GraphJ: A Tool for Big Data Complexity Reduction

Hani Bani-Salameh

The Hashemite University
Zarqa 13115, Jordan
Email: hani@hu.edu.jo

Abdullah Al-Shishani

The Hashemite University
Zarqa 13115, Jordan
Email: abdullah.asendarz@gmail.com

***Abstract*—Software developers, researchers, and industrial companies from all sectors such health, transportation, water treatment, etc., use and deal with big data in order to conduct their research and find better solutions that improve our way of life. Data scientists and software engineers are using generated big data to get accurate information and to extract the maximum value from the data available to them. Big data is applicable in many domains and can help solve many problems. However, analyzing such data is not easy due to its complexity that is resembled by the 6Vs of big data: volume, velocity, value, variety, variability, and veracity. Thus, big data reduction methods and tools are used in order to enhance the data and make it easier to analyze. This paper presents a big data complexity reduction tool called GraphJ. The proposed tool converts a rational database into a graph database, which makes unlocking knowledge patterns much easier than dealing with ordinary rational databases. A case study has been conducted to assess the usefulness and effectiveness of the proposed tool.**

Keywords–Big data; Reduction; Complexity; Graph; Relational database; Neo4J; GraphJ.

I. INTRODUCTION

The term of big data was originated in 1997 and was introduced by two NASA researchers, Michael Cox and David Ellsworth [1]. So far, there is no formal definition for big data, although it is referred to as complex data that are characterized by the well-known Vs properties: huge volume, high value, much variety, low veracity, and big variability that are collected from multiple data streams [2]. The world's data volume keeps growing because the data is continuously produced using numerous data streams *(e.g., mobile devices, cameras, microphones, wireless sensor networks, etc.)* [3]. Hence, such growth in data makes traditional data processing applications useless and requires huge efforts for analysis and processing [4].

Researchers and data scientists working with big data face many challenges, such as: *capturing data, data storage, data analysis, search, sharing, transfer, visualization, querying, updating and information privacy* [5]-[6]. Moreover, big data inherits the *curse of dimensionality*, meaning that the data has a massive set of dimensions, which also makes analysis and processing harder [7][8]. One of the ways to overcome such issues is big data reduction. The term *'reduce'* can relate to either the complexity or the volume of the data. By reducing the complexity and/or the volume of the data, it becomes more manageable, hence easier to analyze. Big data reduction methods can be categorized into five groups, as follows:

- **Network Theory**: or graph theory is one of the significant techniques that are used in reducing high-dimensional unstructured big data into low-dimensional structured data [9]. Trovati et al. [10] proposed a network theory-based approach to extract the topological and dynamical network properties from big data.
- **Dimension reduction**: the dimensions of the data are the attributes of that data *(i.e., id and name of a student, color and speed of a car, etc.)*. The dimensionality can be reduced by either features' selection or features' extraction. Feature selection is done by only considering the important dimensions of that data, as all the dimensions will not be needed. Feature extraction is done by merging multiple sets of dimensions to derive new ones [11]-[12].
- **Deduplication**: the data collected may contain redundancies. Redundancy is not necessarily a duplicate row in the database. Redundancy in the data can be the order of bits or block of memory that is exactly identical to another one. In such case, the original one is kept, and the copy is replaced with a pointer to the original in order to reduce the volume in use [13]-[14].
- **Graph theory**: to reduce the complexity of the data, the topological and dynamical network properties are extracted. To construct topological networks, relationships between data points are established [15]-[16].
- **Compression** [9] such methods are good to handle data reduction in terms of size by maintaining the whole data streams. Compression-based methods involve complex computations that affect the reduction process efficiency and add compression overhead cost. Many big data compression techniques are proposed by academics and researchers, including spatiotemporal, Anamorphic Stretch Transform (AST), parallel compression, sketching, and adaptive compression.

Research showed that these methods cannot be used single-handedly by considering all the Vs properties of big data [15], which motivates the need for more reliable data reduction approaches that combine multiple methods together.

Motivated by knowledge graphs [17]-[18], this paper presents a tool and an approach to reduce big data complexity using graphs. In graphs, data is presented using nodes and edges instead of using an ordinary relational database (see Figure 1), where each row is presented by a node (an object), and relationships between the entities are replaced with edges between the nodes. Presenting the database using a graph database makes the act of unlocking knowledge patterns easier

 ISBN: 978-1-61208-700-9

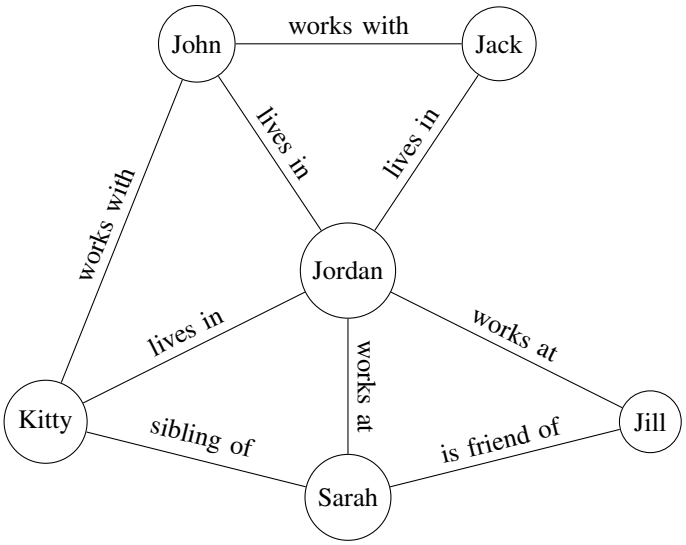

Figure 1. A sample of graph database.

[15], and is most useful when one must deal with highly interconnected data [19].

GraphJ uses MySQL database [20] to convert its schema and data into Neo4J graph database [21]. The reason for choosing MySQL because it is well known and widely used. The reason for choosing Neo4J is because of its high performance [22]-[23]. GraphJ is built using Java, because of its suitability and connectivity support for Neo4J.

The rest of this paper is organized as follows: Section II describes related work; Section III describes the tool and used approach; Section IV presents an experimental evaluation; Section V presents the conclusion.

II. Related Work

There are a lot of tools and approaches for big data reduction. Some of these methods are based on *Network Theory* [10], *Compression* [24]-[25] and others based on *Dimension Reduction* [11][12][26].

The topological properties of the networks have been used to model big datasets and study their structure that faced challenges due to the datasets complexity and dependencies between their parts. Defining the models based on such data properties makes it hard to understand the data and to produce useful information due to their complexity and the data inconsistencies. Trovati [10] introduced a big data analytics tool which allows to extract useful data and to obtain in-depth intelligence from such different big datasets.

Jalali and Asghari [24] introduce a lossy image compression that reshapes the image. It depends on the idea that if the image is sampled in a way that is the same in all cases and at all times, then the sharp features have a higher sampling density than the rough ones. This method is claimed to be applicable for big data compression.

Yang et al. [27] present a solution that makes it possible for the compression method to compress the data efficiently. The solution is based on applying the clustering method to the datasets (input data). It divides the data into several different clusters(groups), and then compress the data according to the assigned clustering information.

TABLE I. GraphJ Environment Variables.

Key	Data Type	Default Value
MYSQL_HOST	String	localhost
MYSQL_PORT	Integer	3306
MYSQL_USER	String	root
MYSQL_PASS	String	root
MYSQL_DB	String	null
QUERY_LIMIT	Integer	1000

Dynamic Quantum Clustering (DQC) is a method that works with big and multi-dimensional data. Weinstein et al. [11] conduct studies that show how DQC works for big real-world datasets that come from five different domains, namely "x-ray nano-chemistry, condensed matter, biology, seismology, and finance". These studies show how DQC help at extracting meaningful data that contain important information. They claimed that this method establishes important results that show how complex datasets contain various different structures that might be missed by the other clustering techniques.

To our knowledge, there are currently no tools or approaches to reducing big data complexity using a graph database or knowledge graphs.

III. GraphJ Tool

GraphJ is a standalone GUI based application written in JavaFX [28] and based on Spring Framework [29] for resource management. It is built to convert a MySQL database [20] into a Neo4J graph database [21]. To convert the database, GraphJ performs the following activities (see Figure 2):

1) **MySQL host**
GraphJ requires a live MySQL host to read the database schema from. There is no need to specify a database in that host because GraphJ will inspect all the databases. The connection is established to the host using MySQL Java database connectivity (JDBC) driver [30]. JDBC may not work on remote hosts due to remote direct access regulations, as most of the databases do not allow direct access to the database.
The connection is made using an interface called *DBConnection*. DBConnection contains abstract methods that allows to extract(*host, port, username, password, and database*). The *database* is used only to define the default database for the connection. To set all the required data, GraphJ reads these properties from the environment variables (see Table I).

2) **Inspect the schema**
GraphJ uses a module called *SchemaInspector*, which provides all the details about a schema under inspection. *SchemaInspector* requires an object of *DBConnection* in order to decide which *host, port, username, and password* that it will be dealing with. However, it does not require a *database* from the *DBConnection* object. It inspects all the databases on that host, as database selection is done later.

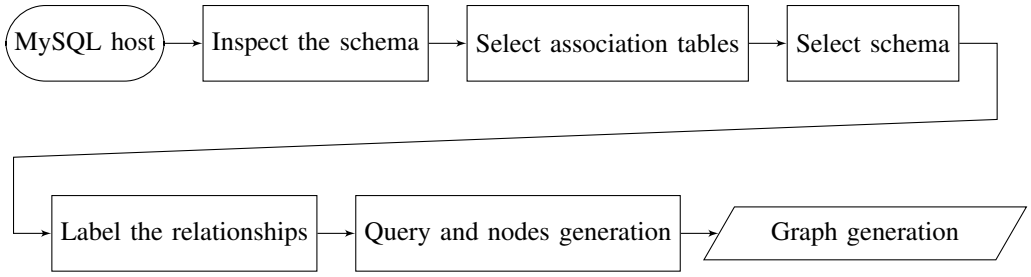

Figure 2. GraphJ flow chart.

SchemaCrawler[31] is used to read the structure of all databases on the provided host. SchemaCrawler is used, although schema can be inspected using *java.sql.DatabaseMetaData*, because it gives a full description of each table, name, columns, primary keys, foreign keys.

3) **Select association tables**
Association or join tables are tables that represent *many to many* relationships between two tables, by using the primary key of each table as foreign keys in an association table. Such tables should be referenced by the user, as they can not be identified programmatically. These tables should be referenced because they will be treated in a different way than *one to one* or *one to many* relationships, since association tables should be omitted and replaced with the original tables.

4) **Select schema**
A single schema should be selected with association tables referenced.

5) **Label the relationships**
Each relationship, including *many to many*, between the tables should be labeled in order to name the relationships between the nodes of the graph.

6) **Query and nodes generation**
GraphJ uses a module called *MySqlConnector* which requires a *DBConnection* in order to establish a connection and execute queries. The queries are executed using *java.sql.Statement*. The reason behind using it instead of *Hibernate* [32] is because HQL Queries suffer from performance degradation (because it should convert HQL to SQL [33], although the performance degradation is trivial, it becomes more tangible when executing queries repeatedly). However, even with SQL, the performance degrades if the database has massive records. To overcome this issue, the queries retrieve limited results. The limitation is read from an environment variable with the key *QUERY_LIMIT*. Inside *MySqlConnector* there are 3 main queries:

- SELECT * FROM table
  table is the table under processing.

  Select all the rows from that table, in order to create a node for each row. After the query is executed, all the attributes are inserted into the node being created. These attributes can be mapped because it is possible to know each column in each table with the help of *SchemaInspector*, each column name is the key. Attributes with a value of *null* are ignored.

- SELECT * FROM r_table WHERE f_k=o_pk
  r_table is the foreign key table.
  f_k is the foreign key in *r_table*.
  o_pk is the primary key in the table that has a relationship with *r_table*.

  Select all the rows from that table that match the foreign in the original table.

- SELECT * FROM j_table INNER JOIN r_table on r_pk=r_fk WHERE o_fk = o_pk
  j_table is the association table
  r_table is the foreign key table.
  r_pk is the primary key in the original table under processing.
  r_fk is the foreign key in *r_table*.
  o_fk is the foreign key reference to the original table under processing.
  o_pk is the primary key in the table that has a relationship with *r_table*.

  Select all the rows from that table that match the foreign in the original table.

Queries are generated based on the selected schema and the relationships between the tables.

7) **Graph generation**
Once the nodes with their relationships are created, the last step is to flush these nodes into the Neo4J database. GraphJ tries to connect to an already instantiated Neo4J database. It read the database path from an environment variable with key *NEO4J_DB*. This database's host should be stopped in order to establish a connection, as it cannot establish two connections at the same time.

IV. Experimental Evaluation

In this section, experimental evaluation details, goals, and results of GraphJ are presented.

 ISBN: 978-1-61208-700-9

TABLE II. SUBJECT DATABASES.

Database	# tables	# all relationships	# M2M relationships	# records
World X	3	2	0	5411
Sakila	16	22	2	47271
Employees	6	6	2	3911245

TABLE III. AVERAGE TOTAL EXECUTION TIMES.

Database	Query Limit					
	500	1000	1500	2000	2500	3000
World X	16 s	26 s	33 s	40 s	50 s	60 s
Sakila	1.5 m	4 m	5.9 m	7.8 m	9.8 m	11.8 m
Employees	17 s	49 s	1.7 m	3.1 m	4.5 m	6.6 m

A. Goals

The goal was to evaluate the performance of GraphJ. Hence, and to cover all the possible cases, GraphJ has been run on two different databases, with different run configurations. In the end, the total execution time of each one of the run configurations has been compared.

B. Subject Databases

For the evaluation, MySQL sample databases [34] were used. Those databases are well known and widely used for testing MySQL queries. Three of the databases were used(see Table II):

- **World X** [35]: Provided by Oracle [36], it has a set of tables containing information on the countries and cities of the world.
- **Sakila** [37]: Provided by Oracle [36], it is designed to represent a DVD rental store. This database borrows film and actor names from the Dell sample database [38].
- **Employees** [39]: Originally developed by Patrick Crews and Giuseppe Maxia and provides a large set of data that consists of 4 million records.

C. Configuration

GraphJ ran on 64-bit Linux machine with Intel Core i5-4200M and 8 GBs of memory. The MySQL and Neo4J databases were located on *localhost* on ports 3306 and 7474. The performance can be affected by the nature of the database and the query limitations, for that, the study tried to simulate real-life cases.

D. Results

The node generation and relationship mapping are done correctly. However, the performance is questionable, thus, GraphJ ran on the three databases with query limited from 500 to 3000 increasing by 500 each time. After running the tool on the three databases (10 times for each query limitation value). The results showed that the performance is significantly affected by the number of tables and relationships, however, the number of rows does not have much effect. The average total execution times are shown in Table III.

V. CONCLUSION

This paper presents a big data complexity reduction (called GraphJ). GraphJ reads a MySQL database and maps its records to nodes in order to insert them into a Neo4J database. The tool converts a relational database into a graph database, which reduces the complexity, as graph databases facilitate the act of unlocking knowledge patterns [40].

Conversion is done by inspecting the schema of a provided relational database, including table names, column names, and column types. Then the relationships between existing tables are inspected, and the user is asked to label these relationships. After that, the queries are generated using the schema data inspected earlier. Finally, the queries are executed to generate nodes in order to be inserted into the graph database using the retrieved data. The relationships between these nodes are created based on the names the user provided. This results in a set of nodes connected together. These nodes are then inserted into a provided graph database.

To assess the effectiveness of the proposed tool, a case study was conducted. Three MySQL sample databases (World X, Sakila and Employees) were used. GraphJ ran with query limitation starting with 500 records and increasing by 500 each time until reaching 3000 records. Each relational database ran 10 times for each query limitation value. The tool was able to convert all databases into graph databases correctly. However, the results showed that the performance is significantly affected by the number of tables and relationships in the relational database.

Following are various opportunities in order to improve the proposed tool:

- Experiment: larger experiments can be performed on the proposed tool to further assess and evaluate its effectiveness.
- Database support: there are a lot of databases available and are widely used. However, the current implementation of GraphJ only supports MySQL and Neo4j. The tool can be extended to support more database implementations.

GraphJ is built using JavaFX with Spring Framework. The tool with the source code is available on GitHub repository: https://github.com/AbdullahAsendar/GraphJ.

 ISBN: 978-1-61208-700-9

REFERENCES

[1] B. Logica and R. Magdalena, "Using big data in the academic environment," Procedia Economics and Finance, vol. 33, 2015, pp. 277–286.

[2] G. W. X. Wu, X. Zhu and W. Ding, "Data mining with big data," IEEE Trans. on Knowl. and Data Eng., vol. 26, no. 1, Jan. 2014, pp. 97–107. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2013.109

[3] T. Segaran and J. Hammerbacher, Beautiful data: the stories behind elegant data solutions. " O'Reilly Media, Inc.", 2009.

[4] D. Che, M. Safran, and Z. Peng, "From big data to big data mining: challenges, issues, and opportunities," in International Conference on Database Systems for Advanced Applications. Springer, 2013, pp. 1–15.

[5] A. A. Tole et al., "Big data challenges," Database Systems Journal, vol. 4, no. 3, 2013, pp. 31–40.

[6] T. Rabl, S. Gómez-Villamor, M. Sadoghi, V. Muntés-Mulero, H.-A. Jacobsen, and S. Mankovskii, "Solving big data challenges for enterprise application performance management," Proc. VLDB Endow., vol. 5, no. 12, Aug. 2012, pp. 1724–1735. [Online]. Available: http://dx.doi.org/10.14778/2367502.2367512

[7] Y. Zhai, Y.-S. Ong, and I. W. Tsang, "The emerging" big dimensionality"," IEEE Computational Intelligence Magazine, vol. 9, no. 3, 2014, pp. 14–26.

[8] J. Fan, F. Han, and H. Liu, "Challenges of big data analysis," National science review, vol. 1, no. 2, 2014, pp. 293–314.

[9] M. H. U. Rehman, C. S. Liew, A. Abbas, P. P. Jayaraman, T. Y. Wah, and S. U. Khan, "Big data reduction methods: A survey," Data Science and Engineering, vol. 1, no. 4, 2016, pp. 265–284.

[10] M. Trovati, "Reduced topologically real-world networks: a big-data approach," International Journal of Distributed Systems and Technologies (IJDST), vol. 6, no. 2, 2015, pp. 13–27.

[11] M. Weinstein, F. Meirer, A. Hume, P. Sciau, G. Shaked, R. Hofstetter, E. Persi, A. Mehta, and D. Horn, "Analyzing big data with dynamic quantum clustering," arXiv preprint arXiv:1310.2700, 2013.

[12] D. Feldman, M. Schmidt, and C. Sohler, "Turning big data into tiny data: Constant-size coresets for k-means, pca and projective clustering," in Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms. SIAM, 2013, pp. 1434–1453.

[13] Y. Fu, H. Jiang, and N. Xiao, "A scalable inline cluster deduplication framework for big data protection," in Proceedings of the 13th international middleware conference. Springer-Verlag New York, Inc., 2012, pp. 354–373.

[14] W. Xia, H. Jiang, D. Feng, and Y. Hua, "Silo: A similarity-locality based near-exact deduplication scheme with low ram overhead and high throughput." in USENIX annual technical conference, 2011, pp. 26–30.

[15] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," Mobile Networks and Applications, vol. 19, no. 2, 2014, pp. 171–209.

[16] A. C. Wilkerson, H. Chintakunta, and H. Krim, "Computing persistent features in big data: A distributed dimension reduction approach." in ICASSP, 2014, pp. 11–15.

[17] A. Singhal, "Introducing the knowledge graph: things, not strings," Official google blog, 2012.

[18] O. Corby and C. F. Zucker, "The kgram abstract machine for knowledge graph querying," in Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on, vol. 1. IEEE, 2010, pp. 338–341.

[19] J. Hurwitz, A. Nugent, F. Halper, and M. Kaufman, Big Data For Dummies, 1st ed. For Dummies, 2013.

[20] A. MySQL, "Mysql," 2001.

[21] N. Developers, "Neo4j," Graph NoSQL Database [online], 2012.

[22] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins, "A comparison of a graph database and a relational database: a data provenance perspective," in Proceedings of the 48th annual Southeast regional conference. ACM, 2010, p. 42.

[23] H. Huang and Z. Dong, "Research on architecture and query performance based on distributed graph database neo4j," in Consumer Electronics, Communications and Networks (CECNet), 2013 3rd International Conference on. IEEE, 2013, pp. 533–536.

[24] B. Jalali and M. H. Asghari, "The anamorphic stretch transform: Putting the squeeze on big data," Optics and Photonics News, vol. 25, no. 2, 2014, pp. 24–31.

[25] K. Ackermann and S. D. Angus, "A resource efficient big data analysis method for the social sciences: the case of global ip activity," Procedia Computer Science, vol. 29, 2014, pp. 2360–2369.

[26] C.-J. Hsieh, M. A. Sustik, I. S. Dhillon, P. K. Ravikumar, and R. Poldrack, "Big & quic: Sparse inverse covariance estimation for a million variables," in Advances in neural information processing systems, 2013, pp. 3165–3173.

[27] C. Yang, X. Zhang, C. Zhong, C. Liu, J. Pei, K. Ramamohanarao, and J. Chen, "A spatiotemporal compression based approach for efficient big data processing on cloud," Journal of Computer and System Sciences, vol. 80, no. 8, 2014, pp. 1563–1583.

[28] J. Clarke, J. Connors, and E. J. Bruno, JavaFX: developing rich Internet applications. Pearson Education, 2009.

[29] R. Johnson, J. Hoeller, A. Arendsen, and R. Thomas, Professional Java development with the Spring framework. John Wiley & Sons, 2009.

[30] G. Hamilton, R. Cattell, M. Fisher et al., JDBC Database Access with Java. Addison Wesley, 1997, vol. 7.

[31] D. O'Neill, "Id3. org," Sualeh Fatehi." SchemaCrawler". SourceForge.[Online]. Available: http://www. its. bldrdoc. gov/fs-1037/fs-1037c. htm (visited March.

[32] C. Bauer and G. King, Java Persistance with Hibernate. Dreamtech Press, 2006.

[33] ——, Hibernate in Action. Greenwich, CT: Manning, 2005. [Online]. Available: http://www.amazon.com/Hibernate-Action-In-Christian-Bauer/dp/193239415X

[34] "MySQL sample databases," https://dev.mysql.com/doc/index-other.html, last accessed: February 9, 2019.

[35] "World X sample database," https://dev.mysql.com/doc/world-setup/en/, last accessed: February 9, 2019.

[36] K. Loney, Oracle Database 10g The Complete Reference, ser. Oracle Press. McGraw-Hill Education, 2004. [Online]. Available: https://books.google.jo/books?id=qMk3xkkIv0QC

[37] "Sakila sample database," https://dev.mysql.com/doc/sakila/en/, last accessed: February 9, 2019.

[38] "Dell dvd store database," https://linux.dell.com/dvdstore/, last accessed: February 9, 2019.

[39] "Employees sample database," https://dev.mysql.com/doc/employee/en/, last accessed: February 9, 2019.

[40] L. Bellomarini, G. Gottlob, A. Pieris, and E. Sallinger, "Swift logic for big data and knowledge graphs," in Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, 2017, pp. 2–10. [Online]. Available: https://doi.org/10.24963/ijcai.2017/1

 ISBN: 978-1-61208-700-9