Journal of Computing and Electronic Information Management
ISSN: 2413-1660 | Vol. 12, No. 3, 2024

Efficient Edge Computing: A Survey of High-Throughput Concurrent Processing Strategies for Graph Data

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
[3] Software Engineering and Digital Innovation Center, Management and Science University, 40100 Shah Alam, Selangor, Malaysia
[4] Department of Computer Science, College of Education for Pure Science, University of Mosul, Mosul, Nineveh, Iraq
[5] College of Internet Finance and Information Engineering, Guangdong University of Finance, 510521 Guangzhou, China

[*] **Corresponding author**: Weirong Xiu

Abstract: This paper reviews the strategies for high-throughput concurrent processing of graph data in edge computing environments. As information technology rapidly advances, particularly in areas such as the Internet of Things (IoT), smart cities, and autonomous driving, the need for real-time and efficient data processing continues to grow. Edge computing is a distributed computing paradigm that can process data at or near its origin, thereby reducing network latency, improving application performance, and reducing the load on central data centers. The discussion includes the application of edge computing in graph data processing, highlighting parallel computing models such as the Bulk Synchronous Parallel (BSP) model and other parallel optimization techniques like data partitioning and task scheduling. The paper also addresses specific challenges of parallel processing in edge computing, such as resource constraints, data communication delays, and strategies for security and privacy protection. Despite the significant potential edge computing has demonstrated in processing graph data, numerous challenges remain. Future research directions involve the development of new resource optimization algorithms, low-latency communication protocols, and technologies to enhance data security, aiming to achieve more efficient and secure graph data processing. This paper provides clear directions and a foundation for the efficient implementation of graph data processing in edge computing environments, positioning edge computing to play an increasingly significant role in real-time data analysis and intelligent applications.

Keywords: Edge computing; Graph data; High throughput; Concurrent processing.

1. Introduction

As information technology rapidly evolves, the generation and processing of data are experiencing unprecedented growth. This is particularly evident in sectors like the Internet of Things (IoT), smart cities, and autonomous driving, where real-time processing of vast amounts of dynamic data has become a critical necessity (Wang et al., 2020). These demands have propelled the development of edge computing, a distributed computing paradigm that processes data near its point of origin to reduce network latency, enhance application performance, and alleviate the load on data centers (Satyanarayanan, 2017). The significance of edge computing lies in its ability to support the growing demands for real-time applications. With the widespread deployment of 5G and IoT technologies, the data processing capabilities within edge computing environments have significantly improved, further advancing graph data processing technologies to support more complex applications such as real-time video analytics and intelligent transportation systems (Zhou et al., 2019; Yang et al., 2023). Graph data, with its unique advantages in representing complex relationships and dependencies between entities, has become an indispensable technology for many real-time applications. However, the processing capabilities for graph data in edge computing environments are limited by computational and storage resources, and face challenges related to network latency, and data sharing and synchronization (Satyanarayanan, 2017; Shi et al., 2016; Wang et al., 2019; Sahni et al., 2019). Therefore, developing efficient graph data processing strategies adapted to edge computing environments can not only optimize data processing performance and accuracy but also support a broader range of application needs, pushing forward the progress of edge computing technology and providing innovative solutions for handling complex graph data.

The purpose of this review is to explore strategies for high-throughput concurrent processing of graph data in edge computing environments. The research aims to enhance the efficiency of graph data processing, optimize the use of edge computing resources, reduce processing delays, and ensure data consistency. This review will summarize the current state of research, identify existing issues, and discuss potential future research directions. The goal is to provide references and guidance for achieving more efficient and stable graph data processing in edge computing environments.

2. Overview of Edge Computing

In the era of rapid technological advancement, the generation and processing of data are experiencing unprecedented growth. Edge computing, as a distributed computing paradigm, addresses the latency and bandwidth limitations inherent in traditional cloud computing by processing data close to its origin. This approach significantly reduces network latency, enhances application performance, and alleviates the burden on data centers (Satyanarayanan, 2017). Additionally, graph data plays a crucial role in real-time applications such as the Internet of Things (IoT), smart cities, and autonomous driving, where there is a demand not only for real-time processing but also for handling large volumes of dynamic data (Wang et al., 2020). With the

widespread adoption of 5G and IoT technologies, the data processing capabilities of edge computing have been enhanced, further advancing graph data processing technologies (Zhou et al., 2019; Yang et al., 2023). This enhancement supports more complex applications, such as real-time video analytics and intelligent transportation systems, expanding the scope and efficacy of edge computing in modern infrastructures.

The definition and architecture of edge computing originated from the concept of Content Delivery Networks (CDN), with the primary goal of transferring data processing tasks from centralized data centers to the network edge to reduce latency and enhance processing speed (Satyanarayanan, 2017). This computing model is particularly suitable for the rapidly growing fields of the Internet of Things (IoT) and smart devices, where rapid response to real-time applications is crucial. By processing data at or near its point of origin, edge computing not only improves mobile computing efficiency and reduces network latency but also optimizes bandwidth usage, prevents network congestion, and significantly reduces communication costs (Shi et al., 2016; Khan et al., 2019).

Compared to traditional cloud computing, edge computing shows significant advantages in real-time data processing. It reduces the need to transmit data to remote cloud data centers, greatly minimizing transmission times and processing delays, thus optimizing the speed and efficiency of data processing. Additionally, edge computing enhances the privacy and security of data processing. By processing data locally, sensitive information does not need to be sent over public networks to remote servers, thereby lowering the risk of data breaches or interception (Satyanarayanan, 2017; Shi et al., 2016; Khan et al., 2019). Furthermore, the distributed nature of edge computing supports system scalability. As the number of devices increases, more computing resources can be added at the network edge to accommodate various loads and application demands. These features make edge computing an ideal computing model for sectors such as IoT, industrial automation, and intelligent transportation, providing robust support for real-time data processing.

3. Graph Data Processing Technologies

3.1. Basic Concepts and Importance of Graph Data

Graph data, composed of nodes and edges, is a data structure that describes complex relationships between entities, widely used in fields such as social networks and intelligent transportation systems. For example, social media utilizes graph data to analyze user interactions and optimize advertising strategies (Chen & Ran, 2019); intelligent transportation systems leverage graph data to enhance route planning and traffic flow management (Wang et al., 2020).

The main characteristics of graph data include sparsity, power-law distribution, and small-world phenomenon. Sparsity implies that most nodes are connected to only a few other nodes, posing challenges for storage and computational optimization. Power-law distribution means that a few nodes have an excessive number of connections, affecting information propagation and network structural stability. The small-world phenomenon demonstrates that even in large-scale networks, the shortest paths between any two nodes can be very short, providing opportunities for optimized pathfinding and accelerated processing in algorithm design. These characteristics highlight the unique value and efficiency of graph data in handling complex network relationships.

3.2. Storage Methods and Data Structures

The storage structure of graph data is crucial for efficient data processing and query performance. Common storage methods include adjacency lists, adjacency matrices, edge lists, and Compressed Sparse Row (CSR) (Chen & Ran, 2019).

1)Adjacency lists: Suitable for sparse graphs, where each vertex corresponds to a list containing directly connected vertices. This method is space-efficient, easy to dynamically modify, and particularly efficient when the number of edges relative to vertices is low.

2)Adjacency matrix: Represents edges between vertices using a two-dimensional array, suitable for dense graphs. The advantage is quick queries for connectivity between any two points, but consumes significant memory when the number of vertices is high.

3)Edge list: Lists all edges directly, with each edge represented by a pair of vertices. This structure is suitable for scenarios where edge attributes are more important than vertex attributes, such as weighted networks, and offers simplicity in implementation and efficient modification.

4)Compressed Sparse Row (CSR): Suitable for applications requiring compressed storage space, achieved by storing non-zero elements, column indices, and row offsets in three arrays, saving space while maintaining access efficiency.

Choosing the appropriate storage structure significantly impacts performance. For instance, adjacency lists and CSR are superior for graph traversal, while adjacency matrices are more suitable for rapid retrieval. Understanding and selecting the most suitable storage method is crucial for enhancing the efficiency of graph data processing.

3.3. Existing Graph Data Processing Frameworks and Tools

There are various graph data processing frameworks and tools available today, such as Apache Giraph, Apache Spark's GraphX (Xin et al., 2013), and Neo4j (Inc, 2022). These tools and frameworks offer rich graph algorithm libraries, supporting efficient processing of large-scale graph data. For example, GraphX is a graph computing framework provided by Spark, which enhances Spark's ability to process graph data through optimized APIs. Additionally, deep learning techniques such as Graph Neural Networks (GNN) (Liang et al., 2020) have also been applied to analyze and process graph data, demonstrating powerful capabilities in handling complex graph structures (Chen & Ran, 2019; Wang et al., 2020).

Scalability: The ability to handle large-scale graph data.

API and Programming Model: Ease of use and development efficiency.

Graph Algorithm Library: Quantity and quality of provided graph algorithms.

Performance: Processing speed and resource utilization.

Community Support: The level of activity and support within the developer community.

Deployment and Integration: Integration with other systems and tools, as well as the ease of deployment.

Table 1. Comparison of Existing Graph Processing frameworks

Framework	Graph Types	Scalability	API and Programming Model	Graph Algorithm Library	Performance	Community Support	Deployment and Integration
Apache Giraph (Xin et al., 2013)	Directed Graph	High	Java	Medium	Low	Medium	Partial
GraphX (Xin et al., 2013)	Undirected, Directed, Weighted Graph	High	Scala/Java	Medium	High	High	Within Spark Ecosystem
Neo4j (Inc, 2022)	Directed, Undirected Graph	High	Cypher, Java, Python	High	Medium	High	Easy
GNN (Liang et al., 2020)	Various Graph Models	Medium	Pytorch, tensorflow	Medium	High	High	Independent

Note: Graph Types: Undirected graphs, directed graphs, weighted graphs, etc.

4. Parallel Processing Strategies

4.1. Parallel Computing Models

The Bulk Synchronous Parallel (BSP) model is a core concept in parallel computing, particularly suited for handling large-scale graph data. In the BSP model, computation is divided into several "supersteps," each consisting of a computation phase where operations are executed in parallel, followed by a communication phase, and then a synchronization point. All processing units must reach this synchronization point before proceeding to the next superstep. This model simplifies the complexity of parallel programming as it reduces the need for concurrent control and ensures data consistency across different processing stages. The BSP model has been widely applied in Google's Pregel system, effectively supporting complex graph computing tasks (Wang et al., 2019).

4.2. Parallel Optimization Techniques

Table 2. Comparison of parallel optimization techniques

System	Parallel Optimization Technique	Data Partitioning Strategy	Task Scheduling Strategy	Parallelism	Computing Model	Performance	Scalability
PowerLyra (Chen et al., 2015)	Synchronous	High	High	High	High	High	High
WarpGraph (Low, 2013)	Asynchronous	Medium	Medium	Medium	Medium	Medium	Medium
GraphLab (Low et al., 2014)	Hybrid	High	High	High	High	High	High
Distributed GraphLab (Low et al., 2012)	Hybrid	High	High	High	High	High	High
BiGraph (Chen, et al., 2015)	Asynchronous	Medium	Medium	Medium	High	Medium	High
PowerGraph (Gonzalez et al., 2012)	Synchronous	High	High	High	High	High	High
S-PowerGraph (Xie, et al., 2015)	Synchronous	High	High	High	High	High	High
PowerSwitch (Xie, et al., 2015)	Synchronous	High	Medium	High	High	High	High
GraphX (Zaharia et al., 2016)	Synchronous	High	High	High	High	High	High
SpecGraph (Jing Nian-qiang, et al., 2014)	Asynchronous	High	Medium	Medium	Medium	Medium	Medium
Maiter (Zhang et al., 2014)	Asynchronous	High	Medium	Medium	Medium	Medium	Medium
Signal/Collect (Stutz et al., 2016)	Hybrid	Medium	High	Medium	High	Medium	Medium
Trinity (Shao et al., 2013)	Synchronous	High	High	High	High	High	High

To enhance the parallelism of graph data processing, data partitioning and task scheduling strategies play a crucial role. Data partitioning strategies involve dividing large-scale graph data into smaller subgraphs, which can be processed in parallel to optimize computational performance and resource utilization. For example, using hash techniques to distribute nodes across different processing units can evenly distribute the workload, reducing processing bottlenecks (Chen & Ran, 2019). Task scheduling focuses on effectively allocating tasks among compute nodes to minimize processing time and communication overhead. For instance, priority-based scheduling systems ensure critical tasks are executed first, thereby optimizing overall execution efficiency (Khan et al., 2019). These techniques support the efficiency and scalability of large-scale graph data processing systems when dealing with complex graph structures.

For instance, GraphLab employs the GAS (Gather-Apply-Scatter) model, which further subdivides the computation

process by executing specific operations at each vertex to increase concurrency. Moreover, systems like GraphX, PowerGraph, and others combine different data partitioning strategies and storage models to handle large-scale graph data, thereby improving processing performance and scalability (Low et al., 2012; Gonzalez et al., 2012).

4.3. Challenges of Parallel Processing Specific to Edge Computing

In edge computing environments, parallel processing faces the dual challenges of resource limitations and network latency. Edge devices typically have limited computing power and storage capacity, which restricts the scale and complexity of parallel processing that can be executed. Additionally, the network connections of edge devices may be unstable or have limited bandwidth, increasing the difficulty of data synchronization and managing concurrent updates.

To develop effective parallel processing strategies suitable for edge computing environments, it is necessary to overcome these device and network limitations. This can be achieved by optimizing data transmission protocols and adopting lightweight parallel algorithms that are more suitable for low-power devices (Luo et al., 2021; Yang et al., 2019). These adaptations help to ensure that even within constrained environments, parallel processing can still effectively support the real-time and distributed nature of edge computing applications.

5. Challenges and Opportunities in Edge Computing Environments

5.1. Resource Limitations Impacting Parallel Processing

Resource constraints in edge computing environments have a significant impact on parallel processing. As described by Chen J. & Ran X. (2019), elastic resource scheduling strategies are particularly important in edge computing because they allow for the dynamic adjustment of resource allocation based on actual workload, optimizing resource use and reducing costs. However, the computational power and storage capacity of edge devices are typically limited, which restricts the types and scales of parallel processing tasks that can be performed. This limitation can lead to reduced processing efficiency and effectiveness.

These constraints necessitate the development of more sophisticated resource management technologies that can efficiently distribute and utilize available resources. Leveraging technologies such as containerization and microservices can help in dynamically scaling applications according to the available computing resources. Furthermore, developing lightweight and efficient algorithms that can perform under constrained conditions is crucial. This also opens up opportunities for innovation in designing software and hardware that are specifically optimized for low-power and low-resource environments, thereby enhancing the capabilities of edge devices in handling complex tasks.

5.2. Data Communication and Latency Issues

Data communication and network latency have a direct impact on the performance of parallel processing. In edge computing environments, while the decentralization of data processing tasks can reduce the load on single points and increase response times, it is essential to effectively manage the synchronization and consistency of data across devices. Khan et al. (2019) discussed the implementation of resource allocation strategies based on priority in resource-constrained environments, ensuring that critical applications and services receive the necessary resources. They also highlighted the potential issues of data latency and synchronization that this strategy might introduce.

In scenarios where data must be consistently synchronized across numerous edge devices, the inherent network variability can lead to significant challenges. Techniques such as data caching and predictive data analytics can be employed to mitigate the impact of latency. Caching frequently accessed data on local devices reduces the need for repeated data retrievals across the network, thereby decreasing latency. Predictive analytics can preemptively adjust resource allocation based on anticipated data flows and processing needs, thus enhancing overall system responsiveness.

Moreover, the development of more advanced network protocols that can dynamically adjust to varying network conditions is crucial. These protocols can help in prioritizing traffic for critical applications and dynamically managing bandwidth to ensure that latency-sensitive tasks are processed in a timely manner. Such advancements in communication technologies and network management play a critical role in enabling effective parallel processing in edge computing environments, ensuring that even under conditions of high network latency, data integrity and processing efficiency are maintained.

5.3. Security and Privacy Protection Strategies

In edge computing environments, maintaining security and privacy is a critical consideration for effective parallel processing. The transmission of data across multiple nodes increases the risk of interception and tampering. Therefore, implementing effective security measures, such as data encryption and access control, becomes crucial to ensuring data security and protecting user privacy. The research by Hong C. H. & Varghese B. (2019) highlights that resource sharing and isolation strategies not only enhance resource utilization efficiency but also ensure the security and stability of the system, which is particularly important for handling sensitive or private data.

6. Current Research Status and Future Prospects

In recent years, significant progress has been made in the technology for processing graph data within edge computing environments. Research has focused on developing parallel graph data processing algorithms that can operate efficiently on resource-constrained edge devices. For example, the Bulk Synchronous Parallel (BSP) model has been widely applied in such environments, supporting complex graph computation tasks (Wang et al., 2019). These advancements provide a solid foundation for the effective processing of large-scale graph data, especially in resource-limited edge computing settings.

Despite some progress, there are still many challenges in processing large-scale graph data in edge computing environments, with resource limitations remaining a major obstacle affecting processing capabilities and efficiency. Additionally, data communication issues, especially data transmission delays and synchronization problems, have a significant impact on parallel processing performance. Security and privacy also remain critical unresolved issues in edge computing environments, particularly when dealing

with sensitive or private data (Chen & Ran 2019; Khan et al. 2019).

Future research will need to develop more efficient resource scheduling strategies to optimize the use of computing resources in edge computing environments; design low-latency data communication mechanisms to enhance the real-time nature of data processing; and strengthen data security and privacy protection mechanisms, especially in environments with multiple users and tasks. Furthermore, as artificial intelligence and machine learning technologies advance, using these technologies to improve the concurrent processing strategies of graph data to enhance processing efficiency and accuracy will also be an important direction for future research (Luo et al. 2021; Yang et al. 2019).

7. Conclusion

This paper has extensively explored the key technologies and challenges associated with graph data processing in edge computing environments, with a focus on parallel computing models, optimization techniques, and specific processing challenges. Research indicates that despite numerous challenges, significant progress has been made in the graph data processing technology of edge computing through continuous technological innovation and optimization.

This paper is significant for understanding graph data processing in edge computing environments. It not only systematically summarizes current research achievements but also points out future directions, providing valuable reference and guidance for researchers in this field. Furthermore, it emphasizes the importance of implementing high-throughput parallel processing strategies in edge computing environments and the key technologies and strategies required to achieve this goal. This provides a direction for future research in this area, aiming to advance edge computing technologies and offer more effective solutions for processing complex graph data.

Acknowledgement

This work is supported by the Scientific Research Foundation of Guangzhou [grant number: 202201011667], Guangdong Provincial Education Science Foundation [grant number: 2023GXJK407], the Philosophy and Social Science Foundation of Guangzhou [grant number: 2021GZGJ145], and Guangdong Province Undergraduate University Teaching Quality and Teaching Reform Project Construction Project (2022SJJXGG992).

References

[1] Wang, S., Cao, J., Wang, H., & Jin, Q. (2020). "Graph Processing on Edge Computing Platforms: A Survey". IEEE Access, 8, 57045-57063.

[2] Satyanarayanan, M. (2017). "The Emergence of Edge Computing". Computer, 50(1), 30-39.

[3] Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). "Edge Computing: Vision and Challenges". IEEE Internet of Things Journal, 3(5), 637-646.

[4] Chen, M., & Ran, X. (2019). "Deep Learning With Edge Computing: A Review". Proceedings of the IEEE, 107(8), 1655-1674.

[5] Jeong, E., Kim, H., & Park, S. (2020). "Efficient Distributed Graph Processing under Edge Computing Environment". IEEE Transactions on Parallel and Distributed Systems, 31(2), 423-437.

[6] Gupta, H., Vahid Dastjerdi, A., Ghosh, S. K., & Buyya, R. (2017). "iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in the Internet of Things, Edge and Fog Computing Environments". Software: Practice and Experience, 47(9), 1275-1296.

[7] Zhou, Z., Chen, X., Li, E., Zeng, L., Luo, K., & Zhang, J. (2019). "Edge Intelligence: Paving the Last Mile of Artificial Intelligence with Edge Computing". Proceedings of the IEEE, 107(8), 1738-1762.

[8] Yang, L., Cao, J., Yuan, Y., Li, T., Han, A., & Luo, M. (2023). "Real-time Data Processing at the Edge: Opportunities and Challenges". IEEE Network, 37(2), 112-120.

[9] Sahni, Y., Cao, J., & Zhang, S. (2019). "Edge Mesh: A New Paradigm to Enable Distributed Intelligence in Internet of Things". IEEE Access, 7, 82527-82539.

[10] Khan, L., Yaqoob, I., Hashem, I. A. T., Inayat, Z., Mahmoud Ali, W. K., Alam, M., & Shiraz, M. (2019). "Edge Computing: A Survey". Future Generation Computer Systems, 97, 219-235.

[11] Zhou, Z., Chen, X., Li, E., Zeng, L., Luo, K., & Zhang, J. (2019). Edge intelligence: paving the last mile of artificial intelligence with edge computing. Proceedings of the IEEE.

[12] Yang, H., Pan, H., & Ma, L. (2023). A review on software-defined content delivery network: A novel combination of CDN and SDN. IEEE Access.

[13] Wang, L., et al. (2014). Big data bench: A big data benchmark suite from internet services. 2014 IEEE 20th International Symposium on High-Performance Computer Architecture (HPCA). IEEE.

[14] Wang, P., Yao, C., Zheng, Z., Sun, G., & Song, L. (2019). Joint task assignment, transmission, and computing resource allocation in multi-layer mobile edge computing systems. IEEE Internet of Things Journal.

[15] Sahni, Y., Cao, J., & Yang, L. (2019). Data-aware task allocation for achieving low latency in collaborative edge computing. IEEE Internet of Things Journal.

[16] Ren, J., He, Y., Huang, G., Yu, G., Cai, Y., & Zhang, Z. (2019). An edge-computing-based architecture for mobile augmented reality. IEEE Network.

[17] Ning, Z., Kong, X., Xia, F., Hou, W., & Wang, X. (2019). Green and sustainable cloud of things: Enabling collaborative edge computing. IEEE Communications Magazine, 57(1), 72-78.

[18] Low, Y. (2013). GraphLab: A distributed abstraction for large-scale machine learning [Dissertation]. University of California, Berkeley.

[19] Ren, S. Q., He, K. M., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards real-time object detection with region proposal networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(6), 1137–1149.

[20] Luong, M. T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (pp. 1412–1421). Association for Computational Linguistics.

[21] Wu, Y. H., Schuster, M., Chen, Z. F., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv:1609.08144.

[22] Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A. R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., & Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four

research groups. IEEE Signal Processing Magazine, 29(6), 82–97.

[23] Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., & Battaglia, P. (2018). Graph networks as learnable physics engines for inference and control. In Proceedings of the 35th International Conference on Machine Learning (pp. 4470–4479). PMLR.

[24] Battaglia, P. W., Pascanu, R., Lai, M., Rezende, D. J., & Kavukcuoglu, K. (2016). Interaction networks for learning about objects, relations, and physics. In Proceedings of the 30th International Conference on Neural Information Processing Systems (pp. 4509–4517). NIPS.

[25] Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Representation learning on graphs: Methods and applications. IEEE Data Engineering Bulletin, 40(1), 52–74.

[26] Brandes, U., Gaertler, M., & Wagner, D. (2003). Experiments on graph clustering algorithms. In Proceedings of the 11th European Symposium on Algorithms (pp. 568–579). Springer.

[27] Fout, A., Byrd, J., Shariat, B., & Ben-Hur, A. (2017). Protein interface prediction using graph convolutional networks. In Proceedings of the 31st International Conference on Neural Information Processing Systems (pp. 6533–6542). NIPS.

[28] Jeong, C., Jang, S., Park, E., & Choi, S. (2020). A context-aware citation recommendation model with BERT and graph convolutional networks. Scientometrics, 124(3), 1907–1922.

[29] Malewicz, G., Austern, M. H., Bik, A. J. C., et al. (2010). Pregel: A system for large-scale graph processing. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (pp. 135-146). ACM.

[30] Valiant, L. G. (1990). A bridging model for parallel computing. Communications of the ACM, 33(8), 103-111.

[31] Avery, C. (2011). Giraph: Large-scale graph processing infrastructure on hadoop. Proceedings of the Hadoop Summit, 11(3), 5-9.

[32] Neo4j, Inc. (2022). Neo4j graph database platform. Retrieved from https://neo4j.com/

[33] Salihoglu, S., & Widom, J. (2013). GPS: A graph processing system. In Proceedings of the 25th International Conference on Scientific and Statistical Database Management, 22.

[34] Seo, S., Yoon, E. J., Kim, J., et al. (2010). Hama: An efficient matrix computation with the MapReduce framework. In Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science (pp. 721-726).

[35] Zhou, C., Gao, J., Sun, B., et al. (2014). MOCgraph: Scalable distributed graph processing using message online computing. VLDB Endowment, 8(4), 377-388.

[36] Yan, D., Cheng, J., Lu, Y., et al. (2014). Blogel: A block-centric framework for distributed computing on real-world graphs. VLDB Endowment, 7(14), 1981-1992.

[37] Quamar, A., Deshpande, A., & Lin, J. (2016). NScale: neighborhood centric large-scale graph analytics in the cloud. The VLDB Journal, 25(2), 125-150.

[38] Low, Y., Gonzalez, J. E., Kyrola, A., et al. (2014). GraphLab: A new framework for parallel machine learning. Computer Science, 1408.2041.

[39] Low, Y., Bickson, D., Gonzalez, J., et al. (2012). Distributed GraphLab: A framework for machine learning and data mining in the cloud. VLDB Endowment, 5(8), 716-727.

[40] Xie, C., Chen, R., Guan, H., et al. (2015). Sync or async: time to fuse for distributed graph-parallel computing. ACM SIGPLAN Notices, 50(8), 194-204.

[41] Gonzalez, J. E., Xin, R. S., Dave, A., et al. (2014). GraphX: Graph processing in a distributed dataflow framework. In Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (pp. 599-613).

[42] Stutz, P., Strebel, D., Bernstein, A., et al. (2016). Signal/Collect. Semantic Web, 7(2), 139-166.

[43] Zhang, Y., Gao, Q., Gao, L., et al. (2014). Maiter: An asynchronous graph processing framework for delta-based accumulative iterative computing. IEEE Transactions on Parallel and Distributed Systems, 25(8), 2091-2100.

[44] Ding, X., Chen, R., & Chen, H. (2015). Hybrid computing mode in distributed graph computing framework. Journal of Chinese Computer Systems, 36(4), 665-670.

[45] Jing, N., Xue, J., Qu, Z., et al. (2014). SpecGraph: A distributed graph processing system for dynamic result based on concurrent speculative execution. Journal of Computer Research and Development, (S1), 155-160.

[46] Shao, B., Wang, H., & Li, Y. (2013). Trinity: A distributed graph engine on a memory cloud. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (pp. 505-516).