2022 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)

EPGraph: An Efficient Graph Computing Model in Persistent Memory System

Baoke Li[1,2], Cong cao[1(✉)], Fangfang Yuan[1], Yanbing Liu[1], Baohui Li[3], Binxing Fang[4]
[1]*Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China*
[2]*School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China*
[3]*Chinese Information Technology Securiry Evaluation Center, Department of System Evaluation, Beijing, China*
[4]*Institue of Electronic and Information Engineering of UESTC in Guangdong, Dongguan, Guangdong, China*
{libaoke,caocong,yuanfangfang,liuyanbing}@iie.ac.cn, delibh@126.com, fangbx@cae.cn

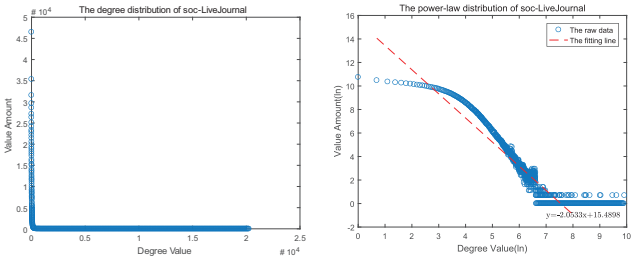
***Abstract*—With the wide range of requirements to analyse large-scale graphs in many real-world applications (e.g., relationship analysis, fraud detection and product recommendation), graph computing recently receives intensive interests. However, the massive volume and the power-law distribution of graphs are objective obstacles to efficient graph computing. Fortunately, Intel Optane DC Persistent Memory (PMEM) has emerged as a new solution, which is expected to play a crucial role in large-scale graph processing. But compared with main memory, PMEM shows much lower bandwidth and higher access latency. Therefore, it becomes paramount to fully exploit the advantages of PMEM in persistent memory systems. In this paper, we propose EPGraph, a novel efficient graph computing model designed by PMEM. To a considerable extent, it improves the spatial locality and the temporal locality of graph computing at the same time. The main contribution of our work lies in three aspects. Firstly, we design a degree-based data layering strategy to reduce the impact of power-law distribution. The hierarchical strategy makes full use of DRAM and PMEM simultaneously. Secondly, we propose a dynamic migration mechanism during the iterative execution of graph computing. The dynamic mechanism equitably schedules the subgraphs which are used in the next iteration. Thirdly, we evaluate the effectiveness of EPGraph on five public graph data sets. Extensive evaluation results show that EPGraph outperforms state-of-the-art graph computing systems by 22.67%-35.03%.**

***Index Terms*—High performance model, persistent memory system, graph computing, data layering, dynamic data migration**

I. Introduction

Graph is one of the most flexible data structures in computer science. By utilizing vertices and edges, it can fully express the relationships of entities in nature. Nowadays, graph computing is widely used in a considerable number of real-world applications (e.g., relationship analysis, fraud detection and product recommendation). These practical requirements can be solved efficiently by graph algorithms with the support of Graph Theory.

However, with the arrival of big data era, the volume of graph data in many fields (e.g., communication, e-commerce, and transportation) has grown exponentially. For examples, Facebook had more than 3.4 billion monthly active users in 2021, and there were about 1 billion commodities and 800 million users in Amazon. The scale-free graphs from all walks of life bring unprecedented challenges to graph computing.



(a) The degree distribution of soc-LiveJournal

(b) The power-law analysis of soc-LiveJournal

Fig. 1: The degree distribution and power-law analysis on soc-LiveJournal.

To storage the huge volume of graph data, distributed graph database (e.g., Neo4j [1], JanusGraph [2] and HugeGraph [3]) is a direction, which consumes lots of hardware resources inevitability. To analyse the massive graphs deeply, more computing resources have to be taken up by graph computing models (e.g., graph convolution networks [4], graph attention networks [6] and graph generative networks [7]).

Besides the massive volume, graphs tend to show power-law characteristic. As Fig.1 (a) shows, according to statistics on soc-LiveJournal, the top 1.95% of vertices are adjacent to 25.3% of the edges, and the bottom 35.8% of vertices are adjacent to 4.18% of the edges. That is, the amount of vertices and the distribution of degrees satisfy the linear fitting relationship: $ln(VertexAmount) = ln(c) - rln(Degree)$. Generally, when r is close to 2, it means the graph follows the power-law characteristic. As Fig.1 (b) shows, for soc-LiveJournal, r is 2.05. Both of them confirm that most graphs follow the power-law distribution.

Both the massive volume and the power-law characteristic of graphs are objective obstacles to efficient graph computing [8]. In order to process the huge and complex graphs, many in-memory graph computing systems(e.g., Pregel [9], Ligra [12] and GraphX [13]) and distributed in-memory systems(e.g., Pregel [9], GraphLab [10] and Gemini [11]) had been designed. Although the performance of in-memory systems can meet some requirements of academic research, main memory

978-1-6654-6497-0/22/$31.00 ©2022 IEEE
DOI 10.1109/ISPA-BDCloud-SocialCom-SustainCom57177.2022.00009

must be big enough to hold the whole vertices and edges. It seriously limits the application of the in-memory graph computing systems.

On the one hand, as a kind of precious computing resources, main memory is economically used by operating systems. The massive volume of graph makes great contradiction with the limited memory. Nevertheless, it's quite expensive that equipped with large capacity memory on a single machine. This means that in-memory graph computing systems have natural limitations in large-scale graph processing.

On the other hand, distributed in-memory solutions must deal with the problems of data synchronization and load imbalance. Due to the limited capability of parallel computing, the overall performance of distributed in-memory solutions degrades significantly. Furthermore, the quality of graph partition can not be guaranteed in distributed systems. Therefore, the communication overhead between different nodes is the main disadvantage of distributed systems.

Fortunately, Intel Optane DC Persistent Memory (PMEM) provides more solutions of large-scale graph computing, such as [17], [20] and [23]. PMEM not only has the byte addressable characteristic of main memory, but also has the persistent characteristic of external storage [15]. So, it represents a new class of memory and storage technology architected specifically for data center usage. Characterized by much larger capacity and lower access latency, PMEM is expected to play a crucial role in large-scale graph processing.

Another major benefit of using PMEM is cost. In distributed systems, each node is an entire computer. Decreasing the number of nodes can reduce the cost significantly. For instance, if each node is equiped with 64 GB DRAM, then, more than 8 machines are required to compute 512 GB graph data in parallel. A single machine equipped with PMEM can have 512 GB of main memory, so the cluster of 8 nodes can be reduced to a single node. It completely removes the costs of 7 nodes. Simultaneously, it can efficient storage graph data, which could reduce the preconditioning costs. What's more, it consumes inconsiderable energy. Therefore, it provides a more cost-effective solution for graph processing.

However, compared with main memory, PMEM shows high access latency and low bandwidth. These performance differences imply that the traditional graph computing models might not be optimized in persistent memory systems [17] [19]. Therefore, it becomes paramount to understand the strengths of PMEM in graph computing.

In this paper, we propose EPGraph which is a novel efficient model specially designed by PMEM. EPGraph focus on the graph data layering strategy and the data migration mechanism simultaneously. By analyzing the power-law distributions of graph data sets and the execution flows of graph computing, EPGraph hierarchically allocates graph data between DRAM and PMEM. What's more, by calculating the ratio of the active vertices, EPGraph selectively migrates graph chunks between DRAM and PMEM. In summary, the main contributions of our work are as follows:

- We design a data-layering strategy to separately storage all the in-memory graph data. The strategy not only reduces the influence of power-law distribution of graph data, but also improves the spatial locality of graph computing.
- We propose a dynamic data migration mechanism to dispatch graph data between the iterative execution of graph computing. The migration mechanism improves the memory access efficiency and the temporal locality at a certain extent.
- We evaluate the performance of EPGraph on five public graph data sets. Extensive experimental results show that EPGraph outperforms state-of-the-art graph computing systems by 22.67%-35.03%.

II. Background and Motivation

Graph data allocation and access efficiency are the key factors of graph computing in persistent memory systems [17]. For one thing, due to the power-law characteristic of real-world graphs, the phenomenon of asymmetric convergence [21] [23] is widespread in the process of graph computing. In other words, different data placement schemes have a great impact on iterative graph computing. For another, data access efficiency is the main bottleneck of the persistent memory systems, due to the bandwidth and latency limitations of PMEM. In addition, the iterative graph algorithms lead to a linear increase of the time overhead. Hence, data migration strategy should be considered by persistent memory graph computing systems.

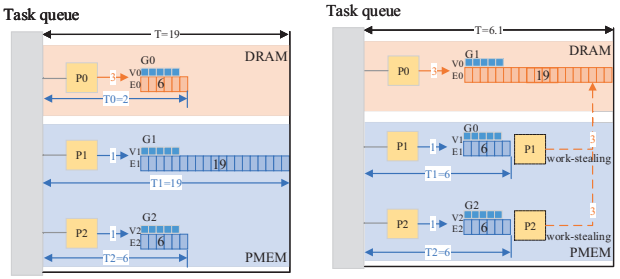
A. Graph Data Layering

Most real-world graph data sets tend to show the power-law distribution. As shown in Fig.1 (a), the degree distribution of different vertices varies greatly. A small amount of the hub-vertices connect more than millions of edges. However, the rest large number of vertices only connect dozens of edges. In order to facilitate the complex graph processing, like most of in-memory systems, the persistent memory graph computing models also need to divide the whole graph into multiple partitions and load them into DRAM and PMEM. Therefore, most memory accesses are concentrated on the hub-vertices during graph processing.

For simplicity, most of the existing systems evenly distribute all the vertices into subgraphs [21]. Then the subgraphs are processed in parallel by multiple processing threads. However, the number of edges of different subgraphs may varies greatly. This phenomenon may cause loading imbalance. In traditional architecture, multi-threading optimization mechanism such as work-stealing mechanism is used to solve this problem.

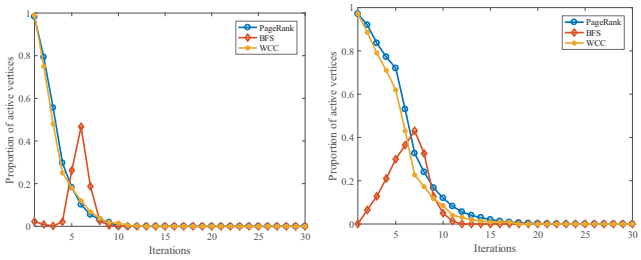
However, the bandwidth and latency gaps between DRAM and PMEM are rather significant. These differences may lead to the running times of different processing units varying greatly. All those indicate that the multi-threading mechanism has great limitations in solving the problem of load imbalance.

To improve the overall performance, graph data loading strategy should be considered in persistent memory systems. Fig.2 shows an example of graph computing in persistent



(a) Random allocation (b) Layered allocation

Fig. 2: An example of graph placing on DRAM and PMEM.



(a) soc-LiveJournal (b) Twitter-2010

Fig. 3: The proportion of the active vertices on soc-LiveJournal and Twitter-2010.

memory system. We assume the reading/writeing speed of DRAM is about 3 times faster than PMEM [23]. The vertices and the corresponding edges of $G(V, E)$ are evenly distributed into three chunks: $G_0(V_0, E_0)$, $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$. These graph chunks are processed by three processing threads in parallel. For simplicity, data dependency between different chunks is ignored. We assume the volumes of the three chunks are 6, 19 and 6.

Fig.2 (a) shows an example in which graph chunks are allocated in DRAM and PMEM randomly. In this case, thread P_0, P_1 and P_2 process G_0, G_1 and G_2 separately. The total execution time $T = max(T0, T1, T2) = max(2, 19, 6) = 19$. IF work-stealing mechanism is adopted in the random case, $T = 9$. Obviously, the overall performance of still has considerable optimization space. If the graph chunks are processed hierarchically, the overall performance can be further improved. It's evident that graph data loading format should be optimized in persistent memory systems.

As shown in Fig.2 (b), if G_1 is allocated to DRAM before the processing, the execution time can be significantly reduced. In the stage of pre-processing, it allocates G_1 which has the largest volume into PMEM. Then, it allocates G_0 and G_2 into DRAM. At $T_1 = 6$, threads P_1 and P_2 complete the processing of G_0 and G_2. They can assist thread P_0 to process the remaining data of G_1. Finally, the total execution time T can be reduced to 6.1. Compared with the case of Fig.2 (a), the performance of data-layering mechanism improves 67.9%.

Evidently, the gaps between DRAM and PMEM make the overall performance of traditional systems no longer outstanding. Therefore, well-designed loading strategy should be considered in persistent memory systems. It inspires the degree-based data-layering strategy in this paper.

B. Runtime Analysis

In this section, we analyse the process of in-memory graph computing systems (e.g., Ligra [12], GraphX [13] and Polymer [14]). Most systems usually include graph construction and algorithm execution. In detail, these systems process graph data with the three indispensable stages: (1) Split the raw graph data into tiny chunks and read all the chunks from the external storage. (2) Convert the raw graph to in-memory graph structures and allocate its value and state space. (3) Allocate other metadata space and do algorithm execution on the whole vertices and edges iteratively.

The above process ignores the runtime analysis. Many systems such as Ligra [12] only provide some interfaces which can execute graph algorithms to process the whole graph data. With the execution of graph processing, the number of active vertices or edges in each partition may change dramatically. So, it causes massive data access overhead that traversing all vertices and edges between DRAM and PMEM iteratively.

Hence, we make an experimental of comparative study on soc-LiveJournal and Twitter-2010, as shown in Fig. 3. By calculating the proportion of the active vertexes, we analysis the runtime characteristics on three common graph algorithms. When running PageRank and WCC, all the vertices will be processed in each iteration. When running BFS, its processing often involves some interlayer vertices. In the calculation process, the value of each vertex is calculated according to the adjacent vertices and edges.

As shown in Fig. 3, the number of active vertices varies greatly on different iterations. Then the algorithms gradually shrinks to a small number of active vertices. That is the phenomenon of asymmetric convergence. In general, the execution time of graph algorithms is proportional to the memory access of the active vertices or edges. Therefor, scheduling mechanism based on local awareness plays a crucial role in large-scale graph processing.

The scheduling function generally does not affect the correctness of the program. But it will affect the execution quality of the program to meet some key requirements, such as delay demand, scalability and energy budget. In persistent memory graph computing system, the scheduling function should realize the effective allocation between DRAM and PMEM during the execution of graph algorithms.

III. Model Design

In this section, we present the model designs and the implementation details of EPGraph, a novel persistent memory graph computing model, as shown in Fig.4.

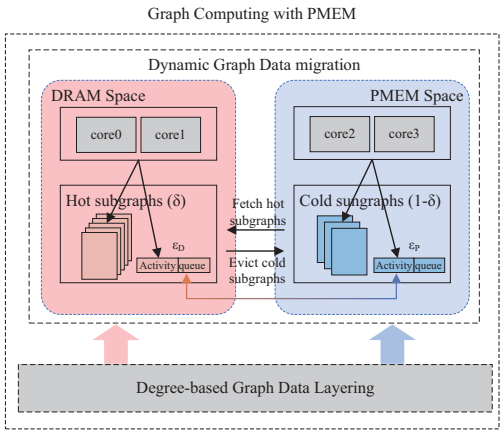


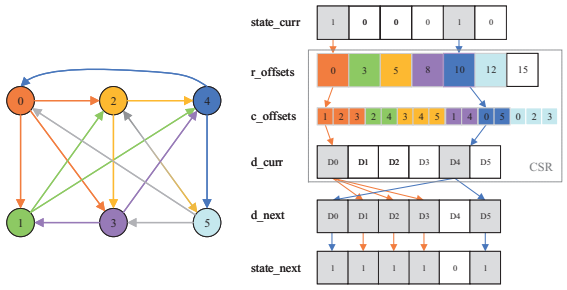
Fig. 4: The architecture of EPGraph.

A. EPGraph Overview

EPGraph is a large-scale graph computing model which is designed by Intel Optane DC Persistent Memory. As shown in Fig.4, based on vertex-centric computing model, EPGraph employs two important parts: degree-based graph data layering strategy and dynamic graph data migration mechanism. After graph partitioning, the strategy of graph data layering selectively loads subgraphs between DRAM and PMEM. In addition, based on the above runtime analysis of graph processing, the data migration mechanism selectively loads the most active subgraphs into DRAM.

Degree-based graph data layering. The graph data layering module is composed of three phases. Firstly, the raw graph dataset is loaded from external storage such as SSD or HDD and all the vertexes and the corresponding edge are divided in tiny blocks and chunks as Fig.4 shows. Secondly, the vertex chunks and the corresponding edge blocks are selectively allocated in DRAM or PMEM by the degrees of vertex blocks. Thirdly, in-memory graph structures and its auxiliary data structures like vertex value and state space are generated before graph processing. It makes full use of the random data access feature of DRAM and the high capacity characteristic of PMEM. The special of data layering strikes a good balance between DRAM and PMEM. We present the details of the graph data layering strategy in Section B.

Dynamic graph data migration. EPGraph divides graph data into tiny blocks and chunks which are the basis of data fetching/evicting. Inspired by D^{2}Graph [21], the data migration is beneficial for the performance of the whole model. By calculating the ratio of the active subgraphs in one iteration of graph computing, EPGraph fetches and evicts blocks and chunks between DRAM and PMEM. It not only focuses on taking advantage of DRAM and PMEM efficiently, but also pays attention to fetching and evicting graph data during the model operation. What's more, to maximize the performance of EPGraph, it also adopts multi-threads strategy. Thus, it improves the spatial locality and the temporal locality of graph computing in persistent memory system. We present the details of the dynamic migration mechanism in Section C.



(a) Raw graph (b) In-memory graph data structures

Fig. 5: An example of raw graph and the execution flows on the in-memory CSR representation.

B. Degree-based Graph Data Layering

As shown in Fig. 5, the raw graph which has 6 vertices and 15 edges is organized by the CSR format. After loaded into memory, the raw graph is converted to in-memory graph structures. The corresponding graph structures are based on CSR format, as shown in Fig. 5 (b). We use $r\_offsets$ and $c\_offsets$ to represent the graph structure data. Moreover, to execute graph algorithms, the graph's attribute data and status data such as d_curr, d_next, $state\_curr$ and $state\_next$ have to be builded. In Fig. 5 (b), started by vertices v_0 and v_4, there are two execution flows on the in-memory structures indicated by orange arrows and blue arrows separately.

Specifically, due to $state\_curr[0]$=1 and $state\_curr[4]$=1, v_0 and v_4 are active vertices in an iterative processing. The vertex v_0 has out-edge set E_0=$r\_offsets[0]\rightarrow c\_offsets[0 \rightarrow 3)$=$\{(0,1),(0,2),(0,3)\}$. Similarly, v_4 has out-edge set E_4=$r\_offsets[4]\rightarrow c\_offsets[10 \rightarrow 12)$=$\{(4,0),(4,5)\}$. So, v_0 has destination vertex set A_0=$\{1,2,3\}$ and v_4 has A_4=$\{0,5\}$. Based on the destination vertex set A_0 and A_4, the execution flows update graph attribute data d_next and graph status data $state\_next$.

$$R_D = N_D/(N_D + N_P) \tag{1}$$

$$N_D = \sum_{v_i \in V} 2 \times (1 + d_{v_i}) \tag{2}$$

$$N_P = \sum_{v_i \in V} (1 + d_{v_i}) \tag{3}$$

- R_D represents the ratio of access to DRAM;
- d_{v_i} represents the degree of vertex v_i;
- N_D represents the numbers of access to DRAM in one iteration, similarly to N_P.

To illustrate the effectiveness of graph data layering strategy, we analyze the access rate of DRAM in persistent memory system. We assume the access number of DRAM is N_D and the access number of PMEM is N_P. Equation (1) expresses the DRAM access rate R_D. In addition, we assume that the capacity of DRAM is C_D and the capacity of PMEM is C_P.

In the random case, we place all the graph data between DRAM and PMEM randomly. N_D and N_P can be replaced

by C_D and C_P approximatively. In general, the capacity of DRAM can be much smaller than that of PMEM. For instance, C_D=32 GB and C_P=256 GB, then R_D=11.1%. As mentioned in Section II, the latency of PMEM is much higher than that of DRAM. Therefore, the overall performance of persistent systems are reduced by the random case.

In the layered case, due to the read-only characteristic, the graph structure data ($r\_offsets$ and $c\_offsets$) can be placed in PMEM. The graph attribute data and status data (d_curr, d_next, $state\_curr$ and $state\_next$) can be placed in DRAM due to the reading and writing characteristics. According to the execution flow on the in-memory graph structures mentioned in Fig. 5(b), N_D can be expressed as equation (2) and N_P can be expressed as equation (3). Evidently, by graph data layering strategy, R_D has increased to 66.7% which is much better than 11.1%. It indicates that graph data layering strategy can improve the performance of persistent memory system.

$$R_D^{layerd} = (N_D + \delta \times N_P)/(N_D + N_P) \tag{4}$$

- δ ($0 \leq \delta \leq 1$) represents the portion of graph structure data loaded into DRAM;
- R_D^{layerd} represents the ratio of access to DRAM when placing graph data structures hierarchically between DRAM and PMEM.

What's more, Equation (4) defines the novel calculation method of R_D^{layerd} by defining variable δ. It represents the portion of graph chunks which are loaded into DRAM. In this situation, R_D is proportional to δ. As mentioned in Section II, the graph attribute and status data are much smaller than graph structure data. That is to say, graph structure data should be as much as possible loaded into DRAM. Therefore, in order to make full use of DRAM, a part of graph chunks can be loaded into DRAM selectively.

Besides δ, d_{v_i} is closely related to R_D^{layerd} by N_D and N_P as mentioned in equation (2) and (3). It means that the degrees of v_i can generate different access times between DRAM and PMEM. Thus, the sum degrees of d_{v_i} should be considered when graph is distributed into tiny chunks. Based on the above two considerations, getting better R_D means getting better performance in persistent memory systems.

Algorithm 1 presents the process of degree-based graph data-layering strategy. In the preliminary work, graph structure data should be distributed into p chunks. To get the chunks state array S which marks G_i loaded into DRAM or PMEM, it firstly calculates the Sum_i of v_i which in chunk G_i. Then, it sorts Sum in descending order and gets the top k entities of Sum. The number k should be in the range of $\delta \times P$. Finally, the chunks are allocated in DRAM or PMEM according to state array S. If $\delta = 0.3$, R_D^{layerd} will increase another 10%. Therefore, based on the above strategy, the persistent system performance gets further optimized. We will quantitatively evaluate the efficiency of the mechanism in section IV.

Algorithm 1 Degree-based Data Layering Strategy

Input: $G_i(V_i, E_i)$, d_v, chunks p, threshold δ.
Output: Chunks state S.

```
state_curr = {1, ..., 1}; k = 0;
S = {0, ..., 0}; Sum = {0, ..., 0};
procedure Data-layering analysis (p, d_v, state_curr)
for (i = 1 : p) do
    for (each v ∈ V_i) do
        Sum[i] += state_curr[i] × d_v;
    end for
end for
end procedure
procedure Get Top-k indexes (Sum, δ, k);
Descending(Sum);
k = δ × Sum.size();
for (i : k) do
    S[Sum[i].getindex()] = 1;
end for
end procedure
return S;
```

C. Dynamic Graph Data Migration

According to the phenomenon of asymmetric convergence, the iterative computing time of different graph chunks varies greatly. So, the above strategy can not resolve the efficiency problem of persistent memory systems once and for all. As shown in Fig. 3, the computing gradually shrinks to the number of active vertices. In general, the execution time of graph algorithms is proportional to the memory access of active vertices and edges. Therefore, the runtime scheduling strategy between DRAM and PMEM should be considered in persistent memory systems.

As mentioned in the above section, in addition to the number of vertices and edges, graph structure data($r\_offsets$ and $c\_offsets$) implies the degree information of each vertex. What's more, graph status data($state\_curr$ and $state\_next$) marks the active state of vertices. As a result, $state\_curr[v]$ and d_v can reflect the activity of graph chunks. So, based on the relative activation of different graph chunks, it executes fetching and evicting operations:

$$\varepsilon = \Sigma_{v \in V_i} state\_curr(v) \times d_v / \Sigma_{v \in V_i} d_v \tag{5}$$

- d_v represents the out-degree of vertex v in V_i;
- $state\_curr(v)$ represents the active state of vertex v;
- V_i represents the vertex block of subgraph G_i.

The goal of formula (5) is calculating the relative activation of each G_i in DRAM and PMEM. The target parameter ε is determined by $state\_curr(v)$ and d_v. On the one hand, the value of ε is proportional to the active vertices. That is, the more active vertices in G_i, the bigger ε is. On the other hand, ε is proportional to d_v. In other words, the more out-edges of active vertices in G_j, the bigger ε is.

Algorithm 2 presents the procedure of dynamic data migration mechanism. During the iterations of graph algorithms such as PageRank, the dynamic migration algorithm calculates the relative activations of the graph chunks in DRAM and PMEM. Due to the difference reading/writing performance,

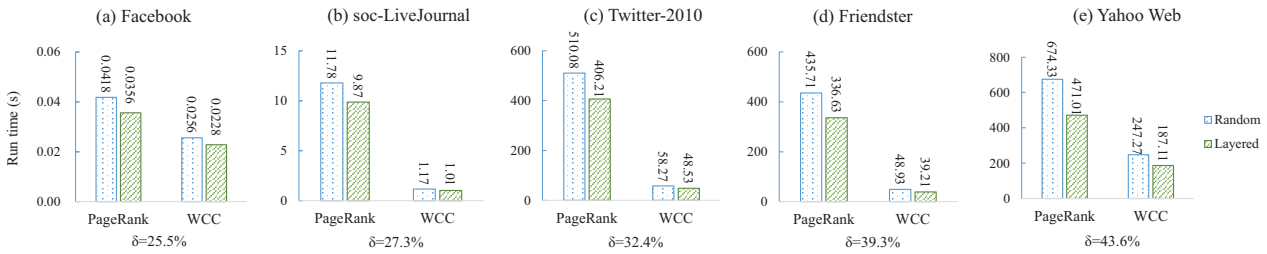


Fig. 6: Execution times of random and layered formats on PageRank and WCC.

it needs to sort ε in DRAM and PMEM separately. $G_{\varepsilon_D[i]}$ represents the subgraph with the maximum ε in DRAM, and $G_{\varepsilon_P[j]}$ represents the minimum ε in PMEM. By utilizing the function of $memcpy()$, it migrates subgraphs $G_{\varepsilon_D[i]}$ and $G_{\varepsilon_P[j]}$, simultaneously.

Algorithm 2 Dynamic Data Migration Mechanism

```
Graph G_i(V_i, E_i), d_v, stat_curr, chunks p.
procedure relative activation(p, d_v, state_curr)
for (i = 1 : p) do
    for (each v ∈ V_i) do
        sum[i] += d_v;
        ε[i] += state_curr[v] × d_v;
    end for
    ε[i] = ε[i]/sum[i]; //Sum normalization;
end for
if (G_i ∈ PMEM) then
    Descending ε_P[i];
else
    Ascending ε_D[j];
end if
end procedure
procedure DataMigrate(ε_P[i], ε_D[i], k)
for (i = 0 : k) do
    MigrateData(G_εD[i], G_εP[j]); //memcpy();
end for
end procedure
```

Undoubtedly, there is a certain of time cost for data migration. But, based on the performance differences between DRAM and PMEM, setting reasonable ε and k can overcome this problem. In this case, it not only reduces the random access of graph structure data, but also maximizes the cache hit rate as much as possible.

Due to the dynamic migration mechanism and the reduction of random memory accesses, it may improve the performance of persistent graph computing systems. We will quantitatively evaluate the efficiency of the dynamic migration mechanism in section IV.

IV. Evaluation

In this section, we firstly introduce our evaluation environment, graph data sets and the graph algorithms. Secondly, we experiment the efficiency of degree-based graph data layering strategy. Thirdly, we evaluate the influence of dynamic graph data migration. Under multi-threading environment, we evaluate the effectiveness of EPGraph by comparing with state-of-the-art systems.

A. Experiment Setup

All experiments are conducted on a Intel Xeon Gold 5218R CPU which is equipped with 20 cores, 40 threads, 32 KB L2 cache and 27.5 MB L3 cache. The experimental platform is built into a persistent memory system which is working for large-scale graph processing. To emulate the experimental setup with limited DRAM resource, it equips with 16 GB DRAM, 256 GB Optane DC persistent memory modules and running Ubuntu 18.04 LTS system.

TABLE I: Five public graph data sets.

Dataset	Vertices	Edges	Type
Facebook	4,039	88,234	Social Graphs
soc-LiveJournal	4,847,571	68,993,773	Social Graphs
Twitter-2010	61,578,416	246,313,664	Social Graphs
Friendster	65,608,366	1,806,067,135	Game Graphs
Yahoo Web	1,413,511,424	5,654,045,696	Web Graphs

All the data sets in the experiments are real-world graphs as shown in Table I. Facebook and LiveJournal are chosen to evaluate the scalability of EPGraph model. Twitter-2010, Friendster and Yahoo Web are social network, game site and web site separately. They consist billions vertices and edges and they have larger diameters. Specifically, the last three graphs are 1.56x, 1.94x and 6.81x larger than the available main memory respectively.

The fundamental algorithms used in our evaluations include the representative sparse matrix multiplication algorithm PageRank and the traversal-based graph algorithm WCC. In experiments, PageRank is set to run ten iterations and WCC runs until it's convergent. These algorithms can provide a comprehensive evaluation and exhibit different computation characteristics. Finally, EPGraph is compared with state-of-the-art systems: Ligra [12] and Polymer [14] which are mentioned in Section V.

B. Effect of Degree-based Graph Layering Strategy

In experiments, we make a full comparison of the efficiency between random allocation and layered allocation. We conduct

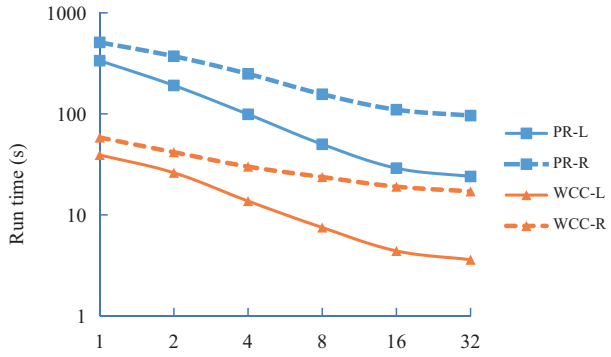


Fig. 7: Multithread execution times for PageRank and WCC on Friendster.

TABLE II: Execution times of data migration on random and layered formats (in seconds)

Algorithm	Dataset	Model-I		Model-II	
		R	R+M	L	L+M
PageRank	Facebook	0.022	0.0203	0.016	0.0147
	LiveJournal	0.81	0.73	0.66	0.58
	Twitter-2010	45.2	38.6	35.1	29.3
	Friendster	96.3	80.3	24.2	19.9
	Yahoo Web	114	91.8	101	76.1
WCC	Facebook	0.016	0.015	0.012	0.0112
	LiveJournal	0.23	0.21	0.12	0.11
	Twitter-2010	7.28	6.34	3.97	3.49
	Friendster	17.1	15.1	3.62	3.02
	Yahoo Web	89.7	76.6	22.4	18.2

single thread experiments with PageRank and WCC on the above five graph data sets. According to the equation (4) and the degree information of vertices in different data sets, δ is setted to 25.5%, 27.3%, 32.4%, 39.3% and 43.6%, respectively. The execution times of the above two allocations on five public graph data sets are shown in Fig. 6.

Obviously, the overall performance of layered mechanism is better than random format. More specifically, for WCC, the performance of graph data-layering mechanism is 12.2%-32.1% higher. For PageRank, the performance of graph data-layering mechanism is 17.4%-43.1% higher. Based on the above observations on the five data sets, we can draw two conclusions: (1) Compared with WCC, the performance of PageRank gets a greater improvement, due to the relatively more memory access which is sensitive to heterogeneous memory. (2) From the trend of the execution times in Fig. 6, we can see that the greater of δ, the greater will be the performance improvement. The above two conclusions can also explain the execution times in Fig. 7.

In order to further verify the effectiveness of data layering strategy, we also experiment it on Friendster in multi-threaded environment (from 1 to 32 working threads). As shown in Fig. 7, for the random and the layered allocations, PageRank and WCC are marked as PR-R, PR-L, WCC-R and WCC-L, respectively.

With the increase of working threads, the execution time gradually decreases. This trend proves that the graph data-layering strategy can take advantages of DRAM and PMEM in persistent memory systems. Similarly, the efficiencies of PR-L and WCC-L get greater promotion. It's obvious that the experimental results in Fig. 6 and Fig. 7 verify the above conjecture in Section II. Therefor, the overall performance of persistent memory system can be considerable development by the graph data layering strategy.

C. Effect of Dynamic Data Migration Mechanism

To evaluate the effectiveness of data migration mechanism, we run PageRank and WCC on the above five data sets, which are organized by the random format and the layered format respectively. It's widely known that parallel computing can make full use of graph computing systems. Thus, we evaluate the effectiveness of dynamic data migration mechanism by 32 working threads.

Uniformly, we treat the dynamic data migration mechanism (M) which based on the random format(R) as Model-I. As shown in Table II, Model-I's execution times are optimized on the five data sets to a certain degree. More specifically, for PageRank, the performance of Model-I is 6.67%-18.75% higher. For WCC, the performance of Model-I is 6.25%-14.6% higher. The experimental results of Model-I show that the migration mechanism is effective in the random allocation format.

In order to further verify the effectiveness of data migration mechanism, we also experiment it with the layered allocation format. Similarly, we treat the dynamic data migration mechanism (M) which based on the layered format (L) as Model-II. As we can see in Table II, the performance of Model-II improves significantly. More detailed observation, the performances of PageRank and WCC both improve remarkable especially on Twitter-2010, Friendster and Yahoo Web, which are 16.5%-24.6%.

By analyzing the experimental results of the Model-I and the Model-II, we can determine that the effectiveness of the dynamic data migration mechanism in persistent memory systems. Based on the dynamic data migration mechanism, EPGraph can improve the spatial locality of graph computing. In other words, EPGraph is an efficient graph computing model which combines the graph data layering strategy and the dynamic data migration mechanism together.

D. Comparison with Other Systems

We compare EPGraph with state-of-the-art in-memory systems: Ligra [12] and Polymer [14]. They all support parallel graph computing. Thus, to get the optimal performance of all the systems, we provide 16 GB DRAM, 256 GB PMEM and 32 working threads. To make it fair, Ligra and Polymer allocate all the graph data randomly. We present the execution times on the five different graph data sets in Fig. 8.

In order to improve the efficiency of memory access, Ligra [12] utilizes the vertex-map model and edge-map model.

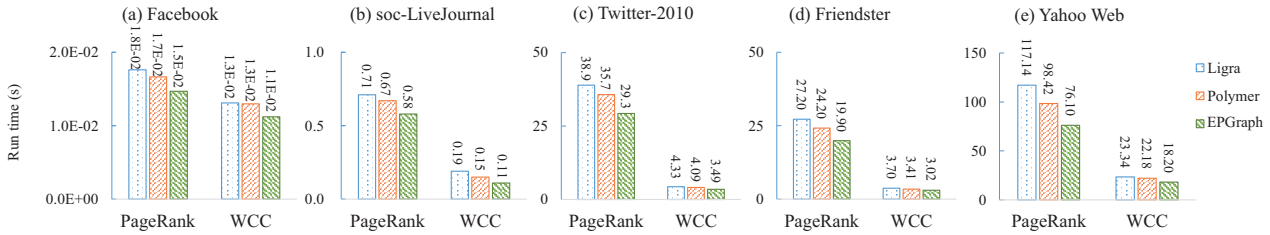


Fig. 8: Comparisons of execution time with state-of-the-art single machine graph computing systems.

However, due to ignoring the dynamic characteristics of graph computing, it traverses all the in-memory graph data which is time-consuming. In addition, it produces large amounts of writing operations of intermediate results, which leads to enormous number of I/O overheads. Based on NUMA structure, Polymer [14] adopts agent to eliminate the remote memory access. However, in each iteration, it ignores the problem of load balancing among multi-threads. That is just the greatest strength of EPGraph. The predictive data migration brings less memory access cost and better utilization of parallelism. Based on the graph data-layering strategy and the dynamic data migration mechanism, EPGraph achieves load balancing between DRAM and PMEM.

It's obvious that EPGraph achieves a significant speedup over Ligra and Polymer, as we can see in Fig.8. Specifically, it outperforms Ligra by 16.37%-35.03% and 14.11%-22.05% on Pagerank and WCC. Moreover, it outperforms Polymer by 13.71%-22.67% and 11.65%-17.95% on Pagerank and WCC, respectively. The improved performance of EPGraph is benefited by the combination of the graph data layering strategy and the dynamic data migration mechanism.

V. Related Work

Single machine graph computing systems enable users to process, analyze and mine large-scale graphs. Current single-machine systems are divided into shared-memory systems and persistent-memory systems.

Shared-memory Systems. They typically utilize a high-end personal computer to load the whole graph data, such as [5], [12] and [14]. The underlying properties such as NUMA, memory locality, and multi-cores are utilized. Specifically, GraphIt [5] is able to run on varying sizes graphs which have different structures. Its graph calculation programs separate calculation and scheduling. Ligra [12] is a simple single-machine parallel graph computing framework which optimizes the graph traversal algorithms and provides two programming interfaces named EdgeMap and VertexMap. It takes advantage of push-pull processing model. By this framework, graph algorithms can be easily implemented. Motivated by NUMA characteristics, Polymer [14] builds a hierarchical barrier to boost parallelism. According to the fact that the bandwidth of sequential remote access is higher than that of random remote access, they proposed a differential data placement strategy to reduce random remote accesses. Moreover, by using vertex replications, it converts the random remote access into a sequential remote access.

Due to main memory is large enough, these systems avoid the disk I/O overhead. However, it's difficult to deploy them when scale-free graphs cannot fit in main memory. Thus, the persistent memory system must be considered in graph computing field.

Persistent-memory Systems. With the unprecedent development of persistent memory (PMEM) technologies, there have been a few studies on graph computing in persistent memory. For instance, Huang *et al.* [17] places vertices in DRAM and edges in PMEM. To reduce the random accessing to vertices, it uses SRAM as data buffer. Based on interval-block graph partitioning, it improves data access efficiency. But it focuses on reducing the energy consumption of graph computing system rather than the performance. Gill *et al.* [19] explores graph computing on Memory Mode and APP-Driect Mode of PMEM, respectively. It points that large-scale graph processing jobs should work on APP-Driect Mode. In addition, huge page technology can improve the overall performance. What's more, it proves that NUMA migration adds kernel time overhead without giving significant benefits.

Based on the above systems, EPGraph realizes the performance optimization in persistent memory system. We use a similar scheme to partition the graph, however, our goal is to make full use of the different characteristics of DRAM and PMEM. Our data layering strategy reduces the consumption of DRAM and fully exploit the large capacity of PMEM. Moreover, we adopt the dynamic data migration mechanism to optimizes graph computing in persistent memory systems.

VI. Conclusion

In this paper, we propose EPGraph which tackles the challenge of large-scale graph computing in persistent memory system. It's a parallel graph computing model which takes advantage of DRAM and PMEM. EPGraph improves the performance of graph computing by utilizing the graph data-layering strategy and the dynamic migration mechanism. Based on the different access patterns of in-memory graph data structures, EPGraph reduces random data accesses between DRAM and PMEM. By the runtime analysis, EPGraph adopts dynamic migration mechanism to selectively schedule the active subgraphs between DRAM and PMEM. To a considerable extent, it improves the spatial locality and the temporal locality. Moreover, it makes full use of the multi-threaded

parallel technology to further improve the overall performance. Based on the above three portions, EPGraph can efficiently process large-scale graphs on a single commodity machine. The experimental evaluation shows that EPGraph outperforms by 22.67%-35.03% when compared with other state-of-the-art frameworks.

EPGraph is designed for graph computing in persistent memory systems with small amount of main memory. Although this model can automatically allocate graph data and dynamically migrate subgraphs, it needs to be extended. In the future, we intend to improve the scalability and the multitasking capability of the model.

ACKNOWLEDGMENT

This research is supported by Strategic Priority Research Program of the Chinese Academy of Sciences under Grant No. XDC02030000 and No.Y9W0013401.

REFERENCES

[1] Fernandes, Diogo, and Jorge Bernardino. "Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB," Data. 2018.
[2] https://docs.janusgraph.org/storage-backend/.
[3] https://github.com/hugegraph/hugegraph/.
[4] Wu, Zonghan, et al. "A comprehensive survey on graph neural networks," IEEE transactions on neural networks and learning systems 32.1 (2020): 4-24.
[5] Y. Zhang, M. Yang, R. Baghdadi, et al. "Graphit: A high-performance graph dsl," Proceedings of the ACM on Programming Languages, vol. 2, no. OOPSLA, p. 121, 2018.
[6] Veličković, Petar, et al. "Graph attention networks," arXiv preprint arXiv:1710.10903 (2017).
[7] Kolda, Tamara G., et al. "A scalable generative graph model with community structure," SIAM Journal on Scientific Computing 36.5 (2014): C424-C452.
[8] Gonzalez, J. E. , et al. "PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs," Usenix Conference on Operating Systems Design Implementation USENIX Association, 2012.
[9] Grzegorz Malewicz, , et al. "Pregel: a system for large-scale graph computing," in SIGMOD'10. 2010. 135–146.
[10] Low, Y., et al. "Distributed GraphLab: A Framework for Machine Learning in the Cloud," Proceedings of the VLDB Endowment 5.8(2012).
[11] Zhou, and Shuheng. "Gemini: Graph estimation with matrix variate normal instances," Annals of Statistics 42.2(2014):532-562.
[12] J. Shun and G. E. Blelloch. "Ligra: a lightweight graph computing system for shared memory," in ACM Sigplan Notices, vol. 48, no. 8. ACM, 2013, pp. 135–146.
[13] Joseph E Gonzalez, Reynold S Xin, et al. 2014. "GraphX: Graph Computing in a Distributed Dataflow System," in OSDI'14. 599–613.
[14] K. Zhang, R. Chen, and H. Chen. "Numa-aware graph-structured analytics," ACM SIGPLAN Notices, vol. 50, no. 8, pp. 183–193, 2015.
[15] https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html
[16] Y. Tian, A. Balmin, S. A. Corsten, S. Tatikonda, and J. McPherson. "From think like a vertex to think like a graph," Proceedings40 of the VLDB Endowment, vol. 7, no. 3, pp. 193–204, 2013.
[17] Huang, T. , et al. "HyVE: Hybrid vertex-edge memory hierarchy for energy-efficient graph processing," 2018 Design, Automation Test in Europe Conference Exhibition (DATE) IEEE, 2018.
[18] T. Hegeman, A. Trivedi and A. Iosup. "Grade: A System for Performance Characterization of Distributed Graph Computing," 2020 IEEE International Conference on Cluster Computing, Kobe, Japan, 2020, pp. 57-68, doi: 10.1109/CLUSTER49012.2020.
[19] Gill G , Dathathri R , Hoang L , et al. "Single machine graph analytics on massive datasets using Intel optane DC persistent memory[J]," Proceedings of the VLDB Endowment, 2020, 13(8):1304-1318.
[20] H. Liu, R. Liu, X. Liao, H. Jin, B. He and Y. Zhang, "Object-Level Memory Allocation and Migration in Hybrid Memory Systems," in IEEE Transactions on Computers, vol. 69, no. 9, pp. 1401-1413, 1 Sept. 2020, doi: 10.1109/TC.2020.2973134.
[21] B. Li et al. "D^2Graph: An Efficient and Unified Out-of-Core Graph Computing Model," 2021 IEEE ISPA, pp. 193-201, doi: 10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00038.
[22] Yu, Jiping, et al. "DFOGraph: An I/O and Communication-Efficient System for Distributed Fully-out-of-Core Graph Processing," 2021.
[23] W. Liu, H. Liu, X. Liao, H. Jin and Y. Zhang. "Straggler-Aware Parallel Graph Processing in Hybrid Memory Systems," 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), 2021, pp. 217-226, doi: 10.1109/CCGrid51090.2021.00031.