ADVANCED DATABASE SYSTEMS (COMP90050 2022 WINTER)

THE UNIVERSITY OF MELBOURNE

GROUP 29 - SURVEY REPORT

Graph Data Management

Group members

Yu Yan	Yuexin Li	Jieyun Peng	Shengqi Ma
1005241	959634	1174519	1262480

Abstract

Graph Database(GDB) uses graph structure to store data as “nodes” and the relationship between nodes as “edges”. It is increasingly used in business environments since it has a flexible and agile structure, also it can reveal the hidden relations between data which is helpful for analysis. The use-cases include E-commerce, social media and supply chain management. This survey provides a review of literature on Graph Data Management over the period 1990-2020. It includes four parts: graph database queries, graph database indexing, graph database system and applications of the graph data management.

Keywords: Graph data; Graph data management; Graph queries; Graph indexing; Graph databases; Survey

July 26, 2022 (Winter 2022)

1 Introduction to the Topic Area

Graph data management focuses on the study and development of well-defined theory and sophisticated technology for storage, processing, and analysis of massive graph data. Graph data management includes two topics: one is graph database models, which concentrate on a conceptual level, whereas the other one is graph data management systems that deal with real-life graph data problems through efficient and direct methods.

A graph database model provides the theoretical basis for a practical database system, it uses graph form to represent the actual data and model any relationships among the entities. A graph database model has three key components: data structures, data manipulation and query, and integrity constraints. Data structures transform the unstructured data into a data graph and determine what graph type is applied. Data querying can be considered a collection of operations that can be used to access data. The basis of these operations hinge on the characteristics of graph data, for example, pathway and graph pattern. Similar to the traditional database model, integrity constraints are the protocols used to guarantee data consistency.

The graph database management system includes two parts: graph databases, and graph-processing frameworks[1]. Graph databases can be native or non-native. This survey will only focus on the native graph databases. The graph database is a graph data management tool that is implemented under the rules of database systems, such as data independence and consistency. The current popularized graph databases such as Neo4j and InfiniteGraph are designed in similar system structures: database language or API supported for data querying, database engine for storing data, indexing tool, and user interfaces for database management.

This survey will discuss the topic of graph database management by concentrating on four aspects: classification of graph queries that are fundamental for a graph database, analysis of indexing methods, comparison of current graph databases, and the applications of graph database management in different industries.

2 Related Work Details

2.1 Graph Database query

Graph query aims to let the computers ‘view’ the graph and then do some actions on it according to the query constraints. We summarized four types of queries that are essential in graph databases.

Adjacency Queries

Nodes/edges queries are the primary notion of the adjacency queries. Finding if two nodes are adjacent and connected by the same edge, or two edges are adjacent and connected to the same node, can test if two nodes/edges are adjacent. However, there is a challenge if the graph data is very sparse[1]. Another typical but more complex queries are neighborhood queries. For example, the k-neighborhood problem will find all neighbors of a node which is reachable through the path of the K-edge. Adjacency queries can solve many problems. Such as recommendation systems to find the "neighborhood" of a user with similar interests, information retrieval to do the web ranking, and so on[3]. In the social media area, there is a classical theory called the six-degrees-of-separation theory, can be validated by adjacency queries. It means that a person can be connected to any other person in the world with no more than 5 intermediates.

Pattern Matching Queries

Pattern matching queries are usually defined as subgraph isomorphism, which will looking through the data and detect all subgraphs which isomorphic with a specific given graph pattern, usually a simple graph with some of the nodes/edges labeled as variables. So it can indicate the unknown data and clarify the query output. Compared to the subgraph search that checks the subgraph isomorphism, pattern matching queries are more flexible because they emphasize the connection between vertices[7]. Pattern matching queries mainly solved two problems: graph isomorphism problem with unknoen computational complexity, and an NP-complete (nondeterministic polynomial time complete) problem called subgraph isomorphism problem[3]. Its practical applications include bio-informatics that can find patterns on protein surfaces to identify proteins with similar functions[4], pattern recognition (PR) problems [5], and so on.

Reachability Queries

A very characteristic tasks in the graph database is checking if two nodes are connected by a path, which is called 'reachability of information'. This kind of query is characterized by path and traversal problems that allow some nodes and edge constraints. The expression of reachability queries is Regular Path Query (RPQ). It will identify a path and detect all pairs of nodes connected by it. In this path, the concatenation of labels will meet the regular expression(τ). Detailed explanation can be found in [32].

The challenge for path problems is the computational complexity, it needs to search through sizable data space. For instance, find the simple paths (paths with no repeated nodes) is an NP-complete problem. Reachability queries can be used in many areas, especially used as real-life graph queries to acquire the shortest route between two nodes (shortest path problem). In social networks, it can be adopted to find users with similar interests. However, compared to pattern matching queries which are looking for a specific pattern in the graph, it is less informative[6].

Analytical Queries

Analytical queries do not consult the graph structure, it is also called summarization queries since they can be used to summarize the query result and usually return a single value via some special operators. It includes some aggregate functions such as average, minimum, and count. Simple analytical queries can output the min/max degree in the graph, calculate the degree of a node by finding how many neighbors the node has, the order of the graph by finding the numbers of nodes on it, and so on.

Complex analytical queries are used for graph mining and analysis. For example, the characteristic path length graph algorithm can calculate the average shortest path length in the networks. Also, complex analytic queries facilitate complex algorithms with large graphs as input, so it is specially used for graph-processing frameworks[1].

2.2 Graph Database Indexing

General Indexing

There are various graph indexing technologies being proposed to accommodate the diversity of the data storage and access scenarios, including early proposed ones that targeted at general graph models. Among them GraphGrep[8] introduced a path-based indexing model that handles each query by enumerating paths in a graph, which correlates the indexing time with the path length instead of graph size. Unlike the path-based indexing mechanism, Gindex[9] introduced data mining technology into graph indexing to support graph-based indexing, i.e. Frequent SubGraphs (FGs) for more stable database updates and better query performance. To reduce the number and size of graph-based indices, only frequent substructures are chosen for indexing, which is combined with two technologies including size-increasing support constraint and discriminative fragments for size compression. In comparison, FG-Index[10] introduced FG-query and IFG-query as an optimization in the aspect of candidate verification, where the queries to FGs are directly answered without candidate verification, and other queries will produce a relatively exact answer set. Moreover, to reduce the size of the index set, a smaller set of representative FGs termed -Tolerance Closed Frequent Graphs (-TCFGs) is introduced.

As is compared in the following table1, Gindex and FG-Index is superior than GraphGrep since they both introduced data mining to get substructures for indexing with the issue of increased index size tackled. Moreover, FG-Index has provided innovative solutions on index construction and query processing and thus outperforms GIndex on these aspects.

	GraphGrep	**Gindex**	**FG-Index**
indexing approach	path-based	graph-based	graph-based
index construction	enumeration of paths	data mining to find substructure	data mining combined with BuildIndex[10]
index size compression	not required	required	required
subgraph matching	Ullman algorithm	GSpan algorithm	FG-Query and IFG-Query[10]

Table 1: A comparison between GraphGrep, Gindex and FG-Index

Multi-dimensional data indexing

Another category of popular indexing model suitable for multi-dimensional data is a subtree structure comparable to R-tree[11], which is used to store spatial objects and perform similarity queries including K-NN and range query. The Related works are compared in the following table 2, in which M-tree[11] is an indexing the organized indices into a hierarchical tree in a dynamic way, with triangle inequality property used for routing. According to experimental data provided by P. Ciaccia et al., M-tree has outperformed R-tree in both index construction costs saving and range queries performance. By contrast, Closure-tree[12], which is proposed by H. He et al., introduced a form of bounding box for graphs termed graph closures to replace MBR. Closure-tree realized effective routing though histogram and pseudo subgraph isomorphism, and is the first index structure that also supports subgraph queries[12], which is proved to outperform that of the GraphGrep.

	R-tree	**M-tree**	**Closure-tree**
node type	MBR	Database Graph	Graph Closure
path pruning	find intersected items given a query point	triangle inequality property[11]	histogram and pseudo subgraph isomorphism[12]
query type	similarity queries	similarity queries	subgraph & similarity queries

Table 2: A comparison between R-tree, M-tree and Closure-tree

Semi-structured data indexing

For semi-structured databases composed of hierarchically nested elements, data can be retrieved through XML structural summaries[8]. DataGuide and 1-index[14] are two previous proposals based on bisimilarity, in which two nodes with identical paths coming to them are considered the same, whereas A(k)-index uses the concept of k-bisimilarity[15] in order to reduce the size of the index structure. It relaxes the equivalence condition by considering two data nodes with identical k paths coming to them as the same. Since the notion of bisimilarity is to precisely encode the graph structure, it is inefficient for DataGuide and 1-index to handle complex semi-structured data, whereas A(k)-index has took advantage of k-bisimilarity to speed up the evaluation, but at the same time, extra validation is required when query path is longer than k. D(k)-index[13], as a robust summary structure for semi-structured data, has a notable advantage of adaptively fitting the query load and source data. While developed based on 1-index and A(k)-index, it outperformed its predecessors both in query load sensitivity and update efficiency.

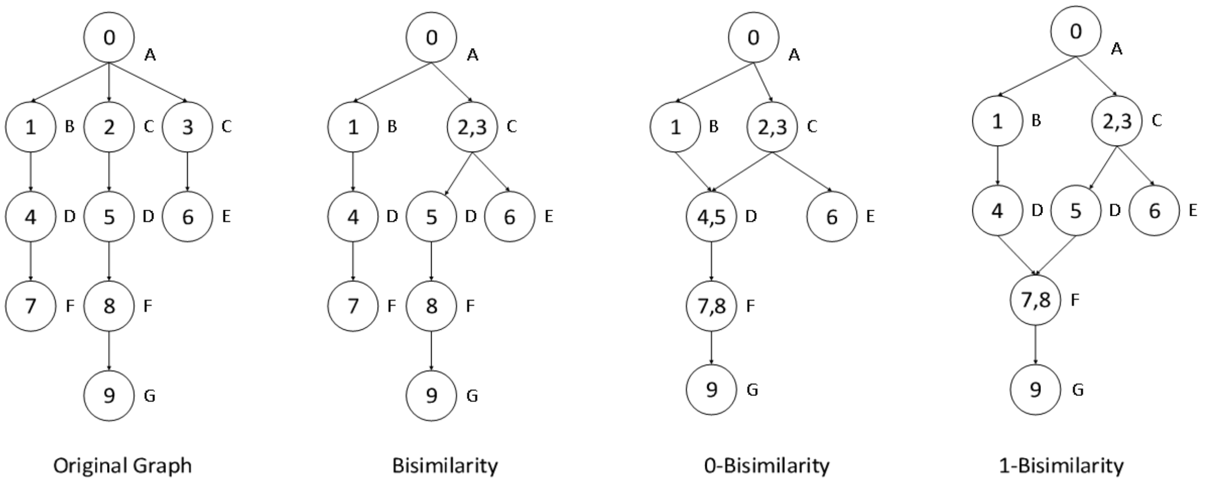


Figure 1: An illustration of bisimilarity and k-bisimilarity

2.3 Graph Database System

AllegroGraph

The AllegroGraph was first implemented in 2004 and was considered as a Resource Description Framework (RDF) database [17]. This graph database is mainly used in businesses to conduct predictive analytics for decision-making from extraordinary complicated and interconnected graph data. One crucial feature of AllegroGraph is that hypergraph is used to allow for greater flexibility and accuracy than a simple graph in the representation of data with more complexity. The data with more comprehensive and accurate expression makes the queries and graph analysis faster and more efficient. These more representative results will better relate to the business issues. Another essential feature is that SPARQL has been a mature query language used in AllegroGraph. SPARQL is optimised to the data with different structured level and support the creation of new queries for the business [18].

Neo4j

Neo4j was released in 2007 and is a disk-based and fully transactional database that stores data in graph form [19]. One of the crucial features is that Neo4j is working based on a property graph model in which the edges come with their own direction arrows, and both edges and nodes can have zero or more labels and can be given any number of properties or attributes [20]. This makes the Neo4j more expressive than the graph databases using a simple graph model. Because of the powerful data representation, Neo4j supports constant time traversals even in a considerable number of graphs for both breadth and depth. This means the complexity of the operation is always O(1), no matter how large the dataset is. In addition, Cypher is a declarative query language and used by Neo4j to create efficient and unique queries for different operations on the graph data [b]. An interesting fact of Cypher is that the process of pattern matching can be visualized.

HypergraphDB

HyperGraphDB was first implemented in 2007 and started from an AI project. HyperGraphDB acted as a component for handling the storage of complex data representation in this project [21]. HyperGraphDB is a graph database that implements hypergraph data model. Hypergraph refers to a concept of graph generalization where the standard edges are replaced by an extended notion of edges named hyperedges. This graph generalization is good at handling the higher-order relationship through reifying every entity in the graph data [21]. Thus, HyperGraphDB is designed for research tasks where a huge amount of graph data needs to be handled, and the complexity of hidden information has to be analyzed [22]. A successful application of this database is in natural language processing. A distributed NLP system named Disko significantly simplifies the task by using the indexing capability of HyperGraphDB [21].

DEX

DEX is a bitmap-based graph database and was first released in 2008. By using the bitmap, a large amount of data can be preserved in a smaller space though applying compression techniques, and binary logic operations may be used to run them with higher efficiency. Thus, DEX is suitable for handling and managing extensive graphs. One important feature of DEX is that queries of DEX are usually implemented by mixing low-level operations in order to make query programming much easier. This is because DEX prefers to produce a more sophisticated algorithm at a higher level [23]. Recently, DEX supports API for several programming languages for graph query implementation and has been applied in many fields such as biology, social networking analysis, and fraud detection [24].

InfiniteGraph

InfiniteGraph was implemented by Objectivity, a business that provides database management technologies. InfiniteGraph was designed to manipulate the graph data, which is large-scaled, interconnected, and distributed from different locations. Hence, InfiniteGraph is usually used by large businesses and governments. InfiniteGraph is also known for its capabilities of graph queries. It can intake enormous and complicated data and create the graph for real-time graph queries concurrently. Additionally, building unique queries that improve analysis performance and accuracy is also supported in InfiniteGraph [25].

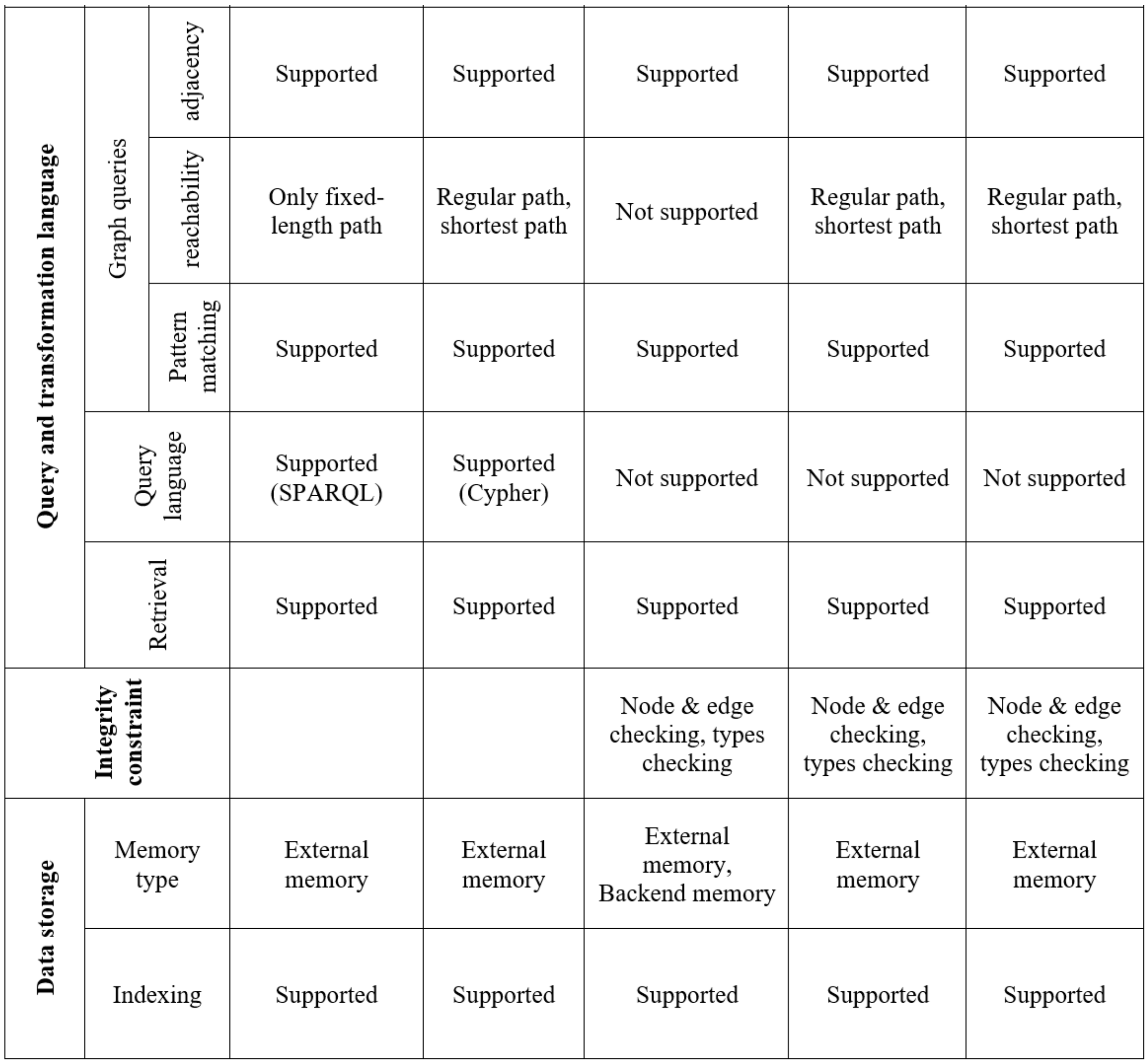
Comparisons

The comparison of current graph databases is conducted based on the following components, data structure, graph query, query language, integrity constraints, and data storage. According to the classification of graph database models [1], four graph data structures are considered: simple graphs, hypergraphs, nested graphs, and property graphs. It is seen from Figure 2 that three out of five databases are based on property graphs, and the inclusion of property or attributes leads to better data retrieval and representation power. The other comparison feature under data structure is to evaluate how expressive the data can be represented. This is assessed by comparing the representation ability in terms of entities, attributes, and relations for both schema and instance level [2].

Data manipulation and queries are critical components in graph databases. It is necessary to compare how well these graph databases support the essential graph queries, adjacency, reachability, and pattern matching. A mature graph database is supposed to be equipped with a powerful and well-defined database language. However, there has not been a formal implementation of a standard database language so far. A substitute that current graph databases provide is APIs for different programming languages [2].

Since real-life graph databases are constantly involved with a considerable amount of data and information, it is crucial to ensure that external memory storage is enabled for the databases in order to handle and manage the data effectively. In addition, indexing is also a critical comparison feature as indexing methods are the basis for enhancing the performance of data retrieval.

			AllegroGraph	**Neo4j**	**HyperGraphDB**	**DEX**	**InfiniteGraph**
Data Structure	Graphs		Hypergraphs	Attributed graphs	Hypergraphs	Attributed graphs	Attributed graphs
	Data Representation	Schema level	Not supported	Not supported	Node types, Relation types	Node types, Relation types	Node types, Relation types
		Instance level	Value nodes, Simple relations	Object nodes, Value nodes, Object relations, Simple relations	Value nodes, Simple relations, Complex relations	Object nodes, Value nodes, Object relations, Simple relations	Object nodes, Value nodes, Object relations, Simple relations



Query and transformation language	Graph queries	adjacency	Supported	Supported	Supported	Supported	Supported
		reachability	Only fixed-length path	Regular path, shortest path	Not supported	Regular path, shortest path	Regular path, shortest path
		Pattern matching	Supported	Supported	Supported	Supported	Supported
	Query language		Supported (SPARQL)	Supported (Cypher)	Not supported	Not supported	Not supported
	Retrieval		Supported	Supported	Supported	Supported	Supported
Integrity constraint					Node & edge checking, types checking	Node & edge checking, types checking	Node & edge checking, types checking
Data storage	Memory type		External memory	External memory	External memory, Backend memory	External memory	External memory
	Indexing		Supported	Supported	Supported	Supported	Supported

Figure 2: A Comparison of Databases

2.4 Graph Database Application

Applications of Graph Data Management in Different Fields

The paper “Graph data management for molecular and cell biology” illustrates applications of graph data management in biology. Network data in biology can be represented as graphs, researchers query and manipulate network data through graph data management.

Systems Biology Graph Extender (SBGE) was introduced to achieve graph data management, SBGE enables Structured Query Language (SQL) queries over biological networks and other graph structures. As demonstrated in [26] SBGE has three main characteristics, firstly SBGE views “a graph as a first-class SQL data type” [26,p.549], so developers use SQL to manipulate graph data in the same manner they manipulate strings and numbers; secondly, SBGE transforms output of SELECT query into graphs; thirdly, SBGE turns a graph into similar data structures.

Authors [26] showed implementations of SBGE, graphs are loaded into SQL by SBGE, and SQL developers create tables to store graph data; SBGE also constructs graphs through a function, the function takes information of graphs as input and generates a one-edge graph for each row of the table; furthermore, SBGE stores graphs in binary format to ensure readability, operations retrieve edge and vertice information, as well as other relevant graph data thanks to binary format.
SBGE manipulates graphs in various ways. For example, as shown in [26], SBGE combines numerous small graphs into one major graph, which rules out irrelevant edges and nodes from small graphs, thus increasing efficiency of the management system. The query over graphs consists of three sections, the first step gathers necessary edges, the second compiles edges into graph instances, the last step returns edge tables, such a table is ready for queries.

The paper "April: An Automatic Graph Data Management System Based on Reinforcement Learning" shows reinforcement learning in graph data management, which is a novel way to tackle graph data. Reinforcement learning enables decision making to be automatic and adaptive, April is a graph data management system [27].
April has three primary goals in data management, which are effectiveness, automation and customization, effectiveness is achieved through three functionalities, which are storage structure selection, index selection, and query optimization; automatic mode generates automatic query plans given fixed data storage and indices; users has the freedom to choose customized mode to store data and create indices [27].
Storage structure selection provides two modes to store data, which are automatic mode and customised mode [27]. Index selection also provides automatic and customised modes for users, automatic mode determines how indices are built with columns, index types and parameters; customised mode enables users to select favourite indices [27]. Query optimization transforms "input SparQL queries into SQL with UDFs" [27,p3466].

The chapter "High-Performance Graph Data Management and Mining in Cloud Environments with X10" illustrates applications of graph data management in cloud settings. X10 is a Partitioned Global Address Space (PGAS) language, X10 creates expandable systems that store and manage graph data on HPC clouds [28].
ScaleGraph library defines "solid abstractions for large-scale graph processing" [28,p178]. X10 operates on three different scales, which are small, medium and large, we choose a scale based on resource and performance [28]. The library is modeled in object-oriented design technique, and contains six parts including "graph, I/O, generators, metrics, clustering, and communities" [28,p180]. The graph package

contains classes of graph representation, graph interface adopts adjacency list representation of graphs, the library reads and writes graph documents, and the library provides algorithms for scalability evaluation [28].

Acacia is a distributed graph database server developed in X10, it has two components, which are master and worker [28]. Front end commands "list the system statistics, upload/delete graphs, get system statistics, run graph algorithms, etc" [28,p185].

The paper "Knowledge Discovery from Social Graph Data" is social network analysis through graph data management. Big data is all sorts of data from different sources, which is represented as big social graphs [29].

Social networks are treated as graph data, interesting and relevant patterns are drawn from social networks through a knowledge-based system. In a social application such as Facebook, each user is considered a vertice, following or followed relationships between users are edges, any two users can follow each other, so the social graph is directed graph [29].

Social graphs give us lots of valuable information, the number of followers of an entity is determined by calculating the number of arriving edges to the node, the number of people an entity follows is determined by calculating the number of departing edges from the node. Other information such as popularity of a user, number of friends-of-friends of a user and isolated users is also inferred from the social graph [29].

The paper "Graph Databases for Large-Scale Healthcare Systems: A Framework for Efficient Data Management and Data Services" is an application of graph data management in large-scale healthcare systems.

Graph database has three modules: data structure types, query operators, and integrity rules; graph databases store data as graphs and has its unique query language, and integrity rules are determined by graph constraints [30]. Relational databases are mapped into graph databases through "3NF Equivalent Graph" (3EG) transformation. A record is transformed into a node, column name of a table is transformed into a predicate, and a cell is transformed into a value [30].

Queries have three types according to their functionalities: queries for direct retrieval, queries for relation mining, and queries for personalized web services [30]. Queries are conducted to test the performance of relational database management system (RDMS) and graph database management system (GDMS) [30]. Results show RDMS is only faster in some cases, query performance of GDMS becomes slower when the number of nodes increases, and query performance of the RDMS becomes slower when number of joining tables increases [30]. Above all, an ensemble system of RDMS and GDMS is suggested due to lower database complexity and higher data accessibility.

Comparisons between RDMS and GDMS

Ways of storing data between two systems are different, GDMS stores relations between data in data points, RDMS stores relations between data in columns of tables. Two systems process data in various formats, GDMS processes data represented as graphs, whereas RDMS processes straightforward data records with clear relationships. For example, social networks are represented as graphs, RDMS cannot manipulate data from social networks as efficiently as GDMS does due to its 5V's characteristics (velocity, volume, value, variety and veracity) [29]. Directed graphs help determine follow or followed relationships, bi-directed and undirected graphs help determine mutual friendships.

When a query or database is more complex, GDMS is faster than RDMS since the latter spends more time joining tables, GDMS consequently manages large-scale healthcare systems. There is a tremendous gap between data management and data service in RDMS, researchers minimize the gap through GDMS, GDMS decreases complexity of datasets and increases users' accessibility to data [30].

GDMS also has its shortcomings. GDMS is a rather new approach to database management compared to traditional management systems such as RDMS, it may not be as reliable as RDMS. In some cases GDMS generates results faster, but GDMS is outperformed by RDMS in other cases, as shown in the query cases from the applications in healthcare, this is why researchers adopt an ensemble framework that combines RDMS and GDMS to handle healthcare systems [30].

In conclusion, no single database management system is suited for every job, it is optimal that we develop an ensemble framework that integrates the advantages of each database management system.

3 Conclusions and Future Directions

With the advantages of intuitiveness and flexibility for managing complex and inter-related data, graph databases have been widely implemented in contemporary fields including biology, cloud, social network, healthcare, etc. This survey paper presents different field-related management systems for graph data, such as SBGE, April, ScaleGraph, Acacia, which are fundamental to the landing of graph databases in actual application scenarios. In addition, we compared five commonly used graph databases regarding their component designs, and saw their variations on data representation and data retrieval power.

Among all design features of a graph database product, we had a further investigation on effective graph query and indexing approaches, which are applied to support faster data operations. Compared to a DBMS, the query conditions and approaches of graph databases are more diversified, in which the query costs are closely dependent on the optimization of mathematical models concerning shortest path, k-neighborhood, NP-complete problem, etc. It's also hard to prove that one indexing technology is obviously superior to another, since the performance evaluation should be constrained by specific query conditions and data structure.

GDMS is fast developing in industries and academia, and also faces challenges from realistic and theoretical aspects.

All data management systems pursue reliable and flawless queries, GDMS brings fresh air into the progress of developing more sophisticated queries, and researchers are more interested in applying tools from graphs to manage large-scale data [31]. However, it is noteworthy that GDMS cannot handle all data problems alone at present, a combination of graph and traditional management systems is capable of solving complex issues effectively [31].

Properties of graphs are both advantages and disadvantages of GDMS. Development of systems are hindered by various kinds of graphs, each graph takes a form that is beyond the control of researchers, forms of each graph do not adapt to the systems, the systems have to adapt to each graph, so design of the systems face significant difficulties [31]. Furthermore, researchers often process large scale databases, the size and dimensions of a graph get incredibly large and relationships become more complex. As a result, the efficiency of representing and storing graphs in memory or cache decreases, the speed of processing data declines, and performance of queries worsens [31]. These are the problems that developers have to solve.

Graph data management has a tortuous road ahead, but its future is bright.

4 References

[1] Angles, R., Gutierrez, C. (2018). An Introduction to Graph Data Management. In: Fletcher, G., Hidders, J., Larriba-Pey, J. (eds) Graph Data Management. Data-Centric Systems and Applications. Springer, Cham. https://doi.org/10.1007/978-3-319-96193-4_1

[2] R. Angles, "A Comparison of Current Graph Database Models," 2012 IEEE 28th International Conference on Data Engineering Workshops, 2012, pp. 171-177, doi: 10.1109/ICDEW.2012.31.

[3] M. Yannakakis, “Graph-Theoretic Methods in Database Theory,” in Proceedings of the 9th Symposium on Principles of Database Systems (PODS). ACM Press, 1990, pp. 230–242.

[4] X. Wang, "Finding patterns on protein surfaces: algorithms and applications to protein classification," in IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 8, pp. 1065-1078, Aug. 2005, doi: 10.1109/TKDE.2005.126.

[5] Conte D, Foggia P, Sansone C, Vento M (2004) Thirty years of graph matching in pattern recognition. Int J Pattern Recognit Artif Intell 18(3):265–298

[6] Zou, L., Chen, L., Özsu, M.T. et al. Answering pattern match queries in large graph databases via graph embedding. The VLDB Journal 21, 97–120 (2012). https://doi.org/10.1007/s00778-011-0238-6

[7] L. Zou, L. Chen, and M. Tamer Özsu. 2009. Distance-join: pattern match query in a large graph database. Proc. VLDB Endow. 2, 1 (August 2009), 886–897. https://doi.org/10.14778/1687627.1687727

[8] R. Giugno and D. Shasha, "Graphgrep: A fast and universal method for querying graphs," in 2002 International Conference on Pattern Recognition, 2002, vol. 2, pp. 112-115: IEEE.

[9] X. Yan, P. S. Yu, and J. Han, "Graph indexing: a frequent structure-based approach," presented at the Proceedings of the 2004 ACM SIGMOD international conference on Management of data, Paris, France, 2004. Available: https://doi.org/10.1145/1007568.1007607

[10] J. Cheng, Y. Ke, W. Ng, and A. Lu, "Fg-index: towards verification-free query processing on graph databases," presented at the Proceedings of the 2007 ACM SIGMOD international conference on Management of data, Beijing, China, 2007. Available: https://doi.org/10.1145/1247480.1247574

[11] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in Vldb, 1997, vol. 97, pp. 426-435.

[12] H. He and A. K. Singh, "Closure-tree: An index structure for graph queries," in 22nd International Conference on Data Engineering (ICDE'06), 2006, pp. 38-38: IEEE.

[13] Q. Chen, A. Lim, and K. W. Ong, "D (k)-index: An adaptive structural summary for graph-structured data," in Proceedings of the 2003 ACM SIGMOD international conference on Management of data, 2003, pp. 134-144.

[14] T. Milo and D. Suciu, "Index structures for path expressions," in International Conference on Database Theory, 1999, pp. 277-295: Springer.

[15] R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes, "Exploiting local similarity for indexing paths in graph-structured data," in Proceedings 18th International Conference on Data Engineering, 2002, pp. 129-140: IEEE.

[16] C.-W. Chung, J.-K. Min, and K. Shim, "APEX: An adaptive path index for XML data," in Proceedings of the 2002 ACM SIGMOD international conference on Management of data, 2002, pp. 121-132.

[17] D. Fernandes and J. Bernardino, "Graph Databases Comparison: AllegroGraph, ArangoDB, Infinite-Graph, Neo4J, and OrientDB," in Data, 2018, pp. 373-380.

[18] X. Wang and W. Chen, "Knowledge graph data management: Models, methods, and systems," in International Conference on Web Information Systems Engineering, 2020: Springer, pp. 3-12.

[19] N. Patil, P. Kiran, N. Kiran, and N. P. KM, "A survey on graph database management techniques for huge unstructured data," International Journal of Electrical and Computer Engineering, vol. 8, no. 2, p. 1140, 2018.

[20] R. Kumar Kaliyar, "Graph databases: A survey," in International Conference on Computing, Communication Automation, 2015: IEEE, pp. 785-790.

[21] B. Iordanov, "Hypergraphdb: a generalized graph database," in International conference on web-age

information management, 2010: Springer, pp. 25-36.

[22] D. Dominguez-Sal, P. Urbón-Bayes, A. Giménez-Vanó, S. Gómez-Villamor, N. Martínez-Bazan, and J. L. Larriba-Pey, "Survey of graph database performance on the hpc scalable graph analysis benchmark," in International Conference on Web-Age Information Management, 2010: Springer, pp. 37-48.

[23] N. Martinez-Bazan, S. Gomez-Villamor, and F. Escale-Claveras, "DEX: A high-performance graph database management system," in 2011 IEEE 27th International Conference on Data Engineering Workshops, 2011: IEEE, pp. 124-127.

[24] N. Martínez-Bazan, V. Muntés-Mulero, S. Gómez-Villamor, J. Nin, M.-A. Sánchez-Martínez, and J.-L. Larriba-Pey, "Dex: high-performance exploration on large graphs for information retrieval," in Proceedings of the sixteenth ACM conference on Conference on information and knowledge management, 2007, pp. 573-582.

[25] R. F. van der Lans, "InfiniteGraph: Extending Business, Social and Government Intelligence with Graph Analytics," Technical report, 20, 2010.

[26]B. A. Eckman and P. G. Brown, “Graph data management for molecular and cell biology,” IBM Journal of Research and Development, vol. 50, no. 6, pp. 545–560, Nov. 2006, doi: 10.1147/rd.506.0545.

[27]H. Wang et al., “April,” Proceedings of the 29th ACM International Conference on Information Knowledge Management, Oct. 2020, doi: 10.1145/3340531.3417422.

[28]M. Dayarathna and T. Suzumura, “High-Performance Graph Data Management and Mining in Cloud Environments with X10,” Computer Communications and Networks, pp. 173–210, 2017, doi: 10.1007/978-3-319-54645-2_7.

[29]P. Braun, A. Cuzzocrea, C. K. Leung, A. G. M. Pazdor, and K. Tran, “Knowledge Discovery from Social Graph Data,” Procedia Computer Science, vol. 96, pp. 682–691, 2016, doi: 10.1016/j.procs.2016.08.250.

[30]Y. Park, M. Shankar, B.-H. Park, and J. Ghosh, “Graph databases for large-scale healthcare systems: A framework for efficient data management and data services,” 2014 IEEE 30th International Conference on Data Engineering Workshops, Mar. 2014, doi: 10.1109/icdew.2014.6818295.

[31]A. Cuzzocrea and I.-Y. Song, “Big Graph Analytics,” Proceedings of the 17th International Workshop on Data Warehousing and OLAP - DOLAP ’14, 2014, doi: 10.1145/2666158.2668454.

[32] Cruz IF, Mendelzon AO, Wood PT (1987) A graphical query language supporting recursion. In: Proceedings of the international conference on management of data (SIGMOD). ACM, New York, pp 323–330