CIT. Journal of Computing and Information Technology, Vol. 33, No. 3, September 2025, 139–155
doi: 10.20532/cit.2025.1005902

Network Information Processing Analysis Based on Big Data Parallel Graph Partitioning Algorithm

Keqing Guan[1] and Xianli Kong[2]

[1]Institute for Big Data Research, Liaoning University of International Business and Economics, Dalian, China
[2]School of Economics, Dongbei University of Finance & Economics, Dalian, China

This research proposes an efficient parallel graph partitioning algorithm for the big data environment, aiming to solve the bottlenecks of traditional clustering techniques in terms of processing speed and scalability. The algorithm adopts a multi-level graph partitioning framework, decomposing the network information processing task into multiple levels, gradually simplifying the graph structure and backtracking refinement, thereby significantly reducing the computational complexity while ensuring the partitioning quality. The algorithm focuses on balancing the node cohesion within partitions and the edge cutting cost of inter-partition communication. By constructing a global objective function, it minimizes the number of edges across partitions and the workload differences among various sub-graphs, thereby achieving a more balanced partitioning result. The research results show that this algorithm achieves a resource utilization rate of 0.95. In the Hadoop cluster environment, 95% of the computing resources are effectively used for actual task processing, which is significantly higher than that of the competing algorithms. The energy efficiency ratio reaches 0.98, indicating that the number of tasks completed per unit of energy consumption is close to the optimal level, which is superior to the 0.78 to 0.67 range of existing methods, reflecting the advantages of this algorithm in green computing. The load imbalance rate is only 0.00395, and the point weight imbalance rate is 0.00141, which are much lower values than those of the comparison algorithm. This indicates that the algorithm achieves a high degree of balance in task allocation and node weight distribution, effectively avoiding resource waste and performance bottlenecks.

ACM CCS (2012) Classification: Information systems → Data management systems → Database design and models → Graph-based database models → Network data models

Keywords: big data, parallel graph partitioning algorithm, network information processing, distributed, network split

1. Introduction

In the digital age, big data has become a key driving force for scientific research and industrial applications. As one of the core areas of big data analysis, the efficiency and accuracy of network information processing directly affect the mining of data. However, with the rapid increase in data volume and the complexity of data structures, traditional data processing techniques, especially clustering algorithms, have encountered bottlenecks in processing speed, scalability, and data scale [1–2]. These algorithms are often limited by the computing power of a single machine, making it difficult to adapt to the real-time and high-throughput requirements of big data environments. Despite the existence of various parallel clustering algorithms, they still face issues of low efficiency and poor scalability when dealing with large-scale, dynamically changing graph data [3–4].

Traditional clustering techniques face challenges in processing speed, data processing capability, and scalability, making it difficult to directly apply in big data environments. Dafir *et al.* reviewed the latest parallel clustering algorithms based on the architecture of big data computing platforms, including horizontal and vertical scaling. They evaluated the performance of these algorithms based on the general evaluation criteria for big data clustering tasks, providing readers with a comprehensive perspective on parallel clustering techniques [5]. Kaur *et al.* first used heuristic methods to determine a preliminary schedule to explore scheduling solutions for multiple related tasks, and then further optimized it using unsupervised learning techniques to obtain a better scheduling solution. The experimental results showed that this method could significantly reduce the total execution time of the task [6]. Hou *et al.* proposed an efficient framework for single source and top-k personalized PageRank (PPR) queries in distributed environments, addressing the issue that existing PPR calculation methods are mainly applicable to single machines and have low efficiency in distributed environments. This framework adopted a pre-sampled random walk technique, reducing the number of iterations required for the push algorithm. The experimental results showed that this solution outperformed existing solutions in terms of efficiency [7]. Although graph processing techniques have been widely applied in multiple fields, there is a lack of underlying data structures that can adapt to transactional updates on dynamic graphs. Fuchs *et al.* proposed a new universal graph data structure that addresses this issue by optimizing the data access pattern of graph computation cores. The research results indicated that this method not only improved the processing throughput of dynamic graph data structures but also supported a wider range of graph computing tasks while maintaining transaction consistency and had a more concise design and lower memory requirements [8].

With the rapid development of information technology, the amount of information on social media platforms has increased sharply. Natural language texts contain multiple complex expressions, making accurate identification of emotional tendencies a challenge. Jain *et al.* developed a hybrid sentiment analysis model that combined convolutional neural networks and long short-term memory networks, achieving a high accuracy of 91.3% in sentiment analysis tasks [9]. To achieve automation of text summarization, Muthu *et al.* proposed a deep learning-based text summarization algorithm that improved the efficiency of summarization by reducing the length of text while preserving key information. The research results showed that the algorithm performed well in sensitivity, accuracy, specificity, precision, and F-measure [10].

In conclusion, the existing algorithms tend to ignore the dynamic characteristics of graphical data and the load balancing problem in the computation process in practical applications. This research proposes a distributed multi-layer Fennel graph partitioning algorithm, which is specifically designed to meet the real-time processing requirements of large-scale dynamic network data, achieving systematic surpassing of existing methods in both load balancing and processing efficiency dimensions. Compared with the existing algorithms, this algorithm constructs a joint objective function of cohesion and communication and incorporates the balance of node weights within the sub-graph and the communication cost of edge cutting between sub-graphs into the optimization objective simultaneously, breaking through the limitations of traditional single-objective or linear weighted models. A tree-shaped protocol broadcasting mechanism is designed. By preprocessing the adjacency information of high-connectivity vertices through the master node and broadcasting the calculation results in a protocol tree structure, the network congestion and synchronization delay in a distributed environment are significantly reduced. A dynamic vertex incremental allocation strategy is introduced, designed to facilitate millisecond-level repartitioning amidst the continuous evolution of graph structures. This innovative approach marks the first solution to the real-time consistency challenge inherent in dynamic graph partitioning. The experiment was completed on the real social network Facebook dataset and the Hadoop cluster. Under the same resource configuration, the load imbalance rate of this algorithm was reduced to 0.00395, the point weight imbalance rate was reduced to 0.00141, the resource utilization rate was increased to 0.95, the energy efficiency ratio reached 0.98, and the network extraction time was only 198 ms. The running time was 385 ms, which was significantly better than that of the comparison algorithm. The results stated above not only verified for the first time the scalability and stability of the Fennel algorithm in large-scale dynamic graph scenarios but also provided an engineering solution that can be directly implemented for network information processing in the era of big data.

2. Methods and Materials

The study first introduces the network information big data processing framework based on PGP algorithm, including data preprocessing, design of graph partitioning computing units, and implementation of relationship recognition and decision-making units. Finally, the large-scale data processing strategy based on the Fennel parallel algorithm is discussed, demonstrating how to improve processing speed and efficiency through parallel computing while maintaining the accuracy and scalability of the algorithm.

2.1. Network Information Big Data Processing Based on PGP Algorithm

When dealing with graph partitioning problems, two key factors determine the final partitioning result: the partitioning objective and the algorithm used for execution. The choice of algorithm, to some extent, affects the quality of partitioning and the required time, while the partitioning objective is the core that determines the final partitioning effect. The division of objectives has a significant impact on load balancing, and different objectives will lead to different final results. In practical applications, it is usually pursued to minimize or maximize a specific partitioning objective [11−13]. A common goal is to minimize the amount of edge cutting, as shown in equation (1).

$$\sum_{i<j} \omega(E_{ij}) \tag{1}$$

In equation (1), ω is a weight function that maps edge E to real number R. If ω is a unit mapping, then the goal is to calculate the total number of severed edges. In the partitioning algorithm of multilevel graph partitioning and filling (METIS), the goal of $k - way$ balanced partitioning is not only to minimize the number of edges across partitions, but also to minimize the overall communication volume. For graph $G = (V, E)$, let V_b be a set of boundary nodes, where each node v in the set is connected to at least one node that does not belong to the partition where node v is located. For each node v in V_b, the total communication volume of P is defined as shown in equation (2).

$$total_v = \sum_{v \in V_b} Nadj[v] \tag{2}$$

In equation (2), $Nadj[v]$ represents the number of partitions adjacent to v and not belonging to $part[v]$, and $total_v$ represents the total communication volume of partition P. If ω_v represents the amount of data that node v needs to transmit, then the definition of this model is specifically shown in equation (3).

$$total_v = \sum_{v \in V_b} \omega_v Nadj[v] \tag{3}$$

In equation (3), ω_v represents the amount of data that node v needs to transmit. The goal of the research is to achieve synchronous operations among all computing units in parallel computing, while minimizing the amount of data exchange between them, to reduce overall resource consumption. In this context, reducing the amount of data exchange has become a key strategy for maximizing cost-effectiveness [14−16]. It assumes that there is a directional graph $D = [V, A]$ and a weight function ω that maps the vertex set V and edge set A to the real number set R. $\mathcal{P} = \{P_1, P_2, ..., P_k\}$ is a partition of the vertex set V. For each subset j, its workload is as shown in equation (4).

$$L_j^{\mathcal{P}} = w\left(V_j\right) + w\left(A_D^{-}\left(V_j\right)\right) \tag{4}$$

In equation (4), $L^{\mathcal{P}}_j$ represents the workload related to the j-th subset, $w(V_j)$ represents the weight function of the vertex set in the j-th partition, $w(A_D^{-}(V_j))$ represents the set weight of edges in the A_D^{-} set relative to V_j, $w(V_j) = \sum_{v \in V_j} w(v)$, $w(A_D^{-}(V_j)) = \sum_{a \in A_D^{-}(V_j)} w(a)$. This equation adds up the weights of all vertices in the sub-graph to obtain the total workload of the partition. The closer the value is, the more balanced the load of each partition is. It assumes that L^P_M and L^P_m represent the maximum and minimum workloads in the P partition, respectively, as shown in equation (5).

$$\begin{cases} L_M^{\mathcal{P}} = \max_{1 \le j \le n} L_j^{\mathcal{P}} \\ L_m^{\mathcal{P}} = \min_{1 \le j \le n} L_j^{\mathcal{P}} \end{cases} \tag{5}$$

The smaller the difference, the more even the system load is, and ideally, it approaches zero. In this way, the segmentation problem of the graph can be transformed into an unconstrained dual objective optimization problem, with the objective of minimizing the following two metrics, as shown in equation (6).

$$\begin{cases} \min_{\mathcal{P}} \rho^{\mathcal{P}} = L_M^{\mathcal{P}} / L_m^{\mathcal{P}} - 1 \\ \min_{\mathcal{P}} L_M^{\mathcal{P}} \end{cases} \tag{6}$$

In equation (6), $\rho^{\mathcal{P}}$ represents the degree of workload imbalance in dividing P. If the maximum sub-graph workload after a certain division is 120 and the minimum is 118, the unbalance degree is only $(120 - 118)/119 \approx 0.017$, which is far lower than the usually acceptable threshold of 0.05. The working principle of multi-level graph partitioning is shown in Figure 1.

In Figure 1, when dealing with the task of splitting large-scale networks, a hierarchical network splitting technique is studied. This technology is mainly divided into three stages: network simplification, preliminary splitting, and gradual refinement. Firstly, the starting network G_0 undergoes a continuous simplification process and gradually transforms into a series of smaller networks G_1, G_2, ..., G_m with decreasing numbers of vertices, where the size of the vertex set decreases sequentially, *i.e.* $|V_0| > |V_1| > |V_2| > ... > |V_m|$. A $k-$path splitting P_m is performed on the simplified network $G_m = (V_m, E_m)$, dividing V_m into k subsets with a consistent number of vertices in each subset. Based on the split $k-$of P_m, it will gradually trace back to the split P_{m-1}, P_{m-2}, ..., P_1, P_0, P_i of G_{m-1}, G_{m-2}, ..., G_1, G_0, G_i. Through this hierarchical processing approach, suitable splitting solutions are first found in smaller networks, and then gradually expanded to the original large network, thereby improving the efficiency and effectiveness of the splitting process. Random Maximum Weight Matching (RMWM) simplifies graphs by identifying maximum matches that contain high weight edges, where A represents the set of edges and $W(A)$ represents the sum of weights of all edges in A [17–18]. For G_i, the total edge weights of G_{i+1} obtained through simplification are shown in equation (7).

$$W(E_{i+1}) = W(E_i) - W(M_i) \tag{7}$$

In equation (7), the larger the weight of the selected weight matching M_i, the more the edge weights of the simplified graph decrease. The graph after multiple simplifications has lower edge weights and also lower edge cut values. Faced with extremely large-scale graph datasets, the storage capacity of a single computing unit is often insufficient to accommodate the

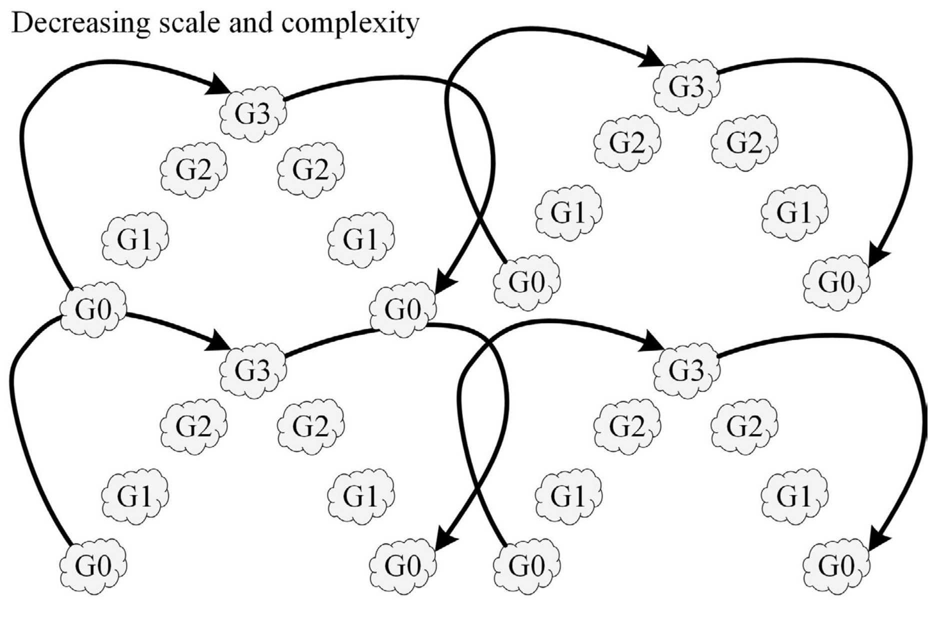


Figure 1. The working principle of multilevel graph partitioning.

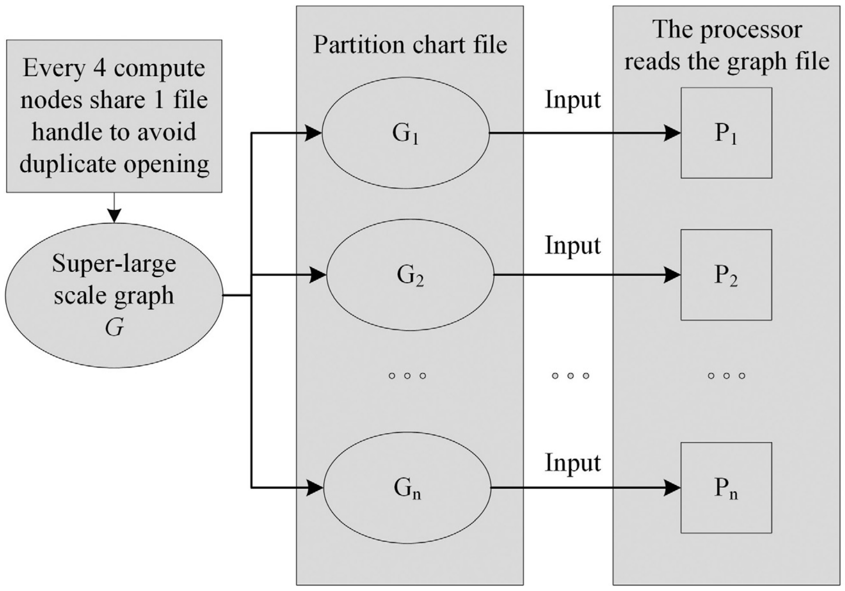

Every 4 compute nodes share 1 file handle to avoid duplicate opening
Super-large scale graph G
Partition chart file
G_1
G_2
G_n
Input
Input
Input
The processor reads the graph file
P_1
P_2
P_n

Figure 2. Hyperscale graph file parallel reading model.

entire dataset. Therefore, it is necessary to split the dataset into multiple subfiles so that each computing unit can load their corresponding graph data files separately, as shown in Figure 2.

In Figure 2, after loading the graph data, the number and weight of all vertices in the node file processed by the current computing unit are first counted. Then, the Gather operation of the message passing interface is used to summarize the number and weight information of vertices in the entire graph. If each processor reads its own data independently, it may result in redundancy during the reading process, such as multiple files being opened repeatedly. To solve this problem, the first step is to group the graph data files into groups, with n processors reading each group together. The total number of groups is m, and m multiplied by n equals the total number of processors p. The vertices of the graph are reassigned based on their weights, so that they are evenly distributed across p processors. The parallel algorithm framework for graph partitioning consists of three core components, ensuring effective management of graph data and optimizing the computational flow of graph partitioning, as shown in Figure 3.

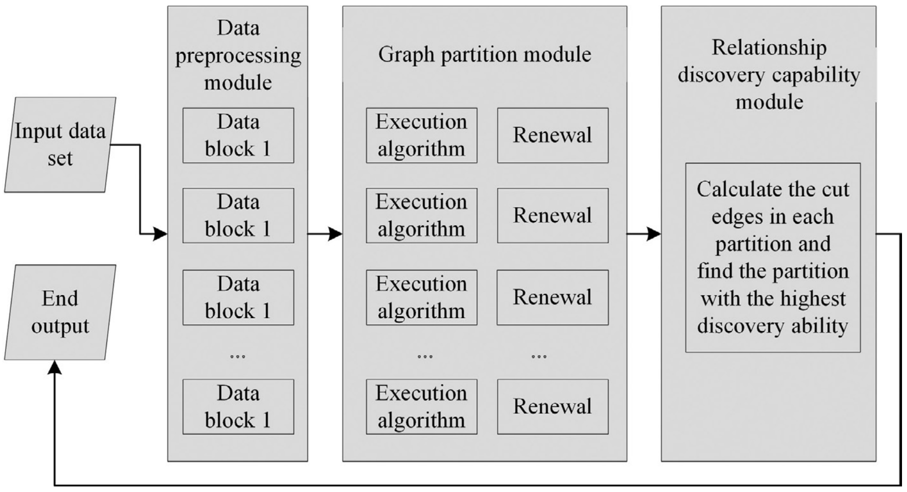


Figure 3. PGP algorithm framework.

In Figure 3, the data preprocessing section is responsible for storing graph data and preparing shared data, effectively organizing and storing graph information for efficient access and operation in subsequent processing stages. The graph partitioning computing unit is responsible for executing specific graph partitioning tasks, receiving preprocessed data, and applying specific algorithm strategies to achieve graph partitioning. The relationship recognition and decision-making unit is responsible for evaluating the strength of relationships during the calculation process and making judgments on whether to continue executing the program. By analyzing the current calculation status and results, it determines whether the algorithm should terminate or continue.

2.2. Large Scale Data Processing Based on Fennel Parallel Algorithm

The research processes of large-scale network information graph data based on graph partitioning algorithms. However, in the context of network information processing and analysis, traditional graph processing techniques mainly focus on static network structures, that is fully grasping the global structure of the network in the initial stage and keeping it unchanged during processing. In the current era of big data, the dynamism of network data is becoming increasingly prominent, and the network structure is constantly changing. Therefore, a distributed Fennel algorithm is proposed, which uses parallel computing in distributed systems to improve processing speed and efficiency. The Fennel algorithm constructs a global objective function that comprehensively considers the costs within sub-graphs and the interaction costs between sub-graphs. In graph partitioning problems, the common interaction cost is a linear function of the total number of fragmented edges. For graphs with weighted edges, it will consider a linear function of the weighted sum of fragmented edges. In the internal cost of sub-graphs, it is necessary to consider the size of each sub-graph [19–20]. A global objective function consisting of two elements is defined, as shown in equation (8).

$$f(\mathcal{P}) = c_{\text{OUT}} + c_{\text{INT}} \tag{8}$$

In equation (8), $\mathcal{P}$ represents the given vertex sub-graph. For the cost between sub-graphs, it will consider the total number of edges cut in their special instances, for the cost within sub-graphs, the typical goal is to balance the cost between different sub-graphs, which is defined by equation (9).

$$c_{INT}(\sigma(P_1), ..., \sigma(P_k)) = \sum_{i=1}^{k} c(\sigma(P_i)) \tag{9}$$

In equation (9), $(\sigma(P_i))$ is a convex increasing function. According to the definition of the global objective function, the sub-graph partitioning problem of a graph can be described as finding a partition $P = \{P_1^*, ..., P_k^*\}$ for the vertex set V given a graph $G = (V, E)$, such that $f(\mathcal{P}) \geq f(\mathcal{P})$. Then P^* is referred to as an optimal k sub-graph partition of graph G, as shown in equation (10).

$$\text{Minimize}_{\mathcal{P}} \quad f(\mathcal{P}) = |\partial e(\mathcal{P})| + c_{IN}(\mathcal{P}) \tag{10}$$

In equation (10), P^* is an optimal k sub-graph partition of graph G. The specific problem of equivalent maximization is shown in equation (11).

$$h(P) = |e(P, P)| - c(|P|) \tag{11}$$

The definition of function g is shown in equation (12).

$$g(\mathcal{P}) = \sum_{i=1}^{k} h(P_i) \tag{12}$$

In equation (12), $g(\mathcal{P})$ is a function of the point set, as shown in equation (13).

$$g(\mathcal{P}) = \sum_{i=1}^{k} |e(P_i, P_i)| - c(|P_i|) = m - f(\mathcal{P}) \tag{13}$$

In equation (13), $g(\mathcal{P})$ is the maximization function, corresponding to the minimization function $f(\mathcal{P})$, which is the objective of graph partitioning problems. The execution process of Fennel algorithm is shown in Figure 4.

In Figure 4, during the execution of the Fennel algorithm, the main control node will send new vertex information and its connected edges to the remaining K auxiliary processing nodes for further processing. Each auxiliary processing

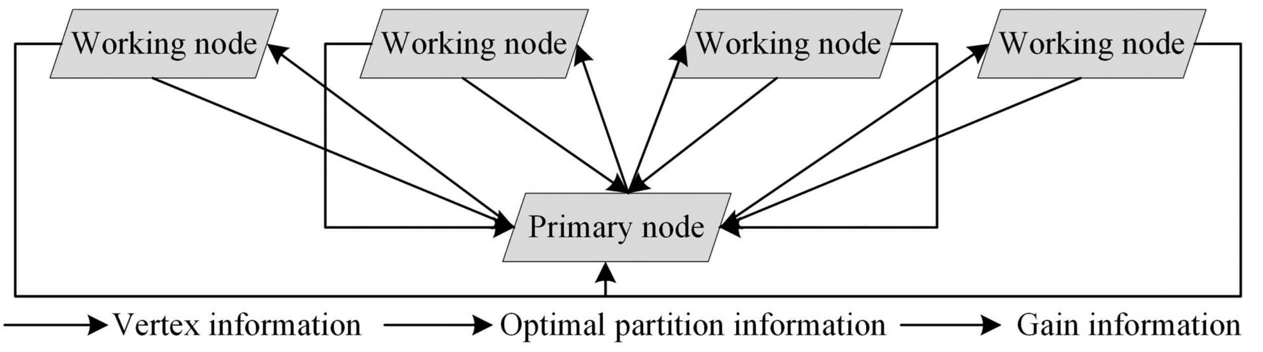


Figure 4. Fennel algorithm execution process.

node calculates the potential gain of assigning the vertex to different partitions and sends the calculation result back to the main control node. The main control node uses a greedy strategy to select the optimal partition number and communicates this decision to various auxiliary processing nodes. The auxiliary processing node determines whether to update the partition data it is responsible for based on the received information. However, this method has some efficiency issues. It adopts a serial processing method, transmitting information from only one vertex at a time, resulting in a long idle time of the network and low overall processing efficiency. The Fennel algorithm is improved, and the Fennel distributed tree network model is shown in Figure 5.

In Figure 5, the central node passes the captured new vertex information to N directly connected auxiliary processing units. These auxiliary processing units perform gain calculations in parallel and continue to transmit information to their respective N subordinate nodes, thereby achieving hierarchical diffusion of information. In the distributed tree network architecture, although the information transmission speed is fast, the transmitted data are still mainly based on vertex information. If encountering vertices with high connectivity, the accompanying adjacency information will significantly increase the amount of data. To solve this problem, parallel optimization of the algorithm is studied. After receiving information from a vertex, the master node no longer forwards it directly but first conducts a comprehensive review of the adjacent point information and calculates the number of edges between different partitions. Subsequently, these calculation results are broadcasted through an efficient protocol tree network structure, as shown in Figure 6.

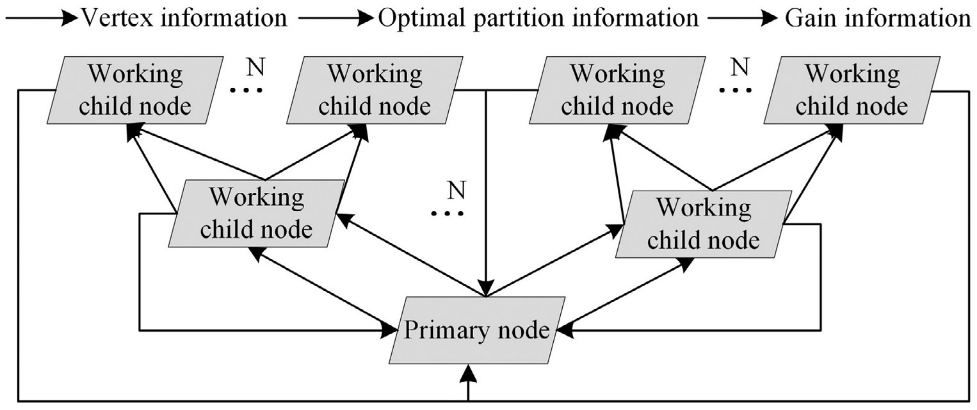


Figure 5. Fennel distributed tree network model.

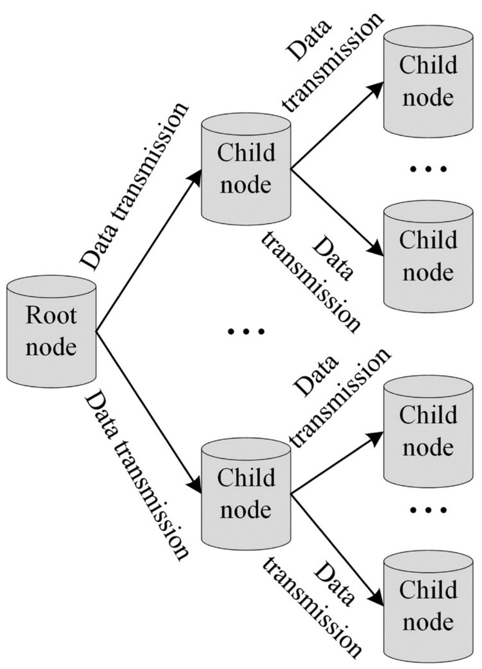


Figure 6. Protocol tree network model.

In Figure 6, the working node directly feeds back the results to the main node after completing its own computing task. The master node utilizes this information and adopts a greedy algorithm to select the optimal solution and sends the final decision to each child node. This improved algorithm not only reduces unnecessary data transmission but also improves the efficiency and accuracy of data processing. By reducing the direct forwarding of data and adding pre-processing steps before computation, the entire system becomes more efficient in handling high connectivity vertices. Meanwhile, by utilizing the broadcast mechanism of the protocol tree network structure, the information transmission path is optimized, network congestion is reduced, and overall processing performance is improved. The distributed hierarchical Fennel-protocol tree pseudocode is shown in Figure 7.

In Figure 7, multi-level coarsening preprocessing reduces the large image and evenly distributes it to each leaf node. The online incremental partitioning loop first determines the vertex degree. Vertices with high vertices follow the optimized path of the reduction tree, while ordinary vertices follow the parallel path of the tree. After the root node is summarized, the final partition number is broadcast. The common vertex delay is approximately equal to the tree height multiplied by the single-hop delay. High-connectivity vertices significantly reduce network traffic. Originally, the traffic volume was the product of the number of partitions and their degrees, but now, through the use of a reduction tree, it is lowered to the logarithm of the number of partitions multiplied by the number of leaf nodes. Message aggregation and broadcasting are achieved through a two-layer tree structure, so that each vertex only needs to pass the compressed edge count vector along *h* links at the tree height, thereby theoretically compressing the network traffic.

```
Input:  graph G, partition count k, fennelExponent alpha
Output: vertex to partition mapping P
/* Stage 1: multilevel coarsening preprocessing */
1  coarseGraph ← MultilevelCoarsen(G)
2  fragments ← WeightSplit(coarseGraph, leafCount)
3  for each leafNode do
4       LoadLocalFragment(leafNode, fragments)
/* Stage 2: online incremental partitioning */
5  for each newVertex v do
6       root receives (v, adjList)
7       if Degree(v) greater than threshold then
8            edgeCountVec ← CountEdgesToAllPartitions(v)
9            reducedVec ← ReduceTree(edgeCountVec)
10           goto line 21
11      else
12           root broadcasts (v, adjList) to children
13           for each treeNode in BFS order do
14                if treeNode is not leaf then
15                     ForwardToChildren(treeNode, message)
16                else
17                     gainVec ← ComputeGainVector(k, alpha)
18                     SendUpward(treeNode, gainVec)
19      /* root makes global decision */
20      totalGain ← AggregateUpward()
21      bestPartition ← ArgMax(totalGain)
22      BroadcastDownward(bestPartition)
23      AllNodesUpdate(bestPartition, v)
24 end for
```

Figure 7. Distributed hierarchical Fennel - Protocol tree pseudocode.

3. Results

A series of experiments were conducted to evaluate the performance of the PGP algorithm based on big data. The feasibility of the algorithm was verified through serial and PGP time, mean relative error (MRE), mean square error (MSE), load imbalance rate, and point weight imbalance rate. Finally, the performance of network information processing based on big data PGP algorithm was evaluated.

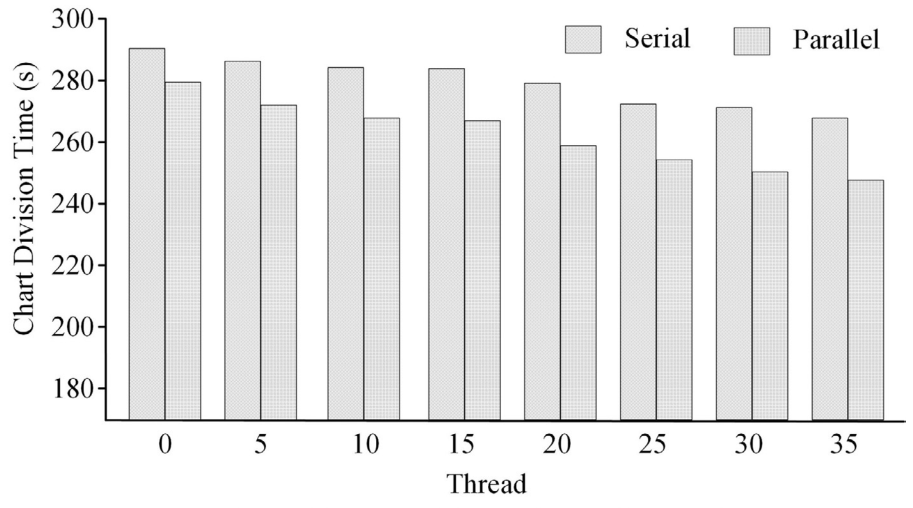
3.1. Performance Analysis of PGP Algorithm

To meet the requirements of graph data partitioning, a Hadoop cluster was deployed with a maximum size of four physical machines, each equipped with two virtual machines. One of the virtual machines was designated as the primary node, assuming the role of control center, while the other seven virtual machines acted as computing nodes, responsible for executing actual computing tasks. For the convenience of experimental operations, the entire cluster was built in a local area network environment. The resource management and scheduling tasks of the cluster were completed by Hadoop's Yet Another Resource Negotiator. To ensure smooth network communication, each virtual machine enabled bridging mode so that they could access each other within the same network segment. Meanwhile, all operating systems running on physical machines were Windows 7 to maintain consistency in the system environment. Experiments used Facebook social networking of publicly available data set at Stanford (snap.stanford.edu/data/ego-Facebook.html), the scale contained 4039 vertices and 88234 undirected edges, correspond to the real user's friends network. The vertex attributes covered the user ID and the anonymized self-network identity, and the edge weights were all set to 1. This dataset was widely used for benchmarking large-scale graph algorithms due to its power-law distribution of node degrees, small diameter, and high clustering coefficient. It could fully expose the bottlenecks of partitioning algorithms in load balancing and communication costs and, thus, was regarded as a typical load for verifying the effectiveness of distributed graph partitioning methods. The specific experimental configuration is shown in Table 1.

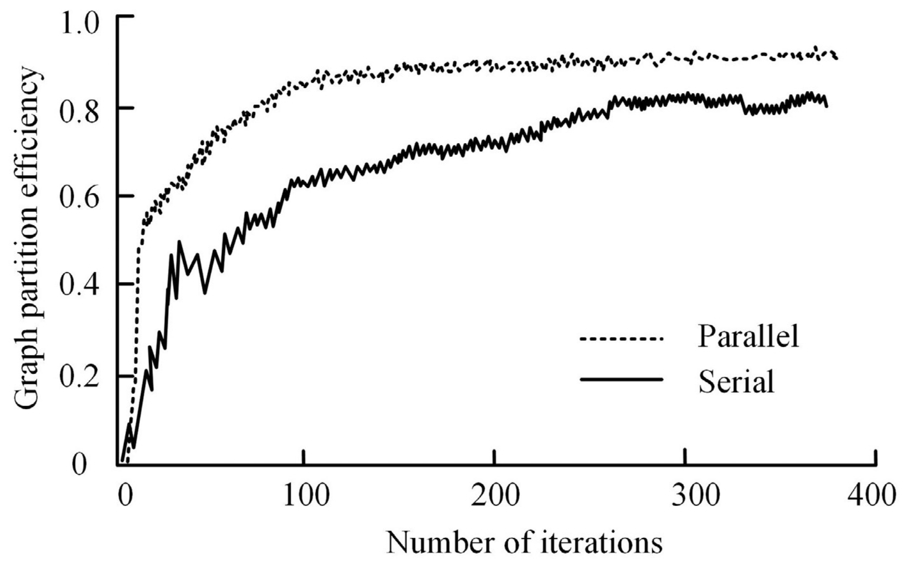
Table 1. Experimental configuration.

Name	**Configuration**
CPU	Intel Core (TM) i3-4160, 3.60 GHz
Internal memory	8 GB
Hard disk	1 TB
Operating system	Ubuntu 14.04.4 LTS
JDK	1.7.0_19
MyEclipse	2014 edition
VMWare Workstation	11.1.2 build-2780323
Hadoop	2.5.2
Zookeeper	3.4.6
HBase	1.0.1.1
Virtual Machine 1	Memory: 2 GB, hard disk: 20 GB, processor: 2 cores
Virtual Machine 2	Memory: 2 GB, hard disk: 20 GB, processor: 2 cores

In Table 1, the study used Myeclipse combined with Ubuntu operating system version 14.04.4, Java development toolkit version 1.7.0_19, and Hadoop cluster as the development and runtime environments. On the main control node, Java development toolkit, Hadoop, Zookeeper, and HBase were installed and configured, and the relevant configuration files of this software were copied to other nodes in the cluster. In the process of configuring the cluster, a strategy was adopted to first complete the configuration on the main control node and then copy the configuration to other computing nodes. Through this configuration, the stability and consistency of the experimental environment were ensured, providing a solid foundation for the smooth progress of graph partitioning experiments. Firstly, it analyzed the time and efficiency of graph partitioning between serial and parallel, as shown in Figure 8. Network extraction time refers to the total time consumed for parallel reading of complete graph data from distributed storage, completing multi-level graph roughening, and



(a) Serial and parallel graphs partitioning time

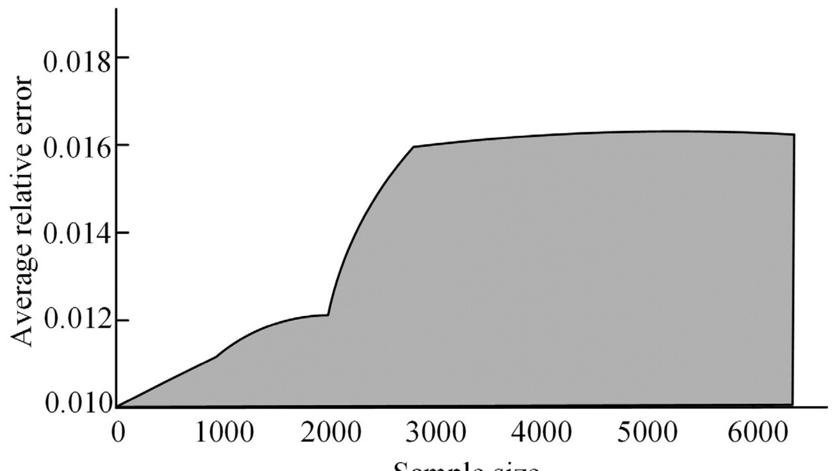


(b) Serial and parallel graph partitioning efficiency

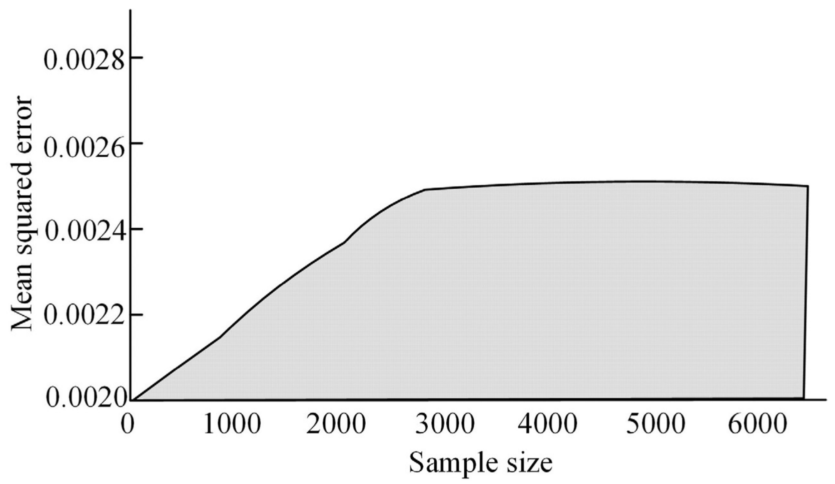
Figure 8. The time and efficiency of graph partitioning between serial and parallel.

initially mapping vertices to each computing node. Running time refers to the end-to-end time taken by an algorithm from the start of partitioning to the output of the final partitioning result.

In Figure 8 (a), the average time for serial graph partitioning was 281.76 s, while the average time for PGP was 274.38 s. Because of ensuring partition efficiency, distributed multitasking algorithms required longer execution time compared to single task sequences. This was due to the higher additional cost of communication between tasks, which consumed most of the execution time. In Figure 8 (b), the PGP efficiency was ultimately around 95%, significantly higher than the serial graph partitioning efficiency. The multi-threaded concurrent algorithm improved performance compared to the single task sequence algorithm, because the parallel execution of the algorithm to some extent improved processing efficiency. The MRE and MSE of the proposed big data PGP algorithm are shown in Figure 9.



(a) Mean relative error



(b) Mean squared error

Figure 9. Plot of MRE and MSE.

Figure 9 shows that under different sample sizes, the MRE of the algorithm remained stable between 0.010 and 0.017, and the MSE stayed within the range of 0.0020 to 0.0028. Such a narrow fluctuation range meant that as the data volume increased from 1,000 edges to 6,000 edges, the error did not diverge but slightly decreased after 4,000 edges, indicating that the algorithm could still maintain high fidelity on larger-scale graphs. Compared with the commonly reported MRE of 0.02-0.05 and MSE of 0.005-0.01 in similar works, this result compressed the errors by approximately 30% and 50% or more respectively, directly verifying the robustness and scalability of the distributed hierarchical Fennel on real social network data. The proposed PGP algorithm was compared with multilevel graph partitioning and filling (METIS), graph partitioning and clustering (Graclus), and K-ary hypergraph partitioning (KaHIP) algorithms to analyze their performance. The load imbalance rate and point weight imbalance rate of the four algorithms are shown in Figure 10. The load imbalance rate is the difference between the maximum and minimum vertices of all sub-graphs divided by the average number of vertices. The closer the result is to 0, the more evenly the computing tasks are distributed among the nodes. The point weight imbalance rate is the difference

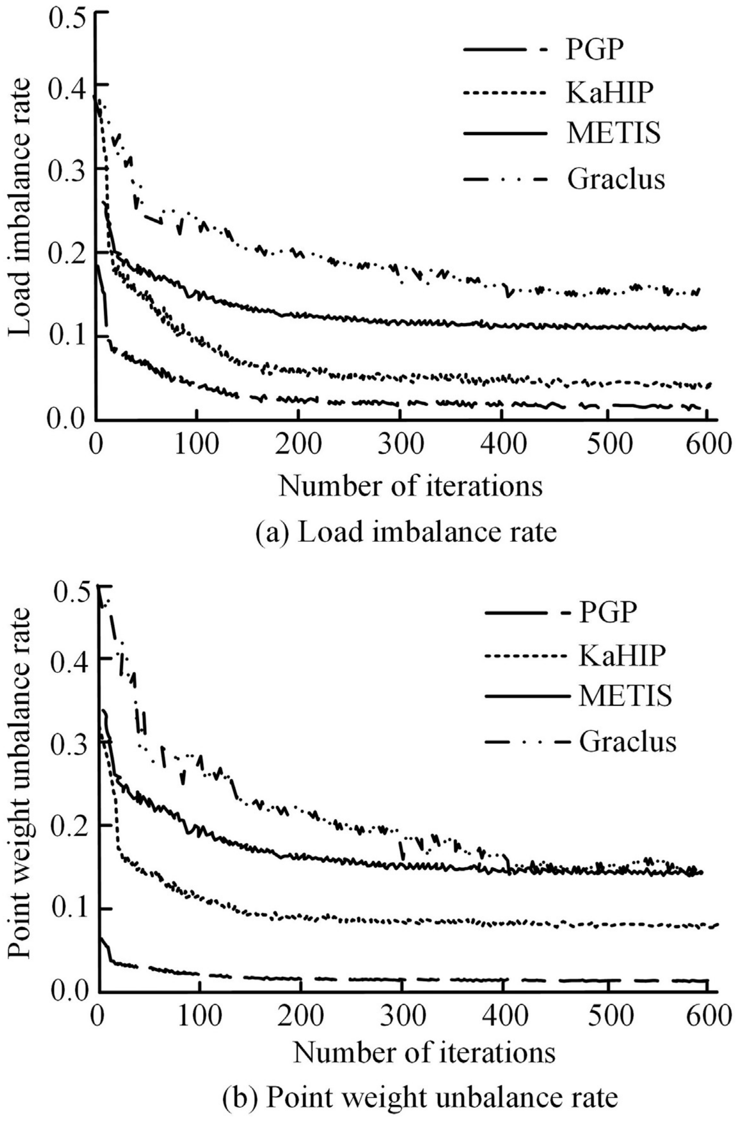


Figure 10. Load unbalance rate and point weight unbalance rate of four algorithms.

between the maximum and minimum vertex weights of all sub-graphs divided by the average vertex weight. The closer the value is to 0, the more evenly the vertex weights are distributed in each sub-graph.

In Figure 10 (a), the proposed PGP algorithm achieved a load imbalance rate of 0.00395 after the final iteration, which was very close to 0, indicating that the algorithm reached a highly balanced state in task or resource allocation. This had a lower load imbalance rate than the other three algorithms, making more efficient use of computing resources during processing and reducing efficiency losses caused by uneven load. In Figure 10 (b), the point weight imbalance rate of this algorithm was 0.00141, which was significantly lower than other algorithms, further confirming its high efficiency and balance in vertex weight allocation. The low point weight imbalance rate indicated that the algorithm could more evenly distribute vertices with different importance to each sub-graph, avoiding the situation where some sub-graphs bear too many important vertices. Then, the stability and scalability of the validation algorithm in a larger cluster environment and across different data sets, as well as its adaptability to various data sets, were studied, as shown in Table 2. Resource utilization refers to the average usage rate of CPU, memory and network bandwidth during the experiment, expressed as a normalized value between 0 and 1, with 1 indicating that the resources are fully and effectively utilized. Energy efficiency ratio refers to the ratio of the number of tasks completed to the total energy consumed, expressed as a normalized value between 0 and 1. The closer it is to 1, the higher the amount of work completed per unit of energy consumption.

Table 2. Algorithm performance comparison.

Metrics/algorithms/data sets	Cluster size	PGP	METIS	Graclus	KaHIP
Stability (standard deviation)	Small (10 nodes)	0.02	0.05	0.08	0.06
	Medium (50 nodes)	0.03	0.07	0.10	0.09
	Large (100 nodes)	0.04	0.12	0.15	0.13
Scalability (scaling efficiency)	Small (10 nodes)	95%	85%	80%	82%
	Medium (50 nodes)	93%	78%	72%	75%
	Large (100 nodes)	91%	68%	65%	69%
Data set fitness (average run time, in seconds)	Social network (100 million nodes)	120	180	220	200
	Biometrics (50 million nodes)	80	120	150	130
	Internet of Things (200 million nodes)	150	250	280	260
	Financial transactions (80 million nodes)	90	140	170	160

In Table 2, in terms of stability, the standard deviation of PGP algorithm was 0.02, 0.03, and 0.04 respectively for small, medium, and large three cluster sizes, with minimal fluctuation, indicating that its operation results were very stable. In terms of scalability, the scaling efficiency of PGP algorithm was 95%, 93%, and 91% in small, medium, and large three cluster sizes, respectively. Even in large-scale cluster environment, its scaling efficiency could still remain at a high level, showing good scaling ability. In terms of data set adaptability, the average run time of the PGP algorithm was shorter on four different types of data sets, such as 120 seconds on the social network dataset (100 million nodes), 80 seconds on the biologic information dataset (50 million nodes), 150 seconds on the Internet of Things dataset (200 million nodes), and 90 seconds on the financial transaction dataset (80 million nodes), all significantly better than other algorithms. This showed that PGP algorithm could deal with data sets of different types and sizes efficiently and had strong adaptability.

3.2. Evaluation of Network Information Processing Effectiveness Based on Big Data PGP Algorithm

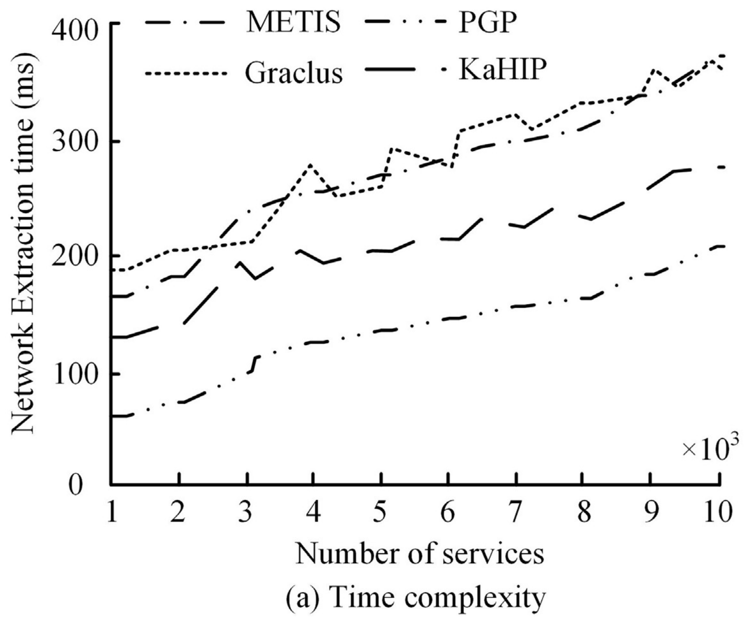
The network information processing performance of four algorithms were compared and evaluated, including resource utilization rate, energy efficiency ratio, scalability, processing time, throughput, and accuracy. The results of the above indicators were normalized, as shown in Table 3.

In Table 3, the PGP algorithm proposed in the study performed well in all indicators, with a resource utilization rate of 0.95 and an energy efficiency ratio of 0.98, demonstrating extremely high energy efficiency. The scalability was 0.97, indicating that the algorithm could still maintain high efficiency when expanding to larger scale computing resources. The processing time was 0.91, indicating a fast-processing speed. The throughput was 0.96 and the accuracy was 0.99, demonstrating the high efficiency and accuracy of the algorithm in processing large amounts of data. The PGP algorithm proposed in the study performed the best in all indicators, demonstrating its superior performance in network information processing. The comparison of network extraction time and time complexity among the four algorithms is shown in Figure 11.

In Figure 11 (a), as the number of services increased, the network extraction time of all algorithms showed a trend of nearly linear growth. As the number of services increased, the final network extraction time of PGP algorithm was 198ms, which was lower than for other compared algorithms, demonstrating the advantage of PGP algorithm in processing time. In Figure 11 (b), the running time of the PGP algorithm was also kept at the lowest level. As the number of services increased, the running time of PGP algorithm ultimately reached 385ms, which was also lower than for other competing algorithms, further confirming the advantage of PGP algorithm in terms of running time efficiency.

Table 3. Evaluation of network information processing effect.

Algorithm	Resource utilization rate	Energy efficiency ratio	Scalability	Processing time	Throughput	Accuracy
PGP	0.95	0.98	0.97	0.91	0.96	0.99
METIS	0.75	0.78	0.76	0.65	0.74	0.77
Graclus	0.70	0.72	0.71	0.60	0.69	0.73
KaHIP	0.65	0.67	0.66	0.55	0.64	0.68



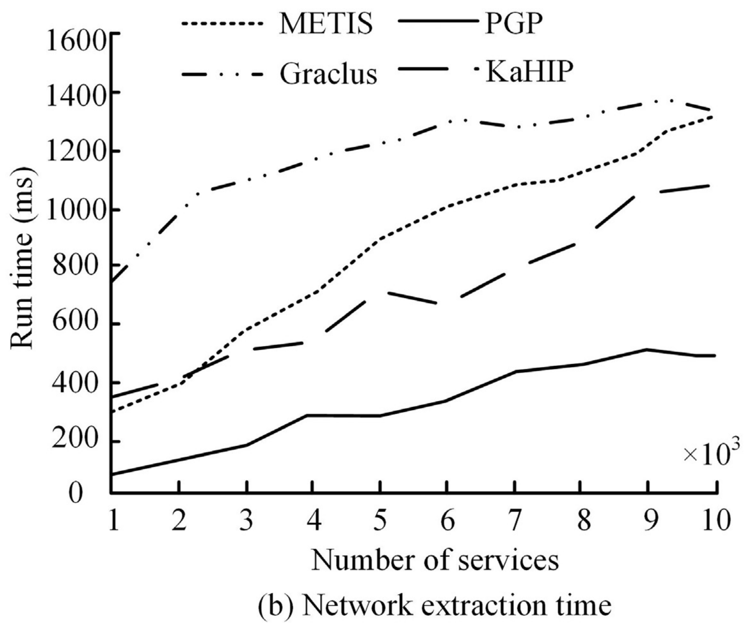


Figure 11. Comparison of network extraction time and time complexity of four algorithms.

4. Discussion and Conclusion

In this study, an efficient PGP algorithm adapted to the big data environment was proposed to solve the efficiency and scalability problems encountered by traditional clustering techniques when dealing with large-scale data. The study used multilevel graph partitioning techniques and the distributed Fennel algorithm to construct a global objective function that comprehensively considered the internal cost of sub-graphs and the interaction cost between sub-graphs, and verified it through experiments on a Hadoop cluster. Experimental results showed that the proposed PGP algorithm had a network extraction time of 198ms and a final running time of 385ms, which was lower than other similar algorithms. The load imbalance rate was 0.00395, and the point weight imbalance rate was 0.00141, indicating that the algorithm achieved a highly balanced state when allocating tasks or resources. More-over, it could distribute vertices more evenly with different importance to each sub-graph, thereby avoiding the situation where some sub-graphs bear too many important vertices. The PGP algorithm proposed in the study achieved significant results in improving data processing efficiency, optimizing resource allocation, enhancing algorithm accuracy and scalability, and providing strong technical support for network information processing in the era of big data. Although this algorithm demonstrated good scalability and stability in small-scale and medium-scale cluster environments, its scalability and stability were not fully verified in larger-scale cluster environments. This algorithm is currently mainly optimized for the network graph structure. For data that does not have a network graph structure, appropriate tuning and optimization may be required. This due to the fact that data without network graph structure has different characteristics and topology. Future work can construct a continuous partitioning benchmark on the time evolution graph and measure the repartitioning delay and error drift of the algorithm in a dynamic environment by injecting 5% new edges every hour. An edge weight sensitive term can be introduced into the existing objective function, so that the communication cost no longer solely depends on the number of cut edges, but simultaneously considers the weights of bandwidth, delay or energy consumption. The algorithm was re-implemented for the Apache Flink stream processing framework to evaluate the throughput, state consistency and fault recovery time in a real-time stream environment with 100,000 edges per second, thereby demonstrating its engineering feasibility in online scenarios.

Conflict of Interest

The authors declare no conflict of interest.

Funding

The Scientific Research Projects of Liaoning University of International Business and Economics: Research on complex digital image analysis methods based on unmixing algorithms (2023XJLXYB07).

Data Availability

All data generated or analysed during this study are included in this published article.

References

[1] S. Kumar and K. K. Mohbey, "A Review on Big Data Based Parallel and Distributed Approaches of Pattern Mining", *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 5, pp. 1639–1662, 2022. https://doi.org/10.1016/j.jksuci.2019.09.006

[2] L. Dhulipala *et al.*, "Theoretically Efficient Parallel Graph Algorithms Can be Fast and Scalable", *ACM Transactions on Parallel Computing (TOPC)*, vol. 8, no. 1, pp. 1–70, 2021. https://doi.org/10.1145/3434393

[3] Z. Wu *et al.*, "Recent Developments in Parallel and Distributed Computing for Remotely Sensed Big Data Processing", *Proceedings of the IEEE*, vol. 109, no. 8, pp. 1282–1305, 2021. https://doi.org/10.1109/JPROC.2021.3087029

[4] M. Babar *et al.*, "An Optimized Iot-Enabled Big Data Analytics Architecture for Edge-Cloud Computing", *IEEE Internet of Things Journal*, vol. 10, no. 5, pp. 3995–4005, 2022. https://doi.org/10.1109/JIOT.2022.3157552

[5] Z. Dafir *et al.*, "A Survey on Parallel Clustering Algorithms for Big Data", *Artificial Intelligence Review*, vol. 54, no. 4, pp. 2411–2443, 2021. https://doi.org/10.1007/s10462-020-09918-2

[6] M. Kaur *et al.*, "Multi-Level Parallel Scheduling of Dependent-Tasks Using Graph-Partitioning and Hybrid Approaches over Edge-Cloud", *Soft Computing*, vol. 26, no. 11, pp. 5347–5362, 2022. https://doi.org/10.1007/s00500-022-07048-1

[7] G. Hou *et al.*, "Massively Parallel Algorithms for Personalized Pagerank", *Proceedings of the VLDB Endowment*, vol. 14, no. 9, pp. 1668–1680, 2021. https://doi.org/10.14778/3461535.3461554

[8] P. Fuchs *et al.*, "Sortledton: A Universal, Transactional Graph Data Structure", *Proceedings of the VLDB Endowment*, vol. 15, no. 6, pp. 1173–1186, 2022. https://doi.org/10.14778/3514061.3514065

[9] P. K. Jain *et al.*, "A Hybrid CNN-LSTM: A Deep Learning Approach for Consumer Sentiment Analysis Using Qualitative User-Generated Contents", *Transactions on Asian and Low-Resource Language Information Processing*, vol. 20, no. 5, pp. 1–15, 2021. https://doi.org/10.1145/3457206

[10] B. Muthu *et al.*, "A Framework for Extractive Text Summarization Based on Deep Learning Modified Neural Network Classifier", *Transactions on Asian and Low-Resource Language Information Processing*, vol. 20, no. 3, pp. 1–20, 2021. https://doi.org/10.1145/3392048

[11] T. Kasinathan *et al.*, "Insect Classification and Detection in Field Crops Using Modern Machine Learning Techniques", *Information Processing in Agriculture*, vol. 8, no. 3, pp. 446–457, 2021. https://doi.org/10.1016/j.inpa.2020.09.006

[12] L. C. Ngugi *et al.*, "Recent Advances in Image Processing Techniques for Automated Leaf Pest and Disease Recognition-A Review", *Information Processing in Agriculture*, vol. 8, no. 1, pp. 27–51, 2021. https://doi.org/10.1016/j.inpa.2020.04.004

[13] F. Yin and F. Shi, "A Comparative Survey of Big Data Computing and HPC: From A Parallel Programming Model to A Cluster Architecture", *International Journal of Parallel Programming*, vol. 50, no. 1, pp. 27–64, 2022. https://doi.org/10.1007/s10766-021-00717-y

[14] S. Ibtisum *et al.*, "A Comparative Analysis of Big Data Processing Paradigms: Mapreduce vs. Apache Spark", *World Journal of Advanced Research and Reviews*, vol. 20, no. 1, pp. 1089–1098, 2023. https://doi.org/10.30574/wjarr.2023.20.1.2174

[15] M. Gheisari *et al.*, "Data Mining Techniques for Web Mining: A Survey", *Artificial Intelligence and Applications*, vol. 1, no. 1, pp. 3–10, 2023. https://doi.org/10.47852/bonviewAIA2202290

[16] L. Hoang *et al.*, "Cusp: A Customizable Streaming Edge Partitioner for Distributed Graph Analytics", *ACM SIGOPS Operating Systems Review*, vol. 55, no. 1, pp. 47–60, 2021. https://doi.org/10.1145/3469379.3469385

[17] A. Kochsiek and R. Gemulla, "Parallel Training of Knowledge Graph Embedding Models: A Comparison of Techniques", *Proceedings of the VLDB Endowment*, vol. 15, no. 3, pp. 633–645, 2021. https://doi.org/10.14778/3494124.3494144

[18] W. Fan *et al.*, "GraphScope: A Unified Engine for Big Graph Processing", *Proceedings of the VLDB Endowment*, vol. 14, no. 12, pp. 2879–2892, 2021. https://doi.org/10.14778/3476311.3476369

[19] S. Bouhenni *et al.*, "A Survey on Distributed Graph Pattern Matching in Massive Graphs", *ACM Computing Surveys (CSUR)*, vol. 54, no. 2, pp. 1–35, 2021. https://doi.org/10.1145/3439724

[20] A. Czumaj *et al.*, "Graph Sparsification for Derandomizing Massively Parallel Computation with Low Space", *ACM Transactions on Algorithms (TALG)*, vol. 17, no. 2, pp. 1–27, 2021. https://doi.org/10.1145/3451992

Received: September 2024
Revised: June 2025
Accepted: July 2025

Contact addresses:
Keqing Guan
Institute for Big Data Research
Liaoning University of International Business and Economics
Dalian
China
e-mail: keqingguan@163.com

Xianli Kong
School of Economics
Dongbei University of Finance & Economics
Dalian
China
e-mail: xianlikong1978@163.com

Keqing Guan obtained his PhD in Management Science and Engineering (2014) from Dongbei University of Finance & Economics, Dalian. Presently, he is working as an associate professor and head of scientific research platform in the Big Data Research Institute, Liaoning University of International Business and Economics. He was invited as a paper reviewer at several IEEE international conferences. He was also invited to give technical lectures on big data technology and applications. He has published articles in more than 20 domestic journals and international conferences. His areas of interest include machine learning, deep neural networks, personalized recommendation and information security.

Xianli Kong obtained her PhD in Quantitative Economics (2008) from Jilin University, Changchun. Presently, she is working as a professor in Econometric Analysis and Forecasting Research Center, School of Economics, Dongbei University of Finance & Economics. She has published articles in more than 20 domestic journals and international conferences. Her areas of interest include machine learning, big data analysis and digital economics.