ISSN 2320 – 2602
International Journal of Advances in Computer Science and Technology (IJACST), Vol. 3 No.2, Pages : 12 – 18 (2014)
Special Issue of ICCSIE 2014 - Held during February 16, 2014,Bangalore, India

An Efficient Index based Query handling model for Neo4j

Anita Brigit Mathew[1], S. D. Madhu Kumar[2]
[1]NIT Calicut, India, anita_brigit@rediffmail.com
[2]NIT Calicut, India, madhu@nitc.ac.in

Abstract : *Relational Database Management Systems (RDMBS) are a predominant technology used for storing and retrieving structured data in web and business applications since 1980. However, relational databases have started losing its importance due to strict schema reliance and costly infrastructure. It has conjointly led to the problem in upgrade relationships between objects. Another important issue of failure is the brobdingnagian growth of BigData. A new database model called NoSQL, plays a vital role in BigData analytics. In this paper one of the NoSQL graph database's particularly Neo4j have been explored. Neo4j, is a reliable graph database which can be scalable to any application. It handles billions of nodes and relationships in a connected structure. Querying in Neo4j is administrated through graph traversals and aggregate operations. Another technique called Multidimenstional indexing was incorporated to speed up query process. Multidimenstional indexing search and insert algorithms have been analyzed and a new Skip list indexing is steered. Multiple skip lists with replication factor of three was incorporated that gave a Skip graph like structure. Analysis have been made and found that Skip list resulted in better performance compared to Multidimenstional indexing.*

Key words : Neo4j graph, Multidimensional Indexing, Skip List.

INTRODUCTION

Background

Graph database is the most shared data structures which have the potential to graciously represent data [1]. Neo4j is an open source commercially supported graph database implemented in Java. It has been designed to be a reliable database which offer a disk based, completely optimized for storing graph structures with high performance and scalability. It stores information in the form of nodes in a graph where each node represents records of data. Each of these records have properties. A node can have single property later which can grow to millions. Nodes share these properties with other nodes. At certain point it needs to distribute the information into multiple nodes that is organized with explicit relationships. These relationships can cluster the nodes to a richly interconnected structure of a table. Each of these relationships could also have properties. All the properties and relationships of each node should be clearly listed before cluster process. This clustering process groups nodes into sets, where each set is assigned the most appropriate label. Labels help during graph traversal for query processing.

Motivation

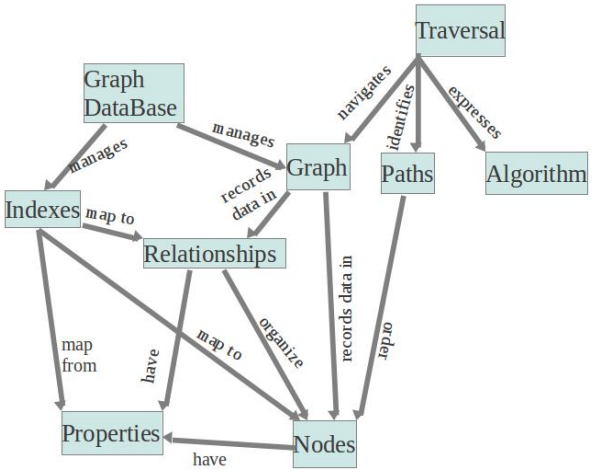


Fig. 1: Neo4j Graph Database Structure

Neo4j graph database which is illustrated in Figure 1 has the flexibility to manage graphs and its connected index. Query process identifies the order of nodes it has to identify for the query response. It navigates the trail of traversal in the graph. Building an index could boost performance potency during query traversal. An Index helps to map from node relationships to node properties or vice versa. Index lookup the simplest path to be navigated for quick query response. The database is queried through Cypher Query Language. Cypher query traversal navigates from starting node to related nodes finding quick responses with the assistance of multidimenstional index.

ISSN 2320 – 2602
International Journal of Advances in Computer Science and Technology (IJACST), Vol. 3 No.2, Pages : 12 – 18 (2014)
Special Issue of ICCSIE 2014 - Held during February 16, 2014,Bangalore, India

Multidimenstional index helps to traverse nodes in an exceedingly consecutive manner until the most accurate node was obtained for the query input. It gave a structured procedure for the unstructured set of nodes. It was found to provide a better response compared to random access procedure on unstructured data. However with associate explosive growth of social data, it has become more demanding to access data at a faster phase. Extreme growth of billions of nodes in social networking, web graphs and knowledge based networks would thus make Multidimenstional index query response performance degradation. Hence it was very much required to develop a much more faster query response index structure. Therefore Skiplist indexing in Skip graph was considered with a replication factor of three.

Neo4j is absolutely powerful when you need to solve issues that demand recurrent probing throughout the network [2]. Compared to traditional databases, during search it does not repeatedly evoke for row column information. In distinction the traditional databases require separate query for each step of search. Here Neo4j graph database is looked into with the available lookup tables of relationships and properties. Study on the Multidimenstional index structure model[3] have been made, its drawbacks and limitations are clearly listed in Section 3. Search and insert algorithms, implementation details and snapshots taken of the present model would be clearly illustrated in Section 4 of this paper. A new Neo4j graph database model with skip list index structure for search and insert operations is recommended with specific illustration algorithms in Section 5. The new model permits faster performance throughout query process. Section 6 talks concerning how analysis is made based on the comparison between search and insert algorithms modeled. Finally Section 7 discuss about the conclusion. Some preliminary ideas for the study of Neo4j graph database is discussed in Section 2.

PRELIMINARY CONCEPTS

Definition 1. Graph A graph G with n vertices and m edges consists of a vertex set V (G) = {v1 , v2 . . . vn } and an edge set E(G) = {e1 , e2 . . . em }, where each edge consists of two distinct vertices called the end points of the edge [4].

Definition 2. Subgraph A subgraph of a graph G is a graph H such that V (H) ⊆ V (G) and E(H) ⊆ E(G) [4].

Definition 3. Path Given a Graph G[V, E], two vertices v1, v2 ∈ V (G) are adjacent when (v1, v2) ∈ E(G). A path in an undirected graph is a sequence of vertices P =(v1 , v2 , . . . , vn) ∈ V n for some positive integer n such that vi is adjacent to vi+1 for 1 ≤ i ≤ n and vi = vj whenever i = j [4].

Neo4j MULTIDIMENSTIONAL INDEX STRUCTURE

Neo4j exclusively deals with optimized graph structure data, consisting of nodes with meta data information and edge links between nodes [2]. This graph model consists of multidimenstional index technique for query processing. This index framework interacts with the back end storage service that has the base tables stored and provides an economical data retrieval interface for faster query processing applications. The main tables of Neo4j constitute specifically Node, Relation and Property as illustrated in Figure 2. Nodes are entities, where each entity has its own property and specify relation with other nodes. Each node is labeled, this label helps to cluster nodes by role of every node and property. Relationships connect entities and structure the node domain model. Properties of each node clearly establish the attributes related to that node and the meta data the node process.

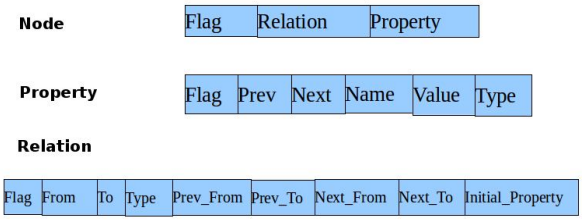


Fig. 2: Table format of Neo4j

Pros of Neo4j are:

- Neo4j database is a highly agile and blazing fast in performance compared to other graph data models of NoSQL databases.
- Developers phrase Neo4j as relationship based, disk based, fully transactional and documentation persistent storage engine [5].
- This powerful data model is good in cases involving deep searching through the networks with the help of multidimenstional indexing.
- It is the most popular graph database for networked operations and runs faster than relational databases.

Cons of Neo4j are:

- Searching for a particular data node with a particular attribute is difficult because it has to traverse through multiple tables.
- Implementing a project requires fore thought, good design work and better planning.

ISSN 2320 – 2602
International Journal of Advances in Computer Science and Technology (IJACST), Vol. 3 No.2, Pages : 12 – 18 (2014)
Special Issue of ICCSIE 2014 - Held during February 16, 2014,Bangalore, India

IMPLEMENTATION DETAILS OF Neo4J MULTIDIMENSTIONAL INDEX

To implement we first collected data from a near by Multi-speciality hospital. The data was taken for a period of time starting from 1st May 2001 to 29th November 2013. It was collected for the department of paediatrics for age group starting from new born infants till 16 years of both male and female sex. Other departments like neonatology, pediatric surgery, microbiology and pathology was also referred and all related data to pediatrics was collected along with details of doctors referred. Nodes represented the associated properties like patient records, departments referred and doctors consulted. Edges represented the relationships and the corresponding properties. Java language is used to program and structure the nodes, properties and relation in the form of tables. Neo4j database uses a key-value store of each node, keys indicate hospital number for each patient and value indicate the name of the patient. This unique key is considered as a primary key to relate the property table and relation table of the corresponding node.

1	CREATE (nname : key, mapp, mapr)	creation of new pa-tient node
2	CREATE (n) − [r : BELON GS T O]− > (m)	patient n belongs to doctor m, r relation
3	START n = node(key)	Start from node with specified key value.
4	START n = node : nodeIndexName (key =value)	Query the index with node auto index (Multidimenstional Index).
5	MATCH (n) − − > (m)	Maps the patient n with doctor m
6	ORDER BY n.property	Sort the result
7	RETURN DISTINCT n	Return unique rows

Table 1: Some Cypher Commands Used

Cypher Query Language

The node or nodes to be fetched is queried using Cypher query language version 1.9 in Neo4j. Cypher language starts with START command and terminates when a RETURN command is fetched. START binds terms using sample lookup directly using known key and index property. Cypher uses MATCH command to find replicated nodes. Lookup helps to traverse nodes match relationships specified on the edge label of the graph. Table 1 lists some of the frequently used commands to built the Neo4j medical record database for pediatrics department.

The pediatrics record database is created using the following steps,

1. Create new graph database service with Embedded Graph Database command
2. Begin new Transaction Service
3. Create node list N
4. For each node v_i where i in 1 to N do
5. Assign unique key to v_ i.flag
6. Set Property list P
7. Insert each v_i flag and P list to Index Table
8. Set Relation list R
9. Create v_i using createRelationship()
10. Insert each v_i flag and R list to Index Table
11. Insert v_i into Embedded Graph Database
12. Transaction finish

Consider a query: “pediatric patients posted for surgery on 2nd November 2013”. Such a query is likely to traverse the record network graph starting from the key associated with 2nd November 2013 node. If one node found then next node (data item) is being fetched. This is a spatially continuous process till all relationships, properties related to the particular node obtained. This semantic locality exposed by graph traversal is based on breadth first search technique used in Neo4j [6]. Inorder to enhance this process Neo4j has suggested simple key-value stores, where an order function on a key serves as a hint to arrange data in an exceedingly table or index. This table or index arrangement of nodes suggested by Neo4j is called Multidimenstional Index. Algorithm for Multidimenstional insert and search index is elaborated within the next subsection.

Insert and Search in Neo4j Multidimenstional Index

Algorithm 1: Insert in Multidimenstional index strcuture

Input: Insert value to GraphDatabase Factory

1. Is GraphDatabase Factory empty
2. then Set GraphDatabase configuration settings as node_key_index, node_property, node_property_value for each node and set relation key_index, rel_prop, rel_prop value for each relationship.
3. Initially assign node_property, node _index_value, node_property_value, rel_prop and rel_prop_value as □.
4. Set node_auto_index and relation_auto_index as true.
5. if Transaction graphDB = empty
6. then node = graphDB_createnode.

ISSN 2320 – 2602
International Journal of Advances in Computer Science and Technology (IJACST), Vol. 3 No.2, Pages : 12 – 18 (2014)
Special Issue of ICCSIE 2014 - Held during February 16, 2014,Bangalore, India

7. node.setProperty (node_property, node_property_ value)
8. while(index_counter! = □)
9. Set node, node_property, node_property_value index structure.
10. while(node _index <= index_ size)
11. then Set node, node_index_value, relation_key_ index, rel_prop, rel_prop_value index structure.
12. Increement node_ index
13. end while
14. Increement index_counter
15. end while
16. else Increement Transaction graphDB counter
17. return index, counter

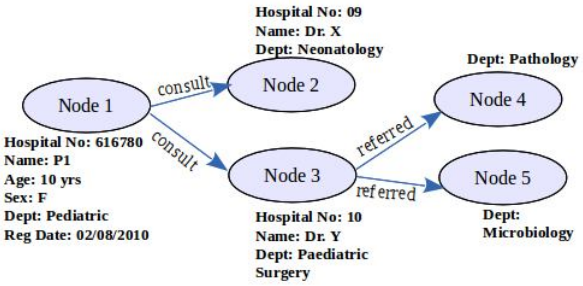
Algorithm 1 illustrates how insert operations are carried out in Neo4j graph database. First it talks concerning insert procedure. In the insert phase it checks for whether the GraphDatabaseFactory is empty or not, if the answer results in a true statement then it will set the node, property and relation tables and link with distinctive flag which is the key to retrieve all the tables. Initially node is created using create command. Node is then assigned properties it process with other nodes related to it. Note that the property of node have to be clearly justified with correct semantics of property structure. The node V_i can be related to other nodes of the node list V. Hence relationships of node should be framed using relationships syntax table structure. One node can have more than one relation. Each relation can have different properties, therefore each of these properties should also be indexed in table.

If the GraphDatabaseFactory is not empty, then increment the counter till reach the end of N node list and insert the new node by setting the above menstioned scenario of node, property and relationship. This new node can share properties of existing nodes then map function can be used by MATCH function in Neo4j and an share the property flag value. Similarly this can be carried out for relations also.

Algorithm 2: Search in Multidimenstional Index Structure
Input: Search node_value
1. while(node_ value! = node)
2. while(autoNode_ index <= index counter)
3. if autoNode _index=node _value
4. then get node auto_index and node_ property
5. get relation_auto_index and relation_property
6. return node_auto_index, relation_auto_index
7. else next autoNode_ index
8. end while
9. end while
10. check if node_value = node
11. then goto step5

Algorithm 2 talks in case of search operation the node index is checked with the inputed node value. Once the required node is found, then the corresponding node's property and relation index is checked through and the required data is retrieved using Cypher Query Language.

START n = node f lag
MATCH n − [: BELON G T O] − − > dr X
WITH n, count(dr X) as patientcount
RETURN n, patientcount



Example in Figure 3 illustrates the relationship between patient P1 , doctor's X and Y and department of pathology and microbiology. Here the number of nodes N= 5. Node1 is a patient P1 with key= 616780, this node belongs to (have relationship consults) with two nodes namely node2(doctor X) and node3 (doctor Y). Node3 also have relationship (referred to) with node4(pathology) and node5(microbiology). Illustration details at each node indicates the property associated with that node. Edges are labeled with relationships between nodes.

Implementation Snapshots

Figure 4 shows how index is constructed in Neo4j 2.0.0. First node index is created. Then as second process for each node f lag key, property and relationship index is built. Property and relationship tables are connected with unique flag key acting as primary key. Figure 5 gives a diagramatic network view in Neo4j 2.0.0.

Here each node v_i where i range from 1 to V is labelled
with integer attribute. Each edge is labelled with string attribute. When node v_i created immediately the index for
property and relationships is asked for and set at the console
panel. Neoclipse panel helps to update the node relations
and properties as the network size increases with new patient
members, doctors and staff.

ISSN 2320 – 2602
International Journal of Advances in Computer Science and Technology (IJACST), Vol. 3 No.2, Pages : 12 – 18 (2014)
Special Issue of ICCSIE 2014 - Held during February 16, 2014,Bangalore, India

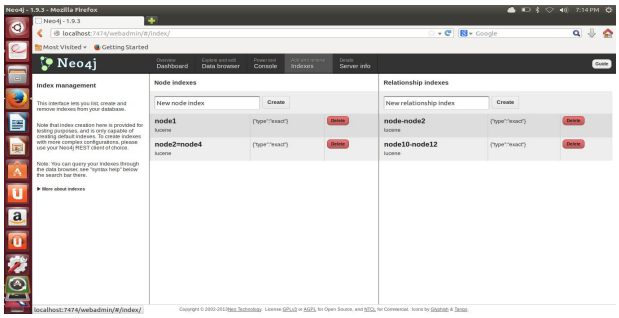


Fig. 4: Index Creation in Neo4j

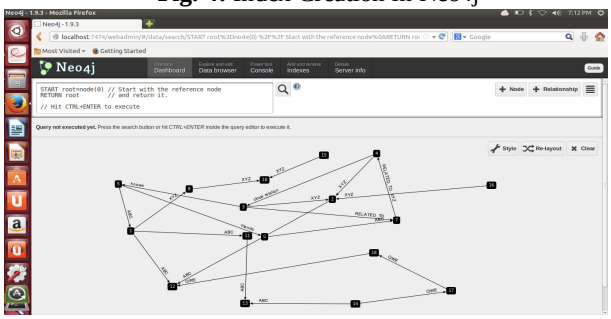


Fig. 5: Graph Network Obtained in Neo4j 2.0.0.

LIMITATIONS OF MULTIDIMENSTIONAL FRAMEWORK

Graph databases, such as the industry leading Neo4j, provide extremely quick access to the types of complex data found in social and science-medicine network systems. Even though they have the ability to push the limits of memory usage in order to meet the performance needs of customers. There are few limitations in its structured frame work. With the use of multidimenstional index structure the performance of Neo4j in query processing is not striking. Some of the issues undergone are illustrated below.

1. When a query is posted it has to get the node and f lag key.
2. With the f lag key it has to search the property and relationship table index to get details.
3. Traversal from one table to another takes more time complexity.

If there was an efficient methodology to retrieve query with better time complexity than multiple tables was appreciable. Hence Skip lists was suggested and a skip graph was modelled. Analysis was made and it was found that Skip lists performed better responses compared to multidimenstional index structure.

SKIP LIST INDEX

Skip List is an interesting data structure that provide the full functionality of a balanced tree where resources are stored in separate nodes [7]. These are designed to be used in search operations in peer-to-peer networks. They also provide the power to perform queries based on key ordering. This organization makes random selections in arranging the entries in such a way that search and update times are only O(1ogn) on the average.

Algorithm 3 : Search for node with searchKey
Input: Given query to search for record name(node)
1. Start from the initial header pointer
2. if (v_i.key = searchKey)
3. then return (v_i, lookup)
4. Assign lookup to join pointer
5. if (v_i.key < searchKey)
6. then while level ≥ 0
7. do
8. if ((v_i.neighbor[R][level].key ≤ searchKey)
9. then return (v_i, lookup)
10. Assign lookup to v.neighbor[R][level]
11. break
12. else level ← level − 1
13. else while level ≥ 0
14. do
15. if ((v_i.neighbor[L][level]).key ≥ searchKey)
16. then return (v_i, lookup)
17. Assign lookup to v_i.neighbor[L][level]
18. break
19. else level ← level − 1
20. if (level < 0)
21. then return (v i, lookup)
22. Set lookup to join pointer

The Algorithm 3 illustrates how modified Skip List search datastructure works in Neo4j graph database. The input to search is given as a pediatric patient record name. This record name has an unique key value attribute to denote the node data holds. Hence this key value stored as flag in node structure v_i of graph G(V,E) is taken as input to search in the node skip list. This flag is compared with search input. Search starts from header pointer, it checks whether flag(v_i) equals the searchKey input then its lookup is returned. This returned value is assigned to join pointer. If the comparison result was no, then the traversal takes back to all levels till it reach the level0. The node v_i's successor and predecessor is denoted by v_i.neighbor[R][ ] and v_i.neighbor[L][ ] respectively. If the comparison results true, then the result from comparison is captured by lookup. There after lookup is assigned to join pointer. This comparison takes continues till the appropriate record name (search input) found. Similarly the same flag is used as input to property and relationship skip list. Finally the result is captured by join pointer. Analysis of the algorithm is carried out and found the search operation in a skip list

ISSN 2320 – 2602
International Journal of Advances in Computer Science and Technology (IJACST), Vol. 3 No.2, Pages : 12 – 18 (2014)
Special Issue of ICCSIE 2014 - Held during February 16, 2014,Bangalore, India

takes O(logn) time for computation. Next insert operation in Neo4j using SkipList is investigated.

Algorithm 4 : Insert for node with searchKey
Input: New node u to be inserted with key flag
1 Set header_pointer ← level
2 v_i ← header_pointer
3 Call Search for node with searchkey
4 u1 ← join_pinter
5 if v i < u < v i + 1 and level ≥ 0
6 while true do
7 Insert u in list at level l starting from v i
8 Scan backwards at level l to find v_i's such
9 that no node process same flag
10 if no such v_i's exists then exit
11 else v_i ← v_i+1
12 l ← l + 1

The Algorithm 4 illustrates insert operation in Neo4j graph database using skip list. Suppose a new node say 'u' have to be inserted to G(V,E) structure of Neo4j then,

1. Node u starts the search from v_i taken as header pointer. Node v_i is chosen by taking the topmost level's first node in the list. Here i ranges from 1 to V of the node list.
2. Skip list search Algorithm 3 is called and neighbouring node is obtained. This obtained result is compared at Step 5 of Algorithm 4 and node u is inserted.
3. After insertion it checks whether any other node process same flag if the result is true, then a pointer is addressed to the location. If the result is false, then level gets incremented.

Analysis of the algorithm was carried out and found the insert operation using skip list took around O(log n) time for computation.

ANALYSIS

The search operation in a Neo4j graph database using skip list with V nodes that is v_i where i range from 1 to V takes O(log n) completion time for search and insert of a given node in its average case, and O(n) completion time for search and insert of a given node in its worst case .

Proof. Let Σ denote the number of nodes in the skip list S, and u be the destination node to be searched from V nodes. From which position should the start begin is an issue. A search for a destination node begins from a randomly selected pinnacle node v_i within the Skip list S. It takes horizontal steps till the node is larger than or adequate to the destination node. From the above mentioned Skip List Algorithm 3 of search operation, we saw if the node is adequate to the destination, then it has been found. If the node is larger than the destination, or the search reaches the termination of list, the procedure is recurrent and moves to succeeding lower list. The number of steps expected in each one of the linked list is at the most 1/v of V. This could be seen clearly by tracing the search path backwards from the destination till reaching a node that seems within the next higher list or reaching the start of the list. Hence to conclude the overall expected cost of search is O(logn).

Similarly in case of insert operation each 2^h -th node, where h = 1, . . . $ceil(\log n)$, includes a reference to 2^h nodes ahead. For instance each 2nd node has a link to 2 nodes ahead; every 8th node has a link to 8 nodes ahead, etc. The header node is not smaller than the largest node on the list. If the Skip List contained 32 nodes and considering search and insert operation in it. Operating down from the highest level, initially we encountered node 16 and have cut the search in half 1/v of total nodes Σ of V. We continued to search again, one level down in either the left or right half of the linked list, again reducing the remaining search in half. We continue this procedure till the destination node u obtained or not. This is quite similar to binary search in an array and is perhaps the effective way to perceive why the utmost range of nodes visited in the linked list is in O(logn) for search and then insert[4].

For most of the queries, the algorithm suggested runs in O(log n) time as average case time computation. In case when all the nodes are level one at search and insert operations then the time computation of Skip lists is O(n). This has been considered as the worst case time of computation.

For finding all node attributes namely property and relationship in an interval, we can modify search by assigning a join coordinator. This coordinator is capable of processing query process simultaneously and perform a encapsulation operation on responses this will only take O(logn) + O(logn)+O(logn) = 3 × O(logn) time which is again constant times O(logn). Let the constant be denoted as k hence it will take O(klogn) as average case and O(kn) as worst case when all node attributes like property and relationship are at level one of Skip list.

Similar to search operation the insert operation in Neo4j graph database using Skip List can perform within a time computation of 0(klogn) as average case and O(kn) as worst case.

Analysis of the Multidimensional index can be categorized based on the following parameters.

Input Data: Input here is a node key value attribute which is checked with the node table. The node table gives the key to the node to be searched. This acts a unique key to property and relationship table.

ISSN 2320 – 2602
International Journal of Advances in Computer Science and Technology (IJACST), Vol. 3 No.2, Pages : 12 – 18 (2014)
Special Issue of ICCSIE 2014 - Held during February 16, 2014,Bangalore, India

Number of Dimensions: The number dimensions here depends on the three tables namely Node, Property and Relation. The number of attributes of Node, Property and Relation are 3, 6, 9 respectively. Based on these three tables the entire pediatric patient record is placed in Neo4j. Each of these three tables have three replicas linked with pointer key attributes in order to provide fast recovery during failure. Hence we can conclude the time taken is
$(3 \times V + 3 \times 6 \times V + 3 \times 9 \times V)3 = O(3 \times m \times V)3$
which is approximately $O(mV)3$ where m is the cross product of vectors from node, property and relationship tables with 3, 6 and 9 columns respectively. 'V' denotes the total of nodes.

Mapping of Results with Dimensions: Map function uses join pointer's that cluster each node its corresponding property and relationships. Here encapsulation operation is performed and all the null values are removed. This will take $O(n1 \times n2)$ to join all the table attributes to obtain the resultant query. Where $n1 \times n2$ denotes the table vector size.

Analysis Test Phase: Here total computation time of the multidimensional index structure for Neo4j is computed. This process is a encapsulation of Dimension and Map operations. Hence it was found it took a time computation of $0(m \times V)3 + 0(n1 \times n2)$ approximately as worst case.

Similar to search operation, the insert operation in Neo4j graph database using Multidimenstional index can perform within a time complexity of
$0(m \times V)3 + 0(n1 \times n2)$ where the value of 'V' gets incremented by one.

CONCLUSION

Neo4j graph database plays a significant role in modeling the web graph administration by google, social networking sites like Facebook to map friendship relationships, likes etc. and protein-protein interaction networks. In this paper we have conferred a multidimensional index structure for pediatric record placement in Neo4j graph database. Search and Insert algorithms were analyzed based on Multidimenstional index structure and found the time of computation was $0(m \times V)3 + 0(n1 \times n2)$ as worst case. Hence a new re placement model called Skip list index structure was steered. Skip list resulted in O(logn) time of computation for search and insert operations in average case. This is clearly illustrated in the Analysis section. It was also found it took a worst case computation of O(n) when all node attributes were at level one of Skip list. As the popularity of NoSQL database applications increases, making the BigData to be efficiently expressed and retrieved using a graphical data structure makes it more user friendly. As part of future work this setup could be tested for distributed network environment with large cluster in each of them. This helps to evaluate the scalability feature of query performance.

REFERENCES

[1] I. Robinson, J. Webber, and E. Eifrem, Graph Databases. O'Reilly Media, 2013.
[2] N. Developers, "Neo4j," Graph NoSQL Database [online], 2012.
[3] J. Partner, Neo4j in Action. O'Reilly Media, 2013.
[4] D. B. West et al., Introduction to graph theory, vol. 2. Prentice hall Englewood Cliffs, 2001.
[5] G. Vaish, Getting Started with Nosql. Packt Publishing, 2013.
[6] M. Pollack, O. Gierke, T. Risberg, J. Brisbin, and M. Hunger, Spring Data. O'Reilly, 2012.
[7] P. Boldi and S. Vigna, "Compressed perfect embedded skip lists for quick inverted-index lookups," in String Processing and Information Retrieval, pp. 25–28, Springer, 2005.