CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. (2016)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3896

SPECIAL ISSUE PAPER

BRGP: a balanced RDF graph partitioning algorithm for cloud storage

Yonglin Leng[1], Chen Zhikui[1,*,†], Fangming Zhong[1], Xiongjiu Li[1], Yueming Hu[2] and Chao Yang[3]

[1]*School of Software Technology, Dalian University of Technology, 116620, Dalian, China*
[2]*College of Natural Resources and Environment, South China Agricultural University, 510642, Guangzhou, China*
[3]*Petroleum Production Engineering Research Institute Of Huabei Oilfield Company, 062550, Renqiu, China*

SUMMARY

The continuous growth of resource description framework (RDF) data poses an important challenge on RDF data partitioning that is a vital technique for effective cloud storage. Recently, many partitioning algorithms for large RDF data have been developed, and most of them are based on graph partitioning. However, existing graph partitioning methods could not partition asymmetric RDF data effectively, resulting in a lower performance for cloud storage. This paper proposes a balanced RDF graph partitioning algorithm for storing massive RDF data on cloud. We first devise a modularity-based multi-level label propagation algorithm (MMLP) to partition RDF graph roughly and then use a balanced K-mediods clustering algorithm for final k-way partitioning. Balanced RDF graph partitioning algorithm designs an effective label update rule and a balanced modification strategy to achieve a high quality coarsening result and make the partition as equilibrium as possible. Experiments are carried on two representative RDF benchmarks and one real RDF dataset by comparison with two representative graph partitioning methods, that is, METIS and MLP+METIS. Results demonstrate that our proposed scheme can produce a high-quality partition for massive RDF data storage on cloud. Copyright © 2016 John Wiley & Sons, Ltd.

Received 27 March 2016; Revised 14 May 2016; Accepted 17 May 2016

KEY WORDS: cloud storage; RDF; balanced clustering partition; multi-level label propagation

1. INTRODUCTION

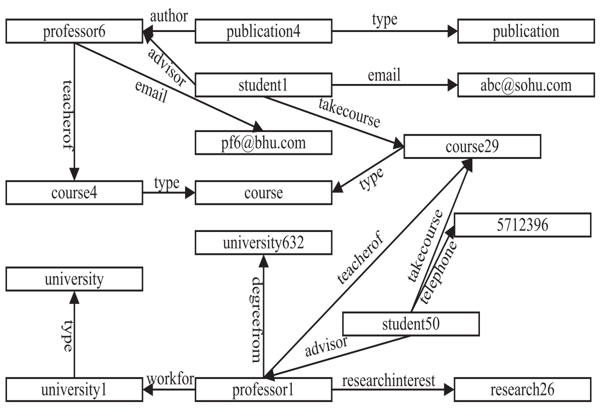
The resource description framework (RDF) [1] has been widely used as the standard data model for Web resources organization and storage because of its simplicity and flexibility. The RDF data model [2] is usually described by a set of triples or a directed graph, as shown in Figure 1. Recently, with the rapid increasing of Web resources, the RDF data size is growing with a high speed. A large number of RDF data poses an important challenge on storage [3, 4]. Specially, a single server cannot store such large scale data. Recently, cloud storage has emerged as an effective tool for storing massive data, even big data, by integrating a lot of storage devices on the Internet. Cloud provides the storage services at a lower cost and a higher scalability for massive data. Obviously, cloud storage can be used for massive RDF data storage [5–7].

To store large RDF data on cloud effectively, RDF data needs to be divided into many data blocks that will be distributed to different storage nodes. An effective RDF data partitioning can improve the query efficiency by reducing the communication overhead among different storage nodes [4].

*Correspondence to: Chen Zhikui, School of Software Technology, Dalian University of Technology, 116620, Dalian, China.
†E-mail: zkchen@dlut.edu.cn

Subject	Predicate	Object
professor1	degreefrom	university632
professor1	workfor	university1
professor6	email	pf6@bhu.com
professor6	teacherof	course4
course4	type	course
university1	type	university
student50	advisor	professor1
student50	telephone	5712396
student50	takecourse	course29
……	……	……
……	……	……

(a)



(b)

Figure 1. Resource description framework (RDF) data model. (a) RDF triples; (b) RDF graph.

Therefore, RDF data partitioning is a vital technique for high efficient cloud storage. Recently, many RDF data partitioning methods have been proposed [4, 8–13]. Most of them are based on graph partitioning that usually distributes closely related nodes to the same storage node. By this way, most of the join operations in SPARQL query [14] can be executed inside one storage node to improve the efficiency of the complex SPARQL query in parallel.

Multilevel graph partitioning method is more effective than other graph partitioning method in face of large scale RDF data. Huang *et al.* [4] utilized a multilevel graph partitioner, that is METIS [15], to optimize RDF data partitioning. The coarsening scheme of METIS is to collapse two adjacent vertices to form a matching. A matching does not allow two of edges to incident on the same vertex. According to the statistics, many real RDF graphs are asymmetric that is the neighbors of vertex (i.e., the degree of vertex) are uneven. For example, over 90% vertices have only less than five neighbors while some vertices with more than 100,000 neighbors in DBpedia. In the asymmetric distributed graph, the degree distribution prevents large matchings of METIS, leading to a slow shrink in the number of edges and the size of coarsened graph [16]. Wang *et al.* [12] introduced a multi-level label propagation algorithm [17] (MLP) to coarsen the graph and then used METIS to partition the coarsened graph. MLP assumed that vertices sharing many common neighbor vertices should have the same label. However, the correlations among vertices are subject-object in the RDF graph, which rarely exist common neighbor vertices. Thus, the update rule based on common neighbor vertices is unsuitable for RDF graph partitioning.

Based on the aforementioned analyses, our goal is to build a suitable RDF graph partitioning algorithm, which must address challenges introduced by RDF graph characteristics, that is, the asymmetric distributed leads to the limitation of coarsening size and the unequal relation of vertices causes the invalid label update.

To achieve the aforementioned challenges, this paper proposes a balanced RDF graph partitioning algorithm (BRGP) for storing massive RDF data on cloud. BRGP provides two more suitable methods in the graph coarsening and cluster partition phase for the asymmetric and the unequal relation of vertices in RDF graph. To summarize, our contributions include the following:

- A label update rule based on modularity is introduced to improve the coarsening quality and efficiency of the asymmetric RDF graph.
- A label energy function is proposed to avoid the 'monster' label, by which the label energy decreases as the label propagation and the capacity of coarsened vertex is more even.
- A balanced adjustment strategy based on the edge weight and vertex weight is integrated with K-medoids to implement k-way partitioning.
- An equivalent pruning strategy is proposed to decrease the size of original graph.
- A set of extensive experiments are executed to verify the partitioning performance and quality of BRGP algorithm.

The rest of this paper is organized as follows: Section 2 reviews some related works about RDF data partitioning. In Section 3, we introduce some notations that are closely related to our work, and then we present our BRGP algorithm and propose an equivalent pruning strategy in Section 4. Section 5 provides a detailed experimental evaluation. Finally, we conclude the paper.

2. RELATED WORK

File and hashing are two RDF data partitioning methods based on triples. In [18–20], the file partitioning and placement policies are employed to partition and distributed RDF data in the vanilla Hadoop. Hash partitioning is another typical method, which is widely used in many distributed RDF engines [8–10]. This method distributes RDF triples into different partitions by computing a hash key over either the subject or the object of each triple. In this way, the triples with a common subject or object would be distributed to the same storage node. A simple SPARQL query with only one center vertex, and one or more edges pointing from the center vertex to other vertices, can be completely executed in parallel without any communication across the storage nodes in the case of the file and hash partitioning. However, file and hashing partition ignore the correlation among triples. Therefore, a complex query with more than one center vertex would lead to substantial I/O cost and communication overhead.

Graph-based approaches [21, 22] have also been applied to partition the RDF data by distributing the closely related nodes to an identical storage node. An optimal graph partitioning is to optimize several given criteria, such as minimizing the number of edges spanning different storage nodes, and leveraging the number of vertices in every storage node. Unfortunately, the optimal graph partitioning is a NP-complete problem. Hence, a lot of approximate algorithms are proposed [23–26]. In particular, the heuristic algorithms are effective in finding the sub-optimal solution. For example, Kernighan–Lin (KL) [23] and Fiduccia and Mattheyses (FM) [24] are two typical heuristic methods. The KL method generates a k-way partitioning by recursive bisection, and the FM approach improves KL algorithm in running time. Additionally, simulated annealing [25] and genetic algorithm [26] are also introduced to improve the performance of graph partitioning. However, these approaches require to access the entire graph randomly, which is inefficient in large-scale data partitioning.

The multilevel partitioning algorithms [12, 15, 27–29] are proposed because of its low running time and high performance in tackling large-scale RDF graph. Such approaches consist of three phases: graph coarsening, initial partitioning, and graph refinement. Huang *et al*. [4] adopt METIS [15] as a graph partitioner to partition RDF graph. However, the maximal matching scheme in METIS reduces the convergence size of graph coarsening for an asymmetric RDF graph. Another multilevel graph-partitioning method (MLP) [12] uses LP [17] to coarsen the graph layer by layer and uses METIS to finish k-way partitioning. MLP is effective in partitioning billion-node graph in a reasonable time. But the label update rule of MLP is not applicable to RDF data.

Lee *et al*. [30] propose a data partitioning framework (SPA), which is scalable and customizable and can partition large RDF graph by vertex-centric blocks partitioning mechanism. Nevertheless, SPA suffers from a large amount of data duplication problem. Wu *et al*. [13] introduce a path strategy-based method, which decomposes the RDF graph into the end-to-end paths. Then such paths are considered as the finest partitioning elements to realize the partitioning. They utilize path based balancing strategy as performance measurement, while our approach employ the node based strategy.

3. PRELIMINARIES

Before going into the details of our approach, firstly, we briefly review some notations that are closely related to this paper. The notations are listed in Table I.

Definition 1 (RDF graph)
We define $G = (V, E)$ as a RDF graph, in which the set of vertices V consists of all the subjects and objects of the RDF triples and the set of edges $E \subseteq V \times V$ denotes the directed edges from the

Table I. Frequently used notations.

Notation	Description
$G/G_i/G^{'}$	an RDF graph / subset of G / coarsened graph of G
$V/V_i/V^{'}$	the set of vertices in G / subset of V / the coarsened vertex set
$E/E_i/E^{'}$	the set of edges in G / subset of E / the coarsened edge set
$v/v_i^{'}$	a vertex in G / a coarsened vertex in $G^{'}$
n	the number of vertices in G
m	the number of edges in G
$N(v)$	the set of neighbors of v
k	the number of partitionings
$deg(v)$	the degree of v
P	a partitioning of G
$e(V_i, V_j)$	the edge cut between V_i and V_j
θ	a float factor
$w(v_i^{'})/w(v_i^{'}, v_j^{'})$	the vertex weight of $v_i^{'}$ / the edge weight between $v_i^{'}$ and $v_j^{'}$

RDF, resource description framework.

subjects to objects. We use $n = |V|$ and $m = |E|$ to represent the number of vertices and edges. Given a vertex $v \in V$, $N(v)$ represents the set of neighbors of v and $deg(v)$ denotes the number of edges connecting to vertex v, where $deg(v) = |N(v)|$.

Definition 2 (RDF graph partitioning)
Given a RDF graph $G = (V, E)$, a k-way partitioning P is to divide G into k disjoint partitions $P = \{G_1, G_2, \dots, G_k\}$, where $G_i = (V_i, E_i), i = 1, 2, \dots, k$. For any $i \neq j$, we have $V_i \bigcap V_j = \varphi$ and $\sum_{i=1}^{k} V_i = V$.

Definition 3 (Edge cut)
Given a k-partitioning P of RDF graph, the edge cut refers to the number of edges spanning different partitions. It can be described as follows:

$$EC(P) = \sum_{i=1}^{k} \sum_{j>i}^{k} e(V_i, V_j) \tag{1}$$

Definition 4 (Balanced partitioning)
The balanced partitioning refers to the number of vertices approximately equal to n/k in each partition. A float factor $0 < \theta < 1$ is allowed to make the partitioning size differ in a small range, that is, the number of vertices of G_i satisfies the following property:

$$n(1-\theta)/k \leqslant |V_i| \leqslant n(1+\theta)/k \tag{2}$$

In order to reduce the communication overhead in a SPARQL query and ensure the parallel execution of all storage nodes, our partitioning goal is to maintain the locality of information and the balanced load distribution. That is to say $|V_i| \approx n/k$ and $EC(P)$ are minimized.

Definition 5 (Coarsened graph)
Given a graph $G = (V, E)$, the coarsened graph $G' = (V', E')$ is to collapse the closely connected vertices in G into a coarsened vertex. We have a coarsened vertex set $V^{'} = \{v_1^{'}, v_2^{'}, ..., v_n^{'}\}$ and a coarsened edge set $E^{'}$, where $(v_i^{'}, v_j^{'}) \in E^{'}$ iff $\exists u \in v_i^{'}, v \in v_j^{'}$, and $(u, v) \in E$. The coarsened graph is a weight graph based on vertex and edge. The vertex weight of $v_i^{'}$ is defined as follows:

$$w(v_i^{'}) = \sum\nolimits_{u \in v_i^{'}} w(u) \tag{3}$$

and the edge weight is denoted as follows:

$$w(v_i^{'}, v_j^{'}) = \sum_{e(u,v) \in E, u \in v_i^{'}, v \in v_j^{'}} w(e(u, v)) \tag{4}$$

4. BRGP ALGORITHM

In this section, we present BRGP algorithm to partition the RDF graph. BRGP algorithm includes two steps: step 1 describes how to coarsen the graph (Section 4.1) and step 2 introduces BKMC algorithm for the final k-way partitioning (Section 4.2). To improve the efficiency and reduce the size of graph, we propose an optimization algorithm to pruning the original RDF graph in Section 4.3 Finally, in Section 4.4, we give the complexity analysis of BRGP.

4.1. Graph coarsening

In this section, we devise MMLP algorithm, which is a modularity-based MLP algorithm. MMLP collapses the vertex of graph to form a coarser graph on the upper level. For each level, LP algorithm is performed to find dense sub-graphs, and each sub-graph forms a coarsened vertex of upper level. As an example in Figure 2, the coarsened vertices C_1^2 and C_2^2 in second level come from two dense sub-graphs in the first level. Furthermore, the coarsened graph is a weight graph based on vertex and edge. In Figure 2, the weight of coarsened vertices C_1^2 and C_2^2 are 6 and 4, respectively, and the weight of coarsened edge between C_1^2 and C_2^2 is 2. In order to avoid the 'monster' label, MMLP further introduces the label energy attenuation strategy to balance the size of coarsened vertex. The details of algorithm are listed in Algorithm 1.

Algorithm 1 MMLP

```
Input: $G = (V, E), \delta, \varepsilon$
output: $C = \{c_1, c_2, ..., c_t\}$
Method:
1. Assign an unique label to each vertex of the graph
2. Assign an initial energy to each label
3. Random sort each node
4. For each vertex $v$                          // the label of $v$ is $l_1$
     For each $u \in N(v)$ and $E_{l_2}(u) > \varepsilon$          // the label of $u$ is $l_2$
         Compute $\Delta Q = \Delta Q_1 + \Delta Q_2$
     End for
     Select label $l_2$ with the maximal gain of modularity and $\Delta Q > 0$
     If $|l_2| \geqslant 1$ then
         Select the label with the least vertices
     End if
     Update the label of $v$ with $l_2$
     Update the label energy of $v$
     End for
4. If exist the update labels, run step 3, until the labels no longer update
5. Output coarsening cluster set $C = \{c_1, c_2, \ldots, c_t\}$
6. If $t$ is too big, make the coarsening set $C$ as a new input, run step 1, until $t$ reaches a
relatively moderate value.
```

(1) Label propagation

LP is originally used for community detection in social network. In this paper, we utilize LP to detect the dense sub-graphs during the graph coarsening process because of two factors: (1) each vertex does not need to access other vertex's neighbors besides its neighbors during the label update process, which is very suitable for parallel computing. (2) LP has a linear time complexity.

The basic idea of naive LP is that each vertex is assigned a unique label in initial phase. Then, we iteratively update the label of each vertex. In the process of iteration, each vertex takes the most frequent label of its neighbors as its own label. If there are multiple candidate

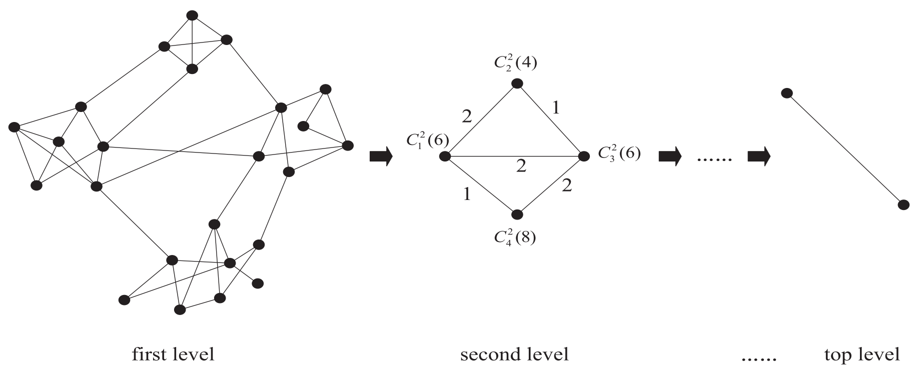


Figure 2. The process of graph coarsening.

labels with the same top frequency, select one randomly. This process terminates until there are no labels change. Vertices with the same label will be distributed to the same community.

However, a large number of experiments show that the label update rule of naive LP would form ‘monster’ label in the asymmetric RDF graph. The reason lies in that the label of high degree vertex has a higher spread probability compared with other vertices. In addition, Wang *et al.* [12] assumed the vertices should be assigned the same label if they shared a lot of common neighbors. Because of the unequal relation among the RDF data vertices, there are few common neighbors between vertex and its neighbors. Therefore, we utilize the gain of modularity and label propagation energy to resolve the aforementioned problems.

(2) Label update rule based on modularity gain

Modularity is introduced by Newman and Girvan [31] to measure the quality of graph clustering, which is a scalar value between -1 and 1. Specially, it is used to measure the density of the intra-clusters edges as compared with the inter-clusters edges. The modularity of clustering graph is described as follows:

$$Q = \sum\nolimits_{i=1}^{k} \left(\frac{I(V_i)}{m} - \left(\frac{\sum_{v \in V_i} \deg(v)}{2m} \right)^2 \right), \tag{5}$$

where $I(V_i)$ is the number of edges inside cluster i. It can be easily inferred that the modularity of cluster i is as follows:

$$Q_i = \frac{I(V_i)}{m} - \left(\frac{\sum_{v \in V_i} \deg(v)}{2m} \right)^2 \tag{6}$$

According to the definition of modularity, the quality of graph clustering increases as the value of modularity increases. Therefore, we use the gain of modularity to update the vertex label. For each vertex v and its neighbor label set $L(v)$, we compute the gain of modularity by changing the label of v into its neighbors label. If a maximum gain exist and it is positive, the label of v will be updated with the neighbor label of maximum gain. Otherwise, it will stay in its original label. If there are multiple maximum labels, then select the label with the minimum vertices.

Because vertex v changes label from l_1 to l_2, the change of modularity only occurs in clusters with label l_1 and l_2. Therefore, we only need to compute modularity gain of two clusters. The clusters with label l_1 and l_2 gain in modularity can be computed by Eqs (7) and (8):

$$\Delta Q_1 = \left[\frac{I(l_1) - N_{l_1}(v)}{m} - \left(\frac{\sum_{j \in l_1} \deg(j) - \deg(v)}{2m} \right)^2 \right] - \left[\frac{I(l_1)}{m} - \left(\frac{\sum_{j \in l_1} \deg(j)}{2m} \right)^2 \right] \tag{7}$$

$$\Delta Q_2 = \left[\frac{I(l_2) + N_{l_2}(v)}{m} - \left(\frac{\sum_{j \in l_2} \deg(j) + \deg(v)}{2m}\right)^2\right] - \left[\frac{I(l_2)}{m} - \left(\frac{\sum_{j \in l_2} \deg(j)}{2m}\right)^2\right], \tag{8}$$

where $I(l_1)$ and $I(l_2)$ are the numbers of edges inside clusters with label l_1 and l_2, respectively, $N_{l_1}(v)$ and $N_{l_2}(v)$ are the neighbor of v with label l_1 and l_2, respectively. Finally, we calculate the total gain by Eq. (9) to find the most appropriate label l_2, and change label of v from l_1 to l_2.

$$\Delta Q = \Delta Q_1 + \Delta Q_2 \tag{9}$$

(3) Label propagation energy

Because of the asymmetric distribution of data, LP algorithm produces several large scale clusters and many small size clusters, which affects the balance of the final partition. To avoid this phenomenon, we assign an initial energy to each label. With the propagation of labels, the label energy gradually decreases accordingly. If the label energy of v is less than the threshold ε, the vertex of v will not be involved into the other vertex update. The label of v is updated with label l; the label energy of vertex v changes as follows:

$$E_l(v) = \underset{j \in N_l(v)}{\arg\max}\, E(j) - \delta, \tag{10}$$

where δ is an attenuation factor. The δ controls the label propagation range and limits the clusters size. In experiment phase, we set δ to a fixed value and a variable value separately, and the variable value changes with the range vertex of degree.

4.2. *The k-way partitioning*

Because of the number of coarsened vertices are uncertain, MMLP cannot achieve the specified k-way partitioning. As is known to all that the coarsened graph is a weight graph based on vertex and edge, in which the weight of vertex represents the number of vertices in the coarsened vertex, and the weight of edge reflects the number of edges spanning two coarsened vertices. Traditional K-medoids clustering algorithm cannot ensure the balanced distribution of vertex. Therefore, in this section, BKMC algorithm is executed on the top-level coarsened graph to attain a balanced k-way partitioning, which not only minimizes the edge cut but also simultaneously keeps the balance of vertex distribution.

(1) K-medoids clustering

K-medoids is a classical clustering technique that clusters n vertices into k groups. In each group, the data points are similar to each other while the data points from distinct groups are dissimilar. K-medoids clustering takes a similarity matrix between data points as an input. In our work, we use K-medoids clustering to cluster the coarsened graph, in which each coarsened vertex is considered as one data point correspondingly, and the weight of edge represents the similarity between data points. The similarity is in proportion to the weight of edge. In order to normalize the similarity, we use a linear function conversion, as shown in Eq. (11), where MIN denotes the minimum value of edge weight and MAX is the maximum value.

$$s(i,j) = \frac{w(i,j) - MIN}{MAX - MIN} \tag{11}$$

The proper initial centroids are essential for finding a good partitioning. In this paper, instead of selecting initial centroids randomly, we identify initial centroids from the density and distance point of view. If the neighbors of vertex i have lower local density and the vertex i is far away from any vertices with a higher local density, then vertex i has a high probability of being the cluster centroid [32]. First, the density ρ_i of vertex i is defined as follows:

$$\rho_i = \sum_{j \in V', j \neq i} \left(1 - e^{-\left(\frac{s(i,j)}{d_c}\right)^2}\right), \tag{12}$$

where d_c is the cutoff distance, and can be specified by user. In [32], the author suggests that d_c should satisfy the average number of neighbors account for around 1% to 2% of all data points in the dataset.

We use $\{q_i\}_{i=1}^n$ to represent a subscript descending order of $\{\rho_i\}_{i=1}^n$. The distance d_i is the maximum similarity from the vertex i to any other vertex with higher density:

$$d_{q_i} = \begin{cases} \max\limits_{q_j : j < i} (s(q_i, q_j)) \; i \geqslant 2 \\ \min\limits_{j \geqslant 2} (s(q_i, q_j)) \; i = 1 \end{cases} \tag{13}$$

Finally, we compute the density and distance product of vertex i using Eq. (14)

$$\gamma_i = \rho_i d_i, i = 1, 2, ..., n \tag{14}$$

According to the product, we sort γ in the descending order and the top k from the sorted list is the initial centroids $c = \{c_1, c_2, \ldots, c_k\}$. And then, each vertex v_i except cluster centroids will be assigned to a cluster according to the balanced adjustment strategy, which will be discussed in the following paragraphs.

After assigning all vertices to k clusters $C = \{C_1, C_2, \ldots, C_k\}$, we update the cluster centroid with the most centrally located vertex in each cluster. Firstly, the 'average point' of each cluster will be computed by Eq. (15).

$$S(\overline{v_i}) = \frac{1}{|C_i|} \sum_{v_k \in C_i} S(v_k, v_j), \forall v_j \in V \tag{15}$$

The vector $S(\overline{v_i})$ is the average similarity for cluster C_i. Then the new centroid c_i in each cluster C_i is calculated by

$$c_i = \arg\min_{v_j \in C_i} \left\| S(v_j) - S(\overline{v_i}) \right\| \tag{16}$$

The other vertices will be assigned again according to the new centroids, and the process is executed iteratively until the clustering objective function converges. Our clustering objective is to minimize the inter-cluster similarity. Thus, the objective function is defined as follows:

$$f(C) = \sum_{i=1}^{k} \sum_{j>i}^{k} d(C_i, C_j), \tag{17}$$

where $d(C_i, C_j)$ represents the similarity between any two clusters; it can be described as follows:

$$d(C_i, C_j) = \sum_{v_i \in C_i} \sum_{v_j \in C_j} s(v_i, v_j) \tag{18}$$

(2) Balanced adjustment strategy

K-medoids clustering only assigns vertices to the closest centroid according to the similarity between the distributed vertex and the centroid without considering the balanced distribution. However, in order to realize the parallel query of storage nodes, we should not only minimize the edge cut but also keep the load balance of storage node. Therefore, we introduce a balanced adjustment strategy based on the edge weight and vertex weight to K-medoids clustering for the k-way partitioning.

In order to keep balance, each cluster has a capacity limitation, which satisfies the definition of balanced partitioning. After specifying the centroids, each super vertex is assigned to their closest cluster centroids. If the new super vertex makes the cluster exceed its capacity, BKMC will adjust the internal vertices of cluster according to the similarity between these vertices and cluster centroid. The steps of adjustment strategy are as follows (all adjustment is executed on the super vertices):

Step 1: Sorting all the internal vertices by ascending according to the similarity between vertices and the cluster centroid;
Step 2: Finding the smallest vertex set s,whose similarity summation to cluster centroid and the weight summation are smaller than the new vertex;
Step 3: If s exist, then replace these vertices with the new vertex;
Step 4: Otherwise assigning the new vertex to sub-optimum cluster.

Details of the BKMC Algorithm is described as follows:

Algorithm 2 BKMC

```
Input: the similarity matrix $S$, the number of clusters $k$ and a float factor of balance $\theta$
output: the partitioning $C = \{C_1, C_2, ..., C_k\}$
Method:
1. Initial the cluster centers of coarsened graph $c = \{c_1, c_2, ..., c_k\}$
2. Repeat until the clustering objective function converge
    For each coarsened vertex $v_i^{'}$
        Select a cluster $C_t$ which has the maximum similarity between $v_i^{'}$ and the centroid $c_t$
        If $w(v_i^{'}) + capacity(C_t) < (1 + \theta)n/k$ then
            Assign $v_i^{'}$ to $C_t$
        Else
            Execute the adjustment strategy
        End if
    End for
3. Update the cluster centroids
4. If the clustering objective function does not converge, repeat the step 2 until convergence.
```

4.3. *Graph optimization*

Through experimental analysis, we conclude that the vertex with only one incoming edge should be merged with its neighbor. For the directed RDF graph, the directed edge connects the subject and its attributed value, that is, object, in which some attribute values only belong to one subject, so these vertices can be collapsed before BRGP algorithm. Given a RDF graph $G = (V, E)$, if $\exists v \in V$, where $deg(v) = 1$ and $N(v) = u$, then v must be assigned to the same storage node with u. For this kind of vertex, which storage node it belongs to is decided by its only neighbor vertex. Therefore, we design an equivalent pruning strategy to decrease the scale of RDF graph and get a better efficiency. The steps of equivalent pruning strategy are as follows:

Step 1: Combine all vertices v with u, which satisfy $deg(v) = 1$ and $N(v) = u$;
Step 2: Repeat step 1, until there is no qualified vertex.

4.4. *Complexity Analysis*

The time complexity of BRGP algorithm is $O(lT_1|E|)+O(kT_2c^2)$, where T_1 and T_2 are the number of iterations in LP and K-medoids separately, l is the coarsening levels, $|E|$ represents the number of edges in original RDF graph G, k is the number of partitions, and c is the number of vertices in coarsened graph $G^{'}$. Because of the liner time complexity of LP algorithm and $l \ll |E|$, therefore, MMLP is still linear. In the final partitioning phase, due to $c \ll |V|$, so BKMC algorithm does not affect the overall time complexity of BRGP. The space complexity is $O(d|V|)$, where d denotes the average degree of vertices.

5. EXPERIMENTS

In this section, a set of experiments are conducted on three representative RDF benchmark datasets and one real RDF dataset to evaluate the partitioning performance and quality of BRGP. The running time and memory consumption are used to measure the performance of different partitioning methods. The partitioning quality is evaluated by the edge cut and balance. Optimization experiments further prove the effectiveness of equivalent pruning strategy in running time, and then query experiments are executed to validate the superiority of BRGP in the complex SPARQL queries. We compare our algorithm against two typical graph partitioning methods, that is, METIS and MLP+METIS. We first introduce the datasets used in our experiments. Then the experimental settings are described. Lastly, the experimental results and comparisons are presented in detail.

5.1. *Datasets*

Two representative RDF benchmarks, **LUBM** (the Lehigh University Benchmark) [33] and **SP^{2}Bench** [34] are used in our experiments. The LUBM features a university domain, and the SP^{2}Bench dataset features a DBLP domain. A real dataset DBLP is also utilized in our experiments, which contains bibliographic descriptions in computer science. We generate three datasets from LUBM and SP^{2}Bench with different sizes: (1) LUBM50 and LUBM2000 covering 50 and 2000 universities, respectively, and (2) SP^{2}B-100M having about one million triples. Table II shows the details of the four generated RDF datasets.

5.2. *Experimental settings*

The graph partitioning experiments are performed on a PC with Intel Xeon at 2.00 GHz$\times$24, 20GB memory running 64-bit Linux. In graph partitioning experiments, each dataset is divided into 4, 8 and 16 subsets, respectively. For the competitor METIS, we set the parameters '-ptype' and '-ctype' as k-way partitioning (kway) and sorted heavy-edge matching (shem). MLP+METIS uses LP to coarsen graph based on common neighbors and METIS to achieve the final partitioning. In the experiment of partitioning balance, we conduct two set of experiments with the proposed BRGP method (BRGP1 and BRGP2) to validate the effectiveness of the label energy function as stated in Eq. (10). The δ in BRGP1 is set to 0.2, and the label initial energy is set to 1. In BRGP2, we set δ to 0.1, 0.2, and 0.5, respectively, according the range of vertex degree. We also validate the effectiveness of the graph optimization for our method, METIS and MLP+METIS.

To evaluate the partitioning method, a set of queries is executed on the partitioning results, which are distributed to a cluster with 16 computing nodes. We follow the implementation and setup of the distributed RDF processing system proposed in [4]. In the distributed system, the graph partitioning machine mentioned earlier is considered as the master machine, and each computing node in the cluster has a 2.33 GHz Intel Xeon processor, 4GB memory, and a RDF-3X 0.3.5 store system. A Hadoop system is also used to join the intermediate results, which used version 0.20.203 running on Java 1.6.0. The configuration settings of Hadoop system adopts default values. For all experiments, we first import the data into computing node by three different partitioning methods and replicate the triples on boundary by the undirected one-hop and two-hop guarantees. In our experiments, we choose the benchmark queries, which are provided by LUBM datasets. All experiments are repeated three times, and average results are reported.

Table II. Statistics of datasets used in experiments.

Dataset	#Vertex	#Edge
LUBM50	1, 706, 230	6, 888, 642
LUBM2000	66, 059, 204	276, 345, 040
SP^{2}B-100M	548, 826	922, 183
DBLP	166, 801	5, 723, 789

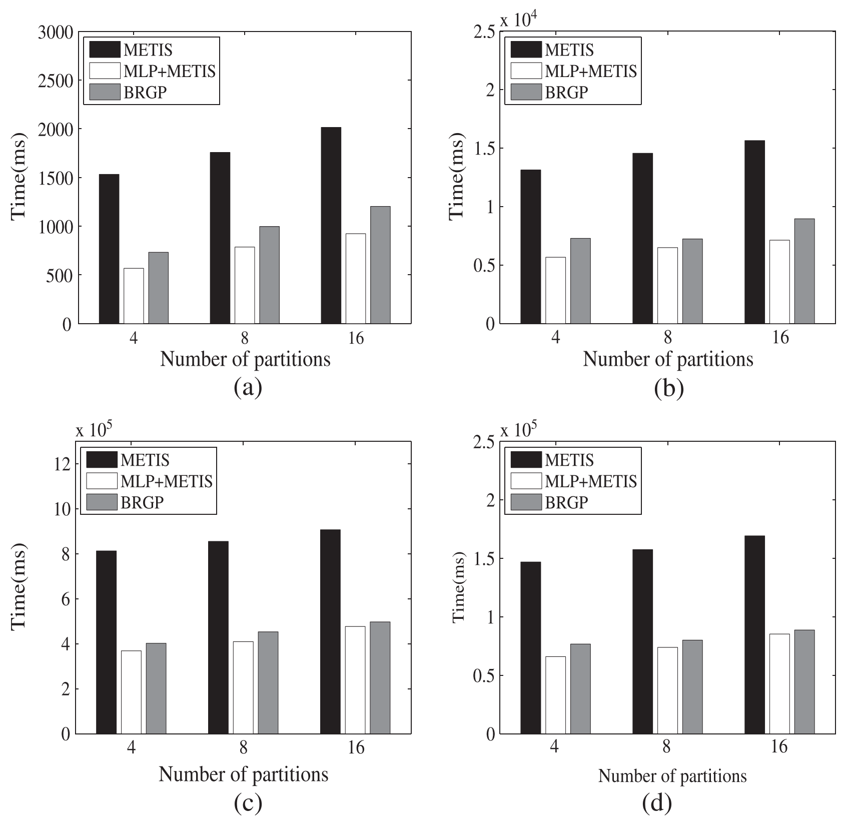


Figure 3. Running time on different datasets. (a) SP^{2}B-100M; (b) LUBM50; (c) LUBM2000; (d) DBLP.

5.3. *Partitioning performance*

Figure 3 demonstrates the comparison of runtime in three different partitioning algorithms. We can draw the conclusion that the running time will increase with the increasing number of partitions. Moreover, the running time of METIS is obviously higher than the other two algorithms. We believe that this is firstly because METIS needs to sort the data by the weight of edges, which is an extremely time-consuming procedure. Secondly, the asymmetric RDF graph prevents large matchings in the coarsening process, leading to a slow shrink in the number of edges and the size of coarsened graph. In contrast, LP chooses a vertex randomly instead of a sorting or indexing mechanism. Thus, the MLP+METIS and the proposed BRGP both outperform METIS substantially on the four datasets. Although our approach is not superior to the MLP+METIS, it is competitive.

Figure 4 presents the results of memory consumption, where the trends are similar to that of running time with the increasing number of partitions. As in each level, METIS will store the coarsening graph and the mappings relationship of the adjacent level; hence, METIS has the highest memory consumption. In contrast, the LP does not storage the intermediate results generated in the coarsening step. Therefore, our approach achieves a superior result to METIS and MLP+METIS.

5.4. *Partitioning quality*

The results of the edge cut ratio on four datasets are reported in Figure 5. Here, the edge cut ratio is computed by $EC(P)$/m. Because the density of edge in SP^{2}B-100M is sparser than that of other three datasets, the results for SP^{2}B-100M are lowest as shown in Figure 5(a). Because of the unequal relationship among RDF data points, the update rule of MLP+METIS is unsuitable for RDF graph partitioning. Hence, the edge cut ratio of MLP+METIS is the highest as shown in Figure 5. As the results show, our approach outperforms METIS and MLP + METIS on all four datasets.

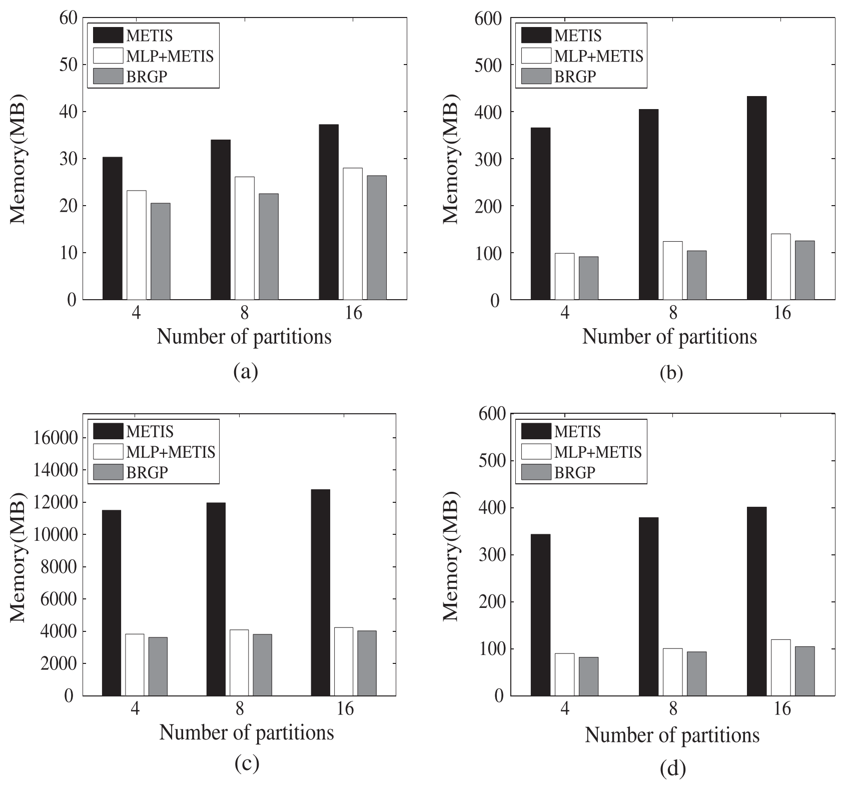


Figure 4. Memory consumption on different datasets. (a) SP^{2}B-100M; (b) LUBM50; (c) LUBM2000; (d) DBLP.

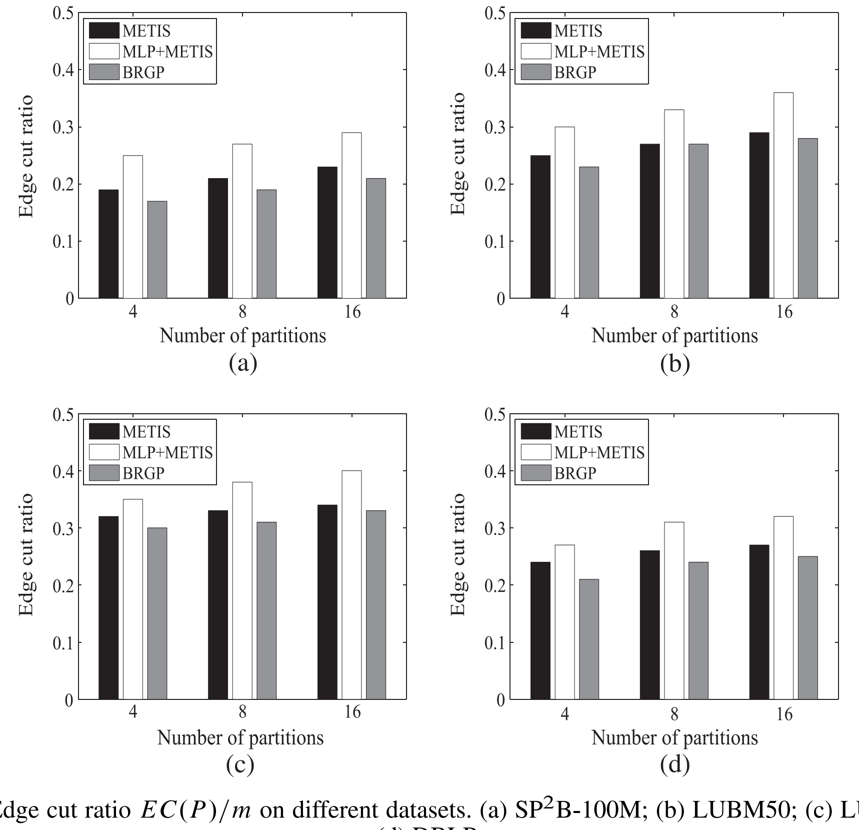


Figure 5. Edge cut ratio $EC(P)/m$ on different datasets. (a) SP^{2}B-100M; (b) LUBM50; (c) LUBM2000; (d) DBLP.

Table III. Partitioning balance.

	METIS	MLP+METIS	BRGP1	BRGP2
LUBM50	1.02	1.03	1.05	1.03
LUBM2000	1.03	1.03	1.04	1.02
DBLP	1.03	1.03	1.06	1.03
SP^{2}B-100M	1.02	1.03	1.04	1.02

Table IV. The optimized result of datasets.

Dataset	#Original vertex	#Optimized vertex	#Pruning vertex	#The rate of decline (%)
LUBM50	1,706,230	1,257,916	448,314	26.8
LUBM2000	66,059,204	50,316,058	15,743,146	21.32
DBLP	548,826	362,225	186,601	34
SP^{2}B-100M	166,801	132,465	34,336	20.59

Table V. The improved ratio by graph optimization.

	Edge cut (%)			Runtime (%)			Memory (%)		
	METIS	MLP+METIS	BRGP	METIS	MLP+METIS	BRGP	METIS	MLP+METIS	BRGP
LUBM50	0.28	0.27	0.24	21.33	20.32	22.69	23.56	23.42	22.40
LUBM2000	0.29	0.21	0.27	23.92	24.86	21.32	21.94	20.77	23.57
DBLP	0.23	0.19	0.21	27.05	30.35	32.75	24.04	25.08	23.62
SP^{2}B-100M	0.16	0.12	0.18	17.96	18.51	18.55	19.80	21.75	21.49

The results of partitioning balance are reported in Table III. Although the BRGP1 that with fixed attenuation factor does not outperform METIS and MLP+METIS, the BRGP2 that with dynamic attenuation factor performs competitively. The comparisons demonstrate the effectiveness of BRGP in the balanced partitioning of RDF graph.

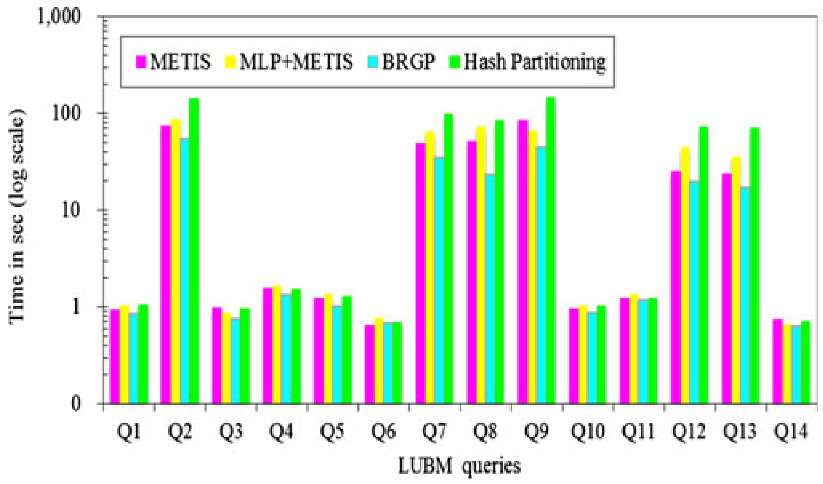
5.5. Optimization

Table IV shows the optimized result of datasets, and Table V shows the partitioning performance and quality for all three methods on four optimized datasets, in which the value indicates the improved rate. Here, the results are obtained from the experiments on each dataset with eight subsets.

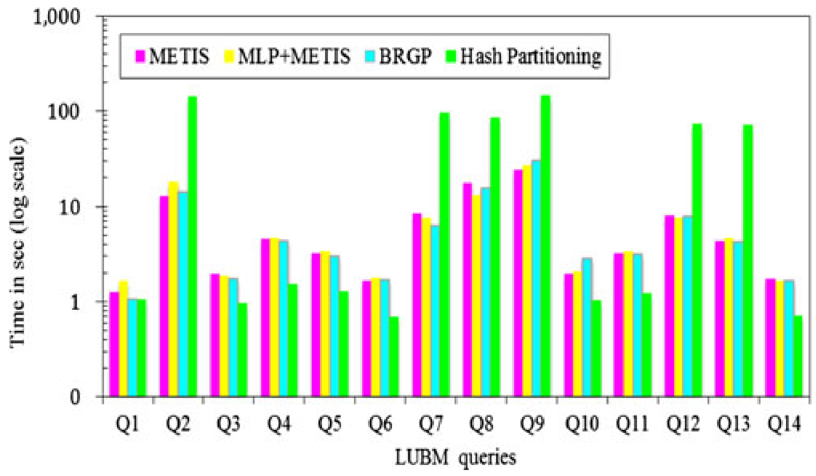
Through the pruning strategy, the number of vertices for partitioning algorithm decrease by 20–30%. So the running time and memory consumption have an obvious improvement. However, for these merged vertices, they would be distributed to the same partition whether pruning or not, so the impact of partitioning quality is less than 0.5%.

5.6. Query processing

Figure 6 shows the comparison of query time on three graph partitioning methods and a hash partitioning method, in which Figure 6(a) is a one-hop guarantee in graph partitioning methods and Figure 6(b) is a two-hop guarantee. Q1, Q3, Q4, Q5, Q6, Q10, Q11, and Q14 are simple benchmark queries of LUBM, which only consist of subject–subject joins and the maximum path between two nodes is two. Therefore, the computing nodes can execute queries in parallel without any network communication. The result shows that there is no obvious difference in the query time on four data partitioning methods for these simple queries. However, Q2, Q7, Q8, Q9, Q12, and Q13 are complex queries, which have subject–object joins, and the maximum path is over 2. Compared with the hash partitioning, the graph partitioning methods execute some subject–object joins inside storage node, which would greatly reduce the communications and improve the query efficiency.



(a)



(b)

Figure 6. Query processing time. (a) One-hop guarantee; (b) Two-hop guarantee.

For the three graph partitioning methods, the number of joins among computing nodes decrease as the number of edge cuts decrease. So the query efficiency in BRGP is better than other two graph partitioning methods in one-hop guarantee. Nevertheless, the partitioning of two-hop guarantee makes most joins be executed inside computing nodes, so the query time on three graph partitioning methods is very little. But the two-hop guarantee stores an extra 20% duplicate triples. Besides, for the complex query, we can conclude that the graph partitioning is superior to hashing partitioning.

6. CONCLUSION

We presented a balanced RDF graph partitioning algorithm (BRGP) for storing massive RDF data on cloud, which works in two major steps. Firstly, BRGP uses the gain in modularity as the label update rule of LP to roughly partition the RDF graph iteratively until the graph is small enough. Secondly, a balanced adjustment strategy is devised in K-medoids clustering to implement the final partition on the coarsened graph. Experimental results demonstrate that the label update rule based on the modularity gain and the label energy function in our approach improved the graph partitioning performance by generating a balanced and low edge cut partitioning scheme in RDF graph. Furthermore, our scheme is effective to reduce the communication overhead when performing query operations among different storage nodes, which is particularly important for cloud storage. For future work, we would like to investigate the incremental graph partitioning for dynamic RDF data storage on cloud.

ACKNOWLEDGEMENTS

We would like to thank all anonymous reviewers for their valuable comments and suggestions. This work is partly supported by the State Key Program of National Natural Science of China (U1301253) and the Science and Technology Planning Key Project of Guangdong Province (2015B010110006).

REFERENCES

1. *Rdf*. (Avaialable from: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/) [Accessed on 23 June 2015].
2. *Rdf model and syntax specification.1999*. (Available from: http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/) [Accessed on 12 April 2015].
3. Gu R, Hu W, Huang Y. Rainbow: A distributed and hierarchical rdf triple store with dynamic scalability. In *2014 International Conference on Big Data*, IEEE, Washington, DC, USA, 2014; 561–566.
4. Huang J, Abadi DJ, Ren K. Scalable sparql querying of large rdf graphs. *Proceedings of the VLDB Endowment*, Vol. 4, 2011; 1123–1134.
5. Kaoudi Z, Manolescu I. Cloud-based rdf data management. In *Proceedings of the 2014 ACM International Conference on Management of Data*, ACM, Snowbird, UT, USA, 2014; 725–729.
6. Kaoudi Z, Manolescu I. Rdf in the clouds: a survey. *The VLDB Journal* 2015; **24**(1):67–91.
7. Punnoose R, Crainiceanu A, Rapp D. Sparql in the cloud using rya. *Information Systems* 2015; **48**:181–195.
8. Lee K, Liu L. Scaling queries over big rdf graphs with semantic hash partitioning. *Proceedings of the VLDB Endowment*, Vol. 6, 2013; 1894–1905.
9. Salvadores M, Correndo G, Harris S, Gibbins N, Shadbolt N. The design and implementation of minimal rdfs backward reasoning in 4store. In *The Semanic Web: Research and Applications*. Springer: Heraklion, Crete, Greece, 2011; 139–153.
10. Kaoudi Z, Kyzirakos K, Koubarakis M. Sparql query optimization on top of dhts. In *The Semantic Web-ISWC 2010*. Springer: Shanghai, China, 2010; 418–435.
11. Wang X, Yang T, Chen J, He L, Du X. Rdf partitioning for scalable sparql query processing. *Frontiers of Computer Science* 2015; **9**(6):919–933.
12. Wang L, Xiao Y, Shao B, Wang H. How to partition a billion-node graph. *30th IEEE International Conference on Data Engineering (ICDE)*, IEEE, Chicago, IL, USA, 2014; 568–579.
13. Wu B, Zhou Y, Yuan P, Liu L, Jin H. Scalable sparql querying using path partitioning. *31th IEEE International Conference on Data Engineering (ICDE)*, IEEE, Seoul, South Korea, 2015; 795–806.
14. *Sparql*. (Available from: http://www.w3.org/TR/2012/PR-sparql11-query-20121108/) [Accessed on 4 October 2014].
15. *Metis*. (Available from: http://glaros.dtc.umn.edu/gkhome/views/metis) [Accessed on 26 May 2016].
16. LaSalle D, Karypis G. Multi-threaded modularity based graph clustering using the multilevel paradigm. *Journal of Parallel and Distributed Computing* 2015; **76**:66–80.
17. Raghavan UN, Kumara S. *Near linear time algorithm to detect community structure in large-scale networks. physics. soc-ph. 19*, 2007.
18. Husain MF, McGlothlin J, Masud MM, Khan LR, Thuraisingham B. Heuristics-based query processing for large rdf graphs using cloud computing. *IEEE Transactions on Knowledge and Data Engineering* 2011; **23**(9):1312–1327.
19. Ravindra P, Hong S, Kim H, Anyanwu K. Efficient processing of rdf graph pattern matching on mapreduce platforms. In *Proceedings of the Second International Workshop on Data Intensive Computing in the Clouds*, ACM, Seattle, WA, USA, 2011; 13–20.
20. Rohloff K, Schantz RE. Clause-iteration with mapreduce to scalably query datagraphs in the shard graph-store. In *Proceedings of the Fourth International Workshop on Data-intensive Distributed Computing*, ACM, San Jose, CA, USA, 2011; 35–44.
21. Tomaszuk D, Skonieczny L, Wood D. Rdf graph partitions: A brief survey. In *Beyond Databases, Architectures and Structures*. Springer: Ustron, Poland, 2015; 256–264.
22. Hose K, Schenkel R. Warp: Workload-aware replication and partitioning for rdf. In *29th IEEE International Conference on Data Engineering Workshops (ICDEW)*, IEEE, Brisbane, Australia, 2013; 1–6.
23. Kernighan BW, Lin S. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal* 1970; **49**(2):291–307.
24. Fiduccia CM, Mattheyses RM. A linear-time heuristic for improving network partitions. In *19th Conference on Design Automation*, IEEE, Las, Nevada, USA, 1982; 175–181.
25. Brooks SP, Morgan BJT. Optimization using simulated annealing. *The Statistician* 1995; **44**(2):241–257.
26. Bui TN, Moon BR. Genetic algorithm and graph partitioning. *IEEE Transactions on Computers* 1996; **45**(7): 841–855.
27. Hendrickson B, Leland R. *A multi-level algorithm for partitioning graphs*, 1995.

28. Barnard ST. Pmrsb: Parallel multilevel recursive spectral bisection. In *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, ACM, Barcelona, Spain, 1995; 27–48.
29. Karypis G, Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* 1998; **20**(1):359–392.
30. Lee K, Liu L, Tang Y, Zhang Q, Zhou Y. Efficient and customizable data partitioning framework for distributed big rdf data processing in the cloud. In *Sixth IEEE International Conference on Cloud Computing*, IEEE, Santa Clara, CA, USA, 2013; 327–334.
31. Newman MEJ, Girvan M. Finding and evaluating community structure in networks. *Physical review E* 2004; **69**(2):026113.
32. Rodriguez A, Laio A. Clustering by fast search and find of density peaks. *Science* 2014; **344**(6191):1492–1496.
33. Guo Y, Pan Z, Heflin J. Lubm: A benchmark for owl knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web* 2005; **3**(2):158–182.
34. Schmidt M, Hornung T, Lausen G, Pinkel C. Sp^2 bench: a sparql performance benchmark. In *25th IEEE International Conference on Data Engineering*, IEEE Computer Society, Shanghai, China, 2009; 222–233.