International Journal of Applied Mathematics
Volume 38 No. 5, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

# AN EDGE-ORIENTED CONCURRENT GRAPH PROCESSING FRAMEWORK: SHARED ACCESS AND TIERED MEMORY OPTIMIZATION DESIGN

**Weirong Xiu [1, 2], Md Gapar Md Johar[3*], Mohammed Hazim Alkawaz[4], Chen Bian [5]**

[1] School of Graduate Studies, Management and Science University,40100 Shah Alam, Selangor, MALAYSIA

[2]School of Information Technology and Engineering, Guangzhou College of Commerce, 511363 Guangzhou, CHINA

e-mail: xiuweirong@vip.163.com

[3]Software Engineering and Digital Innovation Center, Management and Science University, 40100 Shah Alam, Selangor, MALAYSIA

e-mail: mdgapar@msu.edu.my

[4]Department of Computer Science, College of Education for Pure Science, University of Mosul, Mosul, Nineveh, IRAQ

e-mail: mohammed.ameen@uomosul.edu.iq

[5]College of Internet Finance and Information Engineering, Guangdong University of Finance, 510521 Guangzhou, CHINA

e-mail: bianchen0720@126.com

## Abstract

With the increasing deployment of IoT devices and intelligent terminals, large-scale graph data with dynamic relational structures are continuously generated and accessed at the network edge. However, edge nodes typically feature limited computing capacity, heterogeneous hardware conditions, and fluctuating bandwidth, making traditional data center–oriented graph processing frameworks difficult to sustain efficient and stable concurrent execution in such environments. To address this challenge, this study proposes an Edge-Oriented Concurrent Graph Processing Framework (ECGPF). The framework constructs a shared data region within edge nodes and employs a tiered memory access strategy to reduce redundant data loading and mitigate access conflicts during multi-task execution. In addition, a lightweight runtime regulation mechanism dynamically adjusts concurrency levels based on resource conditions, preventing memory pressure accumulation and execution jitter. Experiments conducted on two real-world datasets, Microsoft News and User Behavior, demonstrate that ECGPF consistently achieves high throughput and stable execution latency under varying workloads and node scales, and exhibits desirable scalability

Received: August 07 2025

Volume 38 No. 5, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

in distributed edge settings. These results indicate that optimizing data access paths and concurrent execution strategies enables efficient and stable graph processing under resource constraints, providing practical support for applications such as intelligent recommendation, behavior analytics, and real-time urban computing.

**Math. Subj. Classification 2020:** 68M14, 65Y05, 49M27

**Key Words and Phrases:** Edge computing; Graph data processing; Concurrent execution; Data sharing; Memory access optimization

## 1. Introduction

With the widespread deployment of IoT devices and edge-intelligent systems, large-scale graph data with dynamic relational structures are continuously generated at the network edge. These data underpin many applications, including social network analysis, personalized recommendation, traffic management, and security monitoring. However, graph data are typically large, structurally complex, and frequently updated, requiring efficient concurrent processing. In edge computing environments, limited computational capacity, hardware heterogeneity, and fluctuating network conditions often lead to redundant data loading, cache thrashing, memory contention, and unstable scheduling when multiple graph tasks execute concurrently. Consequently, traditional graph processing frameworks such as MapReduce, Pregel, and GraphX, which are designed for resource-rich cloud environments, struggle to maintain performance and stability at the edge. Although recent techniques—such as graph neural networks, graph compression, and streaming graph processing—have improved the efficiency of dynamic graph analytics, they primarily focus on model optimization or centralized execution and lack support for multi-task concurrency and runtime stability under resource constraints.

To address these challenges, this study proposes an Edge-Oriented Concurrent Graph Processing Framework (ECGPF). The framework introduces a shared data region and a tiered memory access strategy to reduce redundant data replication and mitigate access conflicts. In addition, a lightweight runtime regulation mechanism dynamically adjusts concurrency levels based on resource conditions, preventing memory pressure accumulation and execution jitter. Experiments conducted on the Microsoft News and User Behavior datasets show that ECGPF consistently improves throughput, execution stability, and scalability under various workloads and deployment scales.

The remainder of this paper is organized as follows. Section 2 reviews related work and presents the ECGPF framework. Section 3 reports the experimental evaluation. Section 4 concludes the paper and outlines future research directions.

## 2. Materials and Methods

### 2.1 Research Background and Related Work

Graph data often exhibit sparsity, power-law degree distribution, and small-world characteristics, where nodes and edges encode multi-dimensional relational dependencies [1]. These properties lead to irregular memory access patterns and localized computation

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

hotspots, causing storage imbalance and uneven load distribution. In distributed or edge environments, frequent structure updates further increase consistency maintenance overhead, making real-time graph processing more challenging [2].

Traditional graph processing frameworks, such as MapReduce, Pregel, and GraphX, offer vertex-centric programming models and distributed execution support. However, they are designed for resource-rich and stable cloud environments. When deployed at the edge, the overhead of cross-node communication, graph partitioning imbalance, and synchronization becomes significant, limiting real-time processing and concurrent task execution [3].

To support dynamic graph analytics, several emerging techniques have been developed. Graph neural networks learn complex structural patterns [4] (e.g., large-scale batch-level GNN partitioning [5]). Graph compression and sparse storage techniques reduce memory overhead [6], and streaming graph processing frameworks enable incremental graph updates [7]. However, these approaches mainly target model expressiveness or centralized streaming execution, and provide limited support for multi-task concurrency and runtime stability in edge environments.

Recent research has explored improving graph processing efficiency under resource constraints by employing lightweight graph algorithms, locality-aware storage, and delay-tolerant synchronization [8–9]. Despite reducing consistency overhead, these methods still face challenges such as low shared-data access efficiency, access conflicts, and execution jitter under high concurrency [10]. Conventional concurrency strategies, including multi-threading and distributed partitioning, remain sensitive to data skew and synchronization complexity at the edge [11]. Meanwhile, resource-aware dynamic concurrency control and hardware–software co-optimization have been proposed [12–14], but they often rely on high-performance hardware, complex coordination mechanisms, or centralized optimization frameworks, which limit their applicability in edge environments [15–17].

In summary, existing research either assumes centralized resource sufficiency or focuses mainly on scheduling-level optimization. A low-overhead framework capable of improving multi-task concurrency and runtime stability in edge environments remains lacking. This study addresses this gap by proposing ECGPF, which enhances edge graph processing performance through shared data access, tiered memory organization, and lightweight runtime concurrency regulation.

### 2.2 Overall framework design of the system

To address the performance degradation caused by limited resources, strong hardware heterogeneity, and unstable network conditions in edge computing environments, this study proposes an Edge-Oriented Concurrent Graph Processing Framework (ECGPF). The framework aims to improve data access efficiency, concurrent execution capability, and runtime stability of graph processing on edge nodes, while minimizing communication and synchronization overhead without modifying the upper-layer graph computation logic. This enables real-time and continuously updated graph analytics to be executed efficiently in distributed edge settings.

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

ECGPF is composed of two core mechanisms: a shared data access mechanism and a concurrent memory access mechanism. The shared data mechanism reduces redundant data loading and storage duplication among parallel graph tasks, while the concurrent memory access mechanism optimizes data access paths to mitigate access conflicts and waiting overhead. Together, these mechanisms enhance cache locality and improve overall execution throughput.

2.3 Shared Data Access Mechanism

In traditional concurrent graph processing, multiple tasks typically load the same vertex and edge data independently, resulting in redundant memory consumption and unnecessary data transfers. To eliminate this overhead, ECGPF maintains a shared data region within each edge node to store frequently accessed and reusable graph structural data. A data preprocessing module identifies high-access hot data and loads them into the shared region. Tasks directly perform read operations from the shared region, while updates or modifications are executed on local copies to ensure data consistency and isolation. The framework dynamically adjusts the shared dataset based on access frequency and memory pressure to maximize data locality and cache utilization under constrained resources.

2.4 Concurrent Memory Access Strategy

To further enhance concurrency efficiency, ECGPF adopts a tiered memory access model:

(1) Read Operations → directly access the shared data region;

(2) Write/Update Operations → are executed on private data copies;

(3) Merge Operations → are performed selectively after the computation phase.

A concurrency coordinator determines the execution order based on task-level data dependencies to avoid long waiting chains and access hotspots. Additionally, a lightweight cost model balances shared access overhead against latency to improve throughput without introducing substantial scheduling or merging overhead.

Through this design, ECGPF effectively reduces redundant memory allocation, mitigates access conflicts, and maintains stable performance in highly concurrent, heterogeneous, and dynamically varying edge environments. To visually illustrate the relationship between shared regions and private copies, the concurrent processing strategy is shown in Figure 1.

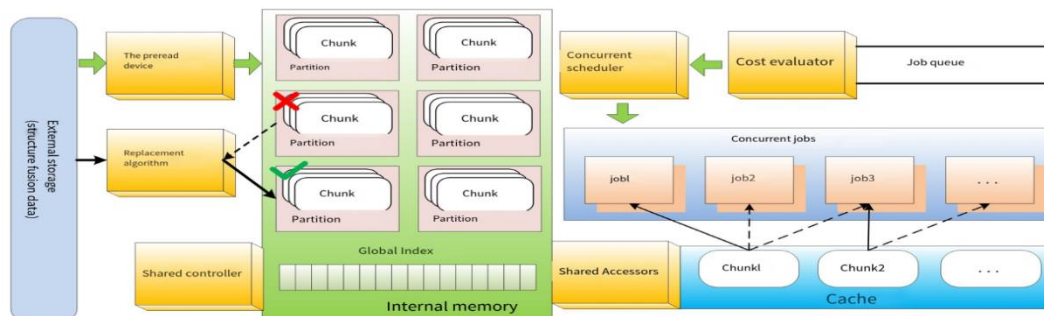



Figure 1: Concurrent Processing Strategy for High-Throughput Graph Data

Received: August 07 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

2.5 Lightweight Runtime Regulation Mechanism

To adapt to the fluctuating resource characteristics of edge nodes, ECGPF employs a resource-aware runtime regulation mechanism. The mechanism periodically monitors key status indicators such as CPU utilization, memory pressure, and task queue length, and dynamically adjusts the concurrency level according to real-time system load. By suppressing excessive parallel expansion under resource constraints and increasing concurrency when resources are sufficiently available, the mechanism alleviates local resource hotspots and prevents memory thrashing and throughput degradation.

Unlike global scheduling frameworks, this regulation mechanism does not rely on a global consistency view, complex prediction models, or pre-training. Therefore, it introduces minimal system overhead and maintains good portability across heterogeneous edge environments. Under conditions of limited resources or significant load fluctuations, this mechanism contributes to maintaining stable execution efficiency and response continuity.

2.6 Dynamic Parallelism Estimation Strategy

While runtime regulation adjusts concurrency adaptively during execution, ECGPF further introduces a dynamic parallelism estimation strategy to determine an appropriate initial concurrency level. During task initialization, the system estimates a safe baseline concurrency level by considering available memory capacity, data partition scale, and expected task computation cost. During execution, incremental adjustments are then applied according to the feedback from the runtime monitoring mechanism described in Section 2.5.

This “baseline estimation + dynamic correction” approach avoids the instability associated with fixed concurrency configurations. The strategy does not require training models, parameter priors, or additional graph sampling operations, resulting in low implementation overhead. Experimental observations indicate that this method maintains balanced throughput performance and resource utilization under multi-node heterogeneous conditions.

2.7 Non-Intrusive Integration with GraphX

ECGPF is integrated into the execution layer of GraphX in a non-intrusive manner, leaving the programming interfaces and upper-level computational logic unchanged. This enables direct incorporation into existing workflows without requiring modifications to graph algorithms.

The integration is achieved through three key steps:

(1) Execution Path Optimization: Frequently accessed vertex features and local adjacency information are preloaded into a shared data region to reduce repeated data reconstruction and communication overhead.

(2) Shared-Read and Local-Write Isolation: Read operations directly access shared data, while write operations are conducted in private copies and merged when necessary, avoiding multi-threaded write conflicts.

(3) Adaptive Concurrency Adjustment: Based on the runtime resource state, the system

Received: August 07 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

adjusts the task submission rate and effective concurrency degree to reduce risks of memory overflow and scheduling jitter.

Through these measures, the framework achieves transparency, ease of deployment, and low intrusion into existing GraphX-based applications.

### 2.8 Implementation of the Concurrency Strategy

The shared data mechanism and tiered concurrent memory access strategy described above are jointly implemented within the GraphX execution process to support stable high-concurrency graph processing. Frequently accessed vertex and edge data are stored in the shared region for direct reuse, while update operations are handled through local copies to prevent consistency conflicts. In parallel, the runtime regulation and dynamic parallelism estimation mechanisms continuously adjust the concurrency degree based on real-time system load, preventing memory pressure spikes and excessive thread contention.

This implementation maintains the original user-level programming model and algorithm semantics, ensuring that the enhancement is transparent to upper-layer computations while improving concurrent execution efficiency and stability.

### 2.9 Summary

This chapter presented the design and implementation of the ECGPF framework. By integrating shared data access, tiered memory organization, and adaptive concurrency regulation, the framework supports efficient and stable concurrent graph processing in resource-constrained edge environments. These mechanisms form the basis for the performance evaluation and analysis discussed in Section 3.

## 3. Results

### 3.1 Experimental Design and Evaluation Indicators

To verify the concurrent graph processing performance of ECGPF in edge environments, experiments were conducted on a distributed platform composed of a central node and multiple heterogeneous edge nodes. The system was deployed based on Apache Spark and GraphX, and all nodes maintained a consistent software runtime environment. Two types of graph datasets, Microsoft News and User Behavior, were selected, representing recommendation graphs and user interaction networks, respectively, in order to cover task scenarios with different structural characteristics. The experiments included two types of computation tasks: WordCount (non-iterative) and PageRank (iterative), which were used to evaluate execution efficiency, stability, and scalability. All experiments were executed repeatedly and the average value was taken to reduce the impact of system fluctuations.

To comprehensively evaluate system performance, the following metrics were adopted, See Table 1 for details.

Table 1 Performance Evaluation Metrics for Edge-Oriented Concurrent Graph Processing

| Metric | Definition | Evaluation Focus |
| --- | --- | --- |

Received: August 07 2025

**International Journal of Applied Mathematics**
**Volume 38 No. 5, 2025**
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

| Throughput | Number of graph operations completed per unit time | Overall processing capability of the system |
|---|---|---|
| Latency | Time required to complete a single task (or iteration) | Stability of concurrent execution |
| Resource Utilization (CPU/Memory) | Level of resource consumption during node execution | Load balance and sustainability of the system |
| Scalability | Impact of changes in the number of nodes on throughput | Horizontal expansion capability of the system |

Throughput and latency reflect concurrency efficiency, resource utilization characterizes execution stability, and scalability indicates adaptability under different deployment scales.

This section evaluates the throughput, latency, and resource utilization performance of ECGPF under different workloads, and compares the results with those of baseline GraphX and static parallelism configuration strategies. To examine performance across different computational patterns, two representative tasks are selected: WordCount, which involves non-iterative processing, and PageRank, which requires multi-round iterative computation. These two tasks allow the evaluation to reflect both single-pass and iterative workloads in edge-concurrent environments, thereby providing a comprehensive assessment of the proposed framework.

3.2 WordCount Task Performance Analysis

Figure 2 illustrates the completion time trends of the three execution approaches under varying levels of data distribution skew. The comparison highlights how each approach responds to changes in data locality and load imbalance.

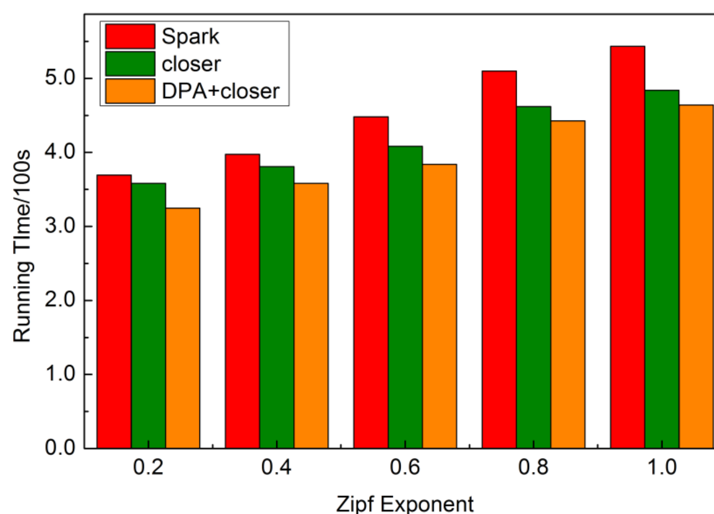



Figure 2. WordCount Experiment

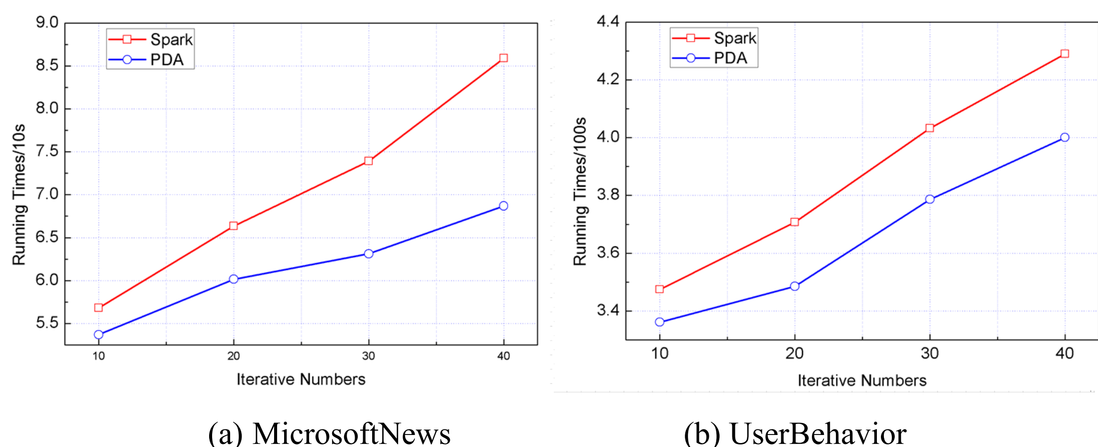
Received: August 07 2025

International Journal of Applied Mathematics
Volume 38 No. 5, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

As the Zipf exponent increases and data skew becomes more severe, the execution time of standard GraphX rises noticeably, indicating that it is sensitive to data distribution and prone to long-tail partition effects. The static balancing strategy can alleviate load concentration to a certain extent, but execution instability still occurs when edge resources are constrained. In contrast, ECGPF maintains a comparatively stable execution curve across different skew levels. The shared data access mechanism reduces repeated structural data reconstruction, while the tiered memory access and lightweight runtime regulation suppress cache thrashing and replica expansion under high concurrency. As a result, ECGPF is able to sustain more stable execution performance even as data skew intensifies.

3.3 PageRank Task Performance Analysis

Figure 3 presents the PageRank experiment results on the Microsoft News (MSN) and User Behavior (UB) datasets, which represent different graph scales and interaction patterns. This experiment evaluates the framework's behavior in multi-round iterative computation scenarios, allowing us to examine execution stability and throughput consistency over repeated synchronization phases.




(a) MicrosoftNews (b) UserBehavior

Figure 3. PageRank Experiment

As the number of iterations increases, both curves exhibit a linear growth trend, which is caused by the wide dependency synchronization inherent in iterative graph computation. Standard GraphX shows a more noticeable increase in latency when the number of iterations and graph size grow. Although the static parallelism scheme improves performance in certain stages, fluctuations still occur during frequent redistributions. In contrast, the execution curve of ECGPF is closer to a steady linear growth, indicating that it can maintain stable concurrency levels and resource usage during the iterative process, reducing jitter caused by concurrency expansion and replica reclamation, thereby lowering additional synchronization overhead and maintaining better execution stability.

3.4 Resource Utilization and Scalability Performance

In terms of resource utilization, ECGPF maintains CPU usage in the range of 70%–85% on edge nodes, and memory usage remains within a safe threshold, without frequent garbage collection or overflow, demonstrating good resource balance. By contrast, standard GraphX

International Journal of Applied Mathematics
Volume 38 No. 5, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

is more likely to experience short-term memory pressure during high concurrency phases, which affects execution rhythm. Scalability tests show that as the number of nodes increases from 2 to 8, the throughput of ECGPF grows approximately linearly; when the number of nodes increases to 10, the growth rate slows, mainly due to edge network synchronization and cross-node communication overhead, which is consistent with scalability characteristics commonly observed in edge distributed systems.

### 3.5 Comparative Discussion

Based on the experimental results of WordCount and PageRank, the differences between the approaches in execution stability and resource utilization are evident. Standard GraphX is sensitive to data distribution and load variation, exhibiting significant execution fluctuation; the static balancing strategy can alleviate data skew to some extent but may still lead to underutilization of resources or temporary overload due to the inability to adapt dynamically to runtime conditions. In contrast, ECGPF maintains more stable performance across different task types and heterogeneous resource conditions. The shared data mechanism reduces cross-task data reconstruction, the concurrent memory access strategy improves cache hit rates under high concurrency, and the lightweight adjustment mechanism effectively avoids resource concentration and execution jitter. These optimizations are performed at the execution layer and do not alter the upper-layer graph algorithms or programming model, providing good transparency and portability.

### 3.6 Summary

The experimental results demonstrate that ECGPF exhibits consistent advantages in throughput, execution latency stability, and resource utilization efficiency, and is able to maintain good scalability in multi-node environments. The framework supports high-concurrency and continuous graph data processing in edge environments without additional hardware cost or modifications to algorithm logic, providing a practical solution and experimental foundation for real-world deployment and further system optimization.

## 4. Discussion

This study proposes an Edge-Oriented Concurrent Graph Processing Framework (ECGPF) to address the challenges of concurrent graph computation in resource-constrained edge environments. The framework is designed around three core mechanisms:

(1) Shared data region construction, which reduces redundant data loading and minimizes memory duplication across concurrent tasks;

(2) Tiered memory access strategy, which improves data locality and mitigates access conflicts under high concurrency;

(3) Lightweight runtime regulation mechanism, which dynamically adjusts concurrency levels based on real-time resource states to maintain stable execution.

These mechanisms are integrated into the execution layer of GraphX in a non-intrusive manner, preserving existing graph programming interfaces and algorithmic semantics. As such, ECGPF can be seamlessly deployed within existing graph processing workflows

Received: August 07 2025

without requiring additional hardware support or modifications to upper-layer computation logic.

Experimental results across non-iterative and iterative graph computation workloads demonstrate that ECGPF provides consistently higher throughput, lower execution latency variability, and more balanced resource utilization compared with standard GraphX and static concurrency configurations. These findings indicate that enhancing data locality and concurrency control at the execution layer is an effective approach to improving both performance and scalability in edge-based graph computing scenarios.

Looking forward, several research directions can further advance this work. One direction is to extend ECGPF to dynamic and continuously evolving graph structures, where efficient shared-state maintenance and update propagation mechanisms will be essential to controlling synchronization overhead. Another direction is to incorporate fine-grained task dependency characterization and adaptive scheduling models, allowing the framework to respond more intelligently to fluctuating workloads and heterogeneous resource conditions. In addition, exploring integration with heterogeneous acceleration platforms, such as GPUs, FPGAs, and emerging graph processing hardware, may enable the framework to support larger-scale graph datasets and higher concurrency demands in real-world deployments.

**Data Sharing Agreement**

The datasets used and/or analyzed during the current study are available from the first author on reasonable request.

**Competing Interests**

The authors have no relevant financial or non-financial interests to disclose.

**Acknowledgement**

This article was supported by the Scientific Research Foundation of Guangzhou (No. 202201011667), Guangdong Provincial Education Science Foundation (No. 2023GXJK407), Guangdong Provincial Higher Vocational Education Teaching Quality Project (No.2023JG452) and Guangdong Provincial Higher Education Teaching Quality Project (No.2022SJJXGG992).

## References

[1] Corbellini A., Godoy D., Mateos C., Schiaffino S., Zunino A. An analysis of distributed programming models and frameworks for large-scale graph processing. IETE Journal of Research, 2022, 68(4): 3065–3073.https://doi.org/10.1080/03772063.2020.1754139

[2] Gabert K., Sancak K., Özkaya M. Y., Pinar A., Çatalyürek Ü. V. ELGA: Elastic and scalable dynamic graph analysis. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2021: 1–15.https://doi.org/10.1145/3458817.3480857

[3] Xu X., Zang S., Bilal M., Xu X., Dou W. Intelligent architecture and platforms for private edge cloud systems: A review. Future Generation Computer Systems, 2024, 160: 457–

471.https://doi.org/10.1016/j.future.2024.06.024

[4] Gill S. S., Golec M., Hu J., Du J., Wu H., Walia G. K., et al. Edge AI: A taxonomy, systematic review, and future directions. Cluster Computing, 2025, 28(1): 18–53.https://doi.org/10.1007/s10586-024-04686-y

[5] Yang S., Zhang M., Dong W., Li D. Betty: Enabling large-scale GNN training with batch-level graph partitioning. Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2023.https://doi.org/10.1145/3575693.3575725

[6] Yao P., Liao X., Jin H., Zhou Y., Xu P., Zhang W., Zeng Z., Pan C., Zhu B. A redundancy-aware energy-efficient graph accelerator. Scientia Sinica Informationis, 2024, 54(6): 1369–1384.https://doi.org/10.1360/ssi-2023-0387

[7] Liang Z., Zheng Y., Bi S., Yao C., Wang J., Xu L., et al. GraphFlow: A distributed streaming graph computation model. Proceedings of the 2024 IEEE International Conference on Parallel and Distributed Systems (ICPADS), 2024: 236–245.https://doi.org/10.1109/ICPADS63350.2024.00039

[8] Zheng X., Zhang C., An Y., Zhang B. A metaheuristic framework with experience reuse for dynamic multi-objective big data optimization. Applied Sciences, 2024, 14(11): 4878.https://doi.org/10.3390/app14114878

[9] Zhang W., Zeadally S., Li W., Zhang H., Hou J., Leung V. C. M. Edge AI as a service: Configurable model deployment and delay–energy optimization with result quality constraints. IEEE Transactions on Cloud Computing, 2023, 11(2): 1954–1969.https://doi.org/10.1109/TCC.2022.3175725

[10] Kim Y., Panda P. Revisiting batch normalization for training low-latency deep spiking neural networks from scratch. Frontiers in Neuroscience, 2021, 15: 773954.https://doi.org/10.3389/fnins.2021.773954

[11] Tabish R., Pellizzoni R., Mancuso R., Gracioli G., Mirosanlou R., Caccamo M. X-Stream: Accelerating streaming segments on MPSoCs for real-time applications. Journal of Systems Architecture, 2023, 138: 102857.https://doi.org/10.1016/j.sysarc.2023.102857

[12] Sagharichian M., Alipour Langouri M. iPartition: A distributed partitioning algorithm for block-centric graph processing systems. The Journal of Supercomputing, 2023, 79(18): 21116–21143.https://doi.org/10.1007/s11227-023-05492-w

[13] Chen Q., Guo S., Wang K., Xu W., Li J., Cai Z., Gao H., Zomaya A. Y. Towards real-time inference offloading with distributed edge computing: The framework and algorithms. IEEE Transactions on Mobile Computing, 2024, 23(7): 7552–7566.https://doi.org/10.1109/TMC.2023.3335051

[14] Du H., Zhang R., Liu Y., Wang J., Lin Y., Li Z., Niyato D., Kang J., Xiong Z., Cui S., Ai B., Zhou H., Kim D. I. Enhancing deep reinforcement learning: A tutorial on

generative diffusion models in network optimization. IEEE Communications Surveys & Tutorials, 2024, 26(4): 2611–2635.https://doi.org/10.1109/COMST.2024.3400011

[15] Alzu'bi A., Alomar A., Alkhaza'leh S., Abuarqoub A., Hammoudeh M. A review of privacy and security of edge computing in smart healthcare systems: Issues, challenges, and research directions. Tsinghua Science and Technology, 2024, 29(4): 1152–1180.https://doi.org/10.26599/TST.2023.9010080

[16] Ding A. Y., Peltonen E., Meuser T., Ott J., Tarkoma S., Xiao Y., et al. Roadmap for edge AI: A Dagstuhl perspective. ACM SIGCOMM Computer Communication Review, 2022, 52(1): 28–33.https://doi.org/10.1145/3523230.3523235

[17] Walia G. K., Kumar M., Gill S. S. AI-empowered fog/edge resource management for IoT applications: A comprehensive review, research challenges, and future perspectives. IEEE Communications Surveys & Tutorials, 2024, 26(1): 619–669.https://doi.org/10.1109/COMST.2023.3338015