A Comparative Study of Streaming Graph Processing Systems

Shaoshuai Du$^{1\,[0009-0006-4991-3494]}$, Jože Rožanec$^{2\,[0000-0002-3665-639X]}$, Ana-Lucia Varbanescu$^{2\,[0000-0002-4932-1900]}$, and Andy D. Pimentel$^{1\,[0000-0002-2043-4469]}$

[1] University of Amsterdam, Amsterdam, Netherlands
`{s.du, a.d.pimentel}@uva.nl`
[2] University of Twente, Enschede, Netherlands
`{a.l.varbanescu, joze.rozanec}@utwente.nl`

Abstract. Streaming Graph Processing Systems (SGPSs) are essential for real-time analytics on dynamic graphs in domains such as social networks and knowledge graphs. Despite increasing interest and a growing number of SGPSs, practical performance comparisons remain limited due to architectural heterogeneity and inconsistent evaluation practices.
This work presents a unified benchmarking workflow for empirically evaluating SGPSs across latency, resource usage, and energy efficiency. We select three representative systems—GraphStream (GS), GraphBolt (GB), and RisGraph (RG)—to reflect non-incremental and incremental design philosophies, and evaluate them using two common graph algorithms (BFS and SSSP) across diverse real-world datasets.
Our results reveal distinct trade-offs: RG excels in low-latency tasks but supports only monotonic algorithms; GB achieves strong batch performance and broader algorithm support; GS maintains stable latency at the cost of higher memory. We also conduct the first empirical comparison of SGPS energy efficiency.
Our findings offer practical guidance for system selection and provide a reproducible foundation for future SGPS benchmarking and optimization.

Keywords: Streaming Graph Processing Systems (SGPSs) · Streaming Graphs · Benchmarking · Architecture Comparison.

1 Introduction

Streaming graph processing systems (SGPSs) have emerged as a critical infrastructure for real-time analytics on dynamic graphs, supporting applications ranging from fraud detection to social network monitoring [1]. Unlike static graph frameworks, SGPSs are designed to handle continuous data changes and real-time queries, enabling dynamic updates and immediate insights into evolving network states [13]. Such capabilities are critical for applications where timely responses to graph mutations—such as newly formed links, disappearing connections, or shifting node attributes—directly influence decision-making and user experience.

Over the past few years, a number of streaming graph systems have been proposed [14, 1, 12, 4], each embodying different design philosophies. Some emphasize fine-grained update granularity, while others adopt batch-oriented or hybrid processing models.

Although several surveys have captured the landscape of SGPS development and some efforts have been made to benchmark selected systems [4, 6], there is a lack of a unified and rigorous empirical comparison across these diverse designs. This makes it difficult for system designers and practitioners to understand the practical trade-offs involved.

Fine-grained metrics such as latency are challenging to evaluate in an automated and comparable manner due to the fundamentally different architectures and diverse algorithms across systems. Nevertheless, understanding these tradeoffs is crucial: for system designers, to guide architectural choices; and for users, to select the most suitable system for their needs.

We reviewed a range of existing SGPSs[3] and selected three representative systems—RisGraph (RG) [7], GraphBolt (GB) [12], and GraphStream (GS) [5], based on their open-source availability, algorithmic capabilities, and architectural diversity. These systems not only support common algorithms for comparison but also represent distinct design paradigms. GS adopts a recomputation-based model, while RG and GB are incremental systems. RG supports only monotonic algorithms, whereas GB additionally supports non-monotonic algorithms. These systems cover key design dimensions relevant to our evaluation, including update models, computation paradigms, and interface types.

We further design a benchmarking workflow that enables the measurement of multiple performance metrics, including latency, resource usage, and energy consumption. Based on this benchmarking workflow, we conduct a fine-grained empirical study.

Our key contributions are summarized as follows:

– We identify the lack of unified empirical comparison as a critical bottleneck in understanding streaming graph system design.
– We develop a comprehensive benchmarking framework that covers both non-incremental and incremental systems, simulating realistic graph dynamics and query patterns.
– Our evaluation reveals that incremental systems are well-suited for low-latency tasks, while non-incremental systems show potential in high-throughput scenarios. We also investigate resource usage and energy efficiency, finding that RG and GB are more efficient than GS.

The remainder of this paper is organized as follows. Section 2 reviews related work, and Section 3 introduces the background and compares three representative SGPS architectures. Section 4 describes the benchmarking workflow, datasets, algorithms, and evaluation metrics. Section 5 presents the experimental results and analysis. Section 6 concludes the paper and outlines future directions.

2 Related Work

Several benchmarking frameworks have been proposed for evaluating streaming graph processing systems (SGPSs). GraphTides [6] provides a benchmark targeting latency and scalability, but its limited extensibility restricts support for new workloads and emerging system designs. G-Bench [10] offers a more comprehensive methodology, proposing best practices for fair evaluation across SGPSs.

From a conceptual standpoint, Besta et al. [3] present a detailed taxonomy of SGPSs, categorizing systems by update models, computational paradigms, and consistency guarantees. While comprehensive, this work remains theoretical and lacks empirical validation. Our system selection and evaluation strategy are informed by such taxonomies, but we focus on architectural traits that directly impact performance, including latency, resource usage, and energy efficiency.

In parallel, hardware accelerators such as Graphicionado [9] and MEGA [8] offer significant performance benefits for dynamic graph processing through custom architectures. However, these approaches target different levels of the system stack and are not directly comparable to software-based SGPSs.

Our work complements these prior efforts by introducing a benchmarking workflow with fine-grained performance breakdowns and by conducting an in-depth empirical evaluation of three representative SGPSs across diverse datasets and workloads.

3 Background and System Overview

3.1 Streaming Graph Processing

Streaming graph processing refers to the continuous computation over evolving graph structures. Existing systems differ in how they represent graph states and propagate updates.

Each system has its own support for update types, such as node addition/deletion, edge addition/deletion, or weight modification [3]. Some systems support per-update processing with a batch size of one, while others allow configurable batch sizes and apply optimizations accordingly [7, 12, 5]. Certain systems push updates to the engine based on a threshold number of accumulated updates, while others do so periodically at fixed intervals.

Some systems rely on full recomputation, while others perform incremental computation [3]. It is also worth noting that different systems support different subsets of algorithms due to their underlying processing models.

The diversified landscape of streaming graph processing offers a variety of options, but also presents significant challenges in understanding, comparing, and selecting suitable systems.

3.2 Target Systems: Design Highlights

As stated in Table 1, to conduct a focused and meaningful evaluation, we select systems based on three main criteria: **open-source availability**, **algorithmic**

Table 1: Comparison of target systems in terms of computation model and supported algorithm types

System	**Computation Model**	**Supported Algorithms**
RG (RisGraph)	Incremental, Parallel Hybrid	Monotonic only
GB (GraphBolt)	Incremental, Batch-based (BSP)	Monotonic and Non-monotonic
GS (GraphStream)	Per-update Recomputation	Basic iterative algorithms

coverage, and **architectural diversity**. These systems not only implement widely-used graph algorithms, enabling consistent comparison, but also exemplify different system designs. Specifically, GS follows a recomputation-based strategy, while both RG and GB adopt incremental update mechanisms. Among the two incremental systems, RG is limited to monotonic algorithms, whereas GB extends support to non-monotonic workloads. Collectively, the three systems span core dimensions of interest in our study, including update handling strategies, computation models, and API-level design.

RG leverages inter-update parallelism and RDMA, enabling fast updates via localized data structures. GB employs a BSP-based, batch-oriented model with dependency-aware refinement. GS recomputes the entire graph per query without batching, prioritizing simplicity but at higher cost. Together, these systems span key architectural dimensions including update granularity, dependency tracking, and query consistency.

For streaming systems, throughput and latency are primary concerns. Additionally, system sustainability—particularly under long-term, high-frequency workloads—is an important yet underexplored aspect. Currently, there is a lack of practical, fine-grained comparison across systems. These architectural differences motivate a systematic evaluation across latency, throughput, and memory usage.

4 Evaluation Methodology

To systematically evaluate representative SGPSs, this section presents our benchmarking methodology, including the experimental platform, datasets, algorithms, and evaluation metrics.

4.1 Benchmarking Workflow

We design a benchmarking workflow that breaks down the key stages of how different systems perform updates and execute computations, as illustrated in Fig. 1. For all systems, *update latency* refers to the time required to apply graph mutations. *Query latency*, however, differs by system type: non-incremental systems perform full recomputation, while incremental systems update dependencies and apply localized refinements.

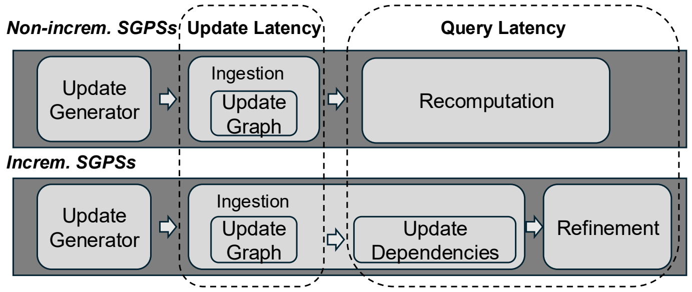


Fig. 1: Benchmarking Workflow and Latency Breakdown.

4.2 Datasets and Graph Characteristics

Table 2: Graphs used in the experiments.

Graphs	Abbr.	Vertices	Edges	Type	Root node
Wikipedia [11]	WK	2.39M	5.02M	Talk	2
LiveJournal [11]	LJ	4.85M	69.0M	Social	0
StackOverflow [11]	SO	2.60M	63.5M	Q&A	13
Pokec [11]	PC	1.63M	30.6M	Social	2
Berkeley-Stanford [11]	BS	685K	7.60M	Web	1

To ensure fairness and simulate realistic streaming scenarios, we follow the approach used in KickStarter [14], GraphBolt [12], and RisGraph [7]. Specifically, we shuffle each dataset's edge list, load 80% of the edges as the initial graph, and treat the remaining 20% as insertions. The same batch of updates is used across different systems and repeated experiments. An equal number of edges from the loaded portion are randomly selected for deletion. Each update batch consists of 10,000 operations: 5,000 additions and 5,000 deletions.

The datasets used are summarized in Table 2, covering diverse domains such as social networks, Q&A platforms, and web graphs, with varied structural characteristics. The "Root Node" column indicates the starting point for traversal algorithms (e.g., BFS, SSSP).

Graph Characteristics: Table 3 summarizes the degree distribution using five-number summaries (Min, Q1, Median, Q3, Max) and the Gini coefficient to reflect skewness. These structural properties significantly impact SGPS performance, especially under parallelism. Highly skewed datasets (e.g., SO, WK) may cause load imbalance and overhead, while more uniform datasets (e.g., PC) yield more stable performance.

Table 3: Degree distribution summary of the datasets.

Dataset	Min Degree	Q1	Median	Q3	Max Degree	Gini Coeff
WK	1	1	1	2	100032	0.74
LJ	1	3	8	28	22889	0.71
SO	1	2	7	22	194806	0.85
PC	1	5	17	49	20518	0.62
BS	1	5	9	19	84290	0.66

4.3 Algorithms

Breadth-First Search (BFS) and Single-Source Shortest Path (SSSP) are fundamental graph traversal algorithms commonly used in graph processing benchmarks. We select these algorithms to ensure a fair comparison across systems with differing algorithmic capabilities. For instance, RisGraph supports only monotonic algorithms, while GraphBolt supports both monotonic and non-monotonic ones. BFS and SSSP are supported by all evaluated systems, making them suitable for consistent and comparable evaluation.

4.4 Metrics

To evaluate SGPS performance, we consider three key metrics:

Latencies: **Query latency** is measured per batch, indicating real-time responsiveness. **Addition** and **deletion latency** represent the average time to insert or remove edges, computed by dividing total addition/deletion time by the number of operations. The overall update latency Δ_u combines addition (Δ_a) and deletion latency (Δ_d), weighted by the number of operations (N_a, N_d), as shown in Equation 1 and illustrated in Fig. 1.

$$\Delta_u = \Delta_a \cdot \frac{N_a}{N_a + N_d} \; + \; \Delta_d \cdot \frac{N_d}{N_a + N_d} \tag{1}$$

Resource Utilization: **CPU usage** reflects processing efficiency, while **memory usage** indicates scalability, with lower usage generally preferred.

Energy Consumption: Measured at the CPU and system levels, this metric captures the energy efficiency of SGPSs, particularly relevant for resource-constrained environments.

4.5 Hardware

We conducted our experiments on a cluster node [2] equipped with two AMD EPYC 7282 processors, each with 16 cores and support for hyper-threading, providing a total of 64 logical CPUs. The system includes 256 GB of main memory and 128 GB of swap space.

5 Performance Analysis of Three SGPSs

We evaluate RG, GB, and GS across multiple workloads, analyze how batch size affects performance, and highlight key observations for future system design. *To strengthen the validity of our observations, we repeat each experiment five times and report the mean and standard deviation. Error bars are included in all latency-related plots.*

5.1 Performance Comparison

Based on the workflow in Section 4.1, we are able to fairly compare GS, RG, and GB, which represent the non-incremental SGPS and incremental ones respectively.

Latencies Fig. 2 shows the latency results of RG, GB, and GS running BFS and SSSP on five datasets, comparing query, update, addition, and deletion operations. We set the batch size to 1×10^4

RG consistently achieves the lowest query latency, followed by GB and then GS, due to its parallel design and optimization for monotonic algorithms. GB adds dependency management overhead, while GS performs full recomputation, leading to the highest latency. In SSSP on LJ, GS is over three orders of magnitude slower than RG.

Update latency follows a similar trend. RG benefits from its Indexed Adjacency Lists for fast edge access and modification, while GB is a bit slower. GS remains the slowest due to its implementation. For instance, in BS-SSSP, GS reaches over 10^7 ns, while RG stays in the 10^6 range. However, the difference in update latency across systems is less pronounced than that in query latency.

Addition and deletion costs vary. RG has higher addition cost due to index updates but handles deletions efficiently. GB favors additions but incurs more overhead on deletions. GS shows the highest variability, with deletions on skewed datasets like WK and SO exceeding 10^7 ns due to supernode effects. SSSP consistently shows higher latency than BFS due to its complexity. GB exhibits the smallest BFS–SSSP gap, thanks to its dependency-tracking mechanism.

Resource Usage Fig. 3 compares CPU and memory usage of the three systems running BFS and SSSP on five datasets.

RG shows the highest CPU utilization, attributed to its parallel architecture and efficient memory design optimized for rapid dependency updates. GS consistently uses the most memory due to its in-memory graph structures and lack of dependency-aware optimizations. While implementation languages contribute slightly, the primary factor is architectural: RG and GB leverage efficient, incremental data structures, whereas GS's classical design leads to higher memory usage and recomputation costs.

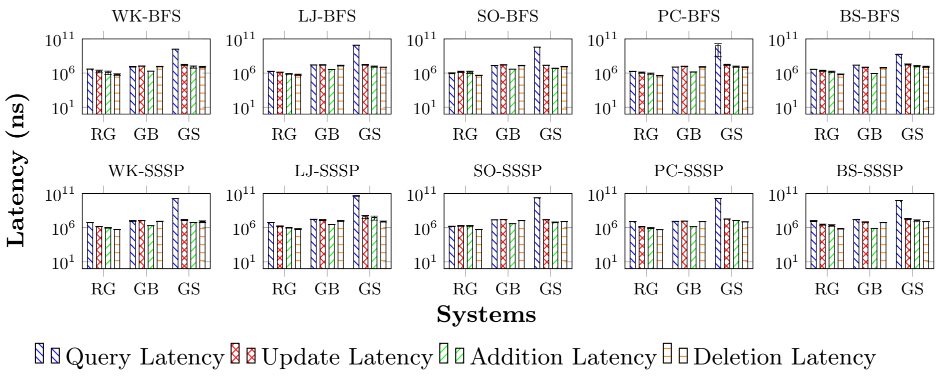
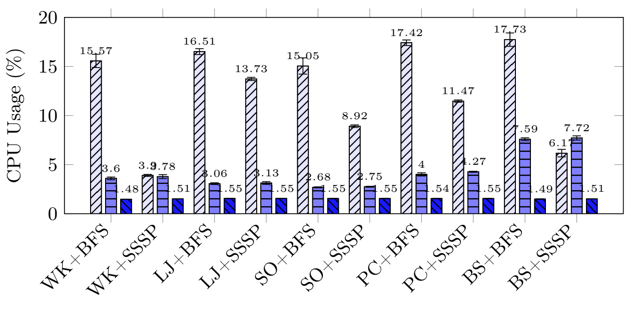
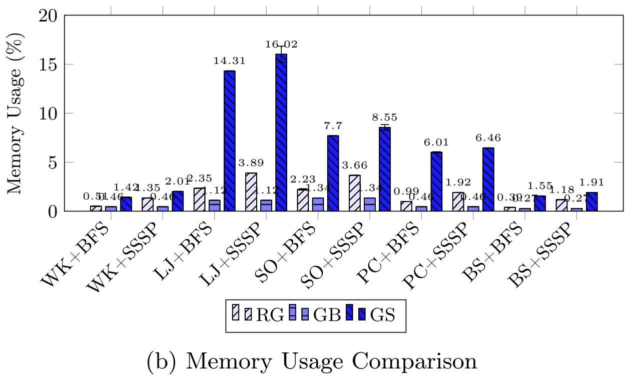


Fig. 2: Latency Comparison of RG, GB, and GS Across Five Datasets.



(a) CPU Usage Comparison



(b) Memory Usage Comparison

Fig. 3: Resource Usage Comparison of Different Graph Systems Across Five Datasets. (a) CPU Usage Comparison (BFS/SSSP, total usage across all CPU cores).(b) Memory Usage Comparison (BFS/SSSP).

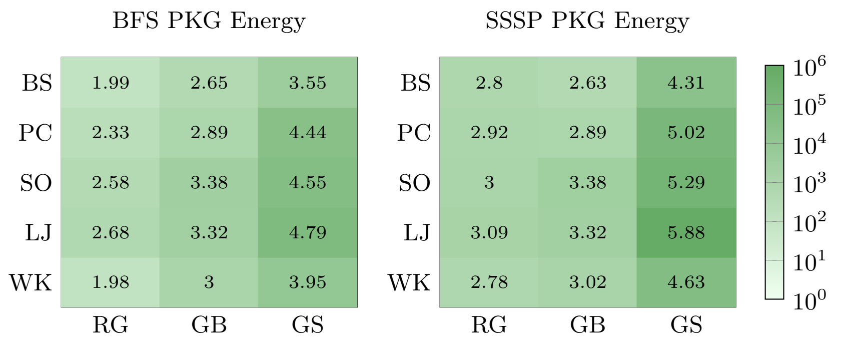


Fig. 4: Energy Consumption Heatmap (J) for RG, GB, and GS Across Five Datasets under BFS and SSSP Operations.

Energy Consumption Fig. 4 compares the package (PKG) energy consumption of RG, GB, and GS for BFS and SSSP algorithms on five datasets. PKG energy represents the total energy consumed by the CPU, including the computational load of cores and additional components like memory controllers, providing a comprehensive view of energy efficiency. Energy consumption was measured using the LIKWID toolkit, which uses hardware performance counters for reasonably accurate energy estimates.

Among the systems, RG exhibits the lowest PKG energy consumption due to efficient parallelized computation, followed by GB with moderate consumption. GS consumes the most energy, reflecting its reliance on computationally intensive in-memory structures.

5.2 Influence of Batch Size on Latencies

To evaluate system behavior under varying update intensities, we test with batch sizes of 10^1, 10^3, and 10^5, representing interactive, moderate, and high-throughput workloads, respectively. This setup enables us to observe how latency evolves under increasing ingestion pressure.

For update, addition, and deletion operations, both RG and GB show clear latency reduction as batch size increases—particularly in GB, where per addition latency drops from over 10^5 ns to below 10^2 ns. This improvement reflects reduced synchronization and amortized overhead in batch processing. In contrast, GS exhibits flat or inconsistent trends, as it lacks batch-aware optimizations.

Query latency behaves differently. RG's query latency increases significantly with batch size—by over three orders of magnitude in BFS—due to cumulative dependency processing. GB shows a more moderate rise, while GS remains largely constant across batches because it performs full recomputation regardless of update size.

Overall, RG excels under small batches with low-latency updates, whereas GB is more robust under large batches, thanks to its dependency resolution

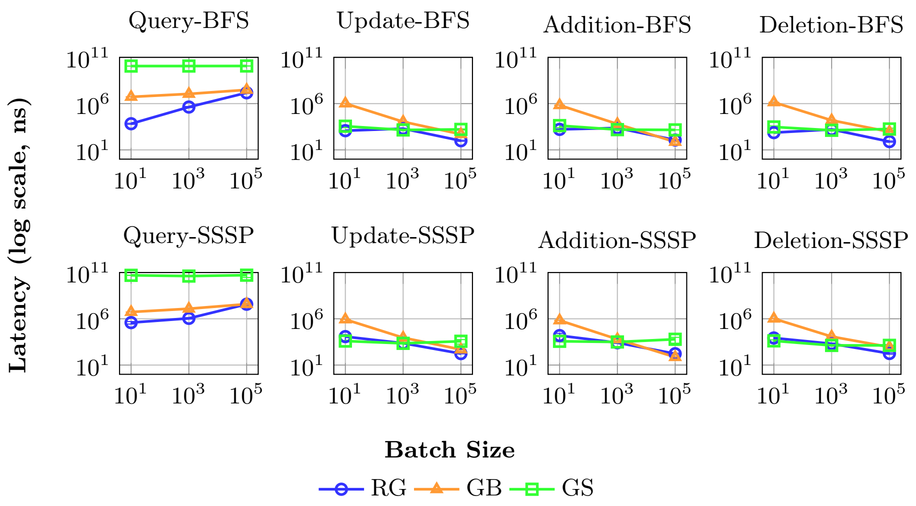


Fig. 5: Latency Trends Across Different Batch Sizes for RG, GB, and GS on the LJ, Averaged per Operation within Each Batch.

and batching mechanisms [12]. GS, without incremental design, shows limited adaptability. These results highlight the trade-offs between fine-grained efficiency and batch scalability in SGPS design.

5.3 Key Findings and Design Implications

System-level summary. Empirical results reveal clear trade-offs among RG, GB, and GS. RG achieves the lowest latency and energy consumption, but uses more CPU and supports only monotonic algorithms. GB scales better with batch size, supporting broader workloads at stable cost. GS, despite higher latency and resource usage, handles complex workloads and may be more suitable at very high throughput (e.g., batch size $< 10^7$).

Our findings suggest three directions for future SGPS optimization: ***Hybrid update models.*** To handle varying throughput demands, SGPSs should integrate both incremental and non-incremental mechanisms. Incremental updates are preferable for small, frequent batches, while non-incremental processing is more stable for large batch sizes. ***Adaptive update handling.***The cost of additions and deletions varies with graph structure, which may shift over time in streaming settings. Designing distribution-aware and workload-adaptive update strategies is key to maintaining performance under dynamic conditions. ***Energy-aware architectures.*** For low-power or resource-constrained scenarios, incremental SGPSs may offer energy advantages due to their lower computational overhead, and could be prioritized where applicable, subject to validation on target hardware.

6 Conclusion and Future Work

In this work, we addressed the lack of empirical understanding of streaming graph processing systems by conducting a systematic evaluation of representative platforms.

We introduced a unified benchmarking workflow to compare non-incremental and incremental SGPS architectures and conducted a detailed analysis of three systems—RG, GB, and GS—across multiple datasets and workloads. Our results reveal key trade-offs in latency and resource usage. In particular, we provide the first empirical comparison of energy efficiency across SGPSs.

These insights provide practical guidance for system designers and users in selecting appropriate platforms and optimizing performance under varying workload scenarios. The proposed benchmarking methodology offers a reproducible and extensible foundation for future SGPS evaluation.

In future work, we plan to extend the benchmark to additional systems, incorporate more complex algorithm classes such as subgraph pattern queries, and evaluate system behavior under dynamic and bursty update streams.

References

1. Aggarwal, C.C., Subbian, K.: Evolutionary network analysis: A survey. ACM Computing Surveys (CSUR) **47**(1), 10 (2014)
2. Bal, H.E., Epema, D.H.J., de Laat, C., van Nieuwpoort, R., Romein, J.W., Seinstra, F.J., Snoek, C., Wijshoff, H.A.G.: A medium-scale distributed system for computer science research: Infrastructure for the long term. Computer **49**(5), 54–63 (2016). https://doi.org/10.1109/MC.2016.127, https://doi.org/10.1109/MC.2016.127
3. Besta, M., Fischer, M., Kalavri, V., Kapralov, M., Hoefler, T.: Practice of streaming processing of dynamic graphs: Concepts, models, and systems. IEEE Trans. Parallel Distributed Syst. **34**(6), 1860–1876 (2023). https://doi.org/10.1109/TPDS.2021.3131677, https://doi.org/10.1109/TPDS.2021.3131677
4. Besta, M., Hoefler, T.: Dynamic graph processing with high-performance update mechanisms. IEEE Transactions on Parallel and Distributed Systems (2021)
5. Dutot, A., Guinand, F., Olivier, D., Pigné, Y.: Graphstream: A tool for bridging the gap between complex systems and dynamic graphs. In: Emergent Properties in Natural and Artificial Complex Systems. Satellite Conference within the 4th European Conference on Complex Systems (ECCS'2007) (2007)
6. Erb, B., Meißner, D., Steer, B.A., Cuadrado, F., Margan, D., Pietzuch, P.: Graphtides: A framework for evaluating stream-based graph processing platforms. In: GRADES-NDA'18: 1st Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA) (2018). https://doi.org/10.1145/3210259.3210262
7. Feng, G., Ma, Z., Li, D., Chen, S., Zhu, X., Han, W., Chen, W.: Risgraph: A real-time streaming system for evolving graphs to support submillisecond per-update analysis at millions ops/s. In: Proceedings of the 2021 International Conference on Management of Data (SIGMOD) (2021). https://doi.org/10.1145/3448016.3457263

8. Gao, C., Afarin, M., Rahman, S., Abu-Ghazaleh, N.B., Gupta, R.: MEGA evolving graph accelerator. In: Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2023, Toronto, ON, Canada, 28 October 2023 - 1 November 2023. pp. 310–323. ACM (2023). https://doi.org/10.1145/3613424.3614260, https://doi.org/10.1145/3613424.3614260
9. Ham, T.J., Wu, L., Sundaram, N., Satish, N., Martonosi, M.: Graphicionado: A high-performance and energy-efficient accelerator for graph analytics. In: 49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2016, Taipei, Taiwan, October 15-19, 2016. pp. 56:1–56:13. IEEE Computer Society (2016). https://doi.org/10.1109/MICRO.2016.7783759, https://doi.org/10.1109/MICRO.2016.7783759
10. Kumar, P., Revillar, S.: G-bench: Fair benchmarking to support innovations in streaming graph systems. In: 2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 179–188. IEEE (2023)
11. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data (Jun 2014)
12. Mariappan, M., Vora, K.: Graphbolt: Dependency-driven synchronous processing of streaming graphs. In: Candea, G., van Renesse, R., Fetzer, C. (eds.) Proceedings of the Fourteenth EuroSys Conference 2019, Dresden, Germany, March 25-28, 2019. pp. 25:1–25:16. ACM (2019). https://doi.org/10.1145/3302424.3303974, https://doi.org/10.1145/3302424.3303974
13. Vaquero, L.M., Cuadrado, F., Ripeanu, M.: Systems for near real-time analysis of large-scale dynamic graphs. arXiv **1410**(1903) (2014)
14. Vora, K., Gupta, R., Xu, G.: Kickstarter: Fast and accurate computations on streaming graphs via trimmed approximations. In: Chen, Y., Temam, O., Carter, J. (eds.) Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2017, Xi'an, China, April 8-12, 2017. pp. 237–251. ACM (2017). https://doi.org/10.1145/3037697.3037748, https://doi.org/10.1145/3037697.3037748