High Performance Dynamic Graph Model for Consistent Data Integration

Bilal Arshad
Ashiq Anjum
b.arshad@derby.ac.uk
a.anjum@derby.ac.uk
University of Derby
Derby, Derbyshire, UK

ABSTRACT

In a distributed environment, data from heterogeneous sources are brought together in a unified and consistent manner for analytics and insights. Inconsistencies arising due to the dynamic nature of sources such as addition/deletion of column or merging of columns can compromise the consistency of the distributed system. This can lead to the linking of inaccurate records and faulty data entries. Resulting in false reports and erroneous analyses. Furthermore, issues such as performance guarantees and scalability fuel the existing challenges. We have proposed an alternate graph-based approach to integrate data using an in-memory environment. The central idea of the approach is the use of graphs to integrate heterogeneous data sources in a distributed environment. The underlying approach provides both high-performance and scalability to address changes in a dynamic system for data integration. This allows the generation of graphs from individual source data and modifications in a consistent manner so that the state of the overall distributed system always remains coherent. It provides a novel way of combining consistent data integration and performance in a distributed system. Our system performs better than existing graph systems for dynamic graph evolution ensuring consistency and provides the necessary scalability guarantees as the size of the data increases. Results also show the correctness of the approach when integrating disparate data-sets.

KEYWORDS

data integration, graphs, dynamic graphs, consistency, scalability, performance

ACM Reference Format:
Bilal Arshad and Ashiq Anjum. 2019. High Performance Dynamic Graph Model for Consistent Data Integration. In *Proceedings of the IEEE/ACM 12th International Conference on Utility and Cloud Computing (UCC '19), December 2–5, 2019, Auckland, New Zealand.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3344341.3368806

UCC '19, December 2–5, 2019, Auckland, New Zealand

ACM ISBN 978-1-4503-6894-0/19/12…$15.00
https://doi.org/10.1145/3344341.3368806

1 INTRODUCTION

Data integration is the process of bringing together data from heterogeneous sources with a multitude of data types (not limited to CSV, relational and XML etc.) in a consolidated and consistent manner [18]. Usually, these sources are both high in volume and velocity of change [6]. Inconsistencies can stem promptly if sources evolve, addition/deletion of a column, change in the column name, merging of columns for instance. The consequence of inconsistencies is startling for data integration: linking of inaccurate records or faulty associations between data entries. This deluge of actions leads to false reports and analyses. Coupled with the volume and velocity of changes, the repercussions of inconsistencies can be very detrimental to the overall distributed system. In addition to these challenges, the dilemma of providing both performance and scalability when integrating data in a distributed environment still remains partially resolved.

In order to address the challenges of data integration in a dynamic environment, several approaches have been proposed. Louie et al. [27], Seoane et al. [1], Lenzerini et al. [25], Ziegler et al. [43], Subhani et al. [38], Subhani et al. [33] and Goble et al. [15] provide a comprehensive and detailed review of various data integration techniques for data integration such as link integration, ontologies, federated integration approach and warehousing. Different techniques explore the inherent trade-off between data consistency and practical aspects such as scalability and performance. Some of these techniques work well for static data or systems that can surrender high availability for performance but do not guarantee both. On one end of the spectrum, ETL (Extract, Transform and Load) tools used in Data Warehousing approaches provide consistency but lack scalability both in terms of data integration and querying. On the other end of the spectrum, NoSQL approaches [19] [41] provide scalability but grant a weak notion of consistency. Meanwhile, ontologies [23] provide neither a strong sense of consistency nor scalability.

Inconsistencies arising as a result of data integration in databases have been studied in great depth, some approaches deal with the means of transformation and cleaning processes when data is accessed by the sources [8] [14]. While others deal with inconsistencies in cases where the database does not guarantee integrity constraints e.g. see [9], [28], [29], [13] and [26]. One of the approaches to address these challenges is the use of Graphs. As links are created between sources in order to integrate, it is intuitive to rely on graph structures to represent the problems faced [10].

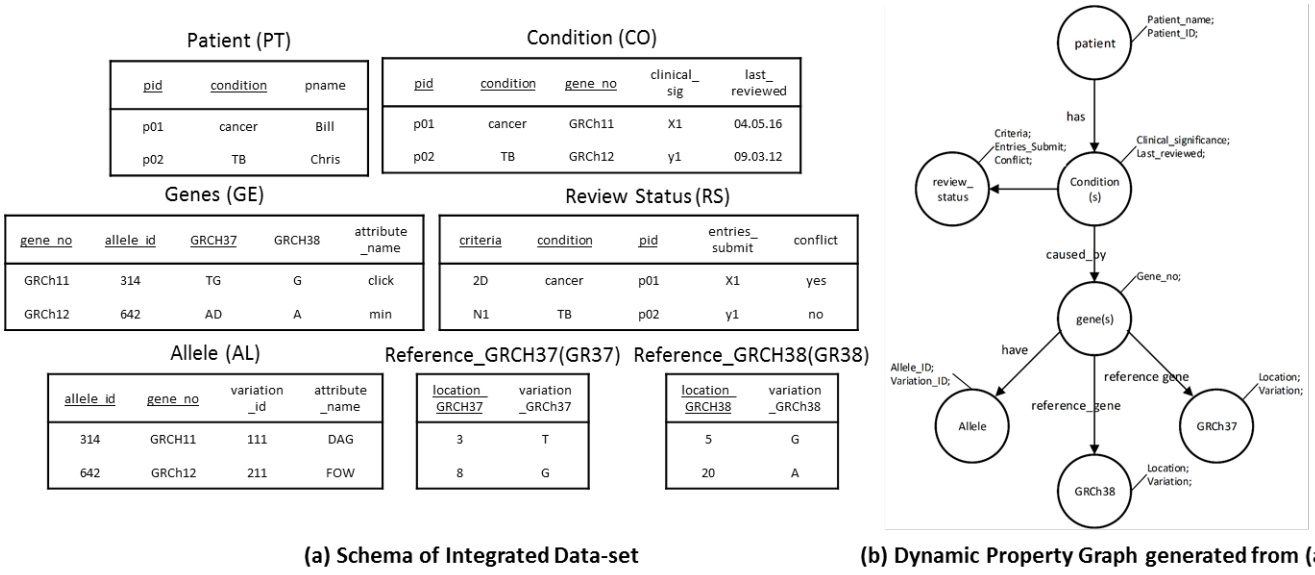

Figure 1: Sample Source Schema [24] and its associated Property Graph

The representation of data integration problem as graphs has been established in prior research works (integration of heterogeneous data sources to generate knowledge graphs Singhal et al. [7] and Bollacker et al. [6]). Graphs can be optimised for efficiently processing interrelated data-sets [32]. Since it does not have to take into account sets of unrelated data, much like the issue with SQL [36], graphs allow for fast traversals. Their flexibility and cost-effectiveness (in terms of computation), makes them an ideal candidate for data integration in a distributed environment prone to changes. In addition to these, graph models can handle high volumes of data and can scale well over billions of nodes hence making the solution scalable and improve the performance of the overall system [32] [4].

For this paper *graphs G = (V,E)* are defined as objects that consist of two sets: Vertices V and Edges E. Vertices or nodes are entities or items of significance; edges serve as the links between them. Both vertices and edges can have an arbitrary number of key/value pairs called properties. Graphs having *multiple edges* i.e. having parallel edges between the same two vertices are known as *multi-graphs*. Directed graph orders the vertices of an edge to denote edge orientation [35]. Property Graphs are directed, labelled, attributed multi-graph. Property graphs [35] not only enable to label both vertices and edges but also to attribute meta-data (i.e. properties) to them. We have used Dynamic Property Graphs [20] [2] to create our graph model. Dynamic Property Graphs allow changes such as edge/vertex insertions (and deletions) to be incorporated within property graphs. Dynamic graphs can be defined as a set of finite or infinite ordered set of pairs (date,events) [2]. All sets of events may tweak the graph structure/configuration/structure and/or attributes of some graph aspects. Dynamic graph models allow querying and updation of graphs as the sources evolve as quickly and efficiently as possible [42]. Once the graph model is created and loaded to a native graph system, data can be browsed and queried quickly and iteratively using data optimisation approaches [21] [17] [5] and distributed globally using high-performance computing, cloud and grid-based approaches [39] [30] [22] [31].

In order to demonstrate the effectiveness of our approach, our research endeavour aims to introduce a dynamic graph model which integrates data from disparate sources to a data warehouse. This work will use clinical data as a running example throughout the paper for data integration. Figure 1 shows an excerpt from a clinical data set to be integrated (Figure 1 (a) and the integrated graph representation Figure 1 (b)). This work only explores inconsistencies at a structural level. All other types of consistency checks are out of the scope of this paper. Consistency is defined as maintaining coherence between the structures in the source (source schema) and structures in the data warehouse (warehouse schema). Some of the contributions of this paper are representing source data as dynamic graphs for integration; quick generation of graphs; modelling changes to the integrated set of graphs (addition/deletion of vertices/edges, updating properties of vertices/edges) and formalising; additionally, allowing integrated data to be inspected quickly, helping to speed up the analytics process significantly.

This paper is organised as follows: Section 2 describes the dynamic graph model for data integration. It then explains how to formally model changes in Section 3 and explains the implementation and experimental setup in Section 4 respectively. Results are presented in Section 5 and in the end, Section 6 concludes this paper by specifying some open problems and giving future directions.

2 RESEARCH APPROACH - DYNAMIC GRAPH MODELLING AND GRAPH GENERATION

In order to address the challenges associated with the dynamic nature of data sources as discussed previously, we present an alternative data model which will allow for quick, efficient and iterative

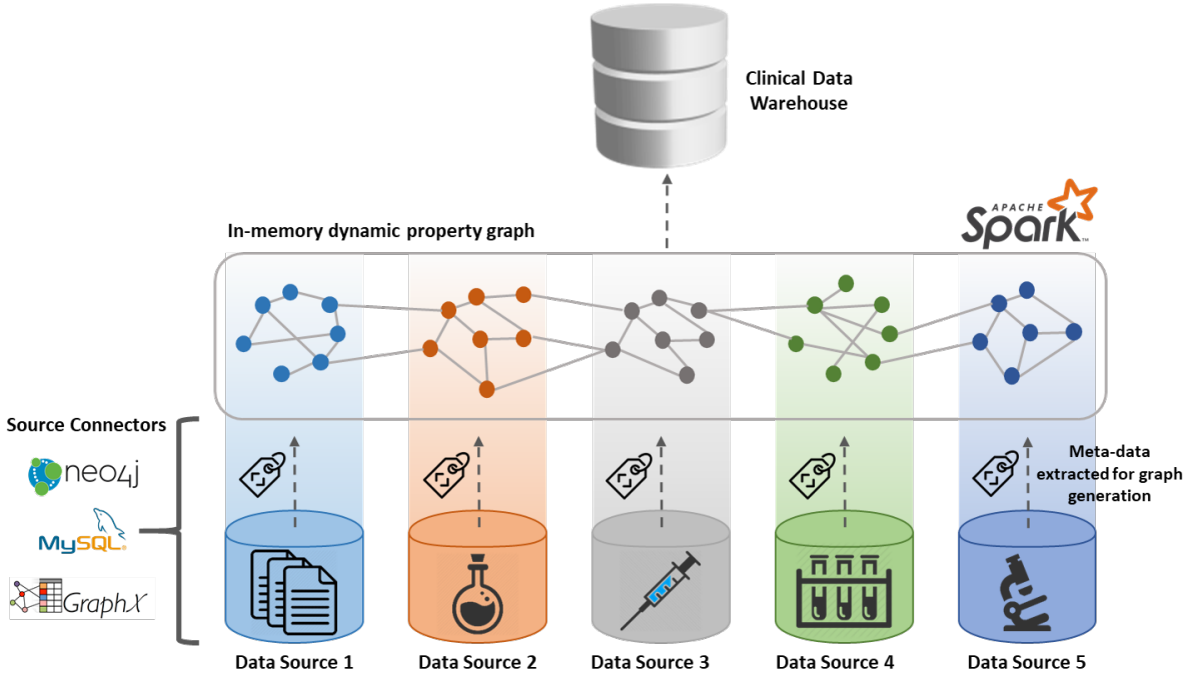

Figure 2: System Architecture

data integration. A unified dynamic property graph is being generated from the meta-data schemas of the source data. Figure 2 shows the architecture of the proposed system. Integrated graphs as shown in Figure 2 are generated using a modified approach from [11]. Virgilio et al.'s approach converts source data to property graphs. Our approach uses a modified version of this approach, using metadata to extract schema paths instead of using schema directly. This approach works well for our endeavour since schema paths from metadata extract structures from source data and essential details about sources that help to generate graphs in the first instance and later populate these graphs. Virgilio's approach attempts to generate graphs only and does not provide any means to update the graph once a change occurs. Our work extends their approach by making the graph dynamic allowing for modifications to the graph in a consistent manner. The first step is to create a graph model to represent the sources as an integrated property graph. Later these integrated graphs are populated using the open-source clinical data sets as running examples. Once these graphs have been populated they will be queried and tested for correctness, scalability, performance and consistency. There are slight modifications to the algorithms employed by Virgilio et al.'s approach to match our needs. Source schema is converted to graphs as follows:

Definition 1 (Schema Graph) *Given a schema* ***m****, the dynamic schema graph for* ***m*** *is a directed multi-graph* $\langle V,E \rangle$ *such that: (i) there is a vertex* $A \in V$ *for each attribute A of a relation in m and (ii) there is an edge* $(A_i,A_j) \in V$ *if one of the following holds:*

(a) A_i *belongs to a key of a relation m in m and* A_j *is a non-key attribute of m;*
(b) A_i,A_j *belong to a key of a relation m in* ***m*** *and*
(c) A_i,A_j *belong to* m_i *and* m_j *respectively and there is a foreign key between* $m_i.A_i$ *and* $m_j.A_j$

Given the meta-data schema **m** and set of full schema paths **SP**, we generate integrated property graphs **g** = (*V,E*) using the modified version of cases described in Virgilio et al.'s approach. Modifications to the algorithms include the use of extracted metadata from source schemas for structural details to construct graphs. Our model runs repeatedly over all the elements of the schema path iterating and analysing schema path from parent A_i to child A_k. **Note: each A_i of *sp* corresponds to an attribute in *m*.** Given schema *m* function **getall(m,A_i)** returns all the values *v* associated to A_i in **m**. Accordingly, when we combine the functioning domain of attribute A_i in the vertices of *g*, we can add that A_i is inspected i.e. A_i is added to the collection of **VS** of visited attributes.

3 FORMAL MODELLING FOR DYNAMIC GRAPH EVOLUTION

In order to add dynamic changes to the integrated graph our work employs Petri-nets to model the changes in the source schema. Petri-nets enables us to describe state changes in a dynamic system like ours with transitions. In a distributed environment, as the data evolves, the state of the sources and the warehouse needs to be captured and integrated. Formal modelling of these state changes enable us to develop a precise specification of the evolving sources and how they can accommodate changes such as an addition, deletion or update.

Petri net elements can be defined as TGG (Triple Graph Grammar) rules from project or object element [37]. This is a form of formal mapping. Figure 3 shows the use of Petri-nets to model graph

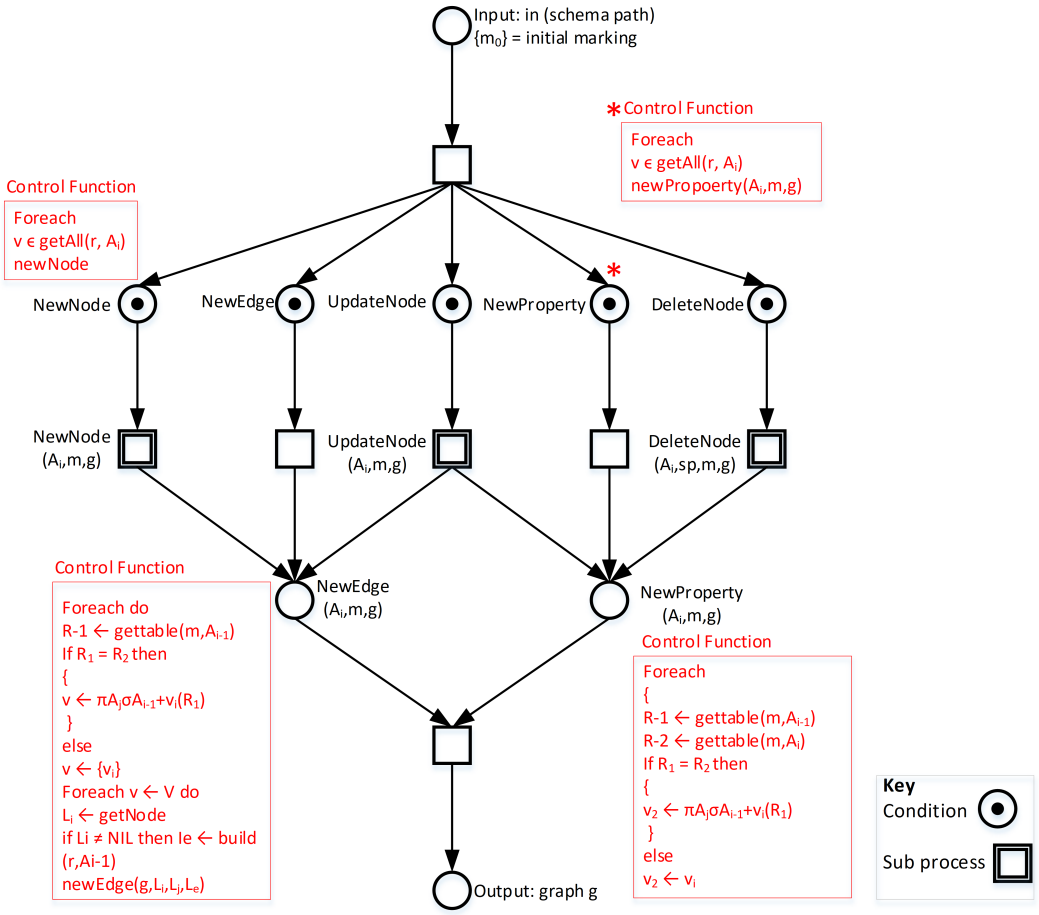

Figure 3: Petri-nets for modelling changes

evolution (insertion, deletion and update). The integrated graph is provided as in input to the Petri-net model. Based on the evolution of the source it can be presented with the following scenarios:

- Vertex/Edge Insertion
- Vertex/Edge Deletion
- Update Properties

Places i.e. circles represent the conditions that are modelled within the graph as mentioned above, while squares represent the events occurring within the graph that may cause changes in the state of the graph. Edges connect places to transitions and transitions to places highlighting the flow of petri-net model. Petri-nets are defined as bipartite graphs or a bipartite digraph that can be represented as a five-tuple (R, T, I, O, M_o), where P is a set of finite set of places, T is a finite set of transitions, I $\subseteq (PxT)$ Input arcs, and O $\subseteq (TxP)$ Output arcs, P $\cup$ T $\neq \phi$ and P $\cap T \neq \phi$,m_0 represents the initial marking. Petri-nets are closely associated with graphs and graph theory and can serve well for visualisation purposes. Control functions presented in red along with the places describe the conditions required by these places to move to transitions within the petri-net model. The following section describes individual places (scenarios) in detail, the petri-net model is broken down to explain the individual places and transitions in detail.

3.1 Modelling changes to Dynamic Graphs - Insertion

Dynamic graphs encode changes in sources to structural changes in the graph. For the purpose of this paper, we have focused on structural changes only. Here we talk about different types of events (types of changes) and how to model them onto a graph.

Insertion: If a new column is added to one of the source's schema for instance "startTime" in the 'Review Status' table (Figure 1 (a)), a new vertex needs to be created in this case. It is worthy to note if the insertion is a new vertex (such as the example scenario) or a new edge (for example relationship between two vertices is altered in the source schema) integrated graph g (current state) needs to be updated to g' (modified state) in order to keep the graph consistent. In situations where the newly added column is a foreign key to another table, it is essential to create edges between the new vertex and the corresponding vertex(-ices).

Figure 4 shows the insertion of a new vertex to the graph, the box labelled NewNode(A_i,m,g) describes the transition from the

Algorithm 1: Insertion in dynamic property graph

Input: Updated SP, graph g
Output: updated graph g'
VS ← ∅;
g ← (V, E);
foreach $sp \in SP$ do
  find vertex A_i;
  if $A_{i+1} \leftarrow \emptyset$ **then**
    add newNode(A_i, m, g);
  else if $A_{i+1} \neq \emptyset$ **then**
    add newNode(A_i, m, g);
    add newEdge(A_{i-1}, sp, m, g);
    add newEdge(A_{i+1}, sp, m, g);
  else
    add newEdge(A_i, sp, m, g);
  end
  VS ← VS ∪ $\{A_i\}$;
  return g';
end

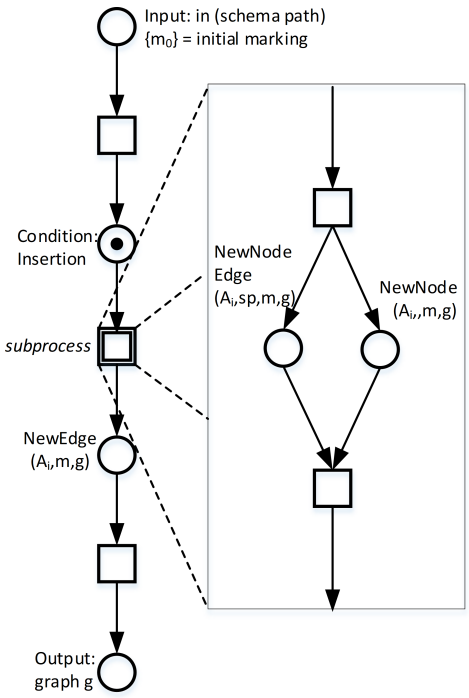

Figure 4: Insertion in Dynamic Property Graph

current state of the graph to the updated state of the graph once a new vertex is added. Control function for this transition include generation of both new vertex as defined by newNode(A_i,m,g) and newNodeEdge(A_i,sp,m,g) to the newly generated vertex. Algorithm 1 checks for conditions if the vertex already exists, if not it adds a new vertex and the associated edges to its parent vertex so the vertex is not lost within the graph. Once the new vertex and edges have been successfully updated, the output graph g is also updated accordingly to the new state. Note: A_{i-1} refers to the parent vertex of the vertex under consideration and A_{i+1} refers to the

3.2 Modelling changes to Dynamic Graphs - Deletion

Deletion: If a column is removed from the schema for instance 'conflict' from the table 'Review Status' (Figure 1 (a)), the corresponding vertex in g needs to be removed along with the edge that connects it to g (current state). In such a scenario it is imperative to take into account if other relationships (edges) exist to that vertex and if that is the case, they need to be removed too. Figure 5 shows the petri-net transition if there is a vertex to be deleted, the deleteNode algorithm takes the current state of graph as the input, iterates to the vertex to be deleted, generates new edges between the parent vertex and the child vertex of the vertex under consideration A_i. This is done under the condition that A_i has any child vertex linked with itself. Once the edges have been created the vertex A_i and the edges from A_{i-1} are deleted from the graph. The updated set of graph g' (modified state) then represents its current state. Algorithm 2 describes how the graph evolves when a vertex and its associated edges are deleted from the graph.

Algorithm 2: Deletion in dynamic property graph

Input: Updated SP, graph g
Output: updated graph g'
VS ← ∅;
g ← (V, E);
foreach $sp \in SP$ do
  find vertex A_i;
  createAll NewEdge(A_i, sp, m, g) from A_{i-1} to A_{i+1};
  delete vertex A_i;
  deleteEdge from A_{i-1} to A_i and A_i to A_{i+1};
  VS ← VS ∪ $\{A_i\}$;
  return g';
end

3.3 Modelling changes to Dynamic Graphs - Update

Update: If a column name is updated, the corresponding vertex name and properties need to be updated in g accordingly. Updates can also occur on edges or properties in scenarios where the associated relationships are updated in the source schema. Figure 6 shows the petri-net model describing the formalisation of updating the graph. The petri-net is provided with the existing graph as the input. If the evolution of source involves creating new linkages i.e. creating new edges between existing nodes the newNodeEdge(A_i,so,m,g) function is called allowing new edges to be created as described in algorithm 3. Similarly, if the property of any vertex or edge is added during the evolution of sources, the newProperty(A_i,sp,m,g) function is used to add properties to the concerned vertex/edge. Once the new edges have been created and properties added to vertex/edges the graph g' is brought to the current state of the graph.

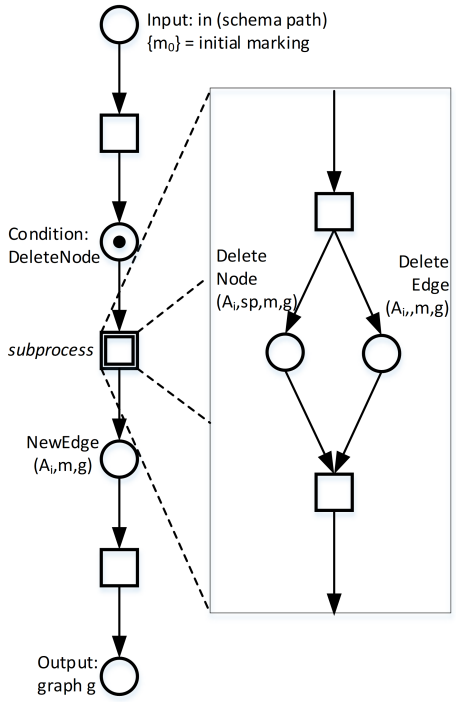

Figure 5: Deletion in Dynamic Property Graph

Algorithm 3: Update in dynamic property graph

Input: Updated SP, graph g
Output: updated graph g'
VS ← ∅;
g ← (V, E);
foreach $sp \in SP$ do
  find vertex A_i;
  create NewEdge(A_i, sp, m, g) from A_{i-1} to A_{i+1};
  createAll NewProperty(A_i, sp, m, g);
  VS ← VS ∪ $\{A_i\}$;
  return g';
end

4 IMPLEMENTATION AND EXPERIMENTAL SETUP

Implementation: Property graph model was used to generate integrated graphs from source meta-data schemas. The changes within a source are then randomly created using a stream of modifications containing vertex/edge insertions, vertex/edge deletions and property updates are implemented based on algorithms in Section 3. Scala Scripting was used to extract schema records to generate Vertex and Edge RDDs in Spark using the GraphX API. Scala is the general-purpose language to be used for distributed processing systems such as Spark. Additionally, Spark uses the GraphX API to process graphs.

We began with data-sets in two major formats; Relational and XML. The relational data-sets were converted to the graph as explained in Section 3. XML data was ingested into Neo4j using

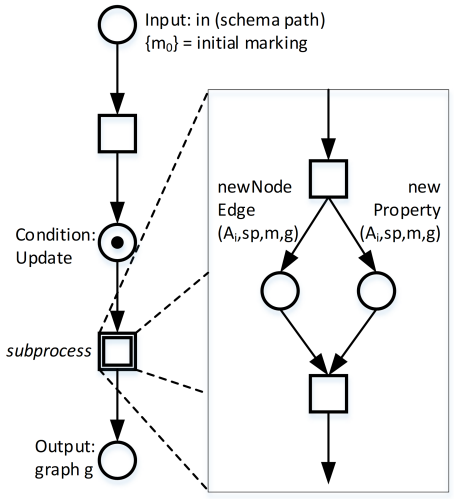

Figure 6: Update in Dynamic Property Graph

Integrated Datasets	Type	No. of Vertices	No. of Edges
DS1	Real World	350	2875
DS2	Real World	11600	65425
DS3	Real World	25767	98598
DS4	Real World	42494	109271
DS5	Synthetic	65821	2386981
DS6	Synthetic	79729	4857647

Table 1: Clinical Data-sets

'apoc.load.xml' provided by Neo4j to bulk load XML data for testing against our proposed approach. In order to generate graphs and ingest data, XML data from ClinVar [24] was used. ClinVar is a freely available clinical repository of clinical data containing human variations and phenotypes.

Evaluation Platform: Our proposed approach is evaluated on an Intel Core i7 CPU running at 3.60 GHz with 16 GB RAM having a Ubuntu 16.04 running on a Virtual Machine. We used GraphX [16] as the underlying technology for our approach (referred to as the proposed approach in the next section) and Neo4j [40] for testing and comparison purposes (referred to as the test approach in the next section). The next section details the results from our experimentation.

Table 1 shows the integrated data-sets used to evaluate the system. Data-sets DS1 to DS6 is an integrated set of data from source schemas of varying size. Synthetic data-sets were generated using DataSynth [3] and Graph500 RMAT data generator [34] due to the limited size of clinical data being publicly available. The structures within these synthetic graphs are similar to the ones present in clinical data to ensure uniformity across the testbeds and results.

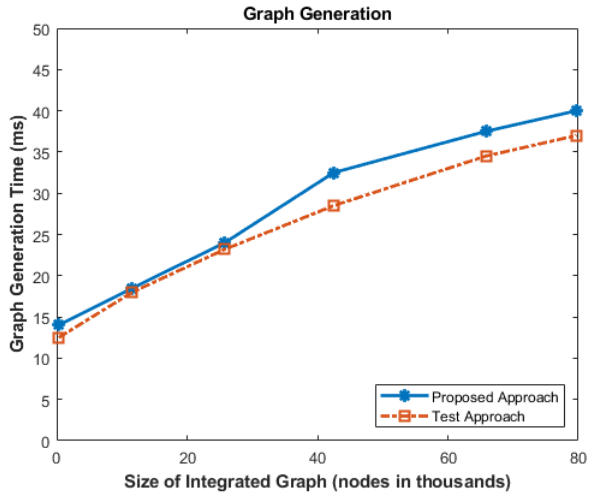

Figure 7: Graph Generation

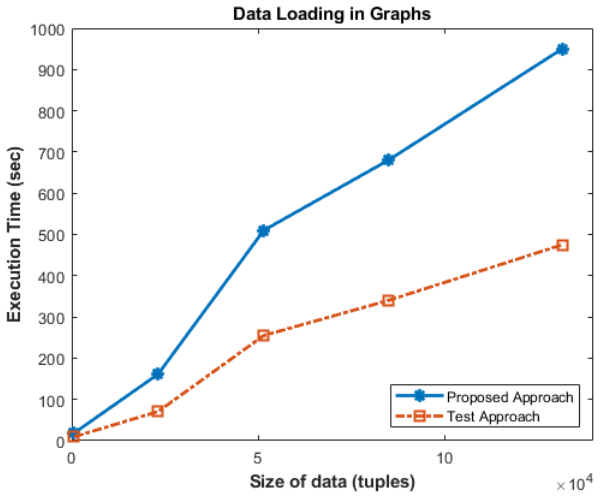

Figure 8: Graph Loading

5 RESULTS

Our results are quite encouraging, we have been able to integrate data-sets resulting in the largest integrated data-set containing approximately 80,000 nodes and close to 49,00,000 edges using ClinVar [24] and synthetic data-sets. The initial effort required to generate a graph and push the data to a graph system is costly, but once it is in the graph system browsing and querying it iteratively is cost-effective in terms of computation because it does not need to be loaded every time there is a change in the source schema. Firstly, we evaluate the two systems with respect to the time they take to generate graphs from raw data-sets. The generation of graphs as mentioned is a costly affair in terms of both time and computation as they generated from scratch at this stage. Figure 7 describes the comparison of graph generation using different techniques. Extraction of metadata is done through the open-source tool (Apache MetaModel [12]) which provides interfaces for various types of data sources not limited to MySQL, CSX, XML files etc. While GraphX and Neo4j have been used to generate graph structure within the native systems.

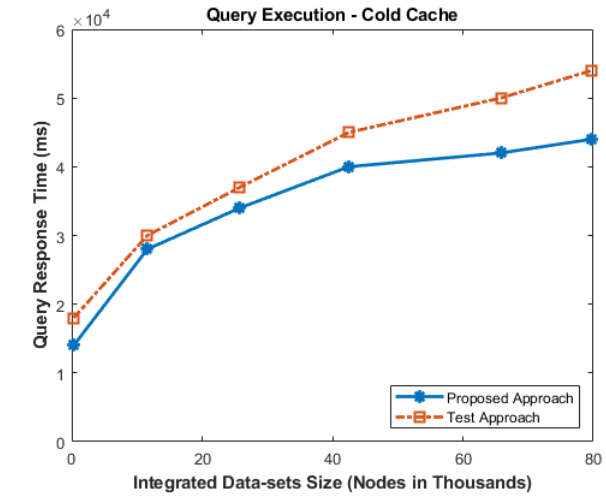

Figure 9: Query Execution time cold cache

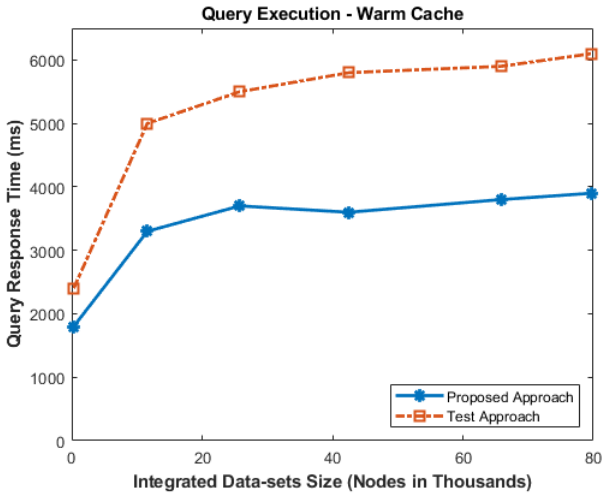

Figure 10: Query Execution time warm cache

Once the graphs have been generated the next stage is to populate these graphs by ingesting data. We evaluate both systems with respect to the time it takes to populate a graph from data sources. We compared our approach against native data importers of Neo4j. We have used a naive model to import a SQL dump in order to populate graph i.e. one vertex for each tuple present in the source data-set and one edge for each foreign key reference when integrating these data-sets. Neo4j importer performed better than our system (approx. two times better). This is because our approach has to process the schema information of the relational database

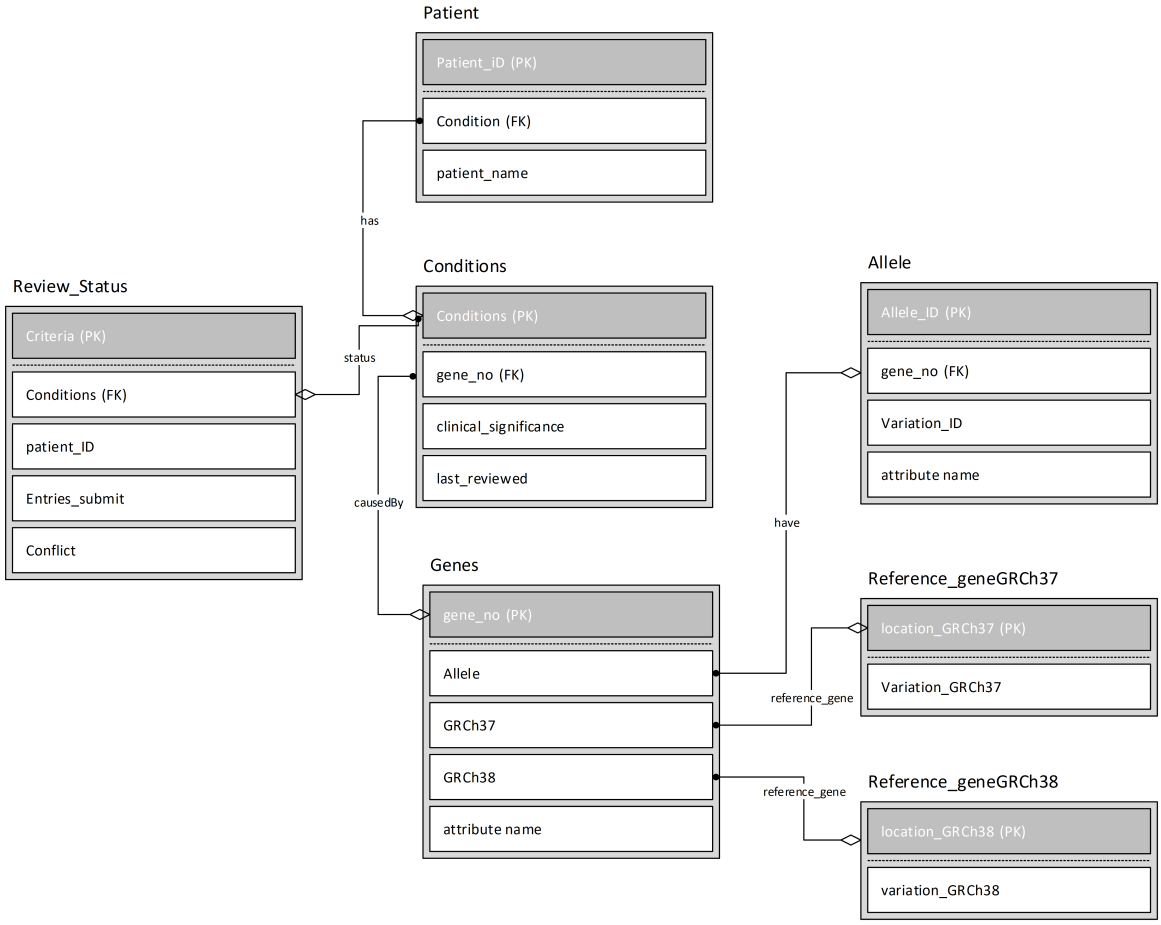

Figure 11: Part of relational schema of integrated data-set DS1

```
scala> clinvardf.printSchema
root
 |-- Name: string (nullable = true)
 |-- Gene(s): string (nullable = true)
 |-- Condition(s): string (nullable = true)
 |-- Clinical significance (Last reviewed): string (nullable = true)
 |-- Review status: string (nullable = true)
 |-- GRCh37Chromosome: string (nullable = true)
 |-- GRCh37Location: string (nullable = true)
 |-- GRCh38Chromosome: string (nullable = true)
 |-- GRCh38Location: string (nullable = true)
 |-- VariationID: string (nullable = true)
 |-- AlleleID(s): string (nullable = true)
```

Figure 12: Snippet from Schema of DS1 extracted by Scala Query

(i.e. the schema graph), while the test system directly imports data values from the SQL dump using Neo4j importer. As seen in Figure 8, the proposed approach takes almost twice as long as the test approach to ingest data into the graphs.

We then evaluated these data-sets based on dynamic graph queries. For each data-set we grouped the queries in five sets (i.e. ten queries per set): each set is homogeneous with respect to its complexity of the queries (e.g. number of connected components, number of results and so on.). For instance, referring to integrated ClinVar data-sets, the first set of queries searches information about colorectal cancer while the second set of queries seeks information about lung cancer. For each set, we ran the queries ten times and measured the average response time. We performed cold-cache experiments (i.e. by dropping all file-system caches before restarting the VMs and running the queries) and warm-cache experiments (i.e. without dropping the cache). Figure 9 shows the performance for cold cache experiments. Due to space constraints, in the figure, we report times only on ClinVar data-sets since the larger size of synthetic data-sets poses more challenges. Our system performs better consistently for most of the queries, significantly outperforming the test approach in some cases (e.g. Figure 10). We highlight how our dynamic evolution algorithms allow GraphX to perform better than Neo4j in DS4 (having a more complex schema). This is due to our strategy reducing space overhead and consequently the time complexity of the overall process with respect to the test approach that spends much time traversing a large number of nodes. Warm-cache experiments follow a similar trend as shown in Figure 10.

In order to test the correctness of the graph, we investigated if the integrated schemas of data-sets match the integrated schemas resulting from the relational model. We tested if integrated schema of data-set 1 (DS1) could be reconstructed without any loss of data by querying the graph in our proposed approach and the tested approach. We then matched these schemas with the schema of the same data-sets being integrated into a relational model. Figure 12 shows part of the schema from data-set 1, this matches the schema obtained from data-set being integrated using a relational model (Figure 11).

6 CONCLUSION AND FUTURE DIRECTIONS

Distributed environments require data from disparate sources to be integrated in a consistent order for various reasons such as querying across sources for interdependent thematics or analytics for reports. These sources are prone to changes and can lead to inconsistencies. Our paper introduces an alternate approach of representing sources as graphs for integration. Representation of integrated data as graphs allows for faster querying, ensures consistency when the sources evolve and provides scalable storage. In addition to all this, the approach provides provenance of changes in the sources and the graph to keep track of all the changes that occur over time. This traceability is important for trust within the sources as they evolve and are used for reporting. Metadata was extracted from sources to obtain structures to generate graphs and then later used to populate graphs for testing purposes. Petri-nets (modelling technique) was used to formalise the model for graph evolution, and algorithms have been proposed for insertion, deletion and update. We then implemented these in a distributed in-memory environment (Spark) for high-performance integration and scalability. The results test the approach by using various sets of queries on the integrated data-sets to validate both correctness and the scalability of the approach. Results show that our approach has significant difference in querying time as compared to the test approach using various set of queries. Furthermore, the proposed system proved to be correct when integrating data from various sources. This was done by matching the source schema to the integrated schema for any differences. To our satisfaction, integrated schema from our graphs was an exact match of the schema being independently integrated at the source level. The proposed system makes it easier to analyse the data since it eliminates the need to load data every time an analysis needs to be performed. The use of in-memory processing will make it easy to add changes to the integrated schema, updating the graph in real-time and be more accurate. While the consistency checks will ensure that the approach provides strong consistency guarantees in addition to performance and scalability. In future, the approach can be tested on other domains for its efficacy.

REFERENCES

[1] Aguiar-Pulido V. R Munteanu C. Rivero D. R Rabunal J. Dorado J. A Seoane, J. and A. Pazos. 2013. Biomedical data integration in computational drug design and bioinformatics.. In *Current computer-aided drug design*, Vol. 9. 108–117.

[2] Damien Olivier Yoann Pigné. Antoine Dutot, Frédéric Guinand. 2007. GraphStream: A Tool for bridging the gap between Complex Systems and Dynamic Graphs.. In *Emergent Properties in Natural and Artificial Complex Systems. Satellite Conference within the 4th European Conference on Complex Systems (ECCS'2007)*. Cray Users Group (CUG). hal-00264043.

[3] Raghav Kaushik Arasu, Arvind and Jian Li. 2011. DataSynth: Generating synthetic data using declarative constraints. In *VLDB Endowment*, Vol. 4. 1418–1421.

[4] C. Avery. 2011. Giraph: Large-scale graph processing infrastructure on hadoop. In *Proceedings of the Hadoop Summit. Santa Clara, USA*, Vol. 11. 5–9.

[5] Ashiq Anjum Richard Hill Nik Bessis Baker, Charlie and Saad Liaquat Kiani. 2012. Improving cloud datacentre scalability, agility and performance using OpenFlow, Vol. 122. IEEE Fourth International Conference on Intelligent Networking and Collaborative Systems, 20–27.

[6] Evans C. Paritosh P. Sturge T. Bollacker, K. and J. Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge.. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 1247–1250.

[7] Evans C. Paritosh P. Sturge T. Bollacker, K. and J. Taylor. 2012. Introducing the knowledge graph: things, not strings.. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. Official google blog. https://blog.google/products/search/introducing-knowledge-graph-things-not/

[8] Mokrane Bouzeghoub and Maurizio Lenzerini. 2001. Introduction to:-data extraction, cleaning, and reconciliation. In *Special issue of Information Systems*, Vol. 26. 535–536.

[9] F. Bry. 1997. Query answering in information systems with integrity constraints.. In *IFIP WG 11.5 Working Conf. on Integrity and Control in Information System*. Chapman Hall.

[10] Joachim Rupp Croset, Samuel and Martin Romacker. 2015. Flexible data integration and curation using a graph-based approach. In *Bioinformatics*, Vol. 32. 918–925.

[11] Maccioni A. De Virgilio, R. and R. Torlone. 2013. Converting relational to graph databases. In *First International Workshop on Graph Data Management Experiences and Systems*. ACM, 1.

[12] Kasper Sørensen et al. Last Accessed: September 2019. Apache MetaModel.

[13] S. Greco G. Greco and E. Zumpano. 2001. A logic programming approach to the integration, repairing and querying of inconsistent databases.. In *17th Int. Conf. on Logic Programming (ICLP'01) of Lecture Notes in Artificial Intelligence*, Vol. 2237. Springer, 348– 364.

[14] Daniela Florescu Dennis Shasha Galhardas, Helena and Eric Simon. 1999. An extensible framework for data cleaning. HAL-Inria.

[15] C. Goble and R. Stevens. 2014. State of the nation in data integration for bioinformatics.. In *Journal of biomedical informatics*, Vol. 41. 687–693.

[16] Xin R.S. Dave A. Crankshaw D. Franklin M.J. Gonzalez, J.E. and I. Stoica. 2014. Graphx: Graph processing in a distributed dataflow framework. In *In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. 599–613.

[17] Ashiq Anjum Richard Mcclatchey Habib, Irfan and Omer Rana. 2013. Adapting scientific workflow structures using multi-objective optimization strategies, Vol. 8. ACM Transactions on Autonomous and Adaptive Systems (TAAS).

[18] Rajaraman A. Halevy, A. and J. Ordille. 2006. Data integration: the teenage years. In *Proceedings of the 32nd international conference on Very large data bases - VLDB Endowment*. 9–16.

[19] E. Haihong Guan Le Han, Jing and Jian Du. 2011. Survey on NoSQL database.. In *6th international conference on pervasive computing and applications*. IEEE, 363–366.

[20] Frank Harary and Gopal Gupta. 1997. Dynamic graph models. In *Mathematical and Computer Modelling*, Vol. 25. 79–87.

[21] Antonio Delgado Peris Ashiq Anjum Dave Evans Stephen Gowdy José M. Hernandez Eduardo Huedo et al. Hasham, Khawar. 2011. CMS workflow execution using intelligent job scheduling and data access strategies., Vol. 58. IEEE Transactions on Nuclear Science, 1221–1232.

[22] Ashiq Anjum Michael Knappmeyer Nik Bessis Kiani, Saad Liaquat and Nikolaos Antonopoulos. 2013. Federated broker system for pervasive context provisioning., Vol. 86. Journal of Systems and Software, 1107–1123.

[23] Michel. Klein. 2001. Combining and relating ontologies: an analysis of problems and solutions.. In *OIS@ IJCAI*.

[24] Lee J.M. Riley G.R. Jang W. Rubinstein W.S. Church D.M. Landrum, M.J. and D.R. Maglott. 2013. ClinVar: public archive of relationships among sequence variation and human phenotype.. In *Nucleic acids research*, Vol. 42. 980–985.

[25] M. Lenzerini. 2002. Data integration: A theoretical perspective.. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 233–246.

[26] J. Lin and A. O. Mendelzon. 1998. Merging databases under constraints.. In *Int. J. of Cooperative Information Systems*, Vol. 7. 55–76.

[27] Mork P. Martin-Sanchez F. Halevy A. Louie, B. and P Tarczy-Hornoch. 2007. Data integration and genomic medicine. In *Journal of biomedical informatics*, Vol. 40. 5–16.

[28] L. E. Bertossi M. Arenas and J. Chomicki. 1999. Consistent query answers in inconsistent databases.. In *18th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'99)*. ACM, 68–79.

[29] L. E. Bertossi M. Arenas and J. Chomicki. 2000. Specifying and querying database repairs using logic programs with exceptions.. In *4th Int. Conf. on Flexible Query Answering Systems (FQAS'00)*. Springer, 27–41.

[30] Andrew Branson Ashiq Anjum Peter Bloodsworth Irfan Habib Kamran Munir Jetendr Shamdasani Kamran Soomro McClatchey, Richard and neuGRID Consortium. 2005. Providing traceability for neuroimaging analyses., Vol. 82.

International journal of medical informatics, 882–894.

[31] Irfan Habib Ashiq Anjum Kamran Munir Andrew Branson Peter Bloodsworth Saad Liaquat Kiani McClatchey, Richard and neuGRID Consortium. 2013. Intelligent grid enabled services for neuroimaging analysis., Vol. 122. Neurocomputing, 88–99.

[32] J.J Miller. 2013. Graph database applications and concepts with Neo4j. In *In Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA*, Vol. 2324.

[33] Anjum Ashiq. Moeez, Subhani. 2019. Multiclass disease predictions based on integrated clinical and genomics datasets.. In *BIOTECHNO2019, The eleventh international conference on Bioinformatics, Biocomputational Systems and biotechnologies*. 20–27.

[34] Kyle B. Wheeler Brian W. Barrett Murphy, Richard C. and James A. Ang. 2010. Introducing the graph 500. In *Mathematical and Computer Modelling*, Vol. 25. Cray Users Group (CUG), 45–74.

[35] M.A. Rodriguez and P. Neubauer. 2010. Constructions from dots and lines.. In *Bulletin of the American Society for Information Science and Technology*, Vol. 36. 35–41.

[36] M.A. Rodriguez and P. Neubauer. 2013. The graph traversal pattern.. In *In Graph Data Management: Techniques and Applications*. IGI Global, 29–46.

[37] A.S. Staines. 2011. Rewriting Petri Nets as Directed Graphs. In *International Journal of Computers*. 289–297.

[38] Ashiq Anjum Andreas Koop Subhani, Moeez M. and Nick Antonopoulos. 2019. Clinical and genomics data integration using meta-dimensional approach.. In *9th International Conference on Utility and Cloud Computing (UCC)*. IEEE/ACM, 416–421.

[39] M. Thomas T. Azim I. Chitnis A. Anjum D. Bourilkov M. Kulkarni et al. Van Lingen, Frank. 2005. Grid enabled analysis: architecture, prototype and status.

[40] J. Webber and I. Robinson. 2014. A programmatic introduction to neo4j. Addison-Wesley Professional, 599–613.

[41] Ruichun Hou Xiang, Peng and Zhiming Zhou. 2010. Cache and consistency in NOSQL. In *3rd International Conference on Computer Science and Information Technology*, Vol. 6. IEEE, 117–120.

[42] Mahmoud Attia Doaa Hegazy Zaki, Aya and Safaa Amin. 2016. Comprehensive survey on dynamic graph models. In *International Journal of Advanced Computer Science and Applications*, Vol. 7. 573–582.

[43] P. Ziegler and K.R. Dittrich. 2007. Data integration—problems, approaches, and perspectives.. In *Conceptual modelling in information systems engineering*. Springer, 35 – 58. Berlin, Heidelberg.