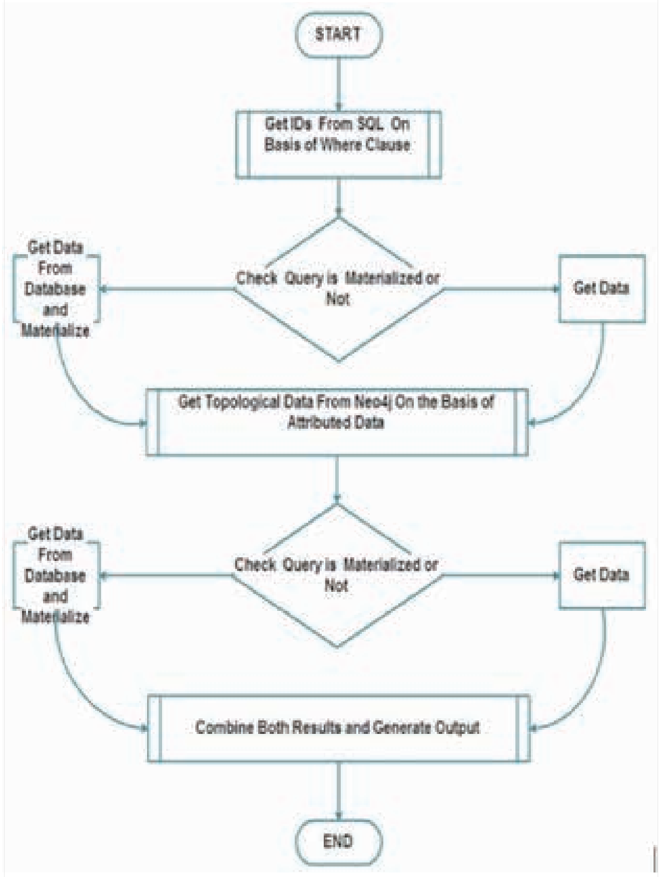

START
Get IDs From SQL On Basis of Where Clause
Check Query is Materialized or Not
Get Data From Database and Materialize
Get Data
Get Topological Data From Neo4j On the Basis of Attributed Data
Check Query is Materialized or Not
Get Data From Database and Materialize
Get Data
Combine Both Results and Generate Output
END

Fig. (7). PAM Engine Execution Flow Chart

G. Softwares Required

Following software's are required for system implementation.

- Neo4j
- SQL Server
- Operating System
- Java(TM) SE Runtime Environment

IV. RESULTS

These below queries are sample queries. The Query1 is written in cypher language. The query executed on a Neo4j database. The author Florian Holzschuher proved experimentally that the Neo4j is best when the database gets larger. Neo4j has shown nearly a constant performance when scaling from 2,000 to 10,000 people [2].

The following queries are model queries taken on the basis related to database. The DBLP database contains the publication data, so we have selected the queries which generate all the publication count by author and other which count the publication type articles of year 2014.

Its division into two parts and the comparison of time taken for processing is given below. The query has two filters one is year and second is publication types. The user wants to return only the author Id and publication count order by author Id.

The Query 2 is simple having no filter it only returns the author Id, year of publication and total publications order by Author Id, year of Publication, and Type of publication

Query 1:

```
MATCH (p:`publication`)-[:`publishedby`]-> (a:`author`)
WHERE p.year = "2014" and p.type = "article"
RETURN a.Id as author, count(*) as pcount
order by a.Id
```

Query 2:

```
MATCH (p:`publication`)-[:`publishedby`]->(a:`author`)
RETURN a.Id as author, p.year as year, p.type as type,
count (*) as pcount
order by a.Id, p.year, p.type
```

The below given Table 1 and Table 2 shows the comparison between the direct query and using PAM Engine with number of fetched records and time taken by each query. The time is given in minutes and seconds format the difference column in the right showing the time saved while using the materialized approach.

Table 1. Comparison results by Direct Query and by using PAM Engine

	Direct Query Results From Graph Database		**First Time Using PAM Engine**		**Difference**
Queries	**Records Returned**	**Time in MM:SS.0**	**Records Returned**	**Time in MM:SS.0**	**Time in MM:SS.0**
Query 1	205402	10:50.2	205400	07:50.2	**03:00.0**
Query 1	205402	11:32.3	205400	08:32.3	**03:00.0**
Query 1	205402	09:15.2	205400	08:15.2	**01:00.0**

Table 2. Comparison results by Direct Query and by using PAM Engine mparison results by Direct Query and by using PAM Engine

	Direct Query Results From Graph Database		First Time Using PAM Engine		Differ-ence
Queries	**Records Returned**	**Time in MM:SS.0**	**Records Returned**	**Time in MM:SS.0**	**Time in MM:SS.0**
Query 2	4893292	20:32.2	4893290	11:16.7	**09:15.5**
Query 2	4893292	21:12.9	4893290	12:37.4	**08:35.5**
Query 2	4893292	21:49.7	4893290	11:20.5	**10:29.2**

The given Table 3 shows the comparison between First materialized query execution time and if the same query executed second time. The difference column in the right shows the time saved while using the reusability materialized approach.

The Figure 8 shows the execution time of queries when we have performed the experiment directly from DBLP dataset or using materialized data and reusing the materialized data. The Blue color shows the time which was taken when have executed the cypher query directly on Neo4j DBLP database, the chart shows it is the most time taking query.

Table 3. Comparison results by Direct Query and by using PAM Engine

	First Time Using PAM Engine		If Query Appears Again in PAM Engine		Differ-ence
Queries	**Records Returned**	**Time in MM:SS.0**	**Records Returned**	**Time in MM:SS.0**	**Time in MM:SS.0**
Query 1	205400	07:50.2	205400	00:35.9	**07:14.3**
Query 1	205400	08:32.3	205400	00:34.8	**07:57.5**
Query 1	205400	08:15.2	205400	00:38.3	**07:36.9**
Query 2	4893290	11:16.7	4893290	01:15.9	**10:00.8**
Query 2	4893290	12:37.4	4893290	01:14.8	**11:22.5**
Query 2	4893290	11:20.5	4893290	01:18.3	**10:02.2**

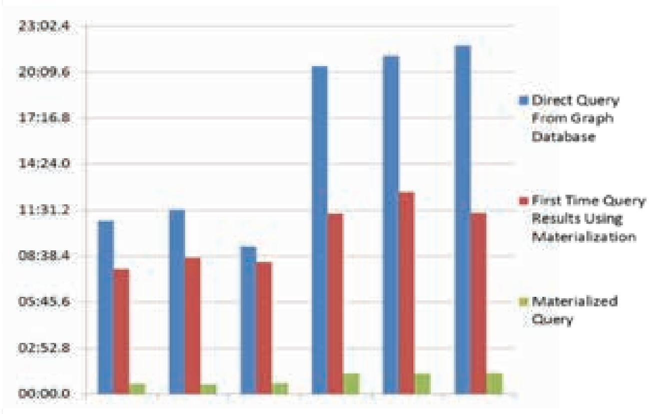


Fig. (8). Query Processing Time Comparison Chart

Red color shows the time which was taken when we have first executed the query in a PAM engine and the query is materialized the difference between the direct query and using materialized is little here but in this step the materialization cost is also included which is onetime cost. The next Green color shows the big difference when we have executed the same query second time in materialized environment. The difference is huge. It improves the response time and decreases the processing time up to 60% which was the goal of this study which have been achieved using the materialization of intermediate results.

V. RELATED WORK

Materialization idea has always works with some minimum cost of updation or modification with almost all databases. Kamel et al. worked on semi materialization to execute repeated quires in a minimum time with efficiency on relational database management systems. The technique is to store repeated or most frequent data that shows the intermediate level of query execution. The most frequent data is used to efficiently build the output of query; thus, is an easy way to maintain results. Then the author compares his results with the other two techniques commonly used to materialize the repeated execution of queries which are full materialization and query modification. The authors has proved systematically and shown by simulation. The results have shown that the technique of semi materialization has improved query response not only for simple queries but also for complex queries, views and joins, under the certain working circumstances [3].

Semi materialization deals with relational databases while in our research we have used the graph databases. We have pre-process the databases and divide it in two parts as topological and aggregate values.

To query databases, there are two important algorithms discussed by Shekhar et al. The algorithms are Materialization and hierarchical routing. The author says these algorithms are very useful in the applications like shortest path, intelligence transportation system (ITS) and travelling sales man problem (TSP). As the graph database size is growing day by day, the materialization technique and hierarchical routing algorithm pre computer and stores the results of shortest path view these results can be reused any time which increases the performance of processing large graph queries [4].

Graphs, network, and Web mining has got much attention in recent years. A bundle of literature has been published in the area of graph processing web and social network analysis. Emerging applications face the need to store and query data that are naturally described as graphs. Building Business intelligence (BI) solution for graph data is a difficult task.

Relational databases are frequently criticized for being unsuitable for managing graph data. Graph databases are gaining popularity but they have not yet reached the same maturity level with relational systems. [5]

Materialization of data worked as a cache in main memory. A materialized view delivers fast access to user data. The materialized view need to be updated when primary data modified or changed. It is just like a cache memory. Cache becomes dirty when the data it copies is updated or modified. A materialized view becomes dirty every time the primary base relations are changed or modified. Sometimes it is costly to maintain a materialized view and get it updated with the primary relational data but in most of the cases it is cheaper and beneficial especially when the data is repeatedly called. A materialized view should not be completely changed when primary relation are updated but there should be in a mechanism which recognize and changes only those parts of materialized view which are changed but not completely materialized view [6].

Abadi et al. study the use of early and late materialization in the C-Store Databases. They focus on standard warehouse-style queries: read-only workloads, with selections, aggregations, and joins. The authors run experiments to determine when one approach dominates the other and develop an analytical model that can be used; for example, in a query optimizer to select a materialization strategy. The results show that on some workloads, late materialization can be an order of magnitude faster than early-materialization while on other workloads; early-materialization outperforms late-materialization by an order of magnitude [7].

Zagorac et al. produced a materialization technique for structured web data such as Amazon and to improve the web search which helps to answer multi domain queries by accessing the heterogeneous web based data [8]. There are so many other techniques for materialization like view materialization author described an incremental maintenance algorithm for views over semi-structured, or schema less, data. The algorithm identifies the needed view changes, based on the information available from the view specification, the update operation, the database state after the update, and some auxiliary data structures that are generated when populating the view [9].

Zhao et al. have communicated the difficulty of supporting warehousing and Online Analytical Processing databases for large multidimensional networks. The authors have exactly calculated the problem and proposed a new data warehousing model for the problem. The author proposed a Graph cube model which was specially developed to improve the aggregation of large graph networks and multi- dimensional attributes [10]. Chen et al. inspects the probability to apply multi-dimensional processing on networked data, and designed a Graph Online Analytical Processing framework, which is categorized into two main classes which are topological Online Analytical Processing and informational Online Analytical Processing [11]. Aksu et al. worked on the dense graph and identified sub graphs of high structure using the k core matrix. The author said that the graph importance is increased day by day and the social network graphs content gets increased but the topologies changes dynamically, the author challenged and proposed that not to just materialize the dense graph but also maintain them and keep updated continuously. The authors said that the graph data is now increased and cannot processed on a single server machine and on its limited memory. The author proposed a distributed algorithm for dense graph view. The algorithm store and process graph on a horizontally scale [12].

VI. CONCLUSION AND FUTURE WORK

By using the materialization of intermediate results for graph technique, we have improved the query response and minimize the processing time and cost for graph data. We have also used a division technique and divided a large graph dataset into two parts and apply some processing while dividing. We have used a PAM engine which gets data from both databases and gives the required output. The materialization of intermediate results has worked as it improved up to 60% time on reusability.

There is always space for improvement, as in our working model we have stored the whole query, so when the same query appears again with the same parameters than we have got the improved response time but there should be a mechanism which partially checks if any of parameter is matched then it will not go for that parameter and get only the remaining parameters from primary database and then store all in a materialized form.

ACKNOWLEDGEMENTS

I would like to express my special thanks of gratitude to my supervisor Dr. Saif-ur-Rahman as well as SZABIST Karachi for giving me this opportunity to do research on the topic of Improving Query Response Time for Graph Data Using Materialization which helped me in doing a lot of research and I learnt new things. I am really thankful.

REFERENCES

[1] M. Adnan "Hybrid approach for storing multi-attributed graph data," submitted for publication. **

[2] F. Holzschuher and R. Peinl, “Performance of graph query languages: comparison of cypher, gremlin and native access in Neo4j” In *Proceedings of the Joint EDBT/ICDT Workshops (EDBT '13)*, 2013, pp.195-204.

[3] M. N. Kamel and S. B. Davidson, “Semi-Materialization: A Technique For Optimizing requently Executed Queries,” *Data & Knowledge Engineering*, vol. 6, no. 2, pp. 101-123, 1991.

[4] S. Shekhar, A. Fetterer and B. Goyal, “Materialization Trade-Offs In Hierarchical Shortest Path Algorithms,” In *Advances in Spatial Databases* (Lecture Notes in Computer Science), M. Scholl and A. Voisard, Eds. Berlin Heidelberg, Springer, 2005, pp. 94-111.

[5] D. Bleco and Y. Kotidis, “Graph Analytics on Massive Collections of Small Graphs” In *Proceedings of International Conference on Extending Database Technology (EDBT),* 2014, pp. 523-534.

[6] A. Gupta and I. S. Mumick, “Maintenance of Materialized Views: Problems, Techniques, and Applications,” In *Materialized views*, Cambridge, MA, MIT Press, 1999, pp. 145-157.

[7] D. J. Abadi, D. S. Myers, D. J. DeWitt, and S. R. Madden, “Materialization Strategies In A Column-Oriented Dbms,” In *Proceedings of IEEE 23rd International Conference on Data Engineering*, 2007, pp. 466 – 475..

[8] S. Zagorac and R. Pears, “Web Materialization Formulation: Modelling Feasible Solutions” in *Database and Expert Systems Applications* (Lecture Notes in Computer Science), H. Decker, L. Lhotska, S. link, M. Spies and R. R. Wagner, Eds. Switzerland, Springer Int. Pub., 2014, pp. 366-374.

[9] S. Abitebouly, J. McHugh, M. Rys, V. Vassalos and J. L. Wiener, “Incremental Maintenance for Materialized Views over Semi Structured Data” In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB ‘98)*, 1998, pp. 38-49.

[10] P. Zhao, X. Li, D. Xin and J. Han, “Graph Cube: On Warehousing and OLAP Multidimensional Networks,” In *Proceedings of the 2011 ACM International Conference on Management of data (SIGMOD '11),* 2011, pp. 853-864.

[11] C. Chen, X. Yan, F. Zhu, J. Han and P. S. Yu, “Graph OLAP: A Multidimensional Framework for Graph Data Analysis,” *Knowledge and Information Systems*, vol. 21, no. 1, pp. 41-63, 2009.

[12] H. Aksu, M. Canim, Y-C. Chang, I. Korpeoglu and O. Ulusoy, “Distributed k - Core View Materialization and Maintenance for large Dynamic Graphs,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 10, pp. 2439-2452, 2014.