


## Article

# Dynamic Partitioning Supporting Load Balancing for Distributed RDF Graph Stores

Kyoungsoo Bok, Junwon Kim and Jaesoo Yoo \* 

School of Information and Communication Engineering, Chungbuk National University, Chungdae-ro 1, Seowon-Gu, Cheongju, Chungbuk 28644, Korea

\* Correspondence: yjs@chungbuk.ac.kr; Tel.: +82-43-261-3230

Received: 5 June 2019; Accepted: 12 July 2019; Published: 16 July 2019



**Abstract:** Various resource description framework (RDF) partitioning methods have been studied for the efficient distributed processing of a large RDF graph. The RDF graph has symmetrical characteristics because subject and object can be used interchangeably if predicate is changed. This paper proposes a dynamic partitioning method of RDF graphs to support load balancing in distributed environments where data insertion and change continue to occur. The proposed method generates clusters and subclusters by considering the usage frequency of the RDF graph that are used by queries as the criteria to perform graph partitioning. It creates a cluster by grouping RDF subgraphs with higher usage frequency while creating a subcluster with lower usage frequency. These clusters and subclusters conduct load balancing by using the mean frequency of queries for the distributed server and conduct graph data partitioning by considering the size of the data stored in each distributed server. It also minimizes the number of edge-cuts connected to clusters and subclusters to minimize communication costs between servers. This solves the problem of data concentration to specific servers due to ongoing data changes and additions and allows efficient load balancing among servers. The performance results show that the proposed method significantly outperforms the existing partitioning methods in terms of query performance time in a distributed server.

**Keywords:** RDF; dynamic partition; cluster; subcluster; usage frequency load balancing

## 1. Introduction

Web technology has continuously evolved, and a variety of information has become available on the web with the increasing number of web users and amount of information on the web. The general web service has provided information suitable for users' search needs through hyperlinks among information resources; however, it has been associated with difficulties in terms of accurate information searches and the meaningful interpretation of information. In other words, a next-generation web that can meet users' various needs and support more accurate searches is now needed. The semantic web is under the spotlight as an alternative to the next-generation web [1,2]. It can define meaningful relationships among data and generate new knowledge through reasoning [3,4].

As metadata and ontology play a core role in the semantic web, the resource description framework (RDF) was proposed by the World Wide Web Consortium (W3C) to describe web data formally [5–7]. The RDF is a language to specify metadata with the capability to express semantic relationships among information resources and sentences with the subject, predicate, and object structure [8–10]. In other words, when RDF documents with subjects, predicates, and objects are used, computers can identify the relationships among information resources and process data more efficiently and accurately [11–13].

In an RDF graph that models RDF documents as graphs, subjects and objects are represented by vertices and predicates are represented by edges. Subject and object can be changed if the meaning of the predicate is changed. That is, the RDF graph has symmetrical characteristics because subject

and object can be used interchangeably if predicate is changed. With the development of big data technologies and the spread of the semantic web, the RDF graph is growing continuously [14,15]. When handling large RDF graphs on a single server, the performance and speed of processing storage make it difficult to service many users [16–19]. Therefore, a RDF graph partitioning method is required to store and manage data using multiple servers [20–22]. The existing RDF partitioning methods focused on minimizing the number of edge-cuts generated by partitioning [23–26]. However, they can concentrate data on a particular server because it partitions the entered data according to predetermined criteria [27–30]. In addition, various query requests from users increase the cost of communication between specific servers [31–33]. For example, partitioning RDF graph data on distributed servers and joining queries on partitioned edges increases the cost of communication between servers and delays response time for queries. To solve this problem, a study of dynamic partitioning methods has been carried out [34–37].

The social partitioning and replication (SPAR) performs data replication in the same server to improve data locality and minimize network load [35,38]. Self evolving distributed graph management environment (Sedge) repartitions graph to reduce the cost of communication between servers and performs data replication to ensure load balancing among servers [37]. However, in environments with ongoing graph data additions and changes, data become concentrated to certain servers due to the ongoing data additions, and if users' query requests become concentrated on the server with data concentration, the performance as a parallel system decreases. In addition, when replicating data in distributed servers, if the original data is modified, additional operations for the replicates are required. In this paper, a dynamic environment is defined as an environment where ongoing additions and changes occur. Accordingly, conventional methods are not suitable for dynamic environments in terms of data storage, load balancing, and query processing. In the existing dynamic partitioning methods, the communication cost between servers increases when join queries are processed.

We propose a basic partitioning strategy for RDF graph considering partition size in previous work [39]. A concept of a cluster and a subcluster for partitioning a large RDF graph in a distributed environment was proposed by [39]. In addition, the RDF graph is partitioned according to the partition size using a cluster and a subcluster. However, [39] did not provide detailed partitioning strategies and algorithms. Also, there is a lack of performance evaluation to prove the improvement of the proposed method. This paper is an extended version of the method proposed in [39] and proposes an RDF partitioning method to provide load balancing without data replication in distributed RDF stores. To meet users' constantly changing query needs, grouping is performed based on the subgraph frequently used by queries, and a partitioning method to minimize the number of edge-cuts among partitions is used. Partitioning considering the size of the generated partitions solves the problem of data concentration to specific servers. In addition, load balancing is possible by partitioning the partitions by calculating the average frequency of data usage on the distributed server. Through this approach, the proposed method can maximize the strengths of the parallel processing system and provide faster replies to queries than existing methods. In order to demonstrate the improvement of the proposed method, we analyze the characteristics of the existing and proposed methods and perform various performance evaluations. The contribution of the method proposed in this paper is as follows. We propose a method to collect the information of the subgraph used in the query to perform the partition using the RDF graph usage pattern.

- We propose a detailed algorithm for creating clusters and subclusters, and propose a partitioning method using clusters and subclusters.
- We analyze existing and proposed methods according to various criteria such as partitioning policy, replication policy, load discrimination, partition size, and partitioning condition.
- We demonstrate the improvement of the proposed method by comparing the performance of the proposed method with that of the existing method as well as evaluating its own performance.

This paper is organized as follows. Section 2 describes the existing graph partitioning methods. Section 3 describes the RDF dynamic partitioning method proposed in this paper. Section 4 describes the results of the performance evaluation to show the excellence of the proposed method. Finally, Section 5 provides the conclusions of this paper and presents future directions.

## 2. Related Work

Hendrickson and Leland proposed a multilevel method for graph partitioning within acceptable execution time [23]. The multilevel partitioning method generates a sequence of smaller graphs approximating the original graph to find easily a good partition via coarsening procedure. In the coarsening step, it merges two vertices connected to an edge and the new vertex retains edges to the union of the neighbors of the merged vertices. The smallest graph is partitioned via a spectral partitioner. The spectral partitioner needs one, two, or three eigenvectors of the Laplacian matrix to partition a graph into two, four, or eight groups, respectively. The partitioning algorithm is able to handle edge and vertex weights, even if the original graph is unweighted. The multilevel partitioning method makes the sums of the vertex weights in each server as equal as possible and the sum of the weights of edges crossing between servers is minimized.

Wang and Chiu proposed a RDF graph partitioning method to minimize the number of edges on different servers [40]. This method first scans the RDF graph and stores subgraph with common vertices into the same components. If the connected components of the graph are found, each connected component is partitioned. After initial partition, each RDF subgraph is treated as an undirected and weighted graph. This method weights the vertices with the number of triples of the same subjects, and uses multi-constraint partitioning to achieve a balanced size of partitions.

Stanton and Kliot proposed a new stream partitioning method for balanced graph partitioning in distributed environments [36]. As the vertices arrive in stream with the set of edges, a partitioner decides to store the vertex on one of several servers. The 10 heuristics were provided for streaming balanced graph partitioning. The first seven heuristics such as balanced, chunking, hashing, (weighted) deterministic greedy, weighted randomized greedy, weighted triangles, and balance big do not use a buffer, but the last three such as prefer big, avoid big, and greedy evocut all use a buffer. The linear weighted variant of the greedy algorithm shows the best performance because it reduces computation overhead and the number of edge-cuts.

Pujol et al. developed SPAR, which partitions online social graph through a joint partitioning and replication to prevent query processing via multiple servers [35,38]. SPAR is a social partitioning and replication system that utilizes the social graph to achieve data locality while minimizing replication. SPAR guarantees that data of all direct neighbors stored on a particular server is co-located on that same server because most of the relevant data for a user in online social networks is one-hop way. When a new server is added, SPAR forces a master replica from another server to a new one, immediately balancing all servers or waiting for a new arrival to fill the server. When the server is removed, it redistributes the master replicas to the other servers equally.

Yang et al. proposed Sedge, which is a distributed graph partition management for minimizing inter-communication during query processing in distributed environments [37]. Sedge uses a two-level partition architecture with complementary and on-demand partitioning. Supplemental partitioning creates multiple partitions so that the partition boundaries do not overlap. On-demand partitioning uses two partitioning methods: partition replication and dynamic partitioning to handle internal hotspots and cross-partition hotspots. Partition replication shares the workload of these partitions by replicating the same data across multiple servers. Dynamic partitioning reconfigures new partitions to handle cross-partition queries locally. Sedge developed a new technique to profile graph queries to perform dynamic partitioning efficiently.

Nicoara et al. proposed a lightweight repartitioner in Hermes for a distributed social graph [34]. The initial partition generated by Metis [24] is repartitioned by a lightweight repartitioner to improve an initial partitioning by decreasing edge-cuts while maintaining almost balanced partitions. Each server

maintains auxiliary data to perform repartitioning. The auxiliary data stores the list of accumulated weight of vertices in each partition and the number of neighbors of each vertex in each partition. The repartitioning process has two steps. In the first step, each server runs the repartitioner algorithm using the auxiliary data to choose some vertices in its partition that should be migrated to other partitions. These vertices are logically moved to the destination partition until no more vertices are selected for migration. In the second step, physical data migration is performed. Vertices and relationships that were marked for migration by the repartitioner are moved to the target partitions.

Troullinou et al. proposed a semantic partitioning method for the RDF store to reduce communication costs and enhance query performance [41]. This method performs vertical partitions utilizing both structural and semantic information to select the most important nodes and then to assign the remaining nodes to the proper clusters. First, the most important nodes are identified to select the centroid for each cluster. The dependence is used to eventually assign the remaining nodes to the cluster maximizing the dependence with the corresponding centroid.

Leng et al. proposed a balanced RDF graph partitioning (BRGP) for storing massive RDF data on cloud [42]. BRGP use a modularity-based multi-level label propagation algorithm (MMLP) to partition RDF graph roughly. In MMLP, LP algorithm is performed to find dense subgraphs, and each subgraph forms a coarsened vertex of upper level. To generate a balanced k-way partitioning, a balanced K-medoids clustering algorithm is used. BRGP not only minimizes the edge-cut but also keeps the balance of vertex distribution.

Hayes and Gutiérrez proposed a model to represent a RDF graph as a bipartite graph [43]. The RDF graph can be represented by a simple ordered 3-uniform hypergraph. The proposed model is intermediate model of RDF between the abstract triple syntax and data structures used by applications. This model provides an approach to stratify an RDF graph into data and schema layers.

Tomaszuk et al. discussed classical graph partitioning methods and presented four ways to transform an RDF graph to a classical graph representation [44]. This paper presented that a directed labeled graph can be easily transformed into a RDF graph, but the reversed transformation is cumbersome. The practical relevance of RDF graph partitioning is evaluated by using the gpmets algorithm.

Akhter et al. compared seven RDF graph partitioning methods in two different evaluation setups [45]. The existing partitioning methods are evaluated in terms of partitioning time, partitioning imbalance, and query processing time. Total communication volume (TCV)-Min leads to smallest query runtimes followed by Property-Based, Horizontal, Recursive-Bisection, Subject-Based, Hierarchical, and Min-Edgecut, respectively. In general, partitioning methods that minimize the total number of selected sources proves to provide better runtime performances.

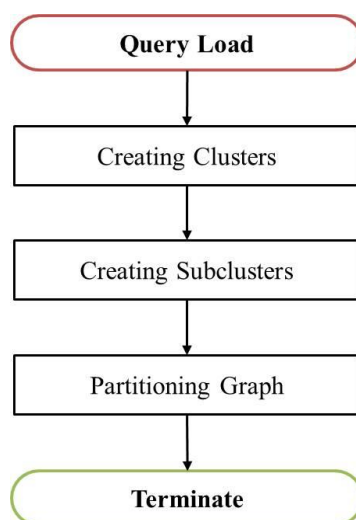
### 3. The Proposed Dynamic Partitioning Method

#### 3.1. Architecture

In a dynamic RDF environment, a large-scale RDF graph is continually added and changed. The existing dynamic partitioning methods have the problem of performing unnecessary work due to data concentration and replicate data. Therefore, dynamic partitioning methods are needed to provide fast query responses in dynamic RDF graph environments. This paper proposes an RDF dynamic partitioning method that take into consideration load balancing by analyzing the subgraph frequently used by queries. The proposed method provides load balancing by grouping, taking into account the frequency of use of the data used in user request queries. An RDF document consists of a subject, a predicate, and an object. When modeling an RDF document as a graph, subject and object are represented by vertices and predicates are represented by edges. Query frequency represents the number of subgraphs used together by query when subject, predicate, and object are modeled as graphs. It groups the RDF subgraphs with higher frequency of queries to create a cluster, and groups the RDF subgraphs with relatively lower frequency of queries to create subclusters. The proposed method

solves the problem of data concentration to specific servers by considering the size of the partitions to which the graph data will be partitioned. It also minimizes the number of edge-cuts connected to the partition, minimizing the cost of communication between servers during join processing.

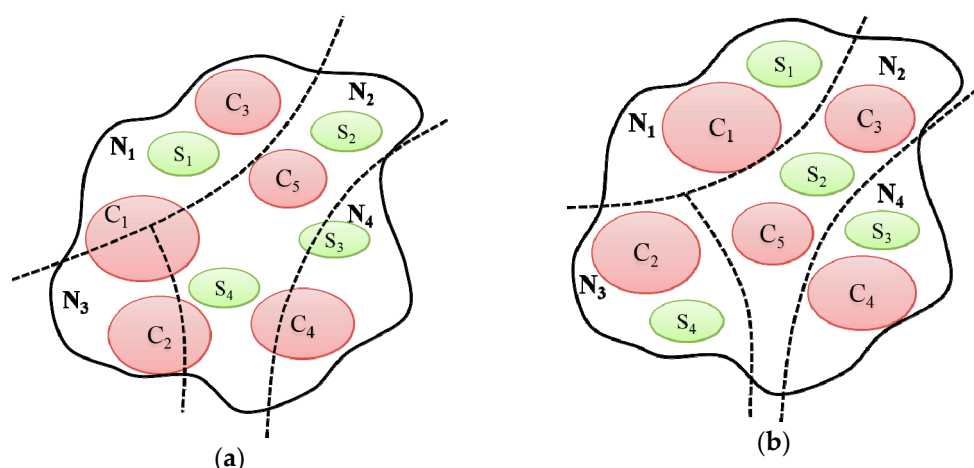
Figure 1 shows the whole process of the proposed dynamic partitioning method. If the queries are concentrated on a specific server and the communication cost between servers increase between servers during query processing, the proposed method repartitions the RDF graph through the following three steps. The first step is the cluster creation, in which RDF subgraphs that are frequently used in queries are grouped to create clusters. The clusters are the criteria for data partitioning for load balancing and are created with RDF graphs with high query frequency. The second step is the subcluster generation, and the subgroups are generated with data with lower query frequency than the clusters. The last step is the graph partitioning, and partitions are generated based on data size, query frequency, and the number of edges.



**Figure 1.** The whole process of the proposed resource description framework (RDF) dynamic partitioning method.

Figure 2 shows RDF graphs partitioned with the proposed dynamic partitioning method. Figure 2a shows the architecture of the distributed storage of graph data in the distributed server environment with four servers from  $N_1$  to  $N_4$ .  $C_1 \sim C_5$  represent clusters with high query frequencies, and  $S_A$ ,  $S_B$ ,  $S_C$ , and  $S_D$  represent subclusters with lower query frequencies. As shown in Figure 2a, when we process a query on cluster  $C_1$ , we process it by using subgraphs stored in distributed servers  $N_1$ ,  $N_2$ , and  $N_3$ . Similarly, queries on  $C_2$  and  $C_4$  are processed in two distributed servers. Therefore, communication costs among distributed servers are required to obtain query processing results. When the communication costs among servers for processing a frequent query increase, the proposed method reduces the communication costs by redistributing  $C_1$ ,  $C_2$ , and  $C_4$  to the distributed servers. A particular server increases loads when it has many frequently used queries and subgraphs. The proposed method also redistributes subgraphs by considering a partition size stored in a distributed server in order to reduce the server loads. Since the loads of server  $N_1$  increase when  $C_3$  is stored along with  $C_1$  in server  $N_1$ , the proposed method stores  $C_3$  in server  $N_2$  considering the partition size. Figure 2b shows the results of repartitioning in the event of a high load due to query requests. In the proposed method, each partitioning adds and redistributes subclusters properly based on clusters, which solves the problem of data concentration to specific servers and performs efficient load balancing among servers.





**Figure 2.** Performance of the proposed dynamic partitioning method: (a) high load of queries; (b) performance of the proposed method.

### 3.2. Statistical Data

To perform dynamic graph data partitioning, the information on past queries requested by users must be managed as statistical data. For efficient graph partitioning in the dynamic RDF graph environment, the proposed method manages statistical data on queries whenever a query is requested by a user. The statistical data is used to group queries based on query frequency in the cluster generation and subcluster generation stages to perform load balancing as well as meet users' diverse query needs. In addition, they provide information for graph partitioning based on query frequency and query size in the graph partitioning stage.

The proposed method is based on a master–slave architecture in order to collect statistical data as shown in Figure 3. A slave is a distributed server that stores the partitioned RDF subgraphs and processes a query. The RDF query processor can determine which subgraphs are used together in a distributed server when performing query processing. Therefore, each slave collects the information of the vertices and edges used in the query processing without any additional computation. Each slave stores the query identifier and vertex and edge information used in the query. Whenever the master performs a query, it generates a lot of updates if it creates statistical information by collecting all vertices and edges used in the entire slave. Therefore, each slave periodically sends the information of the vertices and edges used in the queries to the master, and the master creates statistical information using the subgraph pattern used in the queries. However, since the statistical information used in the proposed method collects the subgraph information used by the past queries, it does not update the statistical information even if the graph is changed. That is, the statistical information is updated only when the subgraph patterns used by queries are changed. For example, when server  $N_1$  processes queries  $Q_1$  and  $Q_3$ , it stores query identifiers and the number of vertices included in the query results. Master updates the query frequencies of  $Q_1$  and  $Q_3$  by collecting their query information from server  $N_1$ . Since the RDF graph can be changed continuously, the number of vertices included in the results of the same query can also be changed. Therefore, master periodically updates the number of vertices included in the query result by using the up-to-date query information.

Table 1 shows statistical data managed in a master server. Here, ID, server, query, frequency, and size mean identifier, distributed server identifier, query identifier, the query frequency, the number of vertices included in the query result, respectively. Server and query are used to classify the types of queries requested by users. Frequency is used to group clusters and subclusters during dynamic partitioning and to perform the load balancing of servers. Size is used to determine the size of a partition stored in each distributed server as the number of vertices included in the query result. In order to grasp the size of a subgraph frequently used in the query, only the number of vertices used in the query is stored in Table 1. The information of the edges is needed in addition to the vertices in

order to identify the subgraph information in the query. Information of vertices and edges in the query, that is, subgraph information, is stored separately with IDs in the statistical data management table.

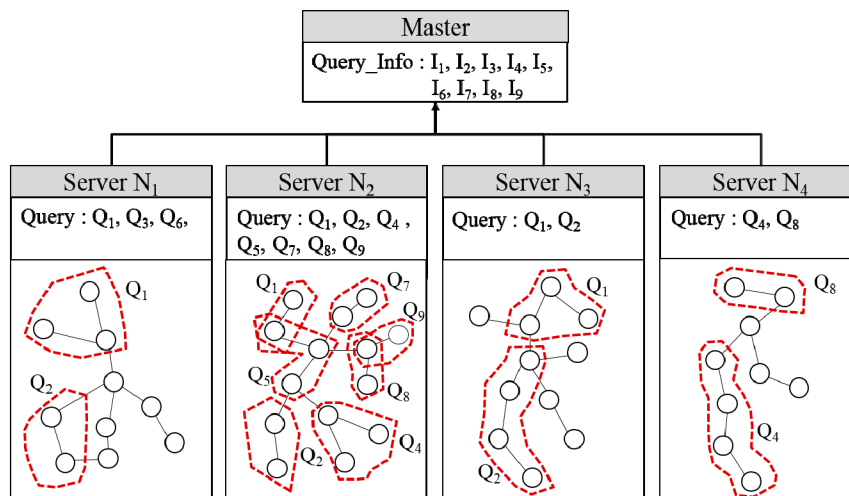


Figure 3. Process to collect statistical data.

Table 1. Statistical data management table.

ID	Query	Server	Frequency	Size
I <sub>1</sub>	Q <sub>1</sub>	N <sub>1</sub> , N <sub>2</sub> , N <sub>3</sub>	47	2000 K
I <sub>2</sub>	Q <sub>2</sub>	N <sub>2</sub> , N <sub>3</sub>	50	1800 K
I <sub>3</sub>	Q <sub>3</sub>	N <sub>1</sub>	25	1200 K
I <sub>4</sub>	Q <sub>4</sub>	N <sub>2</sub> , N <sub>4</sub>	57	2000 K
I <sub>5</sub>	Q <sub>5</sub>	N <sub>2</sub>	22	1000 K
I <sub>6</sub>	Q <sub>6</sub>	N <sub>1</sub>	13	800 K
I <sub>7</sub>	Q <sub>7</sub>	N <sub>2</sub>	7	500 K
I <sub>8</sub>	Q <sub>8</sub>	N <sub>2</sub> , N <sub>4</sub>	3	500 K
I <sub>9</sub>	Q <sub>9</sub>	N <sub>2</sub>	10	800 K

### 3.3. Cluster Creation

To process a user-requested query efficiently in a dynamic distributed environment, it is required to minimize join processing between servers and communication cost. The proposed method expects that subgraph that has been frequently requested by users in the past will be stored on the same server and requested again in the future. Therefore, the frequently requested data is grouped into clusters. The cluster consists of RDF subgraphs that have been frequently requested by users. To meet the diverse needs of constantly changing users, we create clusters based on queries that users have frequently used.

As for clusters, groupings are performed in such a way that the frequency of the subgraph is close to the average value of the cluster frequency based on statistical data. Clusters are also used as reference points in each server to perform the same load balancing between servers, and as many clusters are created as there are servers. By grouping subgraph stored on multiple different servers into the same server, the proposed method can minimize the communication costs that occur during query processing and provide quick query responses.

Equation (1) calculates the mean cluster frequency  $AvqCF$  that is the criterion used to create a cluster in each server, where  $SumOfHighQF$  is the sum of the high frequency values when frequency in Table 1 was sorted in descending order.  $NumofServer$  is the number of servers in the distributed

environment. In the cluster creation step, the proposed method groups the data with query frequencies higher than  $AvgCF$  into clusters. It also groups the data with query frequencies lower than  $AvgCF$  into subclusters.

$$AvgCF = \frac{SumofHighQF}{NumofServer} \quad (1)$$

Figure 4 shows the process of cluster generation. The queries that have been frequently requested by users are generated into the number of clusters equal to the number of servers ( $C_1, C_2, C_4, C_6$ ). In the case of Table 1, the number of clusters generated is close to the value of Equation (1) based on the queries with high frequency. Here,  $AvgCF = 50.25$ . As many clusters are generated as there are distributed servers. Although  $C_3$  and  $C_5$  are originally subclusters, they are grouped together into a cluster to balance the frequency of  $C_3$  with  $C_5$ .

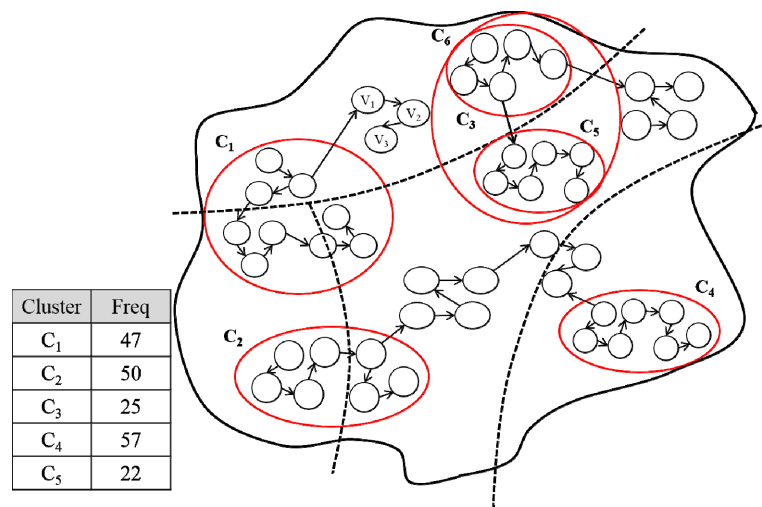


Figure 4. Cluster generation.

Algorithm 1 shows the algorithm used to generate clusters. The input parameters are *Statistical\_data*. *CntServer* is the number of servers in the distributed server, *CntQuery\_data* is the number of the subgraph with high query frequency in descending order in line 2, *Query\_dataQDN* is subgraph based on the order value of *QDN* in *CntQuery\_data*, *Freq\_Query\_dataQDN* is the current subgraph frequency based on the order value of *QDN*, and *AvgCF* denotes the mean cluster frequency. Line 2 sorts user-requested queries in descending order based on frequency to generate clusters based on subgraph with high query frequencies. Lines 3~7 generate as many clusters as the mean cluster frequency based on subgraph frequency.

---

**Algorithm 1** Generating Clusters

---

```

1: procedure Creating_Cluster (Statistical_data )
2:   sort Query_data in data frequency based on descending order
3:   for CN ∈ CntServer do
4:     new ClusterCN
5:     for QDN ∈ CntQuery_data do
6:       add Query_dataQDN in ClusterCN
7:       if Freq_Query_dataQDN ≤ AvgCF then exit
8:     add ClusterCN in Clusters
9:   return Clusters

```

---



### 3.4. Subcluster Creation

The proposed method uses the query frequency based on the average query frequency of the distributed server to perform the same load balancing between servers. The average query frequency for a distributed server is the average query frequency value by which the same load balancing is performed during data partitioning. It is computed based on the frequency of queries from statistical data. Specifically, each server uses a cluster, and a subgroup with a query frequency lower than the cluster frequency performs load balancing on the distributed server by adding the average query frequency of the cluster to the subgroup. In other words, the subcluster creation step creates subclusters by grouping data that is less frequently queried, with the exception of clusters created in the cluster creation phase.

Subclusters refer to subgroups that are grouped based on subgraph as in a cluster but are less frequently queried than in a cluster. They are used to perform the same load balancing between servers, and the subcluster is distributed across clusters of each server to achieve efficient load balancing. That is, subgroups are used to improve the efficiency of load balancing. A subcluster also calculates distance hops using connection edges based on clusters created during the cluster creation step and distributes based on the threshold range that you set. Furthermore, in order to minimize inter-server communication costs, the grouping does not include data that users never request. This avoids the collapse of previously partitioned graph data structures and reduces the cost of additional redistribution through distribution based on the predictive range of queries that users can request in the future.

Figure 5 shows the subcluster generation process. It is performed after the clusters are formed in the cluster generation stage and the subclusters ( $S_1, S_2, S_3, S_4$ ) are generated based on the remaining subgraph excluding the generated clusters ( $C_1, C_2, C_3, C_4, C_5$ ). The grouping is performed based on the subgraph of which frequencies excluding clusters are 1 or larger, using statistical data.

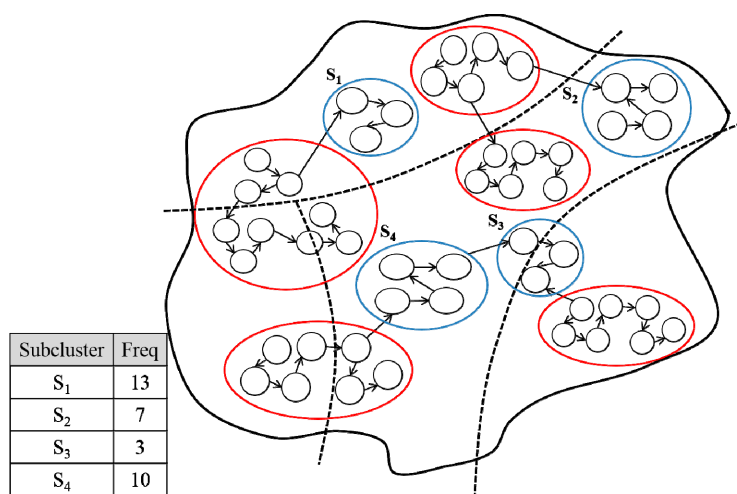


Figure 5. Subcluster generation.

Algorithm 2 shows the algorithm used to generate subclusters. The input parameters include *Statistical\_data* and *Clusters* generated in the cluster generation stage. *CntClusters* is the number of clusters, and *Cnt\_QD\_2HCSN* is the number of subgraphs within a 2-hop distance of the clusters according to the order value of *SN*. Lines 2~7 generate subclusters based on subgraph. Subgraphs that contain connected edges within two hops based on each cluster and appear more than once in the query are included in the subcluster. This allows distribution based on the prediction of the range of queries that can be requested by users in the future, which can reduce additional repartitioning cost.

**Algorithm 2** Generating Subclusters

---

```

1: procedure Creating_Subcluster (Statistical_data, Clusters)
2:   for SN ∈ CntClusters do
3:     new Sub_clusters
4:     for QDN ∈ Cnt_QD_2HCSN do
5:       if Freq_Query_dataQDN ≥ 1
6:         add Query_dataQDN in Sub_clusters
7:       add Sub_clusterSN in Sub_clusters
8:   return Sub_clusters

```

---

**3.5. Graph Partitioning**

The proposed method performs graph partitioning by considering the size of the partitions to alleviate the problem of data concentration to specific servers due to a dynamic RDF graph environment. In order to address the problem of load balancing in the existing dynamic partitioning methods, it calculates the average query frequency of the distributed server. Finally, in order to minimize inter-server communication costs, it uses a technique to minimize the number of edge-cuts that are employed.

The edge-cut minimization technique performs a cut based on an edge connected between a cluster and a subcluster rather than on an edge connected between the vertices of the RDF graph. The subcluster that has a small number of edges connected to subclusters in the same server and a large number of edges connected to subclusters in another server is selected as a candidate to move to another server for edge cutting. Edge-cut minimization is performed by relocating the selected candidates. By relocating the selected candidates to the servers with many connections, the number of edge cuts can be minimized.

Graph data partitioning is performed based on query frequency and size in statistical data. Each server performs grouping into  $PK = \{C_i, S_j, S_{j+1}, \dots, S_{j+k}\}$  based on the subclusters connected with the edges around one cluster using the edge-cut minimization method. The proposed method takes into consideration the size of the partitions to address the problem of data concentration to specific servers. Therefore, it provides the minimum partition size and the maximum partition size. We define the partition size as the number of vertices in a RDF subgraph to be stored in each server. Equation (2) represents the minimum partition size  $PSize_{min}$  to be stored in each server, and Equation (3) represents the maximum partition size  $PSize_{max}$  to be stored in each server. Here,  $Node_{cnt}$  is the number of servers in the distributed environment,  $Size_{full}$  is the total size of the data that can be stored in the distributed server.  $\alpha$  and  $\beta$  are parameters for adjusting the minimum partition size and the maximum partition size to be stored in each distributed server, respectively. A partition represents a RDF subgraph.  $\alpha$  and  $\beta$  are determined by Equation (4), where  $\alpha$  is a value in the range [0–1]. Since  $\alpha$  determines the minimum size of the partition to be stored in the distributed server, if  $\alpha$  is small, the minimum size of the partition is small and  $\beta$  is increased so that the size difference between the partitions stored in the distributed server is small. This increases the likelihood of splitting the graph by edge-cut ratio rather than the size of partitions stored in the distributed server during graph partitioning. In contrast, if  $\alpha$  is large, the minimum size of the partition is increased, and  $\beta$  is reduced accordingly. As such, there is a large difference in the size of the partitions to be stored in the distributed server. Therefore, even if edge-cut ratio increases during graph partitioning, the size of partitions stored in the distributed server can be made similar.

$$PSize_{min} = \frac{Size_{full}}{Node_{cnt}} \times \alpha \quad (2)$$

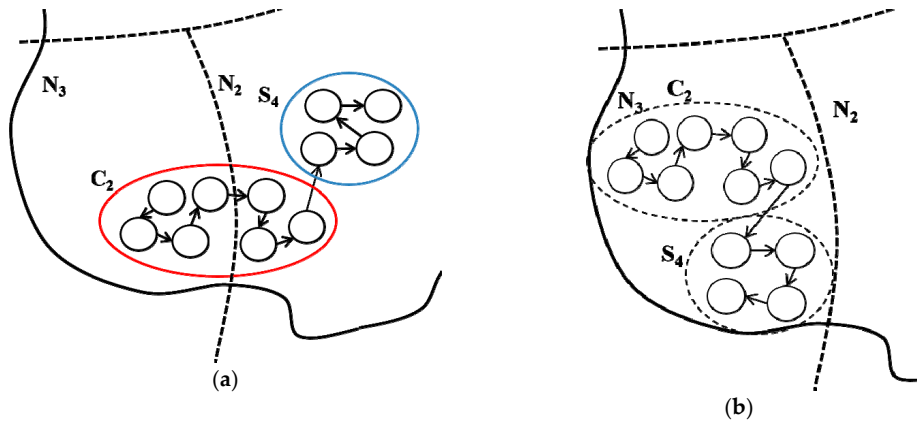
$$PSize_{max} = \frac{Size_{full}}{Node_{cnt}} \times \beta \quad (3)$$

$$\alpha + \beta = 2 \quad (4)$$

To perform effective load balancing of the distributed server, the mean query frequency of the distributed server is considered using Equation (5), where  $Node_{cnt}$  is the number of servers in the distributed server system, and  $QueryFreq_{sum}$  is the sum of the query frequencies in statistical data. If the query frequency of the partitioned graph is smaller than the mean query frequency  $AvgSF$ , it is not generated into a partition.

$$AvgSF = \frac{QueryFreq_{sum}}{Node_{cnt}} \quad (5)$$

Figure 6 shows the process of graph partitioning. It is assumed that 10 million vertices are stored in four servers, and  $\alpha = 0.5$  in Equation (2) and  $\beta = 1.5$  in Equation (3). In addition,  $PSize_{min} = 1,250,000$  in Equation (2), and  $PSize_{max} = 3,750,000$  in Equation (3). In Equation (5),  $AvgSF = 58.5$ . In other words, as for  $PSize_{min}$  and  $PSize_{max}$ , the number of vertices that can be stored in each server has the range of 1,250,000~3,750,000, and the sum of the frequencies of partitions for each server is restricted by the mean query frequency of 58.5.  $C_2$  had 1,800,000 data, and  $S_4$  had 800,000 data. Graph data partitioning is performed by grouping  $C_2$  with  $S_4$ , among subclusters connected with edges, in the same server. Although  $S_3$  is the subcluster connected with  $S_4$  through edge, it is not included in the grouping, because it exceeds the allowed range of Equation (2) and Equation (3), and it would lead to the problem of query concentration to server  $N_3$  based on Equation (4).



**Figure 6.** Graph partitioning process. (a) Cluster and subcluster; (b) partitioning of Server  $N_3$ .

Algorithm 3 shows the algorithm used in the graph partitioning stage. Input parameters include *Clusters* and *Sub\_clusters* generated in the cluster and subcluster generation stage. *CntSub\_clustersPCSN* is the number of subclusters based on *PSCN* order values, *Size\_PartitionPCSN* is the size of the partition currently being generated, and *Freq\_PartitionPCSN* is the frequency of the partition currently being generated. Lines 2~8 generate partitions by appropriately distributing subclusters related to each cluster. If *Freq\_PartitionPCSN* is within the maximum and minimum size of the partition calculated by Equations (2) and (3) and is less than *AvgSF* calculated by Equation (5), the subcluster is included in *PartitionPCSN*.

---

**Algorithm 3** Partitioning Graphs

---

```

1: procedure Partitioning_Graph (Clusters, Sub_clusters)
2:   for PCSN ∈ CntClusters do
3:     new PartitionPCSN
4:     add ClustersPCSN in PartitionPCSN
5:     for SN ∈ CntSub_clustersPCSN do
6:       if  $PSize_{min} < Size\_PartitionPCSN < PSize_{max}$  and  $Freq\_PartitionPCSN \leq AvgSF$ 
7:         add Sub_clustersPCSN_SN in PartitionPCSN
8:     add PartitionPCSN in Partitions
9:   return Partitions

```

---

## 4. Performance Evaluation

### 4.1. Analysis

With the development of various web services, as uses on semantic web have increased, many studies for efficiently storing and managing RDF graphs have been done. Recently, a variety of partitioning methods have been proposed for storing a large scale of RDF graphs in distributed servers. RDF partitioning methods are classified according to various criteria such as partitioning policy, replication policy, load discrimination, partition size, and partitioning conditions. Table 2 shows the analysis results of the existing methods and the proposed method based on such criteria. The partitioning policy is discriminated into the static, partial dynamic, and dynamic, depending on the repartitioning of a partition. The static means the static partitioning method and generates the specific number of partitions by using vertices and edges in the initial RDF graph. The dynamic means the dynamic partitioning method and repartitions the partitioned graph by considering the data size and the queries to adjust the loads of servers. The partial dynamic readjusts partitions stored in the distributed servers due to the addition and deletion of servers and the update of a RDF graph, but it does not consider the loads of servers. The replication policy represents whether a partitioning method supports replication or not. If a partitioning method supports replication, we denote it by ‘○’. Otherwise, we denote it by ‘×’. The load discrimination represents whether a partitioning method discriminates the loads of servers when a partition is repartitioned or not. If it discriminates the loads of servers, we denote load discrimination criteria. Otherwise, we denote it by ‘×’. The partition size represents whether a partitioning method considers the size of a partition or not when it partitions a RDF graph statically or dynamically. Here, SS means that the sizes of partitioned partitions are almost similar, DS means a default partition size that represents a partition with a fixed size, and ‘×’ means that a partitioning method does not consider the partition size. The partitioning conditions means criteria that a partitioning method considers when it partitions the RDF graph.

**Table 2.** Analysis of the partitioning methods according to various criteria.

Methods	Partitioning Policy	Replication	Load Discrimination	Partition Size	Partitioning Condition
MAPG [23]	Static	×	×	SS	Vertex weight, Edge-cut ratio
GPA [40]	Static	×	×	SS	Edge-cut ratio
SPAR [35,38]	Partial Dynamic	○	×	×	Replica
SGP [36]	Partial Dynamic	×	Partition Size	SS	Partition size
Sedge [37]	Dynamic	○	Communication cost, Hotspot	DS	Cross partition query
Hermes [34]	Dynamic	×	Read frequency	Vertex weight	Edge-cut ratio
Proposed	Dynamic	×	Query frequency	$PSize_{min} \sim PSize_{max}$	Cluster/subcluster, Edge-cut ratio, Partition size

Multilevel algorithm for partitioning graph (MAPG) [23] and graph partitioning approach (GPA) [40] that are static partitioning methods generate partitions by considering the edge-cut ratio of a whole graph set. They generate partitions with the similar size. MAPG divides a partition with a specific size within tolerable time. It keeps the numbers of vertices in partitions similarly and decreases the edge-cut ratios among distributed servers. GPA minimizes the edge-cut ratios among distributed servers in order to decrease the communication costs among them when it processes a query. It groups subgraphs connected to the same vertex by scanning the RDF graph. The static partitioning methods are useful for partitioning an initial whole graph set but are impossible to adjust loads due to graph data updates.

SPAR [35,38] and streaming graph partitioning (SGP) [36] that are partial dynamic methods redistribute partitions due to the addition and deletion of servers and the update of a RDF graph. SPAR performs partitioning and replication based on the graph data of social networks in distributed server environments. It supports a replication policy in order to improve locality when it processes a query. SCP performs partitioning by considering a partition size when data is inserted continuously. It partitions the dynamically generated stream RDF graph uniformly by using various heuristics. However, SPAR and SCP do not provide repartitioning strategies considering the query frequency. They also increase communication costs among distributed servers since they do not consider the edge-cut ratio.

Sedge [37] and Hermes [34] perform repartitioning by considering the loads of servers in a similar way as the proposed method. Sedge provides complementary partitioning and on-demand partitioning to improve data locality and to reduce the communication costs among servers. This minimizes the edge-cut ratio by discriminating hot spot data and communication costs as server loads. It also reduces cross partition queries that are processed by multiple servers when repartitioning. Hermes provides lightweight repartitioning for minimizing the edge-cut ratios in order to reduce communication costs among servers. It considers the weight of a vertex as the number of read requests to that vertex and performs repartitioning for evenly distributing the read requests to a particular vertex. Since it replicates the hot spot data, it increases synchronization costs among replicates when a subgraph is changed. Sedge and Hermes minimize the edge-cut ratios in order to reduce server loads. However, they do not consider server loads according to a partition size and the query processing loads. They also do not provide repartitioning criteria considering subgraph patterns used by a query.

The proposed method provides repartitioning in dynamic environments that subgraphs are continuously changed. It does not keep replicates to remove synchronization costs among them and replicates management costs in dynamic environments. It discriminates the query frequency and the sizes of subgraphs in a query as server loads and stores the subgraphs commonly used in several queries in the same server. In order to improve the efficiency of query processing, the proposed method considers edge-cut ratio and partition size between clusters and subclusters. As a result, the proposed method can adjust the loads among servers and reduce the query processing time.

#### 4.2. Evaluation Results

In order to show the excellence of the proposed method, we have conducted various performance evaluations by comparing it with Sedge [37] and Hermes [34], which showed high performance among recently studied methods. We constructed eight PCs as a cluster system for the performance evaluation of the distributed environment and performed the performance evaluation. The specifications of each PC are shown in Table 3. In the evaluation, eight queries were generated, and response time and load distribution were evaluated for each query. Table 4 shows datasets used in the performance evaluation.

**Table 3.** Evaluation environments.

Feature	Value
# of servers	8
Central processing unit (CPU)	Intel® Core™ i3 CPU 540 processor
Random access memory (RAM)	4G
Size of hard disk	1TB
Read speed of hard disk	535MB/sec
Write speed of hard disk	153MB/sec

**Table 4.** Dataset.

Classification	Twitter	DBLP
# of vertices	11.3 million	317 thousand
# of edges	85.3 million	1 million

In order to evaluate the query processing performance, we compared the proposed method with the existing method in terms of query processing time after we create various query types on Twitter and database and logic programming (DBLP) based on query types in SPARQL performance benchmark (SP<sup>2</sup>Bench) [46]. SP<sup>2</sup>Bench provides 12 types of queries for DBLP datasets. The purpose of this paper is to propose a partitioning method to solve the load imbalance in a distributed environment. Therefore, in order to perform performance evaluation for this purpose, we analyzed the query types provided by SP<sup>2</sup>Bench and created query types that perform at least three join operations. Table 5 shows query types used in this experiment. We compare average response times of the partitioning methods that four SPARQL queries on each data set are processed 40 times since a query complexity affects query processing performance. Q1~Q4 are queries on DBLP dataset. Here, Q2 is the most complex and Q1 is the simplest. Q5~Q8 are queries on Twitter data set. Here, Q5 is the most complex and Q8 is the simplest. In order to prove the improvement of the proposed method in the dynamic environment, we measured the average performance of 40 times per time unit of each query.

**Table 5.** Query types.

Type	Query
Q1	<pre> SELECT ?journal ?year WHERE {   ?journal rdf:type swrc:InProceedings.   ?journal dc:type &lt;<a href="http://purl.org/dc/dcmitype/Text">http://purl.org/dc/dcmitype/Text</a>&gt;.   ?journal dcterms:issued ?year }</pre>
Q2	<pre> SELECT ?inproc ?author ?booktitle ?title ?ee ?page ?url ?yr WHERE {   ?inproc rdf:type swrc:Inproceedings.   ?inproc dc:creator ?author.   ?inproc swrc:booktitle ?booktitle.   ?inproc dc:title ?title.   ?inproc rdfs:seeAlso ?ee.   ?inproc swrc:pages ?page.   ?inproc foaf:homepage ?url.   Inproc dcterms:issued ?yr } ORDER BY ?yr</pre>
Q3	<pre> SELECT ?article WHERE {   ?article rdf:type ontology:article.   ?article dc:type dcmitype:Text.   ?article ?property ?value   FILTER (?property=swrc:pages) }</pre>



Table 5. Cont.

Type	Query
Q4	<pre> SELECT DISTINCT ?name1 ?name2 WHERE {   ?article1 rdf:type swrc:article.   ?article2 rdf:type swrc:article.   ?article1 dc:creator ?author1.   ?author1 foaf:name ?name1.   ?article2 dc:creator ?author2.   ?author2 foaf:name ?name2 }</pre>
Q5	<pre> SELECT DISTINCT ?person8 WHERE {   twr:MattKeith foaf:knows ?person2.   ?person2 foaf:knows ?person3.   ?person3 foaf:knows twr:damn.   twr:damn foaf:knows ?person4.   ?person4 foaf:knows ?person5.   ?person5 foaf:knows twr:paul.   twr:paul foaf:knows ?person6.   ?person6 foaf:knows ?person7.   ?person7 foaf:knows person8 }</pre>
Q6	<pre> SELECT ?name1 ?postid1 WHERE {   twr:Roberts foaf:knows ?person2.   ?person2 foaf:name ?name1.   ?post1 sioc:has_creator ?name1.   twr:paul foaf:knows ?person3.   ?person3 foaf:name ?name2.   ?post2 sioc:has_creator ?name2.   FILTER(?name1=?name2) }</pre>
Q7	<pre> SELECT ?person2 ?person3 WHERE {   ?post1 sioc:has_creator twr:paul.   ?post1 sioc:has_creator ?person2.   ?post2 sioc:has_creator ?person2.   ?post2 sioc:has_creator ?person3. }</pre>
Q8	<pre> SELECT ?person1 ?post WHERE {   twr:damn foaf:knows ?person.   ?post foaf:has_creator ?person.   ?post sioc:post ?postid }</pre>

The proposed method adjusts the minimum partition size and maximum partition size stored in a distributed server. In order to perform performance comparison according to partition size, we measure the ratios of distributed storage stored in distributed servers and query response time as we change parameters  $\alpha$  and  $\beta$  in Equations (2) and (3). Since the performance evaluation according to the size of partitions performs the performance comparison according to  $\alpha$  and  $\beta$  shown in Equations (2) and (3) in the dynamic environment, the initial partitioning environment has no significant effect on performance. In the experimental evaluation, the initial graph partition is performed by Metis [24], which is a representative method for performing static graph partitioning on a given graph. Metis performs edge-cut partitioning that minimizes the number of edge-cuts to minimize communication

costs between the partitioned subgraphs. Figures 7 and 8 show the ratios of distributed storage stored in distributed servers according to the change of parameters  $\alpha$  and  $\beta$ . The storage ratio is calculated from Equation (6). Here,  $SR_i$  represents the storage ratio of the server  $i$ ,  $TR_i$  represents the total storage size of the server  $i$ , and  $SS_i$  represents the size of the RDF graph stored in the server  $i$ . The minimum and maximum sizes of partitions stored in the distributed server are determined by Equations (2) and (3). As shown in Equation (4), since it is  $\alpha + \beta = 2$ ,  $\beta$  has a relatively large value when  $\alpha$  is too small. That is, the smaller  $\alpha$  is, the bigger  $\beta$  is, so the difference between the minimum partition size and the maximum partition size to be stored in the distributed server is large. Conversely, the larger  $\alpha$  is, the smaller  $\beta$  becomes relatively small, and the size of partitions stored in the distributed server is likely to be similar. Therefore, the size of partitions stored in the distributed server shows a lot of difference. It is shown through experiments that a partition size stored in a server is very big when the values of parameter  $\alpha$  are 0.1 and 0.3 and the difference of the ratios of distributed storage stored in distributed servers decreases relatively when the value of parameter  $\alpha$  is 0.5 or more.

$$SR_i = \frac{SS_i}{TS_i} \quad (6)$$

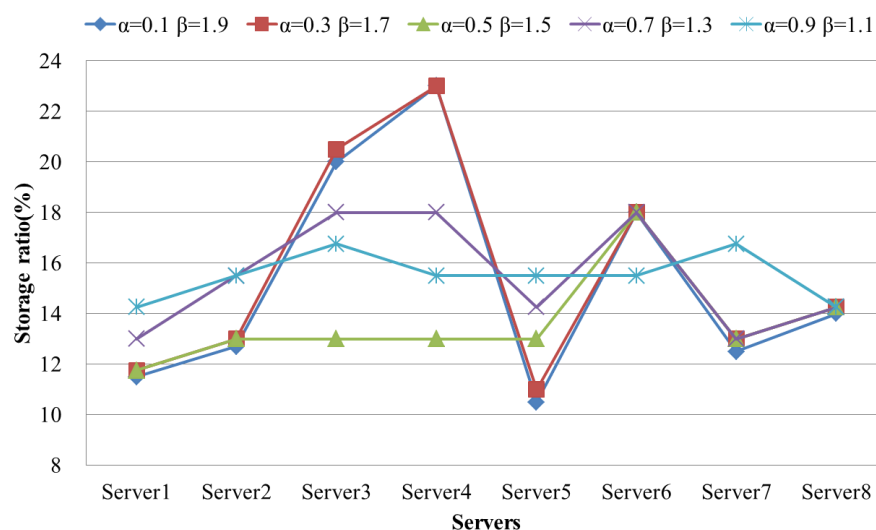


Figure 7. Ratio of distributed storage according to the change of parameters  $\alpha$  and  $\beta$  on DBLP dataset.

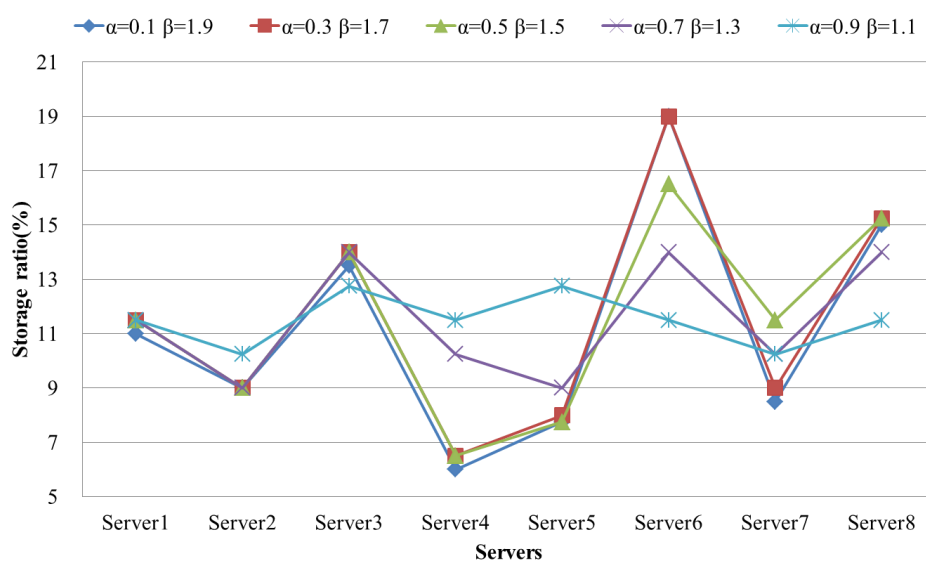


Figure 8. Ratio of distributed storage according to the change of parameters  $\alpha$  and  $\beta$  on Twitter dataset.

Figures 9 and 10 show query response times according to the change of parameters  $\alpha$  and  $\beta$ . Similar to Figures 7 and 8, the smaller  $\alpha$  is, the larger  $\beta$  is, indicating a large difference in the size of partitions stored in the distributed server. Also, the smaller  $\alpha$  is, the more  $\beta$  is increased and the partition is likely to be partitioned by edge-cutting. As shown in the figure, the smaller  $\alpha$  becomes, the lower the relative response time is. When the value of parameter  $\alpha$  is big, each server should store a particular size of partition. Therefore, the edge-cut ratios of clusters and subclusters used in a query increase. It is shown through experiments that the query response time of the proposed method increases when the value of parameter  $\alpha$  is 0.7 or more. Based on these experiment results, we perform performance evaluations by setting the values of parameters  $\alpha$  and  $\beta$  to 0.5 and 1.5.

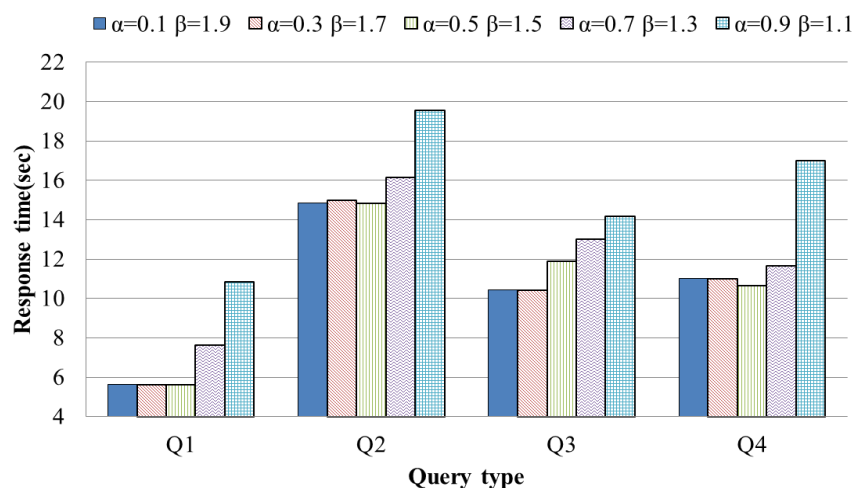


Figure 9. Query response time according to the change of parameters  $\alpha$  and  $\beta$  on DBLP dataset.

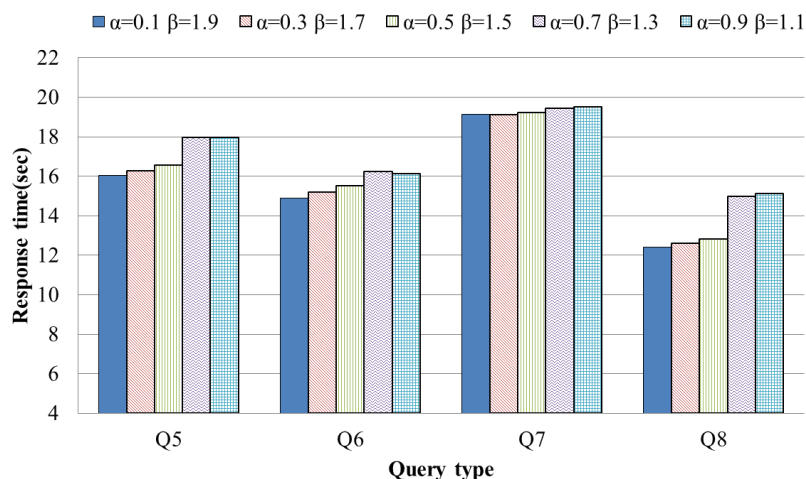


Figure 10. Query response time according to the change of parameters  $\alpha$  and  $\beta$  on Twitter dataset.

The proposed method performs graph data partitioning by minimizing the number of edge-cuts to minimize communication cost. To demonstrate the effective minimization of the number of edge-cuts, edge-cut ratios are evaluated in comparison with existing graph data partitioning methods. Figure 11 shows the results of the comparison of edge-cut ratios among methods when performing RDF graph data partitioning. The results showed the improvement of Hermes in terms of edge-cut ratios. Hermes focuses on minimizing edge-cut ratio in order to reduce communication costs among distributed servers. However, the proposed method performs partitioning by considering various criteria such as a query frequency, a partition size, and the number of connected edges between a cluster and a subcluster. Therefore, although Hermes reduces the edge-cut ratio over the proposed method, the proposed method outperforms Hermes even in terms of the communication cost. The proposed

method also outperforms the existing methods in terms of ratio of distributed storage, query response time, and standard deviation of response time since it considers a query frequency, a partition size, and the number of connected edges between a cluster and a subcluster.

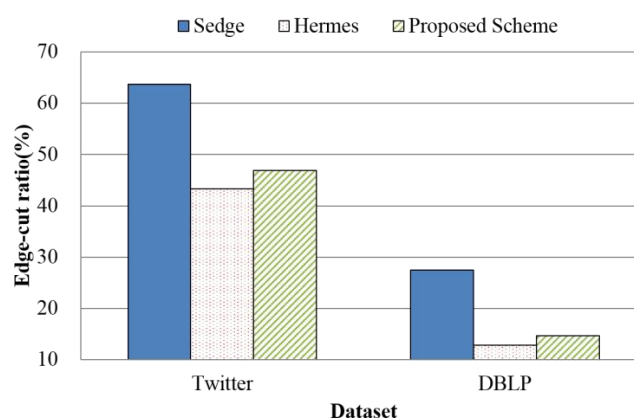


Figure 11. Edge-cut ratios.

In terms of data storage in the dynamic RDF graph environment, to solve the data concentration problem, the proposed method considered the size of the partitions when performing graph data partitioning. To demonstrate the improvement of the proposed method, an experiment on the data size ratio of the data stored across distributed servers was conducted during RDF graph data partitioning in the comparison between existing methods and the proposed method. To prove the improvement of the proposed method, we measured the average storage ratio of 40 times for each query. Figures 12 and 13 show the storage ratio of data stored across distributed servers when RDF graph partitioning is performed with DBLP and Twitter datasets. In the experiment, standard deviations of the size of the data stored in the servers were obtained for evaluation. Evaluation results showed that the proposed method outperformed Sedge and Hermes in terms of reduced data concentration to specific servers and equal storage distribution across servers. In the experiment, data concentration was found in server 4 in Sedge and server 5 in Hermes. Sedge and Hermes try to minimize the edge-cut ratio in order to reduce the communication costs among servers when they process a query. Therefore, they can store a large number of subgraphs in a specific server. However, in the proposed method, the storage ratios of distributed servers are almost similar because both a cluster size and the edge-cut ratios between clusters and subclusters are considered.

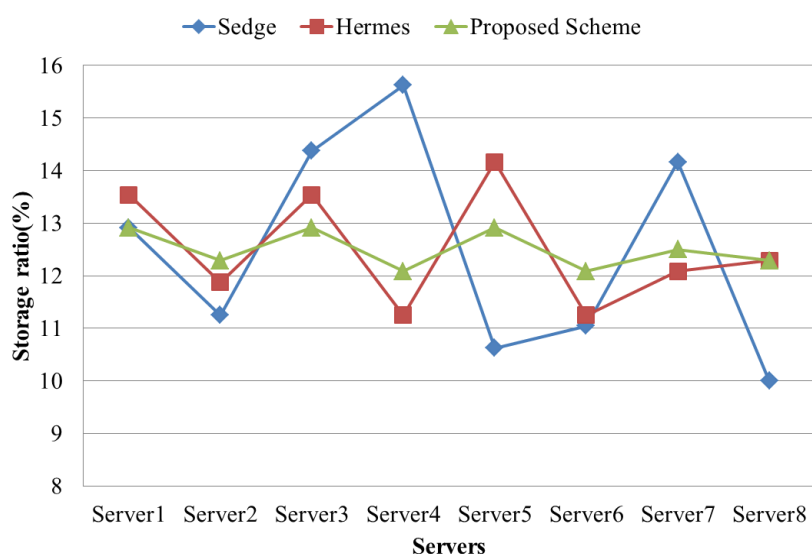


Figure 12. Ratio of distributed storage of DBLP dataset.

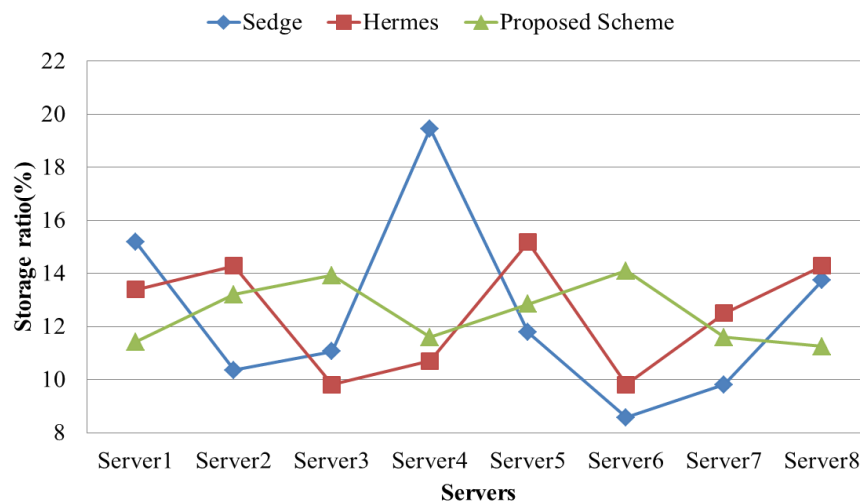


Figure 13. Ratio of distributed storage of Twitter dataset.

The proposed method is designed to provide fast query response while meeting users' diverse needs. To show the improvement of the proposed method in this aspect, an experiment on query response time for various queries was conducted in comparison between the existing methods and the proposed method. Figures 14 and 15 show the results of comparison in query response time of Sedge, Hermes, and the proposed method using eight different queries for the DBLP and Twitter dataset. Sedge and Hermes reduce communication costs among distributed servers since they repartition a partition by minimizing an edge-cut ratio. Hermes outperforms Sedge since it considers read request frequency for repartitioning. However, since Sedge and Hermes do not consider the frequently used subgraphs for processing a query when they perform repartitioning, they can store subgraphs related to the query in several servers. As a result, when they process a query, they increase a query response time due to the communication costs of the distributed servers. On the other hand, the proposed method performs grouping based on the subgraph with high data use frequency, and was designed to create partitions in the size within the range of threshold values that users set. In other words, the proposed method employs the different graph data partitioning process than those of existing methods, and provides fast query response by managing frequently requested subgraph in the same server. Performance evaluation results with the DBLP data showed the query response time of the proposed method was improved by about 10% over Sedge, and about 3% over Hermes. In addition, the results with the Twitter data, the proposed group improved by approximately 7% than Sedge, and approximately 4% than Hermes.

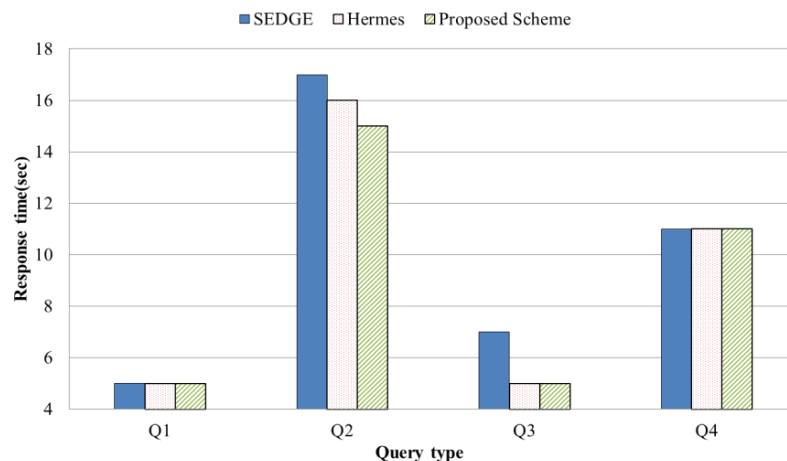


Figure 14. Query response time with DBLP dataset.

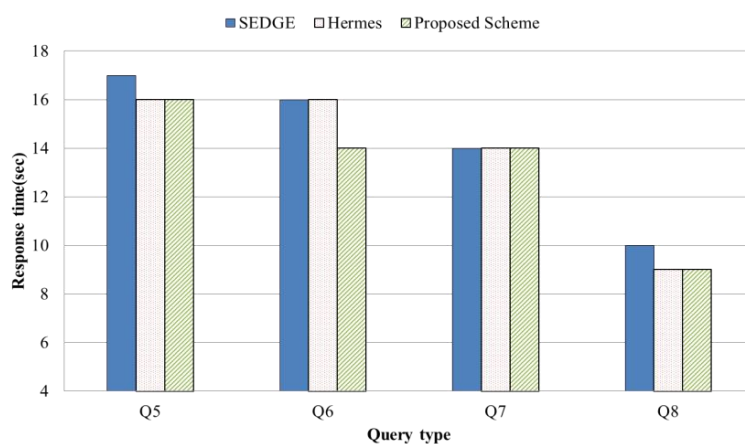


Figure 15. Query response time with Twitter dataset.

The proposed method performs efficient load balancing based on statistical data without performing data replication. To demonstrate the improvement of the proposed method in load balancing, standard deviations of query response time according to the number of query requests were obtained to provide a comparison between existing methods and the proposed method. Figures 16 and 17 show the standard deviations of the response time across the servers when four queries were requested for DBLP and Twitter dataset. In Sedge and Hermes where four queries were performed many times simultaneously, servers experienced high loading with the increased number of query requests, and overall response time increased accordingly. Sedge and Hermes store subgraphs related to a query in several servers and do not consider the frequently used subgraphs for processing a query when they perform repartitioning, they can store subgraphs related to the query in several servers. As a result, when they process a query, they increase a query response time due to the communication costs among the distributed servers. On the other hand, the proposed method performs grouping based on the subgraph with high data use frequency. It was designed to create partitions in the size within the range of threshold values that users set. The proposed method also showed an increase in overall query response time when the number of query requests increased. However, the proposed method demonstrated improvement in response time by performing efficient load balancing based on the mean query frequency of the distributed server when performing graph data partitioning. According to the performance evaluation results, with DBLP data, the proposed method showed an approximately 23% improvement compared to Sedge and an approximately 10% improvement compared to Hermes. With Twitter data, the proposed method showed an approximately 55% improvement compared to Sedge and an approximately 17% improvement compared.

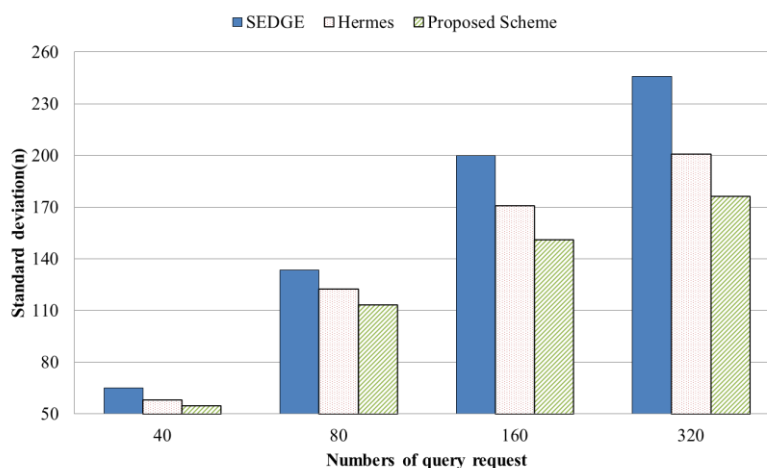


Figure 16. Standard deviation of response time with DBLP dataset.



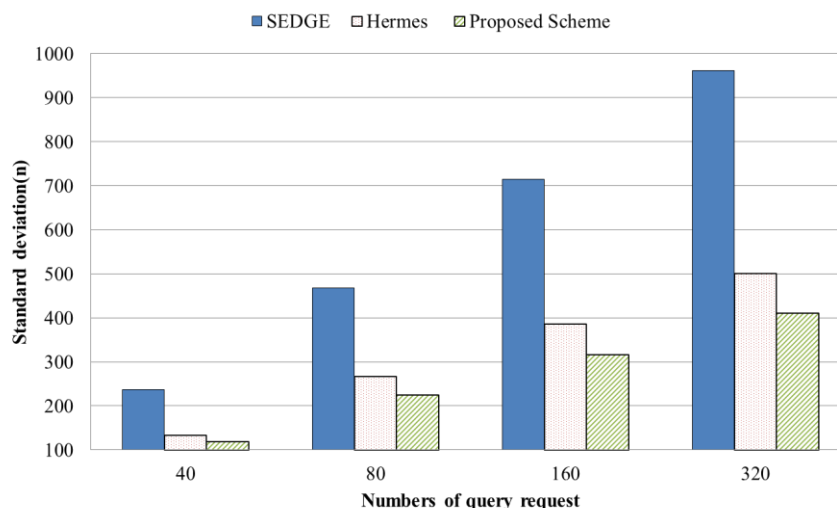


Figure 17. Standard deviation of response time with Twitter dataset.

## 5. Conclusions

In this paper, we proposed a new RDF partitioning method to address the load imbalance of RDF graphs stored in a distributed store in a dynamic environment. The proposed method aims to balance the storage ratio and response time of the distributed store. The proposed method solved RDF graph concentration to specific servers by performing grouping based on a frequently used subgraph and assigning data to the distributed server properly by considering data size. It also performed graph data partitioning by minimizing the number of edge-cuts to reduce inter-server communication costs. These allowed the proposed method to outperform existing methods by maximizing the advantages of distributed processing and providing quick query responses. It was shown through performance evaluation that Hermes reduces the edge-cut ratios over the proposed method since Hermes focuses on minimizing the edge-cut ratio in order to reduce communication costs among distributed servers. However, although Hermes reduces the edge-cut ratio over the proposed method, the proposed method outperforms Hermes even in terms of the communication cost. The proposed method also outperforms the existing methods in terms of ratio of distributed storage, query response time, and standard deviation of response time since it considers a query frequency, a partition size, and the number of connected edges between a cluster and a subcluster. It does not improve query processing performance significantly compared to the existing methods, but makes sense in that it improves the storage ratio. Future research will be conducted to increase the number of servers in various system environments to ensure the scalability of the proposed method.

**Author Contributions:** Conceptualization, K.B., C.K. and J.Y.; methodology, K.B., C.K. and J.Y.; validation, K.B., C.K.; formal analysis, K.B. and C.K.; writing—original draft preparation, K.B. and C.K.; writing—review and editing, J.Y.

**Funding:** This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(No. NRF-2019R1I1A1A01062289), and by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT (No. NRF-2017M3C4A7069432), and by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2019R1H1A2079843).

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Decker, S.; Melnik, S.; Harmelen, F.; Fensel, D.; Klein, M.C.A.; Broekstra, J.; Erdmann, M.; Horrocks, I. The Semantic Web: The Roles of XML and RDF. *IEEE Internet Comput.* **2000**, *4*, 63–73. [[CrossRef](#)]
2. Gomez-Perez, A.; Corcho, O. Ontology Languages for the Semantic Web. *IEEE Intell. Syst.* **2002**, *17*, 54–60. [[CrossRef](#)]

3. Arenas, M.; Pérez, J. Querying Semantic Web Data with SPARQL. In Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, Athens, Greece, 12–16 June 2011; pp. 305–316.
4. Huang, J.; Abadi, D.J.; Ren, K. Scalable SPARQL Querying of Large RDF Graphs. *Proc. VLDB Endow.* **2011**, *4*, 1123–1134.
5. Kim, K.; Moon, B.; Kim, H. R3F: RDF triple filtering method for efficient SPARQL query processing. *World Wide Web* **2015**, *18*, 317–357. [[CrossRef](#)]
6. Neumann, T.; Weikum, G. The RDF-3X engine for scalable management of RDF data. *VLDB J.* **2010**, *19*, 91–133. [[CrossRef](#)]
7. Frey, J.; Müller, K.; Hellmann, S.; Rahm, E.; Vidal, E. Evaluation of metadata representations in RDF stores. *Semant. Web* **2019**, *10*, 205–229. [[CrossRef](#)]
8. Bae, M.; Kihm, J.; Kang, S.; Oh, S. Indexing and querying algorithm based on structure indexing for managing massive-scale RDF data. *J. Intell. Fuzzy Syst.* **2014**, *27*, 575–587.
9. Hammoud, M.; Rabbou, D.A.; Nouri, R.; Beheshti, S.; Sakr, S. DREAM: Distributed RDF Engine with Adaptive Query Planner and Minimal Communication. *Proc. VLDB Endow.* **2015**, *8*, 654–665. [[CrossRef](#)]
10. Fernández, J.D.; Umbrich, J.; Polleres, A.; Knuth, M. Evaluating query and storage strategies for RDF archives. *Semant. Web* **2019**, *10*, 247–291. [[CrossRef](#)]
11. Wylot, M.; Hauswirth, M.; Cudré-Mauroux, P.; Sakr, S. RDF Data Storage and Query Processing Schemes: A Survey. *ACM Comput. Surv.* **2018**, *51*, 84. [[CrossRef](#)]
12. Pan, Z.; Zhu, T.; Liu, H.; Ning, H. A survey of RDF management technologies and benchmark datasets. *J. Ambient Intell. Hum. Comput.* **2018**, *9*, 1693–1704. [[CrossRef](#)]
13. Özsü, M.T. A survey of RDF data management systems. *Front. Comput. Sci.* **2016**, *10*, 418–432. [[CrossRef](#)]
14. Ouksili, H.; Kedad, Z.; Lopes, S.; Nugier, S. Pattern oriented RDF graphs exploration. *Data Knowl. Eng.* **2018**, *113*, 171–183. [[CrossRef](#)]
15. Zou, L.; Özsü, M.T. Graph-Based RDF Data Management. *Data Sci. Eng.* **2017**, *2*, 56–70. [[CrossRef](#)]
16. Galarraga, L.; Hose, K.; Schenkel, R. Partout: A Distributed Engine for Efficient RDF Processing. In Proceedings of the International World Wide Web Conference, Seoul, Korea, 7–11 April 2014; pp. 267–268.
17. Janke, D.; Staab, S.; Thimm, M. Impact analysis of data placement strategies on query efforts in distributed RDF stores. *J. Web Semant.* **2018**, *50*, 21–48. [[CrossRef](#)]
18. Guo, X.; Gao, H.; Zou, Z. Leon: A Distributed RDF Engine for Multi-query Processing. In Proceedings of the International Conference on Database Systems for Advanced Applications, Chiang Mai, Thailand, 22–25 April 2019; pp. 742–759.
19. Hassan, M.; Bansal, S.K. RDF Data Storage Techniques for Efficient SPARQL Query Processing Using Distributed Computation Engines. In Proceedings of the International Conference on Information Reuse and Integration, Salt Lake City, UT, USA, 6–9 July 2018; pp. 323–330.
20. Abdelaziz, I.; Harbi, R.; Khayyat, Z.; Kalnis, P. A Survey and Experimental Comparison of Distributed SPARQL Engines for Very Large RDF Data. *Proc. VLDB Endow.* **2017**, *10*, 2049–2060. [[CrossRef](#)]
21. Leng, Y.; Chen, Z.; Wang, H.; Zhong, F. A Partitioning and Index Algorithm for RDF Data of Cloud-Based Robotic Systems. *IEEE Access* **2018**, *6*, 29836–29845. [[CrossRef](#)]
22. Peng, P.; Zou, L.; Chen, L.; Zhao, D. Adaptive Distributed RDF Graph Fragmentation and Allocation based on Query Workload. *IEEE Trans. Knowl. Data Eng.* **2019**, *31*, 670–685. [[CrossRef](#)]
23. Hendrickson, B.; Leland, R. A multilevel algorithm for partitioning graphs. In Proceedings of the ACM/IEEE conference on Supercomputing, San Diego, CA, USA, 4–8 December 1995; pp. 1–14.
24. Karypis, G.; Kumar, V. *METIS-Unstructured Graph Partitioning and Sparse Matrix Ordering System Version 2.0*; Technical Report; Department of Computer Science, University of Minnesota: Minneapolis, MN, USA, 1995.
25. Malewicz, G.; Austern, M.H.; Bik, A.J.C.; Dehnert, J.C.; Horn, I.; Leiser, N.; Czajkowski, G. Pregel: A System for Large-Scale Graph Processing. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Indianapolis, IN, USA, 6–10 June 2010; pp. 135–146.
26. Chawla, T.; Singh, G.; Pilli, E.S. HyPSo: Hybrid Partitioning for Big RDF Storage and Query Processing. In Proceedings of the ACM India Joint International Conference on Data Science and Management of Data, Kolkata, India, 3–5 January 2019; pp. 188–194.

27. Xu, Q.; Wang, X.; Xin, Y.; Feng, Z.; Chen, R. PDSM: Pregel-Based Distributed Subgraph Matching on Large Scale RDF Graphs. In Proceedings of the Companion Proceedings of the Web Conference, Lyon, France, 23–27 April 2018; pp. 17–18.
28. Liu, J.; Chen, J.; Rao, Z.; Sun, Z.; Yang, H.; Xu, R. A massive RDF storage approach based on graph database. In Proceedings of the International Conference on Geoinformatics and Data Analysis, Prague, Czech Republic, 20–22 April 2018; pp. 169–173.
29. Xu, Q.; Wang, X.; Wang, J.; Yang, Y.; Feng, Z. Semantic-Aware Partitioning on RDF Graphs. In Proceedings of the International Joint Conference APWeb-WAIM, Beijing, China, 7–9 July 2017; pp. 149–157.
30. Al-Ghezi, A.I.A.; Wiese, L. Adaptive Workload-Based Partitioning and Replication for RDF Graphs. In Proceedings of the International Conference on Database and Expert Systems Applications, Regensburg, Germany, 3–6 September 2018; pp. 250–258.
31. Potter, A.; Motik, B.; Nenov, Y.; Horrocks, I. Distributed RDF Query Answering with Dynamic Data Exchange. In Proceedings of the International Semantic Web Conference, Kobe, Japan, 17–21 October 2016; pp. 480–497.
32. Potter, A.; Motik, B.; Nenov, Y.; Horrocks, I. Dynamic Data Exchange in Distributed RDF Stores. *IEEE Trans. Knowl. Data Eng.* **2018**, *30*, 2312–2325. [[CrossRef](#)]
33. Peng, P.; Zou, L.; Özsu, M.T.; Chen, L.; Zhao, D. Processing SPARQL queries over distributed RDF graphs. *VLDB J.* **2016**, *25*, 243–268. [[CrossRef](#)]
34. Nicoara, D.; Kamali, S.; Daudjee, K.; Chen, L. Hermes: Dynamic Partitioning for Distributed Social Network Graph Databases. In Proceedings of the International Conference on Extending Database Technology, Brussels, Belgium, 23–27 March 2015; pp. 25–36.
35. Pujol, J.M.; Erramilli, V.; Siganos, G.; Yang, X.; Laoutaris, N.; Chhabra, P.; Rodriguez, P. The little engine (s) that could: Scaling online social networks. In Proceedings of the ACM SIGCOMM 2010 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, New Delhi, India, 30 August–3 September 2010; pp. 375–386.
36. Stanton, I.; Klot, G. Streaming graph partitioning for large distributed graphs. In Proceedings of the International Conference on Knowledge Discovery and Data Mining, Beijing, China, 12–16 August 2012; pp. 1222–1230.
37. Yang, S.; Yan, X.; Zong, B.; Khan, A. Towards effective partition management for large graphs. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Scottsdale, AZ, USA, 20–24 May 2012; pp. 517–528.
38. Pujol, J.M.; Erramilli, V.; Siganos, G.; Yang, X.; Laoutaris, N.; Chhabra, P.; Rodriguez, P. The Little Engine (s) That Could: Scaling Online Social Networks. *IEEE/ACM Trans. Netw.* **2012**, *20*, 1162–1175. [[CrossRef](#)]
39. Bok, K.; Kim, C.; Jeong, J.; Lim, J.; Yoo, J. Dynamic Partitioning of Large Scale RDF Graph in Dynamic Environments. In Proceedings of the International Conference on Emerging Databases, Busan, Korea, 7–9 August 2017; pp. 43–49.
40. Wang, R.; Chiu, K. A Graph Partitioning Approach to Distributed RDF Stores. In Proceedings of the International Conference on Parallel Processing, Leganes, Madrid, Spain, 10–13 July 2012; pp. 411–418.
41. Troullinou, G.; Kondylakis, H.; Plexousakis, D. Semantic Partitioning for RDF Datasets. In Proceedings of the 11th International Workshop on Information Search, Integration, and Personalization (ISIP), Lyon, France, 1–4 November 2016; Volume 760, pp. 99–115.
42. Leng, Y.; Chen, Z.; Zhong, F.; Li, X.; Hu, Y.; Yang, C. BRGP: A balanced RDF graph partitioning algorithm for cloud storage. *Concurr. Comput. Pract. Exp.* **2017**, *29*, e3896. [[CrossRef](#)]
43. Hayes, J.; Gutiérrez, C. Bipartite Graphs as Intermediate Model for RDF. In Proceedings of the International Semantic Web Conference, Hiroshima, Japan, 7–11 November 2004; pp. 47–61.
44. Tomaszuk, D.; Skonieczny, L.; Wood, D. RDF Graph Partitions: A Brief Survey. In Proceedings of the International Conference on Beyond Databases, Architectures and Structures, Ustroń, Poland, 26–29 May 2015; pp. 256–264.
45. Akhter, A.; Ngomo, A.N.; Saleem, M. An Empirical Evaluation of RDF Graph Partitioning Techniques. In Proceedings of the International Conference on Knowledge Engineering and Knowledge Management, Nancy, France, 12–16 November 2018; pp. 3–18.

46. Schmidt, M.; Hornung, T.; Lausen, G.; Pinkel, C. SP<sup>2</sup>Bench: A SPARQL Performance Benchmark. In Proceedings of the International Conference on Data Engineering, Shanghai, China, 29 March–2 April 2009; pp. 222–233.



© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).