Received April 5, 2021, accepted April 18, 2021, date of publication April 26, 2021, date of current version May 5, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3075457

Dynamic Graph Partitioning Scheme for Supporting Load Balancing in Distributed Graph Environments

DOJIN CHOI[1], JINSU HAN[1], JONGTAE LIM[1], JINSUK HAN[1], KYOUNGSOO BOK[2], AND JAESOO YOO[1]

[1]School of Information and Communication Engineering, Chungbuk National University, Cheongju 28644, South Korea
[2]Department of Software Convergence, Wonkwang University, Iksan 54538, South Korea

Corresponding author: Jaesoo Yoo (yjs@chungbuk.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2019R1A2C2084257), in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2014-3-00123, Development of High Performance Visual BigData Discovery Platform for Large-Scale Realtime Data Analysis), in part by the MSIT (Ministry of Science and ICT), Korea, under the Grand Information Technology Research Center support program (IITP-2021-2020-0-01462) supervised by the IITP (Institute for Information & communications Technology Planning & Evaluation), and in part by the AURI (Korea Association of University, Research institute and Industry) grant funded by the Korea Government (MSS: Ministry of SMEs and Startups). (No. S3047889, HRD program for 2021).

ABSTRACT As dynamic graph data have been actively used, incremental graph partition schemes have been studied to efficiently store and manage large graphs. In this paper, we propose a vertex-cut based novel incremental graph partitioning scheme that supports load balancing in a distributed environment. The proposed scheme chooses the load of each node that considers its storage utilization and throughput as the partitioning criterion. The proposed scheme defines hot data that means a particular vertex frequently searched among graphs requested by queries. We manage and utilize hot data for graph partitioning. Finally, we perform vertex-cut based dynamic graph partitioning by using a vertex replication index, the load each node, and hot data to distribute the load evenly in a distributed environment. In order to verify the superiority of the proposed partitioning scheme, we compare it with the existing partitioning schemes through a variety of performance evaluations.

INDEX TERMS Distributed processing, graph partitioning, graph stream, hot data, vertex-cut partitioning.

I. INTRODUCTION

Graph data are used to express the relationship or interaction between users or objects. As graph data have become actively used, studies have been performed for graph partitioning schemes to efficiently store and manage large graphs [1]–[4]. The graph partitioning schemes are classified as the edge-cut partitioning and vertex-cut partitioning based on the partitioning policy [5]–[8]. The edge-cut scheme partitions a graph based on a certain edge and aims to minimize the number of cut edges that interconnect the partitioned subgraphs. The vertex-cut scheme partitions a graph based on a certain vertex, and replicates and stores the partitioned vertices.

In the early days, studies on static graph partitioning schemes were conducted such that the distributed storage of a static graph that there are no changes in its structures can be performed. As a typical graph partitioning scheme, METIS has proposed an edge-cut-based partitioning scheme to minimize the communication cost between partitioned subgraphs [5]. PowerGraph supports the vertex partitioning scheme that uses the characteristic where vertices and edges presented a power-law distribution [6]. In the social network and Internet of Things (IoT) services, dynamic graph are generated, whereby the vertices and edges constituting the graph change constantly [9]–[12]. For the distributed storage of large dynamic graph data, a partitioning strategy is required for the distributed storage in real time by considering a change of graph data. [13]–[16]. If the throughput of a particular node contained in a cluster is high or the memory space is insufficient, the overall system processing performance can decline, and a partitioning strategy is required to store the subgraphs by considering the load state of the nodes [17]–[20].

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

Therefore, in order to partition dynamic graph data, the load status information of the entire cluster should be managed and determined based on the partitioning policy that considers continuously modified and updated graphs. [21]–[25].

Recently, various schemes have been proposed for the distributed storage of dynamic graphs [26] and [27]. In [21], a vertex-cut-based partitioning scheme was proposed to apply real-time dynamic graphs in the programming model proposed by PowerGraph. In [22], an edge-cut-based partitioning scheme was proposed to minimize the number of cut edges that were generated owing to the addition of new subgraphs. In [23], a partitioning strategy was proposed, in which vertex cutting was performed by considering the processing performance and a memory capacity. However, in [21], the state of nodes existing in a cluster could not be considered since it only considers the ratio of the vertex replication. Furthermore, in [22], although the memory capacity was considered to account for the state of the nodes, the throughput of a node was not considered. In [23], the processing performance and memory capacity were considered when partitioning was performed.

Distributed dynamic graph partitioning schemes consider three main things. The first is throughput, which means the amount of graphs currently being processed by the node. [12] defined the amount of graphs stored for a given period of time. The second is memory utilization. It is a measure of how the nodes utilize the maximum amount of memory that can be managed so that data can be allocated evenly across all nodes. Finally, it is the replication rate. It represents the corresponding proportion to make the vertex split and stored according to the distributed storage characteristics as little as possible. Considering replication rates, communication costs can be reduced. Existing schemes have a problem of considering only some of the three characteristics. Since the three characteristics have very important implications for graph partitioning, new schemes are needed to consider them all. In addition, we need to redefine throughput. The amount of graphs stored over a specific period of time does not actually imply the amount of computation performed by a node to process graph data. Finally, if specific graph patterns or vertices are frequently utilized, storing these data on only one node results in performance imbalances. To prevent performance imbalances, frequently accessed graph vertices should be managed to balance the performance of nodes as much as possible.

In this paper, we propose a novel dynamic graph partitioning scheme to improve the graph query processing performance to handle a large dynamic graph. The proposed scheme presents a partitioning policy that considers the vertex replication ratio, storage utilization, and throughput to improve the system processing performance through the load distribution. The throughput is defined as the number of directly processed queries to facilitate the control of hot data distribution. When a new subgraph that is related with hot data is produced, the new subgraph is regarded as hot data, and by assigning a large weight on the throughput, the load concentration on a particular node is prevented. In this paper, we propose a novel graph partitioning scheme to improve the processing performance of graph queries in a large dynamic graph environment. The contribution of the proposed scheme is as follows.

1) Load management of a node: The proposed scheme manages the load on each node to improve distributed processing performance. In this paper, we define the load of a node as throughput and memory usage. Throughput indicates how many vertices of a query are included in a particular node. Memory utilization refers to the graph size currently stored relative to the maximum storable capacity of each node. The proposed scheme utilizes two values for graph partitioning.
2) Hot data management: Hot data means a particular vertex frequently searched among graphs requested by queries. The proposed scheme records the number of requests per vertex and manages them so that hot data is not driven to a particular node. As with the load of nodes, it is utilized for graph partitioning.
3) Vertex-cut based dynamic graph partitioning on distributed systems: The proposed scheme utilizes a vertex replication index in addition to node load and hot data to perform vertex-cut based dynamic graph partitioning. The vertex replication index means how much the vertex neighbor overlaps with the graph being managed by an existing node for a newly introduced graph to store the graph efficiently. The proposed scheme further considers the vertex replication index because the higher the number of nodes is, the lower the load to store is. It performs efficient dynamic graph partitioning by reflecting these three characteristics.
4) Performance evaluation based on real world graphs: We perform various performance evaluations based on real-world graph data. We show the feasibility and excellence of the proposed scheme compared to the existing vertex-cut and edge-cut based partitioning schemes.

This paper is organized as follows. In Section 2, the characteristics and problems of previous studies mentioned in the introduction are explained; in Section 3, the proposed scheme is described in detail; in Section 4, the superiority of the proposed scheme is demonstrated through performance evaluation. Finally, in Section 5, the conclusions of this study are presented.

II. RELATED WORK

As graph data have become larger, schemes for dividing and storing large amounts of graph data have been studied. [5]–[8], [13], [16]–[18], [20]–[31]. Research on early graph partitioning schemes has mainly been done on static graphs where data does not change in real time. They are divided into vertex-cut based schemes and edge-cut based schemes according to graph partitioning criteria. Graph partitioning schemes aim to minimize the vertices or edges that are split, and the vertices and edges that are split are replicated and stored on each node.

METIS [5] is the most typical static graph partitioning scheme and it is an edge-cut partitioning scheme used to minimize the number of edges cut through partitions. METIS performs a multilevel graph partitioning, consisting of three phases. Phase 1 performs graph compression based on the heavy-edge matching (HEM) algorithm as a graph coarsening phase. Graph compression is compressed into smaller and smaller graphs at multiple levels. After step 1, the original graph is compressed into a graph consisting of a few hundred vertices. Phase 2 performs graph partitioning so that the weights of the vertices equal to the minimum edge-cut can be distributed based on the compressed graph. Graph partitioning is based on a compressed graph, so the computations are very small. Finally, we perform a recovery of the partitioned compressed graph (graph uncoarsening phase). Multithread-based mt-METIS has been further studied to further improve the performance of METIS [28].

PowerGraph [6] is a vertex-cut partitioning scheme that uses the following characteristic: a graph produced in real life shows the distribution shape of the power-law degree. The power-law degree distribution refers to the distribution of graph data in which a small number of vertices constitute the majority of edges, i.e., small numbers of vertices with high degrees exist and the majority of vertices exhibit low degrees. For graphs demonstrating such a distribution, results have shown that the vertex-cut method is more effective than the edge-cut method. PowerGraph performs a balanced p-way vertex-cut for graph partitioning. It allowed p partitions to have uniform edges, with the aim of minimizing the number of vertices that are split. In addition, PowerGraph proposed a GAS (Gather, Apply, Scatter) programming model. The GAS model is a vertex-centric programming model for performing graph algorithms on partitioned graphs.

Since static graph-based partitioning schemes minimize the replication ratio of vertices or edges, the amount of communication is minimized when the graph algorithms are performed. However, the dynamic graph environment that vertices or edges are added or deleted is common in the real world. Because it is very inefficient to perform partitioning whenever graph data changes, studies on dynamic graph-based graph partitioning have been conducted. Dynamic graph partitioning, like static graph partitioning schemes, is divided into vertex cut-based or edge cut-based schemes.

Reference [22] proposed a method for the distributed storage of dynamic graphs in a state where the static graph is partitioned using the edge-cut method. Considering a case of vertices and edges added or deleted in a dynamic graph, a method has been proposed for the distributed storage of a graph. In the case of adding a new vertex, partitioning is performed by considering the communication cost, number of cut edges, and a memory utilization based on the assignment of a new vertex. In the case of an existing vertex being deleted, if a storage utilization of a particular node falls below a particular amount owing to the deletion of a vertex, the vertex stored in a different node is moved to the corresponding node. In the case of a new cut edge being added, if there are more cut edges than the number of internal edges in the edges contained by a particular vertex, the corresponding vertex is moved to a node that has many cut edges. When an internal edge is added, it is simply added. Finally, in the case of deleting an edge, the corresponding edge is simply deleted as well. Recently, an edge-cut based incremental partitioning scheme has been proposed for large-scale RDF dynamic graph processing [29]. It performs partitioning by calculating the minimum edge cut and load balancing index, just like the existing techniques. [29] further deals with graph data that is entered and deleted dynamically by computing partition-specific cohesion and connection indices.

Reference [23] proposed the vertex-cut-based partitioning scheme to partition a dynamic graph in a clustered environment of heterogeneous devices whereby the CPU speed and memory sizes are different. This scheme was performed based on the GAS model proposed by PowerGraph. In a state where a particular number of subgraphs have already been partitioned and stored based on the vertex-cut method, the replication ratio of the vertex, memory utilization, and a processing performance are considered after assigning a new subgraph. The processing performance is calculated by measuring the time for executing each stage of the GAS model, and dividing the size of the subgraph stored in each node by time.

Reference [30] proposed an incremental graph partitioning scheme. Each time a graph change occurs, an increment graph partitioning is performed with the aim of balancing the load, minimum cut-size, and minimizing partition changes. It solved the problem through a heuristic approach because optimizing the three indices is an NP-complete problem. It transforms existing partitioning techniques (vertex-cut, edge-cut) into an incremental method by exploiting the proposed formula and demonstrates validity through experiments. Reference [31] performs hybrid (vertex-cut and edge-cut) partitioning by providing partition transparency similar to reference [30]. It defines the cost-model for partitioning and performs application-driven partitioning that fits the lowest cost through learning.

III. THE PROPOSED DYNAMIC PARTITIONING SCHEME

A. SYSTEM STRUCTURE

In recent years, owing to the use of dynamic graphs, in which vertices and edges composing the graph change constantly, studies are required for the distributed storage of graphs that change in real time. When a new vertex or edge is changed, a node that should store the subgraph has to be determined. In this paper, we propose a distributed storage management scheme of dynamic graphs to improve the query-processing performance of graphs when a large graph has been partitioned based on the vertex-cut method. The proposed scheme aims to improve the processing performance of the entire system by solving the problem owing to deleted subgraphs

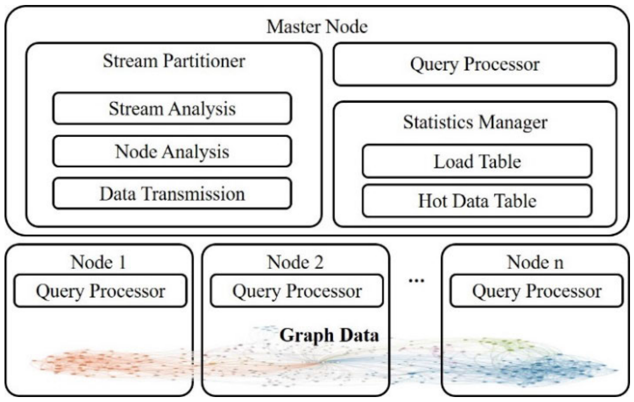


FIGURE 1. **Overall processing structure of the proposed scheme.**

and the assignment policy of added subgraphs. An added subgraph is assigned by considering the storage utilization and throughput of each node composing a cluster. More subgraphs are assigned to a node that has a low storage utilization and low throughput. Here, hot data, which are frequently used in queries, are considered as an additional element for considering the throughput. Depending on whether an added subgraph is hot data, the partitioning policy is changed such that the hot data can be assigned to a node that has a low throughput.

Fig. 1 shows the processing procedure of the proposed scheme. The proposed scheme assumes that an existing graph is distributed through the vertex-cut partitioning scheme in the cluster environment. The system configuration is composed of one master node and four slave nodes. The master node has an Intel(R) Core(TM) i5-6400 CPU @ 2.70 GHz CPU with 32 GB of memory. The slave node has an Intel(R) Core(TM) i7-6700 CPU @ 3.40 GHz CPU and 16 GB of memory. Each server has a 1TB secondary memory. The statistical information management module uses the load table to store the information of throughput and storage utilization for each node according to query processing. Furthermore, subgraph information frequently requested by queries is stored in the hot data table. When a dynamic graph is produced, the stream partitioner accesses the hot data table first and analyzes the existence/non-existence of hot data of the dynamic graph. If a newly generated subgraph is linked with a vertex frequently requested in queries, it is determined to be hot data because there is a high possibility that it will be used together when queries are performed in the future. Once the existence of hot data is determined, the state of each node is verified by referencing the load table. After calculating the cost score for each node by considering the state of each node, the node with the highest score is determined. Finally, the new dynamic graph is sent to the determined node.

B. LOAD MANAGEMENT

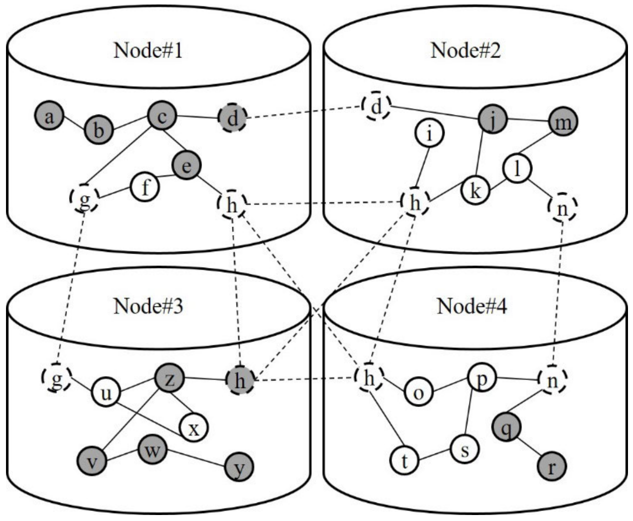
To increase the system performance by ensuring appropriate load distribution, a dynamic graph should be partitioned by considering the state of each node composing the cluster. Hence, the load of each node in the cluster is monitored via load management, and the collected information is used for dynamic graph partitioning. In the proposed scheme, the statistical information management module of the master node manages the storage utilization and throughput of each node. If a newly added subgraph is stored at a node that already has a large number of graphs stored or at a node that has a larger throughput than the other nodes, the performance of the overall system may decline since the load will be concentrated in a particular node. In order to prevent such a decline in performance, the load information is considered when performing dynamic graph partitioning.

When a query is requested to seek a particular subgraph pattern in the master node, the query-processing module determines a node that will perform the search. Subsequently, a slave node that becomes the search target transmits the query-processing result to the query-processing module of the master node and simultaneously sends the results to the statistical information module: the number and ID of the vertices, which are search targets contained in each node among the requested patterns, and the memory utilization rate of the corresponding node. The number of vertices that become the search targets and the memory utilization rate are maintained in the load table. Table 1 shows an example of a load table. Throughput indicates how many query vertices are included in the graph data managed by a particular node. [12] defined the amount of graphs stored for a given period of time. Therefore, the node that handles more query vertices increases throughput, but the node also increases its load. Memory utilization is a measure of how the node utilizes the maximum amount of memory that can be managed so that data can be allocated evenly across all nodes. Memory and storage utilizations mean the same thing and are calculated on a per-node basis as throughput. It means the size of the graph data that each node stores divided by the maximum memory that the node can store. Memory utilization, like throughput, has a higher value for a node storing more graphs.

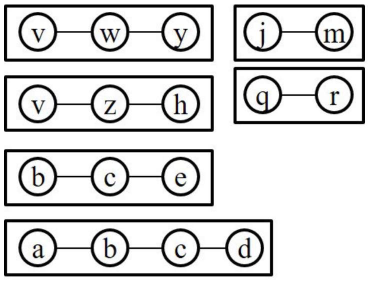
TABLE 1. **An example of load table.**

Node	Throughput	Memory Utilization Rate
1	55%	64%
2	16%	56%
3	55%	62%
4	16%	70%

Fig. 2 shows the process of generating values of throughput and memory utilization rate. Fig. 2 (b) shows a subgraph search query and Fig. 2 (a) shows the subgraph contained in a query among the graphs stored in each node. The number of subgraphs contained in the query among the graphs stored in each node is regarded as the throughput. The nodes exhibit the values of 5, 2, 5, and 2, and to calculate the throughput based on the partitioning policy, they are calculated as values of 5/9, 2/9, 5/12, and 2/12.

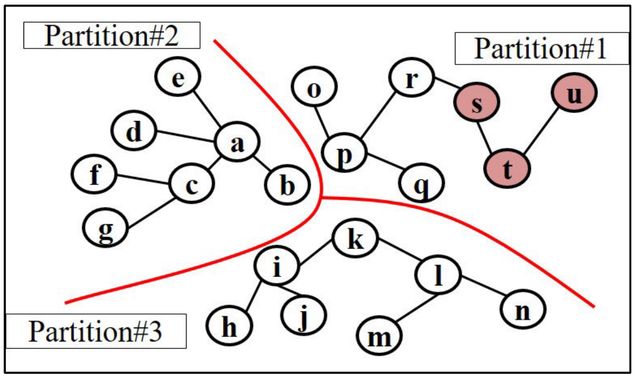


(a) Vertices contained in the subgraph query and data stored in each node

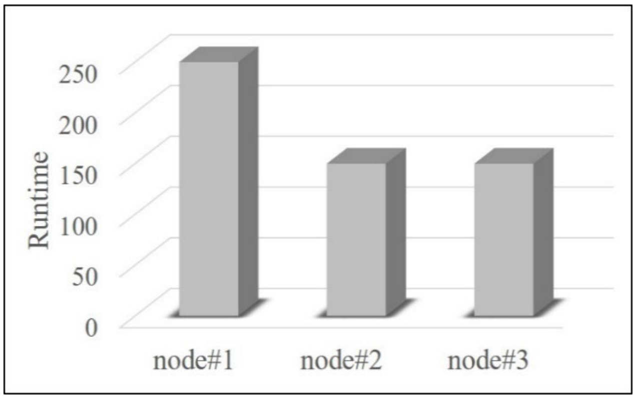


(b) An example of subgraph query

FIGURE 2. Process of generating throughput value.



(a) Generation of hot data



(b) Runtime difference between nodes due to hot data

FIGURE 3. System performance decline due to hot data.

TABLE 2. Example of hot data table.

Rank	Vertex ID	The number of requests
1	b	4
2	a	1
3	c	1

C. HOT DATA MANAGEMENT

The hot data show the subgraphs that are frequently used in the query processing. Suppose two nodes store the same number of subgraphs. Subsequently, the node that contains the hot data will exhibit a relatively higher throughput, and this will induce a performance decline of the entire system. Fig. 3 illustrates the performance decline of the entire system caused by the generation of hot data. As shown in Fig. 3 (a), the entire graph is partitioned and stored as three subgraphs, and subgraph #1 contains s, t, and u vertices, which are hot data. By assuming each subgraph is processed at one slave node, Fig. 3 (b) shows the runtime for each partition. The three nodes process the same amount of data; however, in the case of slave node #1 containing subgraph #1, the processing time is increased, and the other nodes wait for node #1 after processing has completed.

In the proposed scheme, when a newly generated subgraph is connected with the existing hot data, it is regarded as hot data and a higher weight is assigned to the throughput when partitioned. When a subgraph frequently used in the query process is managed using the hot data table and partitioning is performed, whether the corresponding graph is hot data is determined based on the hot data information. The query-processing module of the master node manages the hot data table by sequencing the number of vertices appearing in the subgraph search queries generated randomly, as shown in Table 2. Fig. 4 shows an example of the subgraph query. As shown in Table 2, the request frequency is recorded for each vertex with respect to the subgraphs used in the queries, as shown in the figure.

D. DYNAMIC GRAPH PARTITIONING

The proposed scheme considers the storage utilization, throughput, and vertex replication ratio to determine the node to be used to store the changing subgraph. When many subgraphs are stored at a node, the number of computation operations to be processed increases; consequently, the overall processing performance declines. Therefore, via the throughput, the sizes of the distributed and stored subgraphs are made similar. When the number of processing requests increase for the subgraphs in a particular node, a problem may occur in that the loads of the nodes become

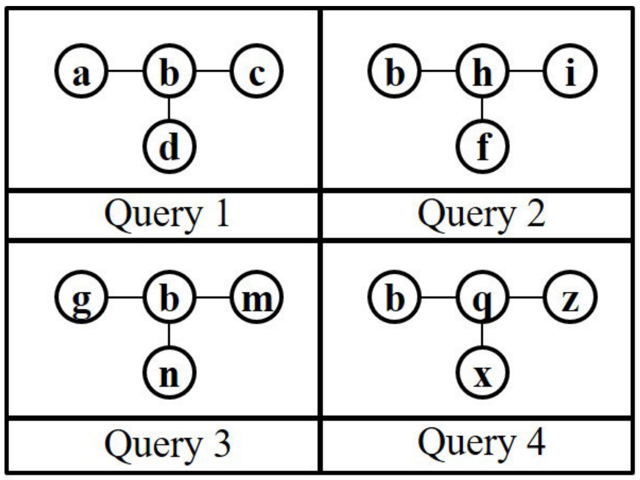


FIGURE 4. Examples of a subgraph query.

concentrated in one node. Therefore, the loads are distributed evenly using the throughput. In a distributed environment, since the communication for join operations is performed through the partitioned vertices, the communication cost may increase if computations occur at the vertices. Therefore, since the communication is performed based on the vertex replication ratio through the partitioned vertices, as the vertex replication ratio increases, the communication cost increases. Therefore, it is necessary to minimize the vertex replication ratio.

RS_i (Replication score) is used to minimize vertex replication by using the vertex replication ratio of node i. The vertex replication ratio, which is an important factor in the vertex-partitioning scheme, shows the number of nodes in which one vertex is partitioned and stored. When the vertex replication ratio is high, it implies that one vertex is stored in multiple nodes; further, when an algorithm such as PageRank is executed to calculate a particular value by regarding a vertex as a unit, this incurs a communication cost between nodes to synchronize the values possessed by vertices. Therefore, the proposed scheme calculates RS_i by (1) to minimize the vertex replication ratio when an edge e is added. To reduce the vertex replication ratio by assigning a large score to a node that has many neighboring edges at the corresponding node. P_i refers to the node i that stores the partitioned subgraphs, and $N(e)$ refers to the group of neighbors of an edge e. The value of $max_j|P_j \cap N(e)|$ is divided by the largest value of the node to normalize the corresponding element with a value between 0 and 1. Fig. 5 shows the concept of $N(e)$. e_{kl} implies the edge that connects vertex k and vertex l, and the neighboring edges of e_{kl} become e_{ik}, e_{lm}, and e_{ln}, as shown in the Fig. 5.

$$RS_i = \frac{|P_i \cap N(e)|}{max_j|P_j \cap N(e)|} \tag{1}$$

US_i (Storage utilization score) is a factor for considering the storage utilization by node i. If a large amount of data is stored in a particular node, considerable computations

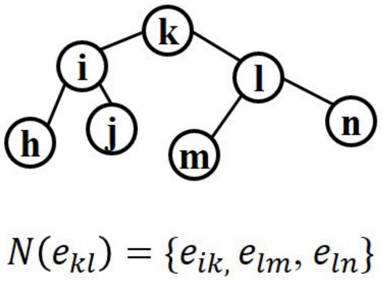


FIGURE 5. Example of $N(e)$.

must be performed to process the queries. To allocate added subgraphs to nodes of relatively low storage utilization to prevent this scenario, the storage utilization US_i is calculated using (2). $|P_i|$ is the size of the graph stored at node i. C_i implies the total memory size of the node i; consequently, US_i shows the memory utilization of the node i. Regarding the US_i value, a node with smaller storage utilization exhibits a higher score.

$$US_i = 1 - \frac{|P_i|}{C_i} \tag{2}$$

CS_i (Computation size score) is a factor for considering the throughput. When subgraphs are constantly added to a node that has a large throughput, the throughput will increase at a particular node only, thereby occurring a workload imbalance. In order to prevent system performance decline owing to load imbalance, the computation size score CS_i is calculated using (3). A high score is given to the CS_i of the node a low throughput during query processing. S_i refers to the number of vertices contained in the node i.

$$CS_i = 1 - \frac{S_i}{\sum_{j \in \{1 \ldots k\}, j \neq i} S_j} \tag{3}$$

Suppose k nodes compose a cluster. Subsequently, the cost score is calculated for each node to determine the nodes that will store the new subgraphs. The cost score of node i is calculated using (4). RS_i is the vertex replication ratio, US_i is the storage utilization, CS_i is the throughput, and $\alpha + \beta + \gamma = 1$. The values are calculated for every node that composes the cluster, and a subgraph is assigned to a node that has the highest TS_i value. If deletion occurs constantly at a particular node, load imbalance will occur from the perspective of storage utilization. Accordingly, the deviation in storage utilization may increase, and even if a small amount of hot data is deleted, imbalance may occur from the perspective of throughput. For the imbalance problem occurring from the perspective of storage utilization, a threshold value is assigned to each node, and when the storage utilization of a particular node falls below a threshold value, β is set to a high value such that the input data will be assigned first to a node of low storage utilization. γ is a weight assigned depending on whether an added subgraph is hot data. If a newly added subgraph is determined as hot data, a relatively large value is

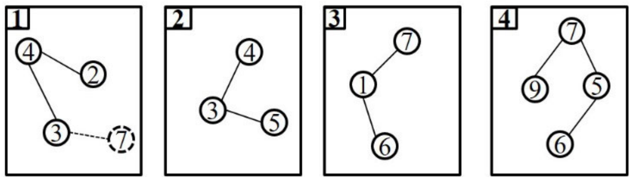
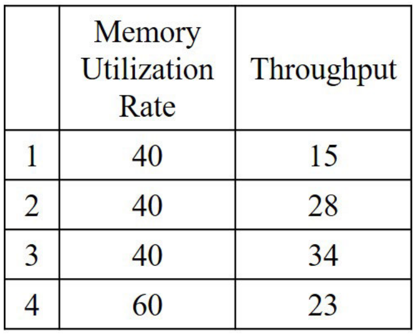
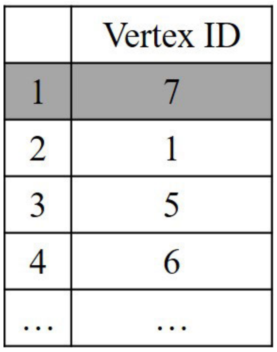


FIGURE 6. Examples of data storage states for each node.



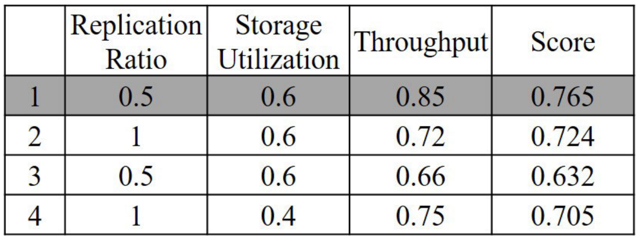
	Memory Utilization Rate	Throughput
1	40	15
2	40	28
3	40	34
4	60	23

(a) Load management table



	Vertex ID
1	7
2	1
3	5
4	6
...	...

(b) Hot data table



	Replication Ratio	Storage Utilization	Throughput	Score
1	0.5	0.6	0.85	0.765
2	1	0.6	0.72	0.724
3	0.5	0.6	0.66	0.632
4	1	0.4	0.75	0.705

(c) Scores in the case of inserting e_{37}

FIGURE 7. Examples of dynamic graph partitioning.

assigned to γ such that the hot data will be assigned to a node of low throughput.

$$TS_i = \alpha RS_i + \beta US_i + \gamma CS_i \tag{4}$$

If a subgraph is generated, the hot data table is verified to determine whether the corresponding dynamic graph corresponds to the hot data. Depending on the existence/non-existence of hot data, the weight used for calculating the partitioning criterion is adjusted. To calculate the partitioning criterion, the information from the load table is used, and a subgraph is stored at a node that has the highest score with respect to the cost score calculated for each node. Fig. 6 and 7 show the dynamic graph partitioning processes with respect to the hot data. Assume that the graph nodes are stored at the nodes of four slaves (Fig. 6 (a), (b), (c), and (d)), to which the numbers 1 through 4 are assigned. When an edge connecting vertex #3 and vertex #7 is inserted, whether a vertex that belongs to the hot data exists among the two vertices is determined. Since vertex #7 is a vertex that corresponds to ranking #1 in the hot data table, it is determined to be hot data. Since the added data are determined as hot data, the weight γ is adjusted. As shown in Fig. 7 (c), as the storage utilization decreases and the throughput decreases, the score increases. In Fig. 7 (c), the value of the last column shows the final calculated score for each node. Because of the generation of hot data, the weight of the throughput was increased, and accordingly, node #1 exhibits the highest value; therefore, the new subgraph is stored at node #1. However, this is merely since no large difference is indicated in the score for storage utilization or replication ratio. Further, it is noteworthy that even when hot data are produced, the throughput is not considered.

TABLE 3. Performance evaluation environment.

Feature	CPU	RAM
Master	Intel(R) Core(TM) i5-6400 CPU @ 2.70 GHz	32GB
Slaves(4 ea)	Intel(R) Core(TM) i7-6700 CPU @ 3.40 GHz	16GB

TABLE 4. Dataset size.

Name	Number of vertices	Number of edges
AS-CAIDA	1,036,474	24,313,233
US-patent	3,774,768	16,518,948

IV. PERFORMANCE EVALUATION

To demonstrate the excellence of the proposed scheme, a performance evaluation was conducted. Table 3 shows the performance evaluation environment. The system configuration consists of one master node and four slave nodes for performance evaluations. The master node has an Intel(R) Core(TM) i5-6400 CPU @ 2.70 GHz CPU with 32 GB of memory. The slave node has an Intel(R) Core(TM) i7-6700 CPU @ 3.40 GHz CPU and 16 GB of memory. Performance evaluation was conducted using Java on Linux. The existing schemes such as [22], [23] were chosen for comparison. [22] was chosen to demonstrate the performance difference between the partitioning schemes of the edge-cut method and vertex cut method, and [23] was chosen to demonstrate the excellence of the proposed scheme using a vertex-cut-based existing scheme that is identical to the proposed scheme.

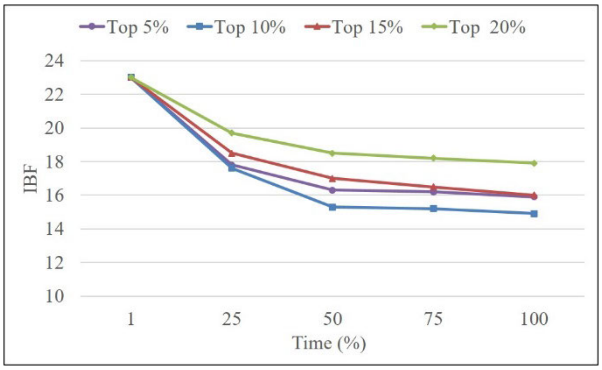


FIGURE 8. Changes in *IBF* value according to generation of dynamic graphs.

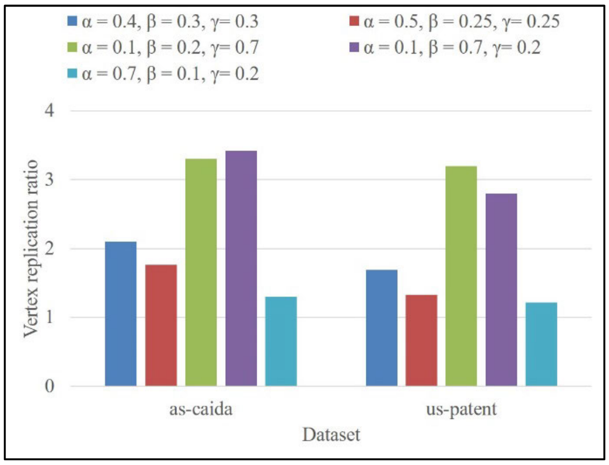


FIGURE 9. Vertex replication ratio according to the weights.

Two datasets were used, as shown in Table 4. The two datasets were provided by SNAP [32]. To generate streaming data for insertion and deletion, the graph data of AS-CAIDA [33] and US-patent citation graph [34] were used. The performance evaluation compares the proposed scheme with the existing schemes in terms of vertex replication ratio, communication cost, and query processing time. The vertex replication ratio is a metric that determines how many vertices are replicated to other partitions based on vertex-cut based graph partitioning. The lower the value of the metric is, the better it is to be partitioned without being replicated. The communication cost means the number of communications between partitions. The number of communications refers to the number of times the data was exchanged between the partitioned vertices to calculate the exact value while the graph algorithm was operating. In general, the higher the vertex replication ratio value is, the greater the probability that the communication cost will increase is. Query processing time means the time to process graph algorithms such as PageRank or fining subgraphs.

The hot data are frequently used data, and the proposed scheme ranks and manages the frequency of the vertices' ID appearing in the queries. However, a criterion is required to determine which vertices are to be regarded as hot data based on the rankings of vertices arranged in the descending order. To establish the criterion, an index was defined first to verify the level of load imbalance. Equation (5) is the equation for calculating the level of load imbalance. *IBF* is the abbreviation for imbalance factor and shows the imbalance level. *MU* is the abbreviation for memory utilization and shows the memory utilization rate. *CS* is the abbreviation for computation size and shows the throughput while having the same value as CS_i. Consequently, the *IBF* value shows the deviations in memory utilization rate and throughput.

$$IBF = \sqrt{MU * CS} \quad (5)$$

In the experimental evaluation, dynamic graphs are added by changing the criterion of the hot data to the top 5%, 10%, 15%, and 20%. Simultaneously, while performing the subgraph search queries, the change in *IBF* value is measured. Figure 8 shows the changes in *IBF* value according to the generation of dynamic graphs. When the top 10% of the data are classified as hot data, the *IBF* value is the lowest. By examining the dataset, it was confirmed that only ~10,000 vertices among ~1,000,000 vertices were the vertices of high degree that had more than 150 edges. Based on the experimental result, the top 10% will be used as the criterion hot data in future experimental evaluations.

To determine the weights α, β, γ of the equation used for dynamic graph partitioning, three experiments were conducted. In the first experiment, the vertex replication ratio is investigated, which is an important factor in the vertex-cut-based partitioning scheme. This is because depending on the number of partitioned vertices, the communication cost for synchronization between the vertices is incurred, which will affect the processing performance. In order investigate the performance from the replication ratio of vertices, the replication ratio of the vertices was verified for the graphs produced after all the dynamic graphs have been generated. Fig. 9 shows the replication ratio according to the weight. When the weight for the replication ratio of the vertex was set to the highest, the lowest replication ratio was indicated.

When the vertex replication ratio is high, it implies that many nodes exist whereby the vertices must be synchronized. Accordingly, a relatively high communication cost will be incurred. To confirm this, the PageRank algorithm was performed, and based on the partitioned vertices, the numbers of sending/receiving data were recorded; subsequently, by multiplying the size of the real-number-type data, which was a delivered factor, the communication cost was calculated. Fig. 10 shows the communication cost result between partitions according to the vertex replication ratio. When $\alpha = 0.7$, $\beta = 0.1$, $\gamma = 0.2$ whereby the vertex replication ratio was the lowest, the lowest communication cost was incurred. When the two data sets were compared, it was found that the US-patent graph data incurred a relatively higher communication cost. Since the data were sent and received between the vertices owing to the characteristics of the PageRank algorithm, more communication cost was incurred in the

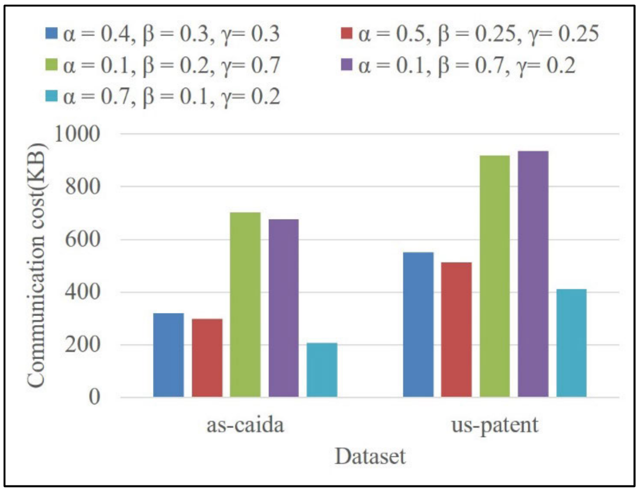


FIGURE 10. Communication cost between partitions according to the weights.

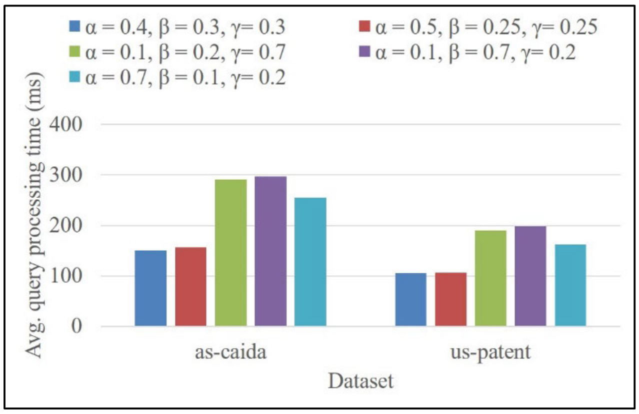


FIGURE 11. Subgraph query processing time according to the weights.

US-patent data, in which the number of vertices was approximately twice higher.

Not all queries for the graphs require computation targeting the entire vertices or communication between vertices. Therefore, it is not appropriate to set the weight based on the communication cost and vertex replication ratio only. In order to investigate the performance in computation for subgraph queries that access partial data, the average processing time of subgraph search queries was measured. Fig. 11 shows the subgraph query processing time according to the weight. A faster processing speed was shown when the weights were assigned evenly rather than setting a high weight for a particular factor. When compared with the two earlier experimental results, the case of $\alpha = 0.5$, $\beta = 0.25$, $\gamma = 0.25$ demonstrated a higher performance than the case of $\alpha = 0.4$, $\beta = 0.3$, $\gamma = 0.3$ in the aspects of communication cost and vertex replication ratio, and a similar performance was shown in the subgraph query processing. Based on this result, the weights of $\alpha = 0.5$, $\beta = 0.25$, $\gamma = 0.25$ were used in the comparative evaluation with the existing schemes.

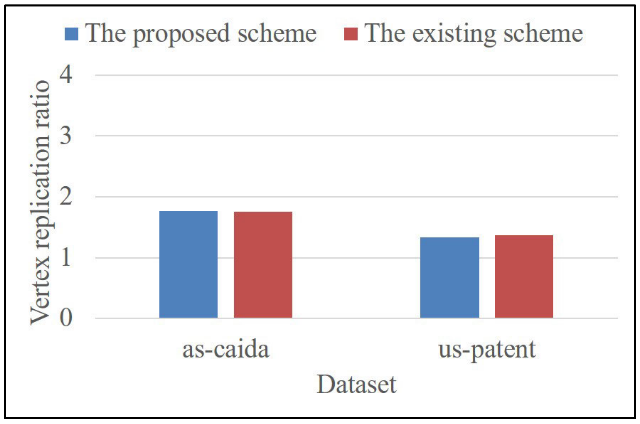


FIGURE 12. Evaluation for vertex replication ratio.

Based on the criterion selected in the in-house evaluation, a comparative evaluation with the existing schemes was conducted. The evaluation was performed for the vertex-cut-based existing scheme and the vertex replication ratio; based on this, the runtime of the PageRank algorithm was evaluated. Although the runtime was comparatively evaluated with the edge-cut-based existing scheme, it was excluded from the results since the vertex replication ratio and the number of cut edges could not be compared. Fig. 12 shows the evaluation results for the vertex-cut-based existing scheme and vertex replication ratio. The y-axis is the vertex replication ratio, and shows the average value calculated by recording the number of nodes where each vertex has been partitioned. The storage utilization and processing performances exhibit values between 0 and 1, whereas the vertex replication ratio exhibits a large number depending on case. As shown in Fig. 12, however, when compared with the proposed scheme, the vertex replication ratio demonstrated a similar performance. This was since in the proposed scheme, the weight α of the vertex replication ratio was set higher compared to the storage utilization and throughput.

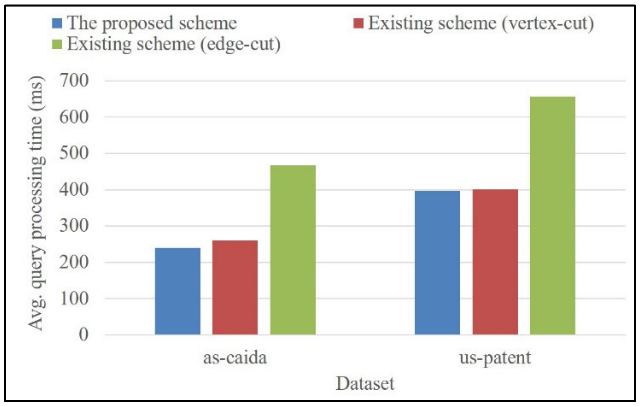


FIGURE 13. Evaluation of PageRank algorithm.

According to the comparative evaluation result of the vertex replication ratio, the evaluation for the runtime of the PageRank algorithm was conducted. Fig. 13 shows the comparative evaluation result for the PageRank algorithm runtime. As shown in the figure, when compared with the vertex-cut-based scheme, a similar or slightly improved performance was shown. When compared with the edge-cut-based scheme, the performance improvement was approximately twice higher, and this was consistent with that proven by PowerGraph. The real-life graphs of the AS-CAIDA and US-patent datasets demonstrated the power-law degree distribution; further, the vertex-cut-based partitioning scheme demonstrated a more improved performance. Furthermore, the runtime of the US-patent graph was relatively long since it contained approximately two times more vertices compared with the AS-CAIDA graph.

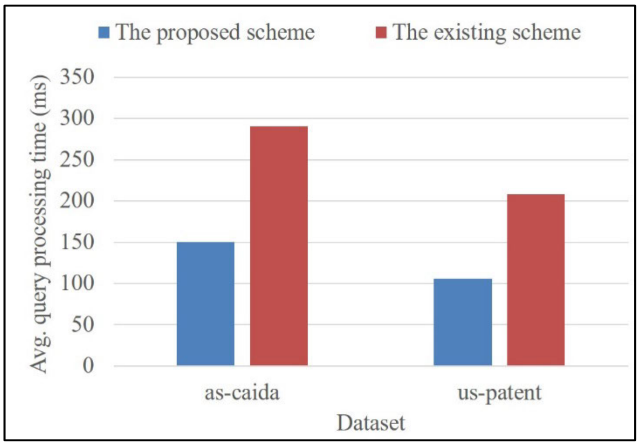


FIGURE 14. Subgraph query evaluation.

A comparative evaluation was performed for the processing time of the subgraph search query that requires access to partial vertices. The evaluation was performed for two datasets, AS-CAIDA and US-patent, and the comparative evaluation was conducted with the vertex-cut-based existing scheme. The AS-CAIDA graphs were based on real graph data stored by time; based on those, changes were extracted and used for the dynamic graphs. However, the US-patent graphs were based on large-scale static graph data, and by deleting a particular portion from the file, the size was reduced by 50%; subsequently, by adding the deleted portion to the dynamic graphs, the experimental evaluation was performed. 1%, 2%, 5%, and 10% were used as the size of graph data deleted for the comparative evaluation using the ratio of added dynamic graphs. Fig. 14 shows the result for the average processing time of the subgraph search queries using the AS-CAIDA graph data. As shown in the result, the performance was improved approximately twice compared with the existing scheme. In contrast to the case of PageRank, a shorter processing time was shown with the US-patent dataset. The reason can be determined by merely observing the size of the datasets. Compared with the AS-CAIDA, because the number of edges is smaller whereas the number of vertices is approximately larger, a shorter runtime is shown in the subgraph queries that performed search based on edges.

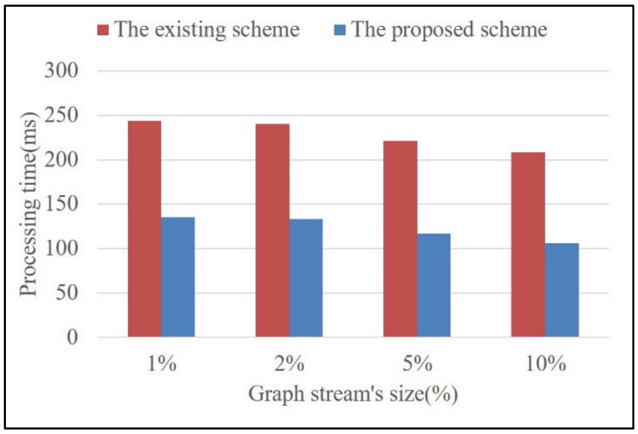


FIGURE 15. Subgraph query processing time according to the size of a dynamic graph.

Fig. 15 shows the query processing time when 1%, 2%, 5%, and 10% of the entire data were used for the size of the dynamic graphs in the experiment using the US-patent graphs only. In the figure, the shortest processing time was shown when the 10% size was used. No significant difference is indicated between the sizes of 1% and 2%; however, when the size was changed to 5% and 10%, some differences occurred. The reason was since whenever a query was processed, time was incurred to generate statistical information, and when dynamic graphs were added in small units, the number of computations increased.

A summary of the performance evaluation results is as follows. We performed our own performance evaluation for hot data selection. This resulted in the lowest performance imbalance (*IBF*) between nodes. In addition, various weight performance evaluations have resulted in vertex replication ratios and low communication costs, which show the best PageRank performance when $\alpha = 0.7$, $\beta = 0.1$, and $\gamma = 0.2$ The vertex replication ratio refers to the number of vertices replicated between nodes. It is closely related to communication cost testing because the lower the vertex replication ratio is, the lower the communication volume is. The weights obtained by self-performance evaluation worked independently of the dataset. However, because the US-patent dataset has a relatively large number of vertices, it can be seen that overall communication costs tend to be high. Since PageRank performs operations at all vertices, the replication ratio has a high weight. However, we could see that Subgraph queries tend to approach only some data and therefore need to be weighted differently according to the query. Even in the processing of subgraph queries, even though the replication ratio has a high weight, giving the three weights (α, β, and γ) equally shows better performance.

After obtaining optimal weights through self-performance evaluation results, we perform comparisons with existing schemes and demonstrate the superiority of the proposed scheme by showing high performance in both graph queries (PageRank, Subgraph). In addition, we have shown the validity of the proposed scheme by comparing it with both vertex-cut and edge-cut based partitioning schemes. In this paper, real-world graph data show a Power-law degree distribution, further enhancing the reliability of the proposed scheme by presenting performance evaluation results consistent with the conclusion [6], where vertex-cut based graph partitioning schemes show better performance.

V. CONCLUSION

In this paper, we proposed a novel dynamic graph partitioning scheme considering the load of a node and hot data to handle a large dynamic graph. The proposed scheme defined the load of a node as throughput and memory usage. We also managed hot data that means a particular vertex frequently searched among partitioned graphs to prevent the concentration of hot data on a particular node. Finally, we perform incremental graph partitioning on a dynamic graph by considering the node load, hot data, and the vertex replication ratio. We showed the superiority of the proposed scheme by conducting various performance evaluation in two graph queries such as PageRank and Subgraph. In addition, we have shown the validity of the proposed scheme by comparing it with both vertex-cut and edge-cut based partitioning schemes on real-world graph data. In this paper, performance evaluations were conducted only for PageRank and subgraph queries. In order to show the superiority of the proposed scheme in various graph applications, we will perform additional performance evaluations on various graph query and analysis algorithms such as DFS, BFS, SSSP, Strongly Connected Component, Cycle Detection, and Graph Coloring. In addition, we will further study graph merging and repartitioning policies considering the failure situation of nodes and load balancing.

REFERENCES

[1] G. Karypis and V. Kumar, "A parallel algorithm for multilevel graph partitioning and sparse matrix ordering," *J. Parallel Distrib. Comput.*, vol. 48, no. 1, pp. 71–95, Jan. 1998.

[2] K. Ammar and M. T. Özsu, "Experimental analysis of distributed graph systems," *Proc. VLDB Endowment*, vol. 11, no. 10, pp. 1151–1164, Jun. 2018.

[3] M. Onizuka, T. Fujimori, and H. Shiokawa, "Graph partitioning for distributed graph processing," *Data Sci. Eng.*, vol. 2, no. 1, pp. 94–105, 2017.

[4] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan, "One trillion edges: Graph processing at Facebook-scale," *Proc. VLDB Endowment*, vol. 8, no. 12, pp. 1804–1815, 2015.

[5] G. Karypis and V. Kumar, "METIS–unstructured graph partitioning and sparse matrix ordering system, version 2.0," Univ. Minnesota, Minneapolis, MN, USA, Tech. Rep., 1995. [Online]. Available: http://www.cs.umn.edu/~metis

[6] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," Presented as the 10th USENIX Symp. Operating Syst. Design Implement. (OSDI), 2012, pp. 17–30.

[7] H. P. Sajjad, A. H. Payberah, F. Rahimian, V. Vlassov, and S. Haridi, "Boosting vertex-cut partitioning for streaming graphs," in *Proc. IEEE Int. Congr. Big Data (BigData Congr.)*, Jun. 2016, pp. 1–8.

[8] D. Nicoara, S. Kamali, K. Daudjee, and L. Chen, "Hermes: Dynamic partitioning for distributed social network graph databases," in *Proc. EDBT*, 2015, pp. 25–36.

[9] P. Liu, L. Zhang, and J. A. Gulla, "Real-time social recommendation based on graph embedding and temporal context," *Int. J. Hum.-Comput. Stud.*, vol. 121, pp. 58–72, Jan. 2019.

[10] H. Yin, D. Guo, K. Wang, Z. Jiang, Y. Lyu, and J. Xing, "Hyperconnected network: A decentralized trusted computing and networking paradigm," *IEEE Netw.*, vol. 32, no. 1, pp. 112–117, Jan. 2018.

[11] E. Alirezaei, S. Parsa, and Z. Vahedi, "Predictive analytics of hyperconnected collaborative network," *Int. J. Bus. Data Commun. Netw.*, vol. 15, no. 1, pp. 17–33, Jan. 2019.

[12] I. Ullah, M. Sohail Khan, and D. Kim, "IoT services and virtual objects management in hyperconnected things network," *Mobile Inf. Syst.*, vol. 2018, pp. 1–19, Jun. 2018.

[13] M. A. K. Patwary, S. Garg, and B. Kang, "Window-based streaming graph partitioning algorithm," in *Proc. Australas. Comput. Sci. Week Multiconference*, Jan. 2019, pp. 1–10.

[14] S. Aridhi, A. Montresor, and Y. Velegrakis, "BLADYG: A graph processing framework for large dynamic graphs," *Big Data Res.*, vol. 9, pp. 9–17, Sep. 2017.

[15] M. Junghanns, A. Petermann, M. Neumann, and E. Rahm, "Management and analysis of big graph data: Current systems and open challenges," in *Handbook Big Data Technologies Anonymous*. New York, NY, USA: Springer, 2017, pp. 457–505.

[16] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "GraphX: Graph processing in a distributed dataflow framework," in *Proc. 11th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2014, pp. 599–613.

[17] A. G. Labouseur, J. Birnbaum, P. W. Olsen, S. R. Spillane, J. Vijayan, J.-H. Hwang, and W.-S. Han, "The G* graph database: Efficiently managing large distributed dynamic graphs," *Distrib. Parallel Databases*, vol. 33, no. 4, pp. 479–514, Dec. 2015.

[18] C. Vlassopoulos, I. Kontopoulos, M. Apostolou, A. Artikis, and D. Vogiatzis, "Dynamic graph management for streaming social media analytics," in *Proc. 10th ACM Int. Conf. Distrib. Event-based Syst.*, Jun. 2016, pp. 382–385.

[19] S. Ma, J. Li, C. Hu, X. Lin, and J. Huai, "Big graph search: Challenges and techniques," *Frontiers Comput. Sci.*, vol. 10, no. 3, pp. 387–398, Jun. 2016.

[20] M. Li, H. Cui, C. Zhou, and S. Xu, "GAP: Genetic algorithm based large-scale graph partition in heterogeneous cluster," *IEEE Access*, vol. 8, pp. 144197–144204, 2020.

[21] C. Xie, W.-J. Li, and Z. Zhang, "S-PowerGraph: Streaming graph partitioning for natural graphs by vertex-cut," 2015, *arXiv:1511.02586*. [Online]. Available: http://arxiv.org/abs/1511.02586

[22] A. Abdolrashidi and L. Ramaswamy, "Continual and cost-effective partitioning of dynamic graphs for optimizing big graph processing systems," in *Proc. IEEE Int. Congr. Big Data (BigData Congr.)*, Jun. 2016, pp. 18–25.

[23] Y. Cao and R. Rao, "A streaming graph partitioning approach on imbalance cluster," in *Proc. 18th Int. Conf. Adv. Commun. Technol. (ICACT)*, Jan. 2016, pp. 360–364.

[24] W. Zhang, Y. Chen, and D. Dai, "AKIN: A streaming graph partitioning algorithm for distributed graph storage systems," in *Proc. 18th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2018, pp. 183–192.

[25] L. Durbeck and P. Athanas, "Incremental streaming graph partitioning," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2020, pp. 1–8.

[26] Z. Abbas, V. Kalavri, P. Carbone, and V. Vlassov, "Streaming graph partitioning: An experimental study," *Proc. VLDB Endowment*, vol. 11, no. 11, pp. 1590–1603, 2018.

[27] S. Verma, L. M. Leslie, Y. Shin, and I. Gupta, "An experimental comparison of partitioning strategies in distributed graph processing," *Proc. VLDB Endowment*, vol. 10, no. 5, pp. 493–504, Jan. 2017.

[28] D. LaSalle, M. M. A. Patwary, N. Satish, N. Sundaram, P. Dubey, and G. Karypis, "Improving graph partitioning for modern graphs and architectures," in *Proc. 5th Workshop Irregular Appl., Archit. Algorithms*, 2015, pp. 1–4.

[29] Y. Leng, H. Wang, and F. Lu, "Artificial intelligence knowledge graph for dynamic networks: An incremental partition algorithm," *IEEE Access*, vol. 8, pp. 63434–63442, 2020.

[30] W. Fan, M. Liu, C. Tian, R. Xu, and J. Zhou, "Incrementalization of graph partitioning algorithms," *Proc. VLDB Endowment*, vol. 13, no. 8, pp. 1261–1274, Apr. 2020.

[31] W. Fan, R. Jin, M. Liu, P. Lu, X. Luo, R. Xu, Q. Yin, W. Yu, and J. Zhou, "Application driven graph partitioning," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 1765–1779.

[32] *Stanford Network Analysis Project*. Accessed: Jan. 19, 2021. [Online]. Available: https://snap.stanford.edu/index.html

[33] *CAIDA AS Relationships Datasets*. Accessed: Jan. 19, 2021. [Online]. Available: https://snap.stanford.edu/data/as-caida.tar.gz

[34] *Patent Citation Network*. Accessed: Jan. 19, 2021. [Online]. Available: https://snap.stanford.edu/data/cit-Patents.txt.gz

DOJIN CHOI received the B.S. and M.S. degrees in computer engineering from the Korea National University of Transportation, South Korea, in 2014 and 2016, respectively, and the Ph.D. degree in information and communication engineering from Chungbuk National University, South Korea, in 2020. His research interests include location based service, big data processing, continuous query processing, and distributed computing.

JINSU HAN received the B.S. and M.S. degrees in information and communication engineering from Chungbuk National University, South Korea, in 2016 and 2018, respectively. His research interests include database systems, graph stream processing social networks, and big data.

JONGTAE LIM received the B.S., M.S., and Ph.D. degrees in information and communication engineering from Chungbuk National University, South Korea, in 2009, 2011, and 2015, respectively. He is currently a Research Professor with Chungbuk National University. His research interests include moving object database, spatial database, location-based services, P2P networks, and big data.

JINSUK HAN received the master's degree in information and communication engineering from Chungbuk National University, in 2013. He is currently working as a Software Developer and CEO of Ubitobe Company Ltd. His research interests include graph data analysis, the IoT, database systems, smart factory, and big data.

KYOUNGSOO BOK received the B.S. degree in mathematics, and the M.S. and Ph.D. degrees in information and communication engineering from Chungbuk National University, South Korea, in 1998, 2000, and 2005, respectively. He is currently an Assistant Professor of software convergence technology with Wonkwang University, South Korea. His research interests include database systems, location based service, mobile ad-hoc networks, big data processing, and social network service.

JAESOO YOO received the B.S. degree in computer engineering from Chungbuk National University, South Korea, in 1989, and the M.S. and Ph.D. degrees in computer science from the Korea Advanced Institute of Science and Technology, South Korea, in 1991 and 1995, respectively. He is currently a Professor in information and communication engineering with Chungbuk National University. His research interests include database systems, storage management systems, sensor networks, distributed computing, big data processing, and social network service.