IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON ADVANCED DATA MINING METHODS FOR SOCIAL COMPUTING

Received June 6, 2019, accepted June 27, 2019, date of publication July 4, 2019, date of current version August 5, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2926753

Parallel Heuristics for Balanced Graph Partitioning Based on Richness of Implicit Knowledge

ZHIPENG YANG[1], RONGRONG ZHENG[2], AND YINGLONG MA[1], (Member, IEEE)

[1]School of Control and Computer Engineering, North China Electric Power University, Beijing 102206, China
[2]State Grid Information and Telecommunication Group Company Ltd., Beijing 100761, China

Corresponding author: Yinglong Ma (yinglongma@gmail.com)

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFC0831404, and in part by the State Grid Corporation of China Science and Technology Project "Research on Key Technologies of Knowledge Discovery Based ICT System Fault Analysis and Assisted Decision".

ABSTRACT Balanced graph partitioning (BGP) has a wide range of applications that involve many large-scale distributed data processing problems. However, most of the existing approaches to parallel graph partitioning neglect the problem of the richness of implicit knowledge (RIK) residing in big graph and the knowledge cohesion after graph partitioning. In this paper, we propose a parallel balanced graph partitioning framework called JA-BE-JA-RIK, which improves the original heuristic JA-BE-JA algorithm by evaluating RIK such that parallel BGP can be more efficiently made under the condition of obtaining the good edge-cut value and high knowledge cohesion. We introduce the concept about RIK and further present a quantitative measure to dynamically evaluate the RIK values during parallel BGP. Based on the Hadoop platform, we also implemented both frameworks and JA-BE-JA. The extensive experiments were made for illustrating efficacy of our JA-BE-JA-RIK approach in comparison with the traditional JA-BE-JA algorithm. The experimental results show that our parallel BGP approach can obtain the RIK value as higher as possible within a closer number of edge-cuts in comparison with the traditional JA-BE-JA algorithm.

INDEX TERMS Graph partitioning, balanced graph partitioning, parallel graph computing, richness of implicit knowledge, knowledge cohesion.

I. INTRODUCTION

In the last decades, large amounts of graph data for modeling real-world data have been produced by various applications like social networks [1], biological networks [2] and bibliographic citation networks [3], etc. Due to the volume and diversity of drastically growing graph data, the computation complexity and time cost of processing massive amounts of graph data are also tremendously increasing. The conventional centralized graph computing methods [4] for graph query [5] and graph partitioning [6], [7], cannot process such massive graph data quickly and efficiently. Therefore, the processing for large-scale graphs [8] has become a remarkable challenge in graph computing. It is necessary to use the parallelism of graph computing methods on distributed environments [9] for processing large scale graph data.

The associate editor coordinating the review of this manuscript and approving it for publication was Shirui Pan.

Balanced graph partitioning (BGP) is a fundamental problem of graph computing for processing big graph data. Finding good partitions is a well-known and well-studied problem in graph theory [10]. Although many algorithms [11], [12] known for graph partitioning have been proposed in the last decades, most of them have a potential assumption that random access to the entire graph is cheap. Moreover, most of existing algorithms require loading the entire large-scale graphs into the main memories, such as METIS [13], [14], and KaFFPa [15], etc. It is crucial for partitioning and storing large-scale graph data on a distributed environment. Parallel balanced graph partitioning [16] therefore has been considered as an efficient way to partition big graph data on distributed environments [17]–[19], which requires partitioning a graph into a predefined number of balanced partitions such that each of them has the almost same number of nodes or edges. Recently, many popular parallel BGP approaches have been proposed such as ParMETIS [20], JA-BE-JA [18], etc. ParMETIS is a parallel graph partitioning

library that extends the functionality provided by METIS and is suited for parallel computations and large-scale numerical simulations. JA-BE-JA is recent a parallel BGP algorithm that can obtain good partitioning results for large-scale graphs by using local search and heuristic methods. Moreover, some researches proposed approaches for partitioning large dynamic graphs [21] and streaming graphs [22] to solve the constraints of memory, communication cost and response time. In conclusion, existing approaches for parallel BGP have illustrated their success in partitioning large-scale graph data by reducing communication overhead within a balanced computation load.

However, existing approaches for parallel BGP neglect to satisfy the requirements of users' graph queries. On one hand, the goal of efficient BGP is to obtain partitions with high cohesion, which can achieve efficient graph queries of the future. A good BGP approach should consider the knowledge cohesion residing in each of partitioned subgraphs such that users can perform graph queries across fewer partitions. During parallel BGP, once the subgraphs to be partitioned are detected to reach the maximum knowledge cohesion, parallel BGP should be terminated even if parallel BGP has not obtained the minimum edge-cuts. In this situation, the computational complexity for parallel BGP can be significantly reduced under the condition of ensuring the effective and efficient graph queries. One the other hand, almost all of parallel BGP approaches do not fully take into account the richness of implicit knowledge in each of partitioned subgraphs. The query knowledge related to specific users is often closely related to each other, so it is desirable if the knowledge to be queried can be assembled in single partitions as much as possible. However, each of subgraphs in existing approaches is partitioned just by their explicit knowledge (e.g., topological structure). Much of implicit knowledge that is derived from the explicit knowledge by using some approaches such as path-based reasoning is neglected for efficient parallel BGP, which impedes efficient graph query based on big graph data. We argue that the richness of the implicit knowledge (RIK) can greatly reflect the knowledge cohesion of a partitioned subgraph besides the explicit knowledge.

In this paper, we propose a parallel balanced graph partitioning framework called JA-BE-JA-RIK, which improves the original JA-BE-JA algorithm by evaluating richness of implicit knowledge such that parallel BGP can be efficiently made under the condition of satisfying users' graph query requirements. We define the concept about richness of implicit knowledge and present an APL-based (Average Path Length) method to quantitatively evaluate the richness of implicit knowledge after graph partitioning. Based on the popular Hadoop platform [23], we also implemented both frameworks JA-BE-JA-RIK and JA-BE-JA. Extensive experiments were made for comparison with the traditional JA-BE-JA algorithm.

This paper is organized as follows. Section 2 is the related work. Section 3 introduces the energy calculation of node color exchange. Section 4 is the overview of our framework JA-BE-JA-RIK. In Section 5, we discuss the parallel algorithm of our framework JA-BE-JA-RIK based on Hadoop. We also present the quantitative measure method of richness of implicit knowledge (RIK) by using approximate average path length (APL) in detail. Next, Section 6 gives the analysis and evaluation of graph partitioning results compared with JA-BE-JA. Finally, Section 7 summarizes the conclusion and future work.

II. RELATED WORK

Balanced graph partitioning (BGP) is a well-known NP-complete problem [24], which has been applied to a wide range of applications such as social networks, biological networks [25], etc. It requires to partition a graph into a predefined number of balanced partitions, each of which has the almost same number of nodes or edges. Graph partitioning can be divided into two categories according to different partitioning objects, i.e., edge-cut partitioning and vertex-cut partitioning [26], [27]. For examples, KaFFPa [15] is a graph partitioning algorithm based on edge-cut partitioning, which focuses on local improve methods and overall search strategies. Pregel [28] is based on vertex-cut partitioning and bulk synchronous parallel computation model [29], but its constraints limit the user program's access to solely the node on which the program is executing by design. The key problem of BGP is to ensure the number of edge-cuts or vertex-cuts as less as possible. The optimal solution for balanced graph partitioning has become an active research area.

BGP approaches are often classified into two main categories according to their search strategies, i.e., local search based partitioning and global graph partitioning. Global BGP has been deeply explored in the last decades, which relies on the global topological structure of the entire graph to be partitioned, such as the Spectral Partitioning algorithm [30]–[32]. Global graph partitioning is relevantly simple and easy to implement, but it has higher computation complexity and needs to consume more memory cost due to that it needs to traverse all nodes and edges in the entire graph. Local search based BGP, such as the KL algorithm [33] and FM algorithm [34], only needs to know the adjacent nodes or a smaller subset of nodes in the graph. In contrast to global BGP, local search based BGP has the lower computation complexity and requires less memory cost. In addition, some hybrid approaches for BGP have recently received more attention. For examples, the multilevel graph partitioning (MGP) [10], [35], has been considered as one of the most popular graph partitioning algorithms, which uses both the strategies of local search based BGP and global BGP. METIS [13] is a famous algorithm based on MGP to improve the quality of graph partitioning by a three-phase heuristic algorithm. The hybrid BGP approaches can partially avoid some limitations of global BGP and local based BGP. Most of the existing approaches are based on centralized environments, and are also easy to understand and implement. However, when the nodes and edges of the

big graph reach the magnitude of million or billion, a single computer is difficult to partition a big graph data. Moreover, a large-scale graph has too large volume of data to be loaded in the memory of a single computer. Partitioning large-scale graph data therefore becomes a very challenging task due to the high computation complexity and tolerant performing time. It is desirable for partitioning large-scale graph data to store and partition big graph data on a distributed environment.

In parallel BGP, a big graph will be divided into many smaller subgraphs such that each of subgraphs can fit into the memory of a single computer. For examples, ParMETIS [20] is a parallel version of METIS [13] on distributed systems. It is faster than METIS and can obtain good partitioning results with minimum edge-cuts. PowerGraph [36] is another distributed graph processing framework that uses vertex-cuts to evenly distribute the edges of graph to multiple machines so that there are minimum vertex-cuts across multiple partitions. JA-BE-JA is a recent most state-of-the-art parallel BGP approach that employs local search and heuristic method. During the partitioning, only the direct neighbors of each node and a small subset of random nodes in the graph need to be known locally. JA-BE-JA has been proved to outperform METIS in finding a better partitioning in some cases [18], especially for graph partitioning of social networks. However, the number of iterations performing the JA-BE-JA algorithm is fixed in order to make the edge-cuts as less as possible, but the edge-cuts are basically unchanged in the large part of latter iterations, which will inevitably bring about extra computation complexity. The work in [37] designs an implementation of JA-BE-JA based on Spark [38], which also explores possibly appropriate number of iterations for reducing computation time. These distributed graph partitioning algorithms have illustrated their success in partitioning large-scale graph data by reducing communication overhead within a balanced computation load.

In this paper, we will focus on the parallel BGP based on the richness of implicit knowledge. We proposed a parallel BGP framework called JA-BE-JA-RIK, which extends the traditional JA-BE-JA algorithm by fully considering RIK such that parallel BGP can be efficiently made under the condition of obtaining the RIK value as larger as possible with a mild increase of edge-cuts.

To the best of our knowledge, this is the first work for efficient parallel BGP to take into accounts the implicit knowledge of large-scale graph data and the knowledge cohesion of graph partitioning. The contributions of this paper are as follows.

First, we proposed a parallel BGP framework called JA-BE-JA-RIK, which improves the traditional heuristic JA-BE-JA algorithm by fully considering richness of implicit knowledge such that parallel BGP can be more efficiently made under the condition of satisfying users' graph query requirements.

Second, we introduce the concept about richness of implicit knowledge (RIK) and further present a quantitative measure to dynamically evaluate the RIK values during parallel BGP.

Third, we implemented our proposed framework JA-BE-JA-RIK and the original JA-BE-JA based on Hadoop.

At last, extensive experiments were made for illustrating efficacy of our approach. The experimental results show that our parallel BGP approach can obtain the richness of implicit knowledge as higher as possible within a closer number of edge-cuts in comparison with the traditional JA-BE-JA algorithm.

III. PRELIMINARIES

Let $G = (V, E)$ represent a graph data, where V is the set of nodes and E is the set of edges. A k-way edge-cut BGP divides V into k subsets with similar equal sizes by a partition function $\pi: V \rightarrow \{1, \ldots, k\}$ that assigns a color to each node. Here, $\pi(p)$, or π_p for short, refers to the color of node p. N_p is the set of neighbors of node p, and $N_p(c)$ represents the set of neighbors of p that have color c:

$$N_p(c) = \{q \in N_p : \pi_q = c\} \tag{1}$$

The number of neighbors of node p is denoted by $d_p = |N_p|$, and $d_p(c) = |N_p(c)|$ is the number of neighbors of p with color c. The energy of the system is defined as the number of edges between nodes with different colors (equivalent to edge-cut). Accordingly, the energy of a node is the number of its neighbors with a different color, and the energy of the graph is the sum of the energy of the nodes as follows.

$$E(G, \pi) = 1/2 \sum\nolimits_{p \in V} (d_p - d_p(\pi_p)) \tag{2}$$

In order to minimize the edge-cuts of the partitioning, we attempt to maximize $d_p(\pi_p)$ for any node p in the graph. Thus, color exchange at each time should decrease the energy of the graph. In other words, for any two nodes, the number of their respective neighbors with a similar color increases after they exchange their colors. Therefore, the energies of node pairs before and after color exchange can be calculated by the following Formula 3.

$$[d_p(\pi_q)^\alpha + d_q(\pi_p)^\alpha] \times T_i > d_p(\pi_p)^\alpha + d_q(\pi_q)^\alpha \tag{3}$$

The formula adds simulated annealing (SA) to avoid local optimization, where the value of T_i decreases from 2 to 1 gradually. Moreover, α is a parameter of node energy function that is not less than 1, and the experiments in paper [18] prove that its optimal value is 2. For instance, assume that there are two kinds of colors including gray and white. If $\alpha = 1$, color exchange for nodes p and q in Figure 1(a) is accepted, as the energy sum of nodes p and q changes from $1^1 + 0^1$ to $1^1 + 3^1$ after color exchange. However, nodes u and v in Figure 1(b) will not exchange their colors, because the energy sums $2^1 + 2^1 \not< 1^1 + 3^1$. If $\alpha > 1$, then nodes u and v will exchange their colors. Although this color exchange does not directly reduce the total edge-cuts of the graph, it increases the possibility of color exchange for node v's two gray neighbors.

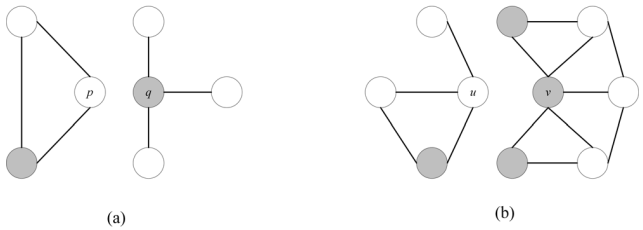


FIGURE 1. Example of the optimal value of α.

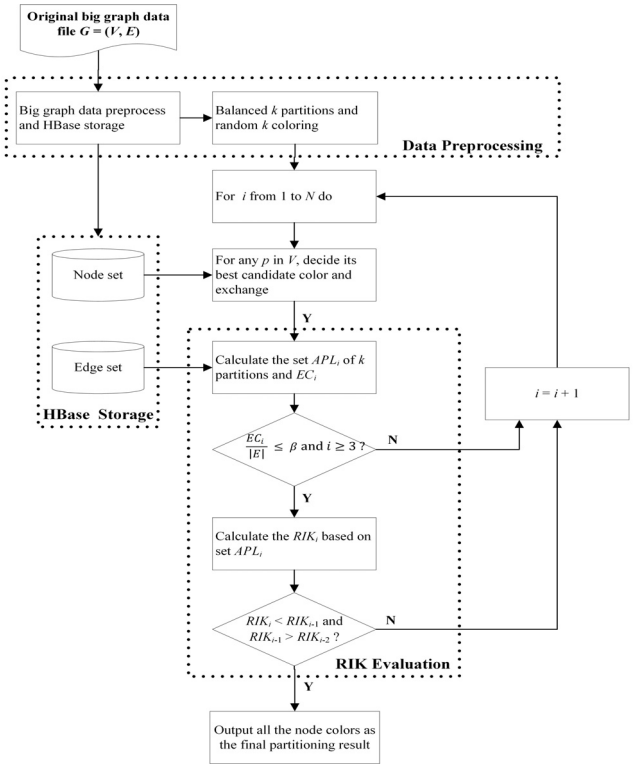


FIGURE 2. The overview of our framework.

IV. OVERVIEW OF FRAMEWORK

In this paper, we present a framework called JA-BE-JA-RIK that is mainly composed of three stages, including data preprocessing, node color exchange and RIK evaluation. The overview of the framework is shown in Figure 2.

In the stage of data preprocessing, the source graph is first preprocessed into the graph data with specified formats, including the node set file and the edge set file. The node set file contains a column of all nodes in the original graph. In the edge set file, each node appending all its direct neighbors is stored as a row. Then, the graph files are stored on HDFS. Moreover, due to the limited memories, the two graph files also need to be stored in HBase for iterative computation of the future. In HBase storage, the entire graph is stored by plenty of key-value pairs where each node is the key and a string consisting of all its direct neighbors is the value of a record. Besides, a color table representing initial colors of all nodes is read into a HashMap structure before the first iteration of performing JA-BE-JA-RIK. In this stage, a predefined number k of partitions should be also specified for the later graph partitioning. Similar to JA-BE-JA-RIK, an initial number N of iterations is set for graph partitioning.

Node color exchange is the second stage of our JA-BE-JA-RIK. The process of node color exchange is implemented based on Hadoop by utilizing the energy computation proposed by JA-BE-JA. It is executed completely in parallel on the machine nodes in the cluster. When graph partitioning begins, each machine node reads only one row from the data blocks at each time. Then, this row of data is split in mapper functions. The results of mapper functions are as the input of reducer functions to perform specific graph partitioning. There are mainly two steps in the reducer functions, i.e. candidate pair selecting and color exchange performing. The first step is that the main node selects the candidate color exchange nodes from its direct neighbors or randomly selected nodes. In the second step, the energies of any two nodes before and after color exchange are calculated to decide whether the color exchange will take place. The two nodes exchange their colors only if the exchange will decrease their energy. Otherwise, there does not need to exchange their colors. Then the program goes to the first step and reads next row in the data blocks for repeating operations above until all nodes in the graph data perform the operations of color exchange. At the end of the node color exchange stage, all the node colors will be updated in the color table.

In the third stage for the RIK evaluation, for each of iterations, we read the node ids and their colors from the updated color table to calculate the APLs (the set APL_i) of all partitions and the edge-cuts (all EC_i) of graph partitioning. Then the ratio of EC_i to the total number of all edges $|E|$ of graph data will be calculated and further is compared with β, where β is an empirical threshold about edge-cuts constraint. If the ratio is more than β, then we need to make the next iteration without calculating RIK_i in the current iteration. Otherwise, RIK_i should be computed based on the set APL_i. At last, we will detect whether there is a peak of RIK values by comparing the current RIK_i with RIK_{i-1} and RIK_{i-2} of previous two iterations. If a peak of RIK values appears, then it means that RIK_{i-1} of the previous iteration possibly reaches the biggest RIK value. At this time, the partitioning results with colors based on our parallel BGP are output, and the iteration will be terminated without performing the remaining iterations.

V. JA-BE-JA-RIK ON HADOOP

In this section, we will discuss the relevant implementation algorithms based on Hadoop, including node color exchange and RIK evaluation.

A. NODE COLOR EXCHANGE

In the beginning, mappers split the row data read from the data block by spaces according to the preprocessing graph data format. The first split value is the current node id, and the

others are the direct neighbor ids of this node. After that, the split values will be sent to reducers to perform the operations of node color exchange. In addition, another configuration type variable representing the current temperature for simulation annealing (SA) should be sent to reducers for sharing the same variable in multiple reducers.

In reducers, the hybrid node selection strategy that combines the local node selection strategy with the random node selection strategy is used to find out the best candidate node q from p's direct neighbors or randomly selected nodes for color exchange. As for a node n_p, we chose three nodes from its neighborhood as candidates, while five nodes chosen randomly from the whole graph. Node q is selected by the energy calculation of node pairs, which means $d_p(\pi_q)^2 + d_q(\pi_p)^2$ is the maximum energy sum.

If node q exists, then compare $[d_p(\pi_q)^2 + d_q(\pi_p)^2] \times T_i$ with $d_p(\pi_p)^2 + d_q(\pi_q)^2$. If the former energy sum is bigger than the latter, node p and q exchange their colors and the color table *NIC* will be updated. Otherwise, their colors are kept and these operations above are performed on the next node in the data block. Moreover, if node q does not exist, mappers will read the next row data and repeat these operations above until all nodes perform the operations of node color exchange. The initial temperature T_0 is set to 2 and decreases by a stride $\delta = 0.003$ per iteration, that is, $T_{i+1} = T_i - \delta$. When T_i drops to 1, it does not decrease any longer.

B. RIK CALCULATION BASED ON APL

The graph data contains a lot of explicit knowledge about topological structure. Utilizing explicit knowledge is one aspect of graph computing [39], but what is more important is to use explicit knowledge for user queries in order to excavate more valuable implicit knowledge from the explicit knowledge.

The richness of implicit knowledge is an indicator that reflects the degree to which the implicit knowledge in the big graph data can be excavated by some graph-based data mining methods [40], [41].

There are many methods for mining implicit knowledge, such as degree distributions, hop-plots, shortest paths, etc. Among these methods, it is proved in [42] that path-based knowledge mining is a typical method by analyzing the existing relationships based on paths between nodes in a graph. For example, we attempt to analyze which researchers have very similar research areas from the paper citation database. Although we can directly their research relevance by the co-author relationship, more potential relevance of research areas can be excavated by analyzing the citation relationship between scientific papers that essentially is indirect relationship and can be reflected by a path between paper citations. The paths based on paper citation are implicit knowledge that can be fully considered to analyze the similar research areas among different researchers.

In this paper, we present a feasible approach that uses average path length (APL) to quantify the richness of implicit knowledge based on the explicit topological structure.

Algorithm 1 Node Color Exchange

Input: (1) $G = (V, E)$, (2) initial random colors of nodes in V. Temperature $T_0 = 2$, $i = 0$.
Output: the color table $NIC = \{v_{\text{id}}, p_{\text{id}}\}$, where v_{id} is the node id, p_{id} is the partition id in which v_{id} is located.

1) **Begin MapReduce Job**
2) **Map** (String *row*)
3) strs ← *row*.split(" "); //split the row of data by spaces
4) *key* ← strs[0];
5) *value* ← strs[1…];
6) emit < *key*, *value*>;
7) **Reduce** (key p, value vl[1…z])
8) **foreach** $j \leftarrow 1$ to n_p **do**// randomly select n_p nodes from sets vl or V as set *Candidates*.
9) *Candidates* ← *Candidates* $\bigcup$ (vl.random() |V.random());
10) **end for**
11) **foreach** q in *Candidates* **do**
12) $EM \leftarrow \mathbf{max}\{d_p(\pi_q)^2 + d_q(\pi_p)^2\}$; //*EM* represents the maximum energy sum.
13) **end for**
14) **if** $EM * T_i > d_p(\pi_p)^2 + d_q(\pi_q)^2$ **then** //T_i represents the temperature at the i^{th} iteration
15) Update color table *NIC* by color exchange between p and q;
16) **end if**
17) $T \leftarrow \mathbf{max}\{T_i - \delta, 1\}$; //the final temperature T.

Taking Figure 3 as an example, the number of nodes and edges in the two graphs are equal. Suppose that the two graphs are in a paper citation database, where nodes and edges respectively represent the co-authors and paper citation relationships between authors. A path can reflect indirect paper citation. The number of edges (direct paths) is 3 in both graphs. However, the graph in Figure 3(a) has 3 indirect paths consisting of edges, which can mine out more implicit paper citation relationships. The average path length is 10/6 in Figure 3(a), while 3/3 in Figure 3(b). Intuitionally, APL can reflect the richness degree of implicit knowledge of graphs to some extent.

If the data graph is not massive, APL can be calculated accurately. However, in many cases, since the time complexity of calculating APL is often $O(n^2)$, the computation time for APLs is enormous and intolerable when the graph to be processed is very large. In order to solve this problem, many methods for approximately calculating APL were proposed [43]–[46]. In this paper, we will use the approximate APL method discussed in [43].

Given a graph $G = (V, E)$, the formula for calculating APL of G is shown in the following.

$$APL^G \approx log|V|/log\bar{K} \tag{4}$$

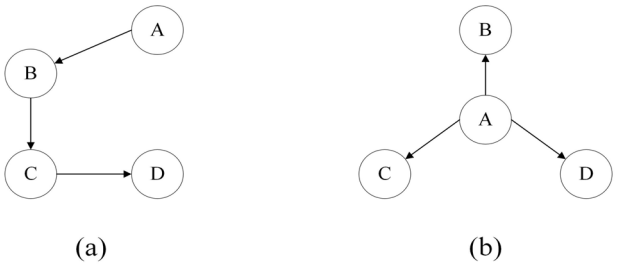


FIGURE 3. Example for RIK based on APL.

where $\bar{K}$ represents the average degree of the sum of all node degrees, which can be calculated by the following Eq. 5.

$$\bar{K} = 1/|V| \sum_{p \in V} D_p \tag{5}$$

where D_p is the degree of node p.

In JA-BE-JA-RIK, the time complexity of calculating $\bar{K}$ in each partition is O(n), so this method can solve the problem of calculating APL very well.

C. RIK EVALUATION ALGORITHM

In JA-BE-JA-RIK, nodes in the data graph are divided into balanced k partitions randomly at the beginning, so at this time there are many edge-cuts between partitions and the number of neighbor nodes with the same color to each node is small. The RIK evaluation algorithm is shown in Algorithm 2.

Considering the randomness of color assignments in our approach, the number of neighbors with the same color to each node will fluctuating increase as the partitioning proceeds, which means $\bar{K}$ of each partition will be fluctuating increase. In the situation, the RIK value will overall decrease but fluctuate locally with more iterations. The RIK evaluation algorithm is just to find the first peak of RIK (often the largest RIK) when the number of edge-cuts are confined within a certain range.

At the end of each iteration i, we calculate the APLs of k partitions and the edge-cuts of the partitioned graph with traversing the updated table *NIC*. Then the ratio of EC_i to the total number of all edges $|E|$ of graph data will be calculated and further is compared with β, where β is a threshold about edge-cuts constraint. By comparing and analyzing the ratio of the edge-cuts to the total number of edges in different graph datasets in JA-BE-JA, we obtained an empirical value of β: 0.17.

If EC_i / $|E| \leq \beta$ and i is bigger than 3, further calculations will be made to get RIK_i of the partitioned graph with set APL_i obtained above. After that, we compare RIK_{i-1} with RIK_i and RIK_{i-2}. If RIK_{i-1} is greater than both RIK_i and RIK_{i-2}, the first peak of RIK appears and the iterations end. Otherwise, the framework enters next iteration. The operations of RIK evaluation will be continuously performed until it finds the first peak of RIK that is supposed to have the maximum value. At last, the algorithm outputs all node colors as the final graph partitioning result.

Algorithm 2 RIK Evaluation

Input: (1) The edge set of G stored in *HBase* (*key* as the node id and *value* as corresponding neighbors' ids),
(2) The color table $NIC = \{v_{\text{id}}, p_{\text{id}}\}$, where v_{id} is the node id, p_{id} is the partition id in which v_{id} is located.

1. **foreach** $< v_{\text{id}}, p_{\text{id}} >$ in *NIC* **do** //RIK evaluation at the i^{th} iteration.
2. $partition[p_{\text{id}}]$ ++; //count the node number of each partition.
3. $neighbors \leftarrow HBase$.get($v_{\text{id}}$); //get direct neighbors of node v_{id} from *HBase* database.
4. **foreach** p_{id}' in *neighbors* **do**
5. **if** $p_{\text{id}} = p_{\text{id}}$' **then**
6. $totalDegree[p_{\text{id}}]$ ++; //count the total degree of partition p_{id}.
7. **else**
8. EC_i + +; //count edge-cuts.
9. **end if**
10. **end for**
11. **foreach** $j \leftarrow 1$ to k **do**
12. $avgDegree[j] \leftarrow totalDegree[j]$ / $partition[j]$; //count the average degrees of k partitions.
13. $APL_{i,j} \leftarrow log\ partition[j]$ / $log\ avgDegree[j]$;
14. **end for**
15. **end for**
16. **if** $i \geq 3 \wedge EC_i/|E| \leq \beta$ **then**
17. **foreach** $j \leftarrow 1$ to k **do**
18. $RIK_i \leftarrow RIK_i + (APL_{i,j} \times 1000)^2$;
19. **end for**
20. **if** $RIK_i < RIK_{i-1} \wedge RIK_{i-1} > RIK_{i-2}$ **then**
21. exit(); //The first peak of *RIK* appears and the iterations end.
22. **end if**
23. **end if**

VI. EXPERIMENTAL EVALUATION AND ANALYSIS

In this paper, our proposed approach will be implemented based on Hadoop, HDFS and MapReduce [23], [47], [48]. Since JA-BE-JA has been compared the edge-cuts with METIS and this paper aimed to optimize JA-BE-JA, the experimental results were only compared with JA-BE-JA. All experiments were repeated five or more times. We report the average of these measurements. The settings of our experiment environments will be introduced in the following in detail.

A. EXPERIMENT SETTINGS

In this paper, we use the MyEclipse + Ubuntu14.04.4 + JDK1.7.0 + Hadoop2.5.2 cluster. The Hadoop cluster has up to eight machine nodes deployed in virtual machines on eight computers. The hardware settings and software versions are as follows.

B. DATASETS

We have used three graph datasets, including add20 and 3elt from the Walshaw archive [50], and the well-known

TABLE 1. Hardware settings.

Name	Physical Host	Virtual Machine
CPU	Intel (R) Core (TM) i3-4160，3.60GHz	2 cores
Memory	8GB	2GB
Disk	1TB	20GB

TABLE 2. Software versions.

Software	Version
Ubuntu	Ubuntu-14.04.4-server LTS
Hadoop	2.5.2
Zookeeper	3.4.6
HBase	1.0.1.1
JDK	1.7.0_79
MyEclipse	Professional 2014
VMware Workstation	11.1.2 build-2780323

TABLE 3. Details about datasets.

Dataset	|V|	|E|	Type	References
add20	2395	7462	Walshaw	[50]
3elt	4720	13722	Walshaw	[50]
Facebook	63731	817090	Social	[51]

social network graph Facebook [51]. The details about these datasets are listed in Table 3.

C. FEASIBILITY AND VALIDITY ANALYSIS

These datasets are calculated on our Hadoop cluster with different numbers (i.e., 2, 4, 8) of machine nodes respectively. In order to evaluate the performance of our framework JA-BE-JA-RIK, we performed related experiments and analyzed the experimental results from three aspects: *Speedup*, *Scaleup*, and *Sizeup*.

1) SPEEDUP ANALYSIS

Speedup analysis records the running time on different number of machine nodes, so as to analyze the performance of our framework in different datasets. The formula of *Speedup* is shown as follows.

$$Speedup(m) = T_1/T_m \tag{6}$$

In Formula 6, m represents the number of machine nodes used in the cluster, T_1 represents the time required for a single machine to process the dataset, and T_m represents the total time required to process the dataset on m machines.

We performed the *Speedup* experiments respectively on two, four and eight machine nodes. The communication cost between machine nodes increases as the number of machine nodes increases, but Figure 4 shows that the parallel graph partitioning on different datasets is approximate linear, which shows that our partitioning approach has a good speedup for parallel graph partitioning.

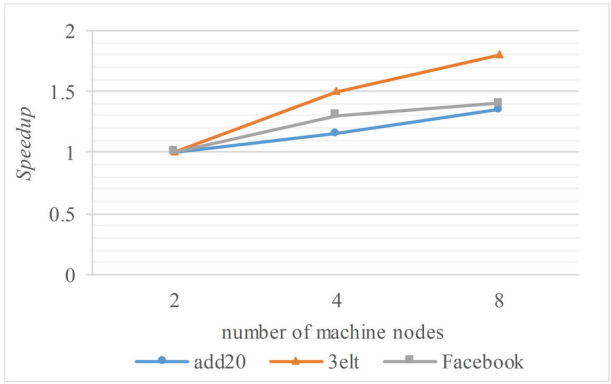


FIGURE 4. *Speedup* for different graphs.

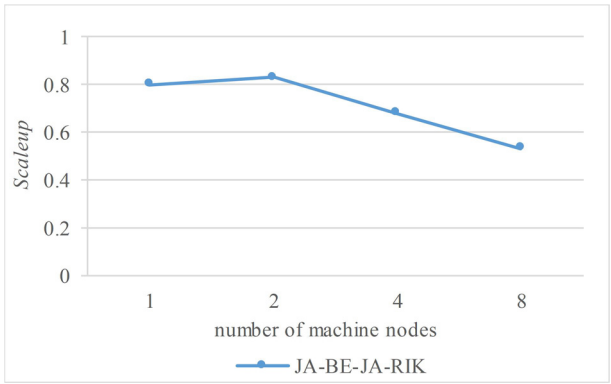


FIGURE 5. *Scaleup* of JA-BE-JA-RIK.

2) SCALEUP ANALYSIS

Scaleup analysis mainly compares the respective partitioning time based on the same increase scale of graph datasets and the number of machines. The formula of *Scaleup* is shown as follows.

$$Scaleup(D, m) = T_{D1}/T_{Dm} \tag{7}$$

where D is a dataset of a specified size, m is the multiple of dataset growth, T_{D1} is the execution time of calculating dataset D on a single machine, and T_{Dm} is the execution time when the size of processing dataset is increased by m times as well as the number of executing nodes in the cluster is also increased by m times. If the value of *Scaleup* is maintained at around 1.0 with the increase of m, it indicates the parallel algorithm has good scalability for large datasets.

From the experimental results in Figure 5, it can be seen that the number of machine nodes in the cluster increases with the growth of datasets, and it is difficult to control the running time changing with the same scale. As the number of machine node increases the communication cost of the cluster, the overall scalability of JA-BE-JA-RIK on Hadoop is maintained between 0.5 and 0.9, indicating that the algorithm is within an acceptable range.

3) SIZEUP ANALYSIS

Sizeup analysis is mainly to place the graph datasets of different sizes on clusters with equal numbers of machines.

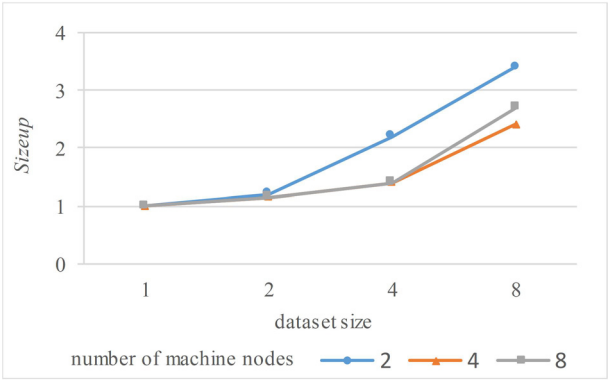


FIGURE 6. *Sizeup* of different machine nodes.

It mainly compares the execution time of processing data with different sizes on fixed number of machine nodes. The formula of *Sizeup* is shown as follows.

$$Sizeup(D, m) = T_{Sm}/T_{S1} \tag{8}$$

where D is a dataset of a specified size, m is the multiple of data growth, T_{S1} indicates the execution time when the data is run in a cluster of a fixed number of machines, and T_{Sm} indicates the execution time of data expanding m times with the same number of machines.

The experiment in Figure 6 calculates all the datasets on 2, 4, and 8 machine nodes, and counts the total running time. In one experiment, keep the number of machine nodes constant, then calculate the running time of different scale datasets. It can be seen from it that the larger the dataset size is, the more significant the *Sizeup* growth is. When the number of machine nodes in the cluster increases, the *Sizeup* growth may decrease. This is mainly due to the difference in machine performance when calculating, and the final time is based on the last machine that ends up, resulting in increasing the total running time.

D. EXPERIMENTAL ANALYSIS COMPARED WITH JA-BE-JA APPROACH

Considering JA-BE-JA was originally implemented on PeerSim [49], we implemented a Hadoop version of JA-BE-JA algorithm (called JA-BE-JA-Hadoop) for the experiments and analysis of the future, compared with our approach (JA-BE-JA-RIK). The specific comparison and analysis will be made from three perspectives, i.e. the number of edge-cuts, the number of iterations, and the RIK value of partitions.

1) COMPARISON AND ANALYSIS OF EDGE-CUTS

By changing the number of partitions, we compare the obtained edge-cuts of JA-BE-JA, JA-BE-JA-Hadoop and JA-BE-JA-RIK in the following. Figure 7 shows the edge-cuts of the three algorithms with different datasets and number of partitions after iterations.

Through comparing the experimental results between JA-BE-JA and JA-BE-JA-Hadoop, it can be seen that the edge-cuts obtained by JA-BE-JA-Hadoop is very close to JA-BE-JA on add20 and Facebook graph, slightly larger difference on 3elt graph due to its own characteristics. Overall, the smaller the number of partitions is, the closer the edge-cuts are. When JA-BE-JA-Hadoop processes different numbers of partitions respectively, the edge-cut value of add20 is average 5.5% higher than JA-BE-JA, and that of 3elt is 24.9% and Facebook 3.5%. From the above data, it can be seen that the edge-cuts of JA-BE-JA-Hadoop with different number of partitions increases by an average of 12% in contrast to JA-BE-JA, indicating that JA-BE-JA-Hadoop can obtain a good edge-cut value.

JA-BE-JA-Hadoop consistently produces partitions with very comparable edge-cuts (less than 12% difference) to the original JA-BE-JA algorithm. As a result, we can use JA-BE-JA-Hadoop as the standard on Hadoop for further experimental comparison. Based on the Hadoop platform, we compare the edge-cuts of JA-BE-JA-Hadoop with JA-BE-JA-RIK obtained from partitioning results on different datasets.

Through comparing JA-BE-JA-Hadoop with JA-BE-JA-RIK, it can be seen that the edge-cuts obtained by them are very close, although JA-BE-JA-RIK obtains more edge-cuts than JA-BE-JA-Hadoop. In conclusion, the average increases of edge-cuts in JA-BE-JA-RIK are less than 10%, which fulfills our predetermined condition, i.e., obtaining the RIK value as larger as possible with a mild increase of edge-cuts.

2) COMPARISON AND ANALYSIS OF ITERATIONS

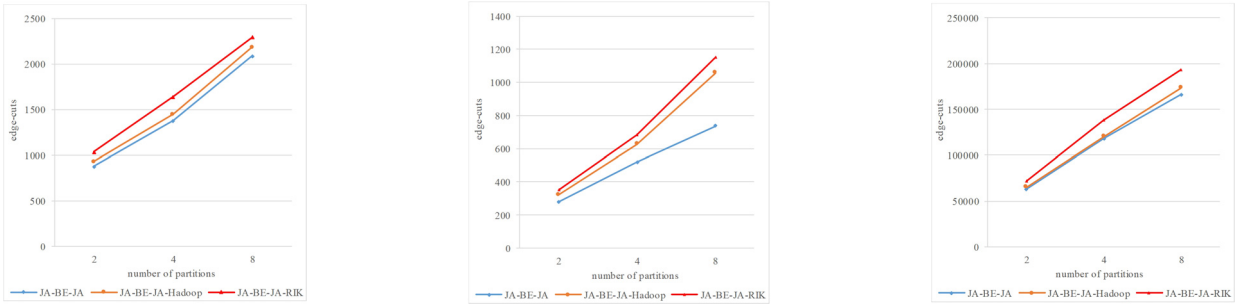
In the following, we compare JA-BE-JA-Hadoop with JA-BE-JA-RIK from the number of iterations performing graph partitioning.

As can be seen from Figure 8, the number of iterations of JA-BE-JA-RIK is greatly reduced, which is lower than 20% of JA-BE-JA-Hadoop. The early iterations of graph partitioning are very effective, because the color exchange is easy to happen. As the program goes, it becomes harder for nodes to find a candidate to exchange color with. As a result, the iteration at the later stage is useless for reducing edge-cuts, but wastes a lot of resources and generates extra computation. Though the number of iterations is dropped drastically, it does not affect the graph partitioning results and a good edge-cuts value still can be obtained.

3) COMPARISON AND ANALYSIS OF THE RICHNESS OF IMPLICIT KNOWLEDGE

In the following, we will compare JA-BE-JA-Hadoop with JA-BE-JA-RIK from the RIK value of partitions of the final graph partitioning results.

Figure 9 shows the RIK value of partitioning results obtained by the two approaches. Compared with JA-BE-JA-Hadoop, the RIK value obtained by JA-BE-JA-RIK has an average increase of 10%, which means our framework outperforms JA-BE-JA in obtaining partitions with higher RIK value to some extent.



(a) add20 graph. (b) 3elt graph. (c) Facebook graph.

FIGURE 7. Comparison of edge-cuts with different numbers of partitions (JA-BE-JA vs. JA-BE-JA-Hadoop vs. JA-BE-JA-RIK).

TABLE 4. Comparison of the overall partitioning results.

Dataset	JA-BE-JA-Hadoop			JA-BE-JA-RIK		
	iterations	edge-cuts	RIK	iterations	edge-cuts	RIK
add20	1000	927	4.3	58	1015	5.0
3elt	1000	321	3.3	156	351	4.0
Facebook	1000	65333	3.4	87	71682	3.1

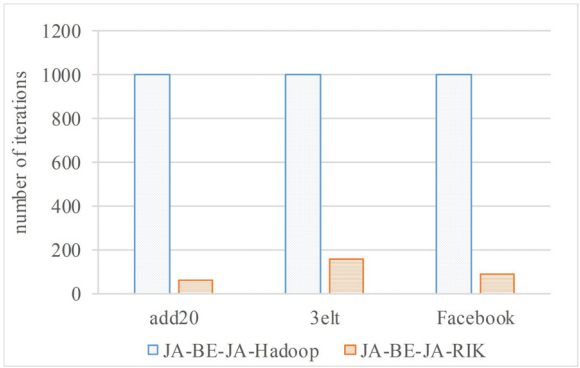


FIGURE 8. Comparison of the number of iterations on different graphs (JA-BE-JA-Hadoop vs. JA-BE-JA-RIK).

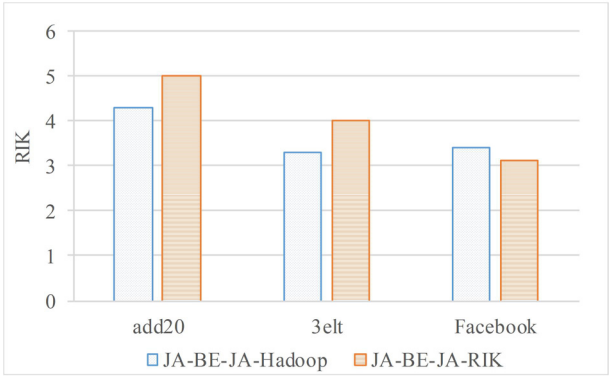


FIGURE 9. Comparison of RIK value on different graphs (JA-BE-JA-Hadoop vs. JA-BE-JA-RIK).

4) COMPREHENSIVE COMPARISON OF PARALLEL BALANCED GRAPH PARTITIONING RESULTS

We also comprehensively compare JA-BE-JA-Hadoop with JA-BE-JA-RIK according to the partitioning results. Table 4 lists the graph partitioning results obtained by both the approaches in details. Through comparison and analysis, our approach can greatly reduce the number of iterations and obtain the larger richness of implicit knowledge (*RIK*) within a closer number of edge-cuts in comparison with the traditional JA-BE-JA algorithm. It is worth noting that our approach can significantly reduce the number of iterations (decreasing the running time of BGP) with a mild increase of edge-cuts.

VII. CONCLUSION

We proposed a parallel BGP framework called JA-BE-JA-RIK by defining the concept about richness of implicit knowledge for parallel BGP, which can improve the efficacy of graph partitioning compared with JA-BE-JA. We designed and implemented our proposed framework on the Hadoop platform. Related experiments were made for illustrating that our approach outperforms JA-BE-JA, and can obtain the larger richness of implicit knowledge within a closer number of edge-cuts in comparison with the traditional JA-BE-JA algorithm.

There still exists some work to be done. Although the experimental results prove our framework can obtain the graph partitioning result with good RIK value, more work should be done to utilize our approach to optimize users' graph queries. We should solve the problem that how to combine our partitioning approach with graph queries so that graph queries can be made efficiently.

REFERENCES

[1] J. Tang, J. Zhang, L. Yao, and J. Li, "Extraction and mining of an academic social network," in *Proc. KDD*, Las Vegas, Nevada, USA, Aug. vol. 2008, pp. 990–998.
[2] Y. Tian, "SAGA: A subgraph matching tool for biological graphs," *Bioinformatics*, vol. 23, no. 2, pp. 232–239, Jan. 2007.

[3] Y. Ding, "Scientific collaboration and endorsement: Network analysis of coauthorship and citation networks," *J. Informetrics*, vol. 5, no. 1, pp. 187–203, Jan. 2011.
[4] X. Liu, Y. Zhou, X. Guan, and C. Shen, "A feasible graph partition framework for parallel computing of big graph," *Knowl.-Based Syst.*, vol. 134, pp. 228–239, Oct. 2017.
[5] C. Nabti and H. Seba, "Querying massive graph data: A compress and search approach," *Future Gener. Comput. Syst.*, vol. 74, pp. 63–75, Sep. 2017.
[6] G. Karypis and V. Kumar, "Multilevel*k*-way partitioning scheme for irregular graphs," *J. Parallel Distrib. Comput.*, vol. 48, no. 1, pp. 96–129, Jan. 1998.
[7] P. Sanders and C. Schulz, "Distributed evolutionary graph partitioning," in *Proc. Algorithm Eng. Expermiments (ALENEX)*, Kyoto, Japan, Jan. 2012, pp. 16–29.
[8] V. Bhatia and R. Rani, "A parallel fuzzy clustering algorithm for large graphs using pregel," *Expert Syst. Appl.*, vol. 78, pp. 135–144, Jul. 2017.
[9] R. Power and J. Li, "Piccolo: Building fast, distributed programs with partitioned tables," in *Proc. OSDI*, Vancouver, BC, Canada, Oct. 2010, pp. 293–306.
[10] B. Hendrickson and R. Leland, "A multi-level algorithm for partitioning graphs," in *Proc. SC*, San Diego, CA, USA, Dec. 1995, Art. no. 28.
[11] H. Meyerhenke, B. Monien, and T. Sauerwald, "A new diffusion-based multilevel algorithm for computing graph partitions of very high quality," in *Proc. ISPDP*, Miami, FL, USA, Jun. 2008, pp. 1–13.
[12] H. Meyerhenke, B. Monien, and S. Schamberger, "Graph partitioning and disturbed diffusion," *Parallel Comput.*, vol. 35, nos. 10–11, pp. 544–569, Oct. 2009.
[13] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, Aug. 1999.
[14] Y. Low, D. Bickson, J. Gonzalez, A. Kyrola, and J. M. Hellerstein, "Distributed GraphLab: A framework for machine learning and data mining in the cloud," in *Proc. VLDB*, Istanbul, Turkey, vol. 5, no. 8, Aug. 2012, pp. 716–727.
[15] P. Sanders and C. Schulz, "Engineering multilevel graph partitioning algorithms," in *Proc. ESA*, Saarbrücken, Germany, Sep. 2011, pp. 469–480.
[16] P. Sanders and C. Schulz, "Think locally, act globally: Highly balanced graph partitioning," in *Proc. SEA*, Rome, Italy, Jun. 2013, pp. 164–175.
[17] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.
[18] F. Rahimian, "JA-BE-JA: A distributed algorithm for balanced graph partitioning," in *Proc. SASO*, Philadelphia, PA, USA, Sep. 2013, pp. 51–60.
[19] E. G. Talbi, *Metaheuristics: From Design to Implementation*. Hoboken, NJ, USA: Wiley, Jun. 2009.
[20] G. Karypis and V. Kumar, "Parallel multilevel series k-way partitioning scheme for irregular graphs," *SIAM Rev.*, vol. 41, no. 2, pp. 278–300, 1999.
[21] L. M. Vaquero, F. Cuadrado, D. Logothetis, and C. Martella, "Adaptive partitioning for large-scale dynamic graphs," in *Proc. IEEE 34th Int. Conf. Distrib. Comput. Syst.*, Jul. 2014, pp. 144–153.
[22] I. Stanton and G. Kliot, "Streaming graph partitioning for large distributed graphs," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, Beijing, China, Aug. 2012, pp. 1222–1230.
[23] A. Gialecki, M. Cafarella, D. Cutting, and O. Malley, (2018). *Hadoop: A Framework for Running Applications on Large Clusters Built of Commodity Hardware*. [Online]. Available: http://hadoop.apache.org/
[24] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," *IEEE Internet Comput.*, vol. 7, no. 1, pp. 76–80, Jan./Feb. 2003.
[25] H. Bolouri, "Modeling genomic regulatory networks with big data," *Trends Genet.*, vol. 30, no. 5, pp. 182–191, May 2014.
[26] F. Rahimian, "A distributed algorithm for large-scale graph partitioning," *ACM Trans. Auto. Adapt. Syst.*, vol. 10, no. 2, Jun. 2015, Art. no. 12.
[27] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "GraphX: A resilient distributed graph system on Spark," in *Proc. GRADES*, New York, NY, USA, Jun. 2013, Art. no. 2.
[28] G. Malewicz, "Pregel: A system for large-scale graph processing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Jun. 2010, pp. 135–146.
[29] L. G. Valiant, "A bridging model for parallel computation," *Commun. ACM*, vol. 33, no. 8, pp. 103–111, 1990.
[30] S. T. Barnard and H. D. Simon, "Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems," *Concurrency Pract. Exper.*, vol. 6, no. 2, pp. 101–117, Apr. 1994.
[31] M. Fiedler, "A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory," *Czech. Math. J.*, vol. 25, pp. 619–633, 1975.
[32] A. Pothen, H. D. Simon, and K. P. Liou, "Partitioning sparse matrices with eigenvectors of graphs," *Siam J. Matrix Anal. Appl.*, vol. 11, no. 3, pp. 430–452, 1990.
[33] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, Feb. 1970.
[34] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. 19th Design Autom. Conf. (DAC)*, Las Vegas, NV, USA, Jun. 1982, pp. 175–181.
[35] V. Osipov and P. Sanders, "n-level graph partitioning," in *Proc. ESA*, Liverpool, U.K., Sep. 2010, pp. 278–289.
[36] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "PowerGraph: Distributed graph-parallel computation on natural graphs," in *Proc. OSDI*, Hollywood, CA, USA, Oct. 2012, pp. 17–30.
[37] E. Carlini, P. Dazzi, A. Esposito, A. Lulli, and L. Ricci, "Balanced graph partitioning with Apache spark," in *Proc. Eur. Conf. Parallel Process.*, pp. 129–140, Aug. 2014.
[38] C. Metz. (2013). *Spark: Open Source Superstar Rewrites Future of Big Data*. [Online] Available: http://www.wired.com/2013/06/yahoo-amazon-amplab-spark/all/
[39] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich, "A review of relational machine learning for knowledge graphs," in *Proc. IEEE*, vol. 104, no. 1, pp. 11–33, Sep. 2015.
[40] X. Yan, J. Zhang, Y. Xun, and Q. Xiao, "A parallel algorithm for mining constrained frequent patterns using MapReduce," *Soft Comput.*, vol. 21, no. 9, pp. 2237–2249, 2017.
[41] M. K. Najafabadi, A. H. Mohamed, and M. N. Mahrin, "A survey on data mining techniques in recommender systems," *Soft Comput.*, vol. 23, no. 2, pp. 627–654, Jan. 2019.
[42] A. Chemchem and H. Drias, "From data mining to knowledge mining: Application to intelligent agents," *Expert Syst. Appl.*, vol. 42, no. 3, pp. 1436–1445, Feb. 2015.
[43] R. Albert and A.-L. Barabasi, "Statistical mechanics of complex networks," *Rev. Modern Phys.*, vol. 74, no. 1, pp. 48–94, Jan. 2002.
[44] B. Bollobás, "Random graphs," in *Modern Graph Theory* (Graduate Texts in Mathematics), vol. 184. New York, NY, USA: Springer, 1998, pp. 215–252.
[45] D. J. Watts, *Small Worlds: The Dynamics of Networks Between Order and Randomness*. Upper Saddle River, NJ, USA: Princeton Univ. Press, 1999.
[46] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.
[47] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[48] M. R. Ghazi and D. Gangodkar, "Hadoop, MapReduce and HDFS: A developers perspective," *Procedia Comput. Sci.*, vol. 48, pp. 45–50, May 2015.
[49] A. Montresor and M. Jelasity, "PeerSim: A scalable P2P simulator," in *Proc. IEEE 19th Int. Conf. Peer-to-Peer Computing (IEEE P2P)*, Seattle, WA, USA, Sep. 2009, pp. 99–100.
[50] C. Walshaw. (2016). *The graph partitioning archive—Test Graphs*. Accessed: Aug. 25, 2016. [Online]. Available: http://chriswalshaw.co.uk/partition/.
[51] B. Viswanath, A. Mislove, M. Cha, and K. Gummadi, "On the evolution of user interaction in facebook," in *Proc. 2nd ACM Workshop Online Social Networks(WOSN)*, Barcelona, Spain, Aug. 2009, pp. 37–42.

ZHIPENG YANG received the B.S. degree in computer science from North China Electric Power University, Beijing, China, in 2017, where he is currently pursuing the M.S. degree in computer science. His research interests include big data analysis and processing, and parallel graph computing.

He is a member of China Computer Foundation (CCF).

RONGRONG ZHENG received the B.S. degree in computer science from the Beijing University of Post and Telecommunication, Beijing, China, in 2017. She is currently a Senior Engineer with State Grid Information and Telecommunication Group Company Ltd. Her research interests include electric power informationization and big data analysis.

YINGLONG MA (M'13) received the M.S. degree in computer science from Northwestern University, Xi'an, China, in 2002, and the Ph.D. degree in computer science from the Institute of Software, Chinese Academy of Sciences, Beijing, China, in 2006.

From 2006 to 2009, he was an Associate Professor with North China Electric Power University, Beijing, China, where he has been a Full Professor with the School of Control and Computer Engineering, since 2016. From 2010 to 2011, he was a Visiting Professor with the School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA. He is the author of two books, more than 70 articles in some international journals and international conferences, and more than six inventions. His research interests include artificial intelligence and knowledge engineering, big data analysis and processing, and software engineering. He is also a member of ACM and a Senior Member of China Computer Foundation (CCF).