Distributed Storage and Query for Domain Knowledge Graphs

Xiaohuan Shan[1], Xiyi Shi[1], Wenyuan Ma[2], and Junlu Wang[1](✉)

[1] School of Information, Liaoning University, Shenyang, China
wangjunlu@lnu.edu.cn

[2] School of International Education, Beijing University of Chemical Technology, Beijing, China

Abstract. The development of knowledge graph needs the support of a vast quantity of data. However, the amount of data increases rapidly is placing increasing demands on machines. Centralized data storage requires high-performance hosts to store data, which is costly and have single point of failure. Distributed data storage can reduce the cost of the machine greatly, and there is no single point of failure, but it has requirements for partition and storage of data collection. In the knowledge storage of specific domain, the way of graph data partition and storage vary from the different domain knowledge. To solve the above problems, a scheme of graph partition and distributed storage for domain-specific knowledge graphs is proposed. The proposed graph partition scheme pays attention to the correlation between the data, and divides the nodes affiliated each other into the same or similar partition. A distributed aggregation storage scheme is designed, which makes full use of cluster performance and solves the problem of data consistency during data insertion and update. The proposed distributed storage scheme based on HBase combines Neo4j to realize visual query effectively. Experimental results show the efficiency and the effectiveness of the proposed method in partition time, the number of edge-cut and update time.

Keywords: Knowledge graph · Distributed storage · Graph partition · Visualization

1 Introduction

The rapid development of information and Internet technology has led to the continuous increase in the scale of data and an increasing demand for its application. As a typical application of knowledge data in a specific field, knowledge graphs can further explore the internal connections of various knowledge to infer new knowledge based on the visualization and data analysis [1]. Therefore, in addition to being used in intelligent question and answer system [2], intelligent search system [3] and personalized recommendation [4], knowledge graphs are also widely applied in different domains [5, 6]. For example, using an anti–fraud knowledge graph of credit card application, we can quickly query whether an entity has fraud risks and analyze whether a relationship is suspicious. We can also use an enterprise knowledge graph for enterprise risk assessments. Financial knowledge graphs can be utilized to predict financial risks and investment. However, faced with the explosive growth of data scale, how to effectively store and manage the knowledge graphs will face enormous challenges.

The graph database organizes data by entity dimensions, which can better obtain the attributes of an entity and its relationship with other entities. So it is more efficient for identifying the characteristics of entities. Therefore, the mainstream method of knowledge graph storage is utilizing graph database at present. The centralized storage pattern can manage only one server. Although it is easy to implement, there are problems such as the inaccessibility of data caused by single point failure and the inability to meet the storage requirement of big data. The multi-point backup of distributed storage can effectively solve the impact of single point of failure on data query. The dynamic expansion feature enables the distributed storage pattern with great scalability, which can meet the continuously increase of the amount of data. At the same time, it can also alleviate the excessive processing pressure on a single server, so as to raise the speed of query. However, knowledge graphs in different fields have their own features. If a unified distributed storage model is adopted, the characteristics of the graph cannot be satisfied. So, it is of great significance to design a corresponding distributed storage model for the domain-specific knowledge graph.

To this end, this paper conducts in-depth research on knowledge graphs in specific fields, and proposes a distributed storage and query method for domain-specific knowledge graphs. The main contributions of this patter are as follows:

- A graph partition method based on node density and modularity is proposed, which applies the node density to initially divide the graph, and then use modularity detection to assign the remaining nodes to the more close-knit partitions. So far as possible to ensure that closely connected nodes and their relationships are stored centrally on a server to reduce network communication overhead.
- A distributed aggregation storage mode is designed. This storage schema makes full use of cluster performance, which can effectively reduce the redundancy overhead caused by repeated data storage. At the same time, it solves the problem of data consistency during data insertion and update.
- In order to realize the distributed storage and query of domain-specific knowledge graph, we design to combines HBase distributed database storage and Neo4j visual query effectively. Related entities and relationships in HBase are extracted according to the query semantics from master node. Then the information is imported into Neo4j to form sub-graphs that users concern for visual display.

The rest of this paper is organized as follows. In Sect. 2, we review related works. The details of the storage and query are discussed in Sect. 3 and 4. Experimental results and analysis are shown in Sect. 5. We finally conclude in Sect. 6.

2 Related Work

2.1 Knowledge Graph Storage

The knowledge graph is composed of nodes and relationships. Through the association of different knowledge, a network-like knowledge structure is formed to intuitively model various real-world scenarios. There are two main current mainstream ways to store the knowledge graphs.

The first kind is RDF storage [7]. An important design principle of RDF is that the data should be easy to publish and share. But RDF stores data in the form of triples and does not contain attribute information, meanwhile there are problems such as high update and maintenance costs. The other is graph database storage. Compared with the RDF storage, the graph database is more general. It generally takes attribute graphs as the basic representation form, which realizes the storage of graph data by nodes, edges, and attributes in the graph structure. So it is easier to express real world business scenarios. Typically, open source graph databases are Neo4j [8], Arangodb [9], etc. Since the database itself has provided a perfect graph query language and supports various graph mining algorithms, it has efficient graph query and search functions. However, the distributed storage of graph database is expensive. With the growth of graph scale, distributed storage has become the mainstream. How to reduce the overhead of graph database distributed storage has become one of the research hotspots of knowledge graph storage.

2.2 Graph Partition

The knowledge graph stored in the graph database is to represent the relationship structure of a certain thing in the form of a graph, which is composed of nodes and edges and attributes. Therefore, the data partition problem of knowledge graph can be transformed into graph partition problem.

The problem of graph partition has been extensively studied in many application fields. Finding the optimal partition of graphs is an NP-complete problem [10], and many solutions have been proposed. In recent years, several multi-level partition algorithms have been proposed [11–13]. The idea is to coarsen large graphs into small ones, and then apply classic algorithms on the small graph. However, the optimal partition on the roughened graph may not be suitable for realistic scale-free graphs. In addition, the coarsening algorithm is very expensive and cannot be scaled on a graph of large-scale nodes. To improve scalability, several parallel partitioning solutions have been proposed. The JA-BE-JA algorithm [14] combines simulated annealing algorithm and data migration algorithm for edge cutting and point cutting, and uses the heuristic method centered on vertex to solve the equilibrium graph partitioning problem by processing each vertex of the graph data in parallel. DFEP [15] randomly assigns a partition for each node and gives an initial funding. Each node uses its funding to find its neighbors. The principle of random selection has a certain influence on the stability of the partition quality.

2.3 Semantic Query

In this paper, the query processing techniques for knowledge graphs on existing graph databases are investigated. Neo4j is the current mainstream graph database system, which uses Cypher language to perform centralized query of graphs. The main clause in Cypher is the MATCH clause, which indicates the matching of a graph pattern on the property graph. And an empty table is added to the matching result to form a new table as a result. With the scale of graphs keeping growing, the index and maintenance efficiency of centralized processing methods is greatly reduced and replaced by distributed query processing technology. Compared with the centralized processing method, distributed query processing assigns the sub-graphs among different compute nodes, decomposes the query graph according to the data division, and realizes the parallel query on each compute node. GeaBas graph database can query the relationship information quickly and efficiently through the unique data organization method and distributed parallel computing algorithm. JanusGraph is a distributed graph database, which is an attribute graph database system that supports the TinkerPop framework. Its query mechanism is Gremlin graph traversal language.

3 Distributed Storage Based on HBase

3.1 Graph Partition Based on Node Density and Modularity

In order to ensure the high-cohesion and low-coupling of graph partition, closely related nodes and relationships are divided into a sub-graph to minimize network communication overhead. This paper proposes a graph partition method based on node density and modularity (GP-NDM). Because of data increasing rapidly and complex relationships in specific fields, GP-NDM first selects hotspot nodes (nodes with large degrees) as the initial partition according to the node density. Then it uses the modularity to detect the partition effect, and the graph partition is completed by dynamically adding nodes. The algorithm in this paper effectively avoids the poor partition due to the high degree of data aggregation.

If there are N storage servers, first the N nodes with the largest degrees and no direct relationship (i.e., the direct relationship (v_i, v_j) is not allowed, the indirect contact (v_i, v_k), (v_k, v_j) exists) are selected randomly. For all the remaining nodes, GP-NDM calculates the modularity when joining a partition and adds each node to the server that makes the modularity larger, so as to ensure the tightness of internal connections in the sub-graph. The modularity formula is given as follows:

$$Q = \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j) \quad Q \in [-1, 1) \; \delta(u, v) = \begin{cases} 1 & when \;\; u == v \\ 0 & else \end{cases} \tag{1}$$

A_ij represents the edges weights between two nodes. Since in this paper the edge in the knowledge graph is a relation, rather than a edge with weight, the weight of the edge is regarded as value 1. The k_i indicates the degree of node i. c_i represents the community to which the node i belongs. The m is the sum of the weights of all edges. The formula of modularity can be simplified as:

$$\begin{aligned} Q &= \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j) = \frac{1}{2m}\left[\sum A_{ij} - \frac{\sum_i k_i \sum_j k_j}{2m}\right]\delta(c_i, c_j) \\ &= \frac{1}{2m}\sum_c\left[\sum cin - \frac{\left(\sum tot\right)^2}{2m}\right] = \sum_c\left[\frac{\sum cin}{2m} - \left(\frac{\sum tot}{2m}\right)^2\right] \end{aligned} \tag{2}$$

$\sum cin$ represents the sum of weights in community c. $\sum tot$ represents the sum of the weights connected to the nodes within the community c. As the value of modularity, Q represents the degree of tightness of nodes within the community, the larger the modularity, the better the community partition. An example is illustrated in Fig. 1.

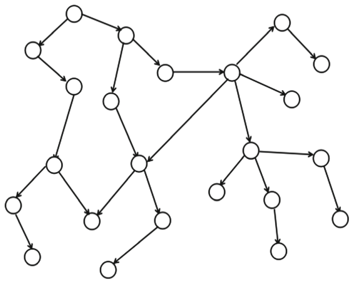


Fig. 1. The directed data graph G

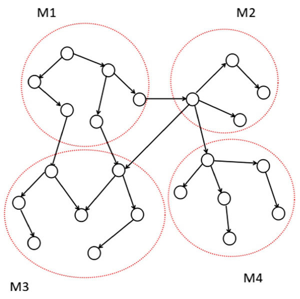


Fig. 2. The partition result of G

Suppose that a node in graph is assigned to m_1, it can be seen from formula of modularity, the modularity degree before moving is

$$Q_1 = \left[\frac{\sum cin}{2m} - \left(\frac{\sum tot}{2m}\right)^2\right] + \left[\frac{0}{2m} - \left(\frac{k_i}{2m}\right)^2\right] \tag{3}$$

The modularity after assigning the node i to m_1 is Q_2. E_{i} indicates that the number of new edges after node i join m_1.

$$Q_2 = \left[\frac{\sum cin + E_i}{2m} - \left(\frac{\sum tot + k_i}{2m}\right)^2\right] \tag{4}$$

The gain of the modularity is:

$$\Delta Q = Q_2 - Q_1 = \frac{E_i}{2m} - \frac{k_i \sum tot}{2m^2} \tag{5}$$

After modularity detection, each node is assigned to corresponding servers according to the criterion of maximum modularity. The results are shown Fig. 2. The specific algorithm is shown in Algorithm 1.

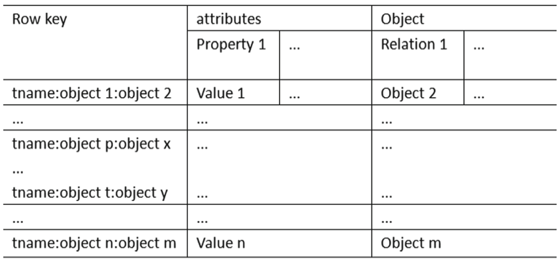
```
Algorithm 1 GP-NDM Algorithm
Input: $G(V,E)$, N
Output: $N$-way partition results
1  G'=G, Ni={0}($1 \le i \le n$);
2  N={N1,N2,N3,....Nn},q=$-\infty$;
3  foreach(Ni in N(N1,N2,...Nn)) do
4    dispatchInitNode(Ni,V);
5    C the index of community of each nodes of G';
6    Initialize each nodes with its own community;
7  foreach(Vk in G(V,E)) do
8    foreach(Ni in N(N1,N2,...Nn)) do
9      while q<Q(G',C) do
10       q=Q(G',C);
11       if(LOAD(Ni<1.3*AVG(LOAD(Ni)) then
12         chooseServer(N);
13         C= MoveNodes(G');
14         G'=Aggregate(G',C);
15         C=put each node of G' In its own community;
16       else select another server;
17 return the $N$-way partition results;
```

3.2 The Design of Distributed Storage Mode

Faced with enormous entities and relationships, the knowledge graph storage access system needs to design a scalable storage schema to enhance the storage performance. This paper takes the knowledge graph in the financial domain as an example to illustrate. We design the storage mode based on HBase, and store enterprise entities and relationships in it.

This paper takes full advantage of the sparse, distributed, consistent, multidimensional sorting characteristics of HBase. And it is designed as a distributed storage model with a single table and multiple-column clusters. The mode not only can effectively avoid data redundancy, but also has the overall storage load distribution balance, preventing the query nodes from re-traverse all entries to meet the need for scalability of the knowledge graph storage. The storage mode is shown in Fig. 3.

As shown in Fig. 3, the distributed storage mode designed in this paper uses a table for each entity type to store relationships and attribute values between entities. Each row stores the attribute values and relationship objects of an entity. The row key stores the table name to which the entity belongs, the entity and another entity associated with it. The attributes store the entity's attribute values, and object stores the corresponding entity. The value in the cell stores the corresponding attribute value or entity, where the entity representation is consistent with the entity representation of the row key. Storing attributes and relationships with two column clusters is not only easy to manage, but also reduces the number of columns loaded in memory during query, which helps to load more entities into memory and speed up query. In order to meet the requirements



Row key	attributes		Object	
	Property 1	...	Relation 1	...
tname:object 1:object 2	Value 1	...	Object 2	...
...	...		...	
tname:object p:object x	...		...	
...				
tname:object t:object y	...		...	
...	...		...	
tname:object n:object m	Value n		Object m	

Fig. 3. The storage mode

of the knowledge graph, the row key format of this storage mode is designed as “tname: object1:object2”. “tname” is the table name corresponding to the stored entity data, “object1” corresponds to the first entity, “object2” corresponds to the second entity associated with first entity, and “:” is the custom delimiter.

The distributed aggregation storage mode designed in this paper not only reduces data redundancy caused by repeated data storage, but also avoids data inconsistency caused by multiple table data insertion and update operations. It can also make full use of cluster performance, improve parallelism and ensure load balancing. The entity storage has the characteristics of overall storage load distribution and local node aggregation storage which fully meets the requirements of knowledge graph storage.

3.3 Load Balancing

In order to solve the problem of load balancing, this paper adopts an improved algorithm of consistent hashing method [16], i.e., a method of increasing the upper limit of the load to solve this problem. The algorithm adding to each server a maximum load limit, where the maximum load limit is $(1 + e)$ times the average load. The custom e value is 0.3, i.e., the average load of the storage node server is 100, if the current node and relationship to be added to the N_i server will result in the load of this server to be 30% higher than the average load, the current selected server would be excluded. And the node will select the server with the second largest modularity to further determine the load of the server at this time. If the maximum load limit is met, add the node to the server, otherwise continue to filter the server.

4 Visual Query Based on Neo4j

4.1 Problem Description

Neo4j is currently the most widely used open source graph database, which can intuitively and vividly represent real-world application scenarios. Therefore, this paper takes the knowledge graph in the financial field as an example to study the visual display and query strategy based on Neo4j. The complex, difficult and diverse knowledge in the financial field is visually presented in a graphical way. The implicit knowledge is manifested and the external knowledge is materialized. And the relevant knowledge in the financial field will be constructed, connected and integrated. In this section, we use the constructed financial knowledge graph as the back-end data source. By designing specific query statements, the query result obtained is the sub-graph that meets the limited conditions, which is inside the users' scope of concern.

Figure 4 shows the visual query architecture of the knowledge graph in this paper. Among them, the user enters a query statement through the query interface, and the query condition is sent to the background processing program. The HBase database interface is logically called by the background code, according to the query condition to find the satisfaction in the HBase database. Query the condition data and write it in Neo4j. Neo4j correlates the corresponding nodes according to the written data, draws the map, and finally returns the map to the user.

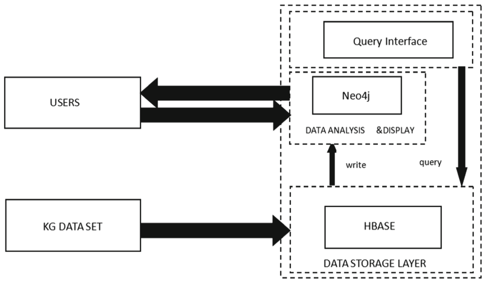


Fig. 4. Visual query architecture

4.2 Visual Query Based on Property Graph

The knowledge graph created by Neo4j is based on the property graph, and the query language on it is Cypher, which can query, modify and update data through the nodes and relationships in the pattern matching graph database without writing complex query statements. It lays a good foundation for data analysis and expansion of knowledge graphs in the financial field. The entity information and semantic relations involved in the graph are stored in the visual query architecture of this paper. The operation content and implementation method of data update and query of financial knowledge graph based on Cypher language are introduced in detail below.

(1) Create financial graph entity nodes, including enterprise entities and business personnel entities. Formally expressed as (Variable: Lable1: Label2 {Key1:Value1, Key2:Value2}). The label of the node is equivalent to the table name of the relational database (RDB), and the attribute is equivalent to the column of the relational database. Each node contains the default internal property id. When creating nodes, Neo4j graph data server will automatically assign an integer id to it. In the entire graph database, the id value of the node is incremented by default and unique. As shown in Fig. 5, creating a Cypher statement for an enterprise entity:

(2) Query the entity node of the financial graph. Cypher query language depends on matching a graph model. The keyword MATCH graph pattern is used to match the

$$
\begin{array}{l}
CREATE\left(n: CorporationBackground: Company\left\{\begin{array}{l} Name:"APPLE", \\ \mathrm{Re}\,g\_Address:"California", \\ Industry:"mobile, techno\log y", \\ Type:"enterprise", \\ Chairman:"ArthurD.Levinson",... \end{array}\right\}\right) \\
RETURN \quad n;
\end{array}
$$

Fig. 5. Creating an enterprise entity by Cypher

existing entity nodes in the database. The MATCH clause is used to specify the search model (pattern). And the WHERE clause adds a predicate to the MATCH pattern constrain the pattern. Query the node of the specified attribute. Such as the Cypher statement of querying "MicroSoft" is "*MATCH* (*n* {*Name*: "*MicroSoft*"}) *RETURN n*".

(3) Create entity relationship of a financial graph. Similar to the syntax of the node, the relationship type (RelationshopType) and attributes are defined in the square brackets of the relationship. The relationship type is similar to the node label. When creating a relationship, specifying the relationship type is essential, but only one relationship type can be specified.

(4) Query entity relationship of financial graph. By using MATCH clause to specify the search mode, the WHERE clause adds a predicate to the MATCH mode to constrain the StartNode node, so as to query the information of the StartNode node and its related nodes with a layer of direct relationships.

4.3 The Display of Neo4j Visual Query

By inputting the corresponding query statement to the knowledge graph, the user can concentrate on querying the graph inside his scope of concern, while the other unrelated partial graphs are not displayed. In this paper, through the collection of financial data from professional financial network, we realize the construction of the financial knowledge graph by the process of data cleaning and extraction. A total of 3548 entities have been constructed in the knowledge base.

In Neo4j, the financial domain knowledge graph stores knowledge with nodes and edges. A part of the entities and relationships of financial knowledge that contains the keyword "GuangdongHongdaBlasting Co., Ltd." are displayed in Neo4j as shown in Fig. 6. Different entity classes are distinguished by different colors. Red nodes represent business entities, blue nodes represent product entities, and orange nodes represent human entities. These nodes are accompanied by their own attribute information. The relationship between entities and entities is represented by edges. The edges contain relationship attributes, start node *id* and end node *id*.

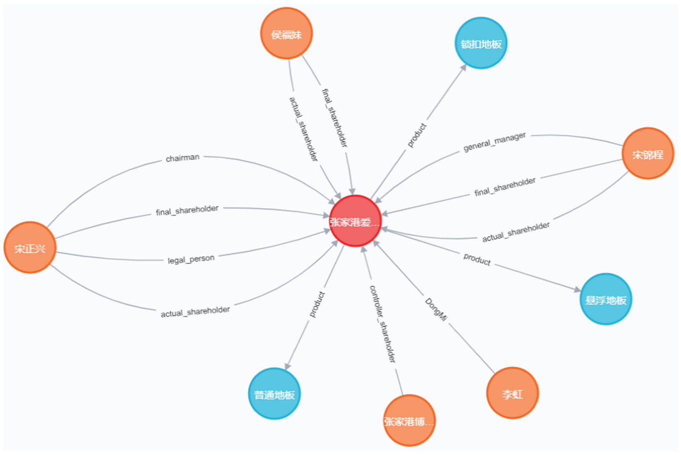


Fig. 6. Partial financial knowledge queried by user

5 Experiments

5.1 Experimental Settings and Datasets

The experimental environment of this paper is Inter Core i7–8750 CPU @ 1.8 Hz 2.00 GHz processor, and the computer with 8G memory is used as the host of Neo4j graph database. Three other servers with 64 GB memory and 512 GB hard disk and Ubuntu operating system are applied as devices for deploying HBase distributed database. The host and server communicate with each other through deployment in the same local area network.

In the selection of data sources, the financial knowledge graph studied in this paper covers the basic information of the company, shareholder information, executive information, corporate news, and corporate credit information. Use crawlers for large-scale data acquisition on selected data sources, such as Royalflush Finance News, Sohu Finance News and Tianyancha. After obtaining the data, we perform basic statistical analysis and processing for subsequent data partition and distributed storage. Build basic connections between enterprises. The original datasets is split into two sets of different sizes. The node numbers of both datasets are 13832 and 21299; the numbers of edges are 69160 and 106440.

5.2 Efficiency Analysis of Graph Partition

This paper compares the proposed GP-NDM algorithm with BS and DynamicDFEP in terms of the partition time and the number of edge cuts. Each partition algorithm is executed three times, and the result is averaged. With the increasing graph scale, the partition time of GP-NDM does not increase greatly as shown in Fig. 7, and the effect is good for large-scale dynamic graphs. DynamicDFEP is more complicated and its time complexity is relatively large. In the experiments, DynamicDFEP is used to cyclically allocate nodes on *N* servers. As the result the time complexity of the algorithm will further increase, and the algorithm is inefficient. As for BS architecture, the BS algorithm is fast, but if the graph changes dynamically, the BS cannot respond to the graph division in real time.

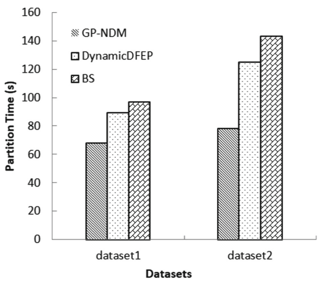


Fig. 7. Comparison of partition time

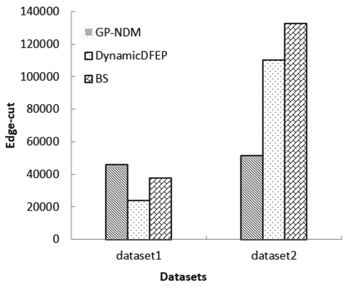


Fig. 8. Comparison of edge-cut

Figure 8 shows the comparison of edge cutting on different datasets. Edge cutting is the number of edges with endpoints in different partitions, and the quality of the partition can be verified by edge cutting. In order to ensure the partitioning principle of high cohesion and low coupling, fewer cutting edges indirectly prove that the segmentation quality is better. BS switches nodes to improve the division result, but ignores the influence of weight on closeness. DynamicDFEP utilizes the principle of random selection, which has a certain influence on the segmentation quality. GP-NDM considers the influence of modularity and node degree, so we exchange nodes to improve the node's modularity.

5.3 Performance Analysis of Incremental Dynamic Maintenance

We evaluate the update efficiency on different numbers of partitions. Since BS is an algorithm for static graphs, it can be updated only by repartitioning the graph, which is much lower than the incremental update method. Therefore, we compared GP-NDM with the DynamicDFEP. As shown in Fig. 9, DynamicDFEP algorithm is sensitive to update, and it is not applicable to the situation where the freshness of the graph exceeds 20%. In the stage of graph update and maintenance, the DynamicDFEP assigns initial "funds" to the initial graph partitioning interactions up to the final vertex. In repeating the initial graph partitioning steps, the information of the boundary nodes needs to be stored in the partition. Due to the incremental update of the graph which leads to the expansion of the graph size, the cost of space and time is close to running the DFEP algorithm starting from the beginning. GP-NDM only need execute to perform the module degree calculation to complete the update according to the newly added nodes or edges. And the algorithm does not need to be run from scratch.

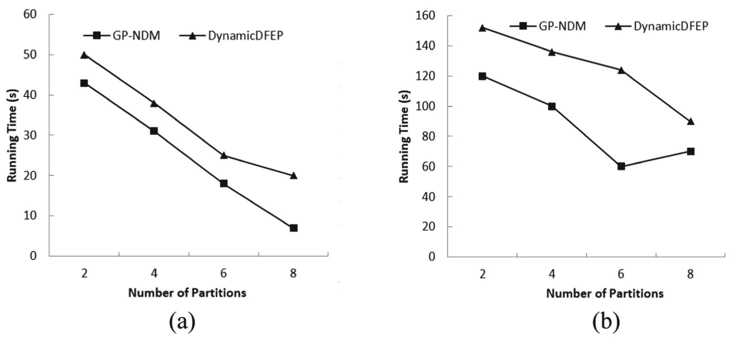


Fig. 9. Comparison of incremental dynamic maintenance

6 Conclusion

This paper conducts an in-depth study of knowledge graphs in specific fields, and proposes a distributed storage and query method for domain-specific knowledge graphs. A graph partition method based on node density and modularity is proposed to ensure as far as possible that nodes closely connected and their relationships are stored in the same partition. The network communication overhead is reduced. A distributed aggregate storage model is designed. This storage model makes full use of cluster performance and can effectively reduce the redundancy overhead caused by storing repeated data. At the same time, it solves the problem of data consistency when inserting and updating data. In order to achieve distributed storage and query of domain-specific knowledge graphs, we manage to combines HBase distributed storage with Neo4j visual query effectively. According to the query semantics of the master node, the related Entities and relationships in HBase are extracted and imported into Neo4j to form sub-graphs that users are interested in for visual display. The proposed method has a wide range of practical application value.

Acknowledgements. This work was supported by National Natural Science Foundation of China under Grant (No. 61472169, 61502215, 62072220, 61702381, U1811261); China Postdoctoral Science Foundation Funded Project (2020M672134); Science Research Fund of Liaoning Province Education Department (LJC201913); Liaoning Public Opinion and Network Security Big Data System Engineering Laboratory (No. 04-2016-0089013).

References

1. Xin, H., Jiangli, D., Jiangli, D.: Scalable aggregate keyword query over knowledge graph. Future Gener. Comput. Syst. **107**, 588–600 (2020)
2. Li, W., Song, M., Tian, Y.: An ontology-driven cyberinfrastructure for intelligent spatiotemporal question answering and open knowledge discovery. ISPRS Int. J. Geo-Inf. **8**(11), 496 (2019)
3. Barnawi, A., Alharbi, M., Chen, M.: Intelligent search and find system for robotic platform based on smart edge computing service. IEEE Access **8**, 108821–108834 (2020)

4. Fensel, A., Akbar, Z., Kärle, E., et al.: Knowledge graphs for online marketing and sales of touristic services. Information **11**(5), 253 (2020)
5. Sousa, R.T., Silva, S., Pesquita, C.: Evolving knowledge graph similarity for supervised learning in complex biomedical domains. BMC Bioinform. **21**(1), 6 (2020)
6. Yuan, L., Huang, Z., Zhao, W., et al.: Interpreting and predicting social commerce intention based on knowledge graph analysis. Electron. Commer. Res. **20**(1), 197–222 (2020)
7. Fan, T., Yan, L., Ma, Z.: Storing and querying fuzzy RDF(S) in HBase databases. Int. J. Intell. Syst. **35**(4), 751–780 (2020)
8. Zhu, Z., Zhou, X., Shao, K.: A novel approach based on Neo4j for multi-constrained flexible job shop scheduling problem. Comput. Ind. Eng. **130**, 671–686 (2019)
9. Fernandes, D., Bernardino, J.: Graph databases comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB. In: DATA 2018, pp. 373–380 (2018)
10. Soudani, N.M., Fatemi, A., Nematbakhsh, M.: An investigation of big graph partitioning methods for distribution of graphs in vertex-centric systems. Distrib. Parallel Databases **38**(1), 1–29 (2019)
11. Filippidou, I., Kotid, Y.: Online partitioning of multi-labeled graphs. In: Proceedings of the GRADES 2015, pp. 1–6. ACM (2015)
12. Lotfifar, F., Johnson, M.: A serial multilevel hypergraph partitioning algorithm. Comput. Res. Repository (2016)
13. Preen, R.J., Smith, J.: Evolutionary n-level hypergraph partitioning with adaptive coarsening. IEEE Trans. Evol. Comput. **23**(6), 962–971 (2019)
14. Rahimian, F., Payberah, A.H., Girdzijauskas, S., et al.: JA-BE-JA: a distributed algorithm for balanced graph partitioning. In: IEEE International Conference on Self-adaptive & Self-organizing Systems, pp. 51–60. IEEE (2013)
15. Guerrieri, A., Montresor, A.: DFEP: distributed funding-based edge partitioning. In: Träff, J.L., Hunold, S., Versaci, F. (eds.) Euro-Par 2015. LNCS, vol. 9233, pp. 346–358. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48096-0_27
16. Mirrokni, V., Thorup, M., Zadimoghaddam, M.: Consistent hashing with bounded loads. In: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, pp. 587–604 (2018)