A Roadmap to Graph Analytics

A discussion on research from a panel in ACM Sigmod/Pods 2024

Angela Bonifati (Univ. Lyon 1, CNRS & IUF, France), M. Tamer Ɵzsu (University of Waterloo, Canada), Yuanyuan Tian (Microsoft Gray Systems Lab, USA), Hannes Voigt (Neo4j, Germany), Wenyuan Yu (Alibaba Group, China), Wenjie Zhang (University of New South Wales, Australia)

OPEN CHALLENGES

Graphs are ubiquitous data structures used in a large spectrum of applications, spanning from transportation networks, financial networks, social networks, product-order transactions and biomedical applications [33]. A recent survey on the usage of graph applications from real users has highlighted the fact that analytics is the most time-consuming task as opposed to testing, cleaning and ETL [32].

Despite the industrial needs of graph analytics, the area is still in its infancy needing standardization and integration into current graph systems.

To understand the burden of graph analytics [44], one has to resort to the data processing pipeline behind it. Graph data is manipulated and queried with online transactional processing (OLTP) operations, such as selections, joins and transitive closures expressed in concrete ISO/IEC standard graph query languages [15,16]. While the results of graph queries in the first versions of these standards are sets of relational tuples, their extensions to return and manipulate paths is expected in future versions. In particular, graph OLTP operations would then manipulate paths instead of tuples, as shown in recent work on graph path algebras [6]. The data is further analyzed, enriched, and condensed with online analytical processing (OLAP) operations, such as (1) grouping, aggregating, slicing, dicing, and roll-up, (2) graph algorithms, such as shortest path counting [42], in-betweenness centrality and PageRank and (3) a handful of user-defined functions. These OLAP operations can again be defined on paths and subgraphs, apart from tuples. Finally, graph data is disseminated and consumed by a variety of applications, including machine learning, such as ML libraries and processing frameworks, and large language models (LLMs), under the form of RAG (retrieval-augmented generation) [29].

Graph databases have extensive applications across various industries [26, 39, 40, 43], including fraud detection in finance and insurance (commonly involving graph pattern matching queries), bill-of-materials analysis in manufacturing and supply chain (typically utilizing graph traversal queries), and customer segmentation and recommendation in marketing (often leveraging graph algorithms).

How might forthcoming graph processing systems deliver extensive scalability, efficiency, and versatile querying and analytical functionalities to meet the diverse demands of real-world scenarios?

Whereas standardization is already taking place for graph query and update languages [9], it would also be worthwhile to reach a common understanding of the building blocks of graph algorithms and analytical APIs for graphs. Graph abstractions, such as graph pattern calculus and graph algebra [6,7] for graph transformations and queries, need to cater for analytical operators and to be extended to include linear algebra operators.

Novel graph processing and graph database architectures are then needed to ensure the unification of OLTP and OLAP operations. The data injected in these future graph systems is expected to be of different format and might require multi-model, converged and/or polystore databases [38].

The panel (moderated by Angela Bonifati) discussed the following research questions:

- **Q1:** Is there a demand for more expressive languages and libraries for analyzing relationships in a graph?
- **Q2:** Do we need OLAP/OLTP architectures or their hybrid version (HTAP for graphs) in order to execute graph analytical workloads? Is on cloud better than on premise?
- **Q3:** What are the requirements in terms of scalability, performance and benchmarking?
- **Q4:** Are graph-only stores sufficient or are polystores needed for the future of graph analytics?
- **Q5:** What is needed in terms of DSL and APIs for enabling graph analytics for data science and ML (and LLM) tasks?

- **Q6:** Since graphs are continuously evolving data structures, what is desirable in terms of analytical operators for dynamic, incremental and streaming graphs?

In the remainder, we present discussions for each of the challenges around the development of research and industrial applications for graph analytics.[1]

Q1 – EXPRESSIVENESS

MTO: We want languages that are sufficiently expressive to accomplish what we want. In general, graph query languages should support two fundamental functions: navigation (path queries) and subgraph matching. In addition, they should have the following capabilities: ability to query both the graph topology and the graph properties (assuming we are in the property graph domain), aggregation operations, support for paths as first-class objects, closedness to facilitate query composition similar to relational languages, ability to be mapped to algebraic operators for processing [6], and incremental computation that is important for dynamic and streaming graphs. This should be the minimal capability set, only partially covered by the recent SQL/PGQ [15] and GQL [16].

Related to analytics, it is important to fix what we mean by graph analytics. One definition characterizes it as workloads that (a) traverse the entire graph, and (b) iterate until a fixpoint or a termination condition is reached. These include, for example, PageRank computation, all-paths shortest path, connected components, and many machine learning tasks. This definition is used in contrast to what are called *online queries* that typically access a portion of a graph that can be aided by indexes, and even if they access the entire graph, they are not iterative (e.g., reachability, recursive path queries, single-source shortest path, subgraph matching). If this type of graph analytics workload is targeted, then one set of features is required in the language.

However, there is another definition of graph analytics more aligned with how we define analytics in relational DBMSs: OLAP over graphs. In this case, the language will need to have additional features supporting drill-down, roll-up, slide-and-dice. The language requirements in this case have been discussed by Gómez et al [12].

YT: The short answer is yes—new use cases continually demand more expressive languages and libraries for graph analysis. Graph workloads are diverse, each requires different levels of expressiveness and user interface capabilities. For instance, Graph Traversal, Pattern Matching, and BI use languages like Cypher [10], Gremlin [30], GQL [16], and SPARQL [41], with growing demand for more advanced constructs. Graph algorithms rely on libraries like Pregel API for complex computations, where customization and built-in algorithms are increasingly needed. In Graph ML, libraries are essential for integrating machine learning with graph data. The emerging field of Graph + LLM further emphasizes the need for expressive tools combining graph reasoning with natural language understanding.

While expressiveness is important, ease of use is equally critical. Today, the barrier to entry for working with graphs is still quite high, limiting it to a niche group of graph specialists. To broaden adoption among SQL users, data scientists, and business users, *it's important to meet users where they are*, whether that's through familiar interfaces like SQL (e.g., SQL/PGQ), Python, or even natural language. Powerful yet user-friendly tools can greatly improve adoption and productivity.

HV: In the current analytics space, there is indeed a demand for more expressive languages and libraries to analyze relationships in a graph. For products, this space is highly dynamic, with constant pressure to incorporate the latest machine learning (ML) and artificial intelligence (AI) algorithms. Presently, most analytic capabilities are provided through libraries, which play a crucial role by offering complex algorithms behind relatively simple APIs, e.g. [5]. This allows for high-value, rapid results for customers while maintaining an easy learning curve for developers and users. These libraries are designed with only a few built-in assumptions, making them flexible and easy for vendors to extend. Currently, no alternative abstraction offers the same balance of learning ease and value delivery that libraries provide, making them the go-to solution. While this could evolve in the future, there is little indication of such a shift occurring at the moment.

WY: Yes, there is a clear demand for more expressive languages and libraries for graph analysis, driven by the diversity and fragmentation of its applications. However, increasing expressiveness in languages and libraries comes with costs, including greater complexity and potential performance trade-offs. While there is a clear distinction between tasks like graph traversal, pattern matching, analytics, and what graph neural networks (GNNs) can accomplish, there is also some overlap in their characteristics. Designing languages and libraries that can cater to these diverse needs while maintaining usability and performance is a significant challenge. Looking ahead, natural languages enhanced by large

[1]The following abbreviations are used: MTO (M. Tamer Özsu), YT (Yuanyuan Tian), HV (Hannes Voigt), WY (Wenyuan Yu), WZ (Wenjie Zhang).

language models (LLMs) could provide a promising solution for simplifying complex queries and improving accessibility. Future directions may focus on balancing expressiveness with performance, potentially leveraging LLMs to bridge current gaps.

WZ: Yes, the demand is growing as graph databases expand their capabilities to support more complex and diverse analyses at various levels. Current graph analytics support in main-stream graph platforms can be summarized as: (1) Node-level: Algorithms like centrality, node similarity, and coreness are commonly supported to measure node importance or influence. (2) Path-level: Standard graph traversal methods (DFS/BFS, shortest paths) and more advanced algorithms (MST, Steiner trees) are widely used to study connectivity. (3) Subgraph-level: Community detection, motif counting, and label propagation are essential for identifying clusters or patterns. (4) Learning-oriented: Machine learning integration, such as node embedding and dynamic graph inference, is becoming increasingly important, with graph databases like Neo4j, TigerGraph, Alibaba GraphScope, and Amazon Neptune implementing these algorithms. The demand for more expressive languages and libraries stems from user and domain-specific workloads. Future graph analytics may include supporting new graph types such as bipartite graphs and emerging applications such as integration with LLMs.

Q2 – OLAP/OLTP/HTAP & Cloud

MTO: The answer to both questions is yes. Graph research is divided into three categories: knowledge graphs and semantic web, graph DBMSs executing the online queries, and graph systems executing some understanding of graph analytics. The first category typically uses RDF graphs and SPARQL, while the other two use property graphs. There are real use cases that require a combination of at least two of these categories, so HTAP makes sense. However, it is not clear how to architect these systems.

The move to the cloud is unmistakable. Foundry Cloud Computing reported in 2023 that the cloud deployments were already higher than on premise deployments (52% to 48%). Their forecast was that in 18 months these ratios would tip further in favour of cloud deployments: 63% to 37%. The challenge is that the architecture of cloud-based graph systems will be different than what we are working on now. It may be useful to focus on cloud architectures.

YT: When it comes to architecture, one size definitely doesn't fit all. Similar to relational databases, graph OLTP and OLAP workloads differ significantly. Graph OLTP focuses on low-latency traversal and pattern matching, requiring efficient indexing for performance. In contrast, graph OLAP processes the entire graph in long, iterative operations where indexing offers little benefit. This difference is why most graph databases use separate architectures for OLTP and OLAP workloads. However, from a user interface perspective, it is highly desirable to have a unified front-end that abstracts these complexities.

As for the question of cloud versus on-premise, the trend is unmistakably towards the cloud, due to the scalability support, ease of management, and better support in integrating graph and non-graph workloads within a single application.

HV: Cloud adoption is rapidly accelerating, becoming the norm for modern enterprise systems. When deciding between cloud and on-premise for graph analytical workloads, it's less about technical superiority and more about meeting business needs. Graphs represent an organization's domain knowledge, helping answer business questions in high-value use cases like detecting fraud [24], digital twins [25], inventory management [23], supply chain optimization, customer 360 and product 360. Many use cases blend transaction processing and graph analyticsbecause timely insights and decision-making often require analyzing relationships and patterns in interconnected data while simultaneously updating and managing that data through transactions. However users care less about these technical distinctions and more about delivering business value. Cloud platforms simplify this by hiding technical complexity, enhancing ease of use compared to on-premise solutions. This flexibility allows vendors to offer a seamless, integrated experience for managing both transactional and analytical graph workloads.

WY: While there is a need for solutions to efficiently execute graph analytical workloads, combining OLAP and OLTP into a single system may not always be necessary. Many hybrid approaches currently employ a decoupled design, typically using a graph database for transactional or online requests while relying on specialized systems for OLAP tasks. However, these solutions often encounter challenges related to consistency, capability, complexity, and data freshness, especially when the primary data is stored in relational databases. At GraphScope [13], we propose an alternative approach, GART [36], which involves managing transactional requests within relational databases while processing OLAP tasks on a synchronized graph that streams data from the relational database's binlog. This method can enhance scalability, consistency, and overall capability.

Cloud-based solutions offer advantages like scalability, flexibility, and easier integration into exist-

Dataset	$\|V\|$	$\|E\|$	Size	Single Machine
Live Journal	$\sim 4M$	$\sim 68M$	1.08GB	6.3GB
USA Road	$\sim 24M$	$\sim 58M$	951MB	9.09GB
Twitter	$\sim 41M$	$\sim 1.4B$	26GB	128 GB
UK0705	$\sim 82M$	$\sim 2.8B$	48GB	247GB
World Road	$\sim 682M$	$\sim 717M$	15GB	194GB
CC2014	$\sim 1.7B$	$\sim 64.4B$	1.3TB	Out of memory

Table 1: Expansion of Graph Data when Loaded (CC2014 stands for CommonCrawl2014)

ing big data and AI infrastructures. However, on-premise solutions may still hold benefits, especially for leveraging specialized hardware for processing tasks or for organizations with strict data security and privacy requirements.

WZ: Given the growing complexity and adoption of graph workloads in industry, there is a clear and increasing demand for OLAP/OLTP architectures or their hybrid variant, namely HTAP for graphs. When comparing Cloud vs. On-Premise solutions, cloud-based options generally offer greater flexibility, scalability, and cost-efficiency, making them well-suited for graph workloads. The ability to scale resources up or down based on demand is particularly advantageous for managing the high concurrency and large datasets typically associated with graph workloads. Cloud-native graph architectures and HTAP on the cloud are likely to play a significant role in the future of graph processing. However, on-premise solutions may still be preferable for organizations that prioritize enhanced security, strict compliance requirements, or need for specific infrastructure customization.

Q3 – SCALABILITY & PERFORMANCE

MTO: Scalability is a major pain point – see our surveys [31,32]. One way of addressing scalability is through "scale-up" – using multithreading and additional resources on a single machine [19]. The main argument is that graph data sets are of reasonable size and can fit in the main memory of a modern large workstation. However, in the long run, scale-out on a parallel cluster is required [34]. Scale-up is important, but relying alone on that is challenging. The raw datasets may be small, but when they are loaded to a real graph DBMS for processing with appropriate data structures, their size balloons. Table 1 reports an experiment we performed using PowerLyra [8]. The heavy computation of graph processing along with the advantages of computation parallelization should also be considered.

Performance of these systems is another pain point as highlighted in our above cited survey. Part of this is due to the fact that the community has been predominantly focused on algorithmic issues and less attention has been paid to the architectural concerns. Hardware accelerators (predominantly GPUs) have been studied to address performance, but these are mostly singular solutions. Incorporating these accelerators into a complete system execution in a deep way remains an issue.

Finally, there is the issue of performance benchmarks. Although there are graph DBMS benchmarks, how well they represent real user applications is questionable. In addition to application-level (or macro)benchmarks, there is a need for microbenchmarks to test system architectural design decisions [20].

YT: Scalability is critical, and scale-out architectures are a must. The survey in [31] identified scalability as a key challenge for users. I also observed that most graph customers require the ability to scale out their graph databases, even if their current datasets can still be managed by a single-node, scale-up solution. When they invest in a graph solution, they are not just addressing their current needs but are also planning for future growth. Performance is another area where continuous improvement is essential. Customers consistently demand lower latency and higher throughput to meet the increasingly complex and real-time requirements of their graph workloads. When it comes to benchmarking, the requirements are diverse, reflecting the wide range of graph use cases. Although lacking standard benchmarks, the Linked Data Benchmark Council (LDBC) has made significant strides in this area by providing a comprehensive suite of benchmarks [1]. A broader adoption of such benchmarks will be crucial in setting performance standards and guiding future development in graph analytics.

HV: Scalability is a critical requirement for graph analytics, involving both scale-out and scale-up strategies. It's not just about handling larger data sizes but also managing growing workloads. While scalability is important, single query performance, query throughput, and update performance also matter, making it a multi-dimensional challenge. The priorities depend on the specific use case, organization, and data. Moreover, achieving correct, reliable, and trustworthy distributed computing is a complex task, particularly when sharding graphs, given their highly interconnected nature. There is no one-size-fits-all solution; it requires a blend of techniques that will mature over time, creating an adaptable system capable of meeting diverse scalability needs.

WY: Future requirements for graph analytics in terms of scalability, performance, and benchmarking may include more efficient integration with upstream and downstream AI and big data systems to provide

holistic insights and value. Achieving end-to-end scalability and performance is important; it is not only about optimizing graph computations but also about managing data format transformations, loading and writing costs, and ensuring these processes work effectively with other types of workloads. Additionally, future benchmarking efforts could benefit from covering more real-life scenarios, including a wider variety of graph types from different domains, expanding workloads beyond common algorithms like Connected Components (CC) and PageRank, and considering factors such as single-machine versus distributed environments, as well as the ease of programming and integration with existing systems.
WZ: Large-scale graphs pose challenges in real-time query response. Techniques like scale-out/scale-up and query-specific summarization or indexing can help improve performance. Well-designed query sets are as important as datasets for ensuring meaningful performance evaluations.

Q4 – GRAPH STORES

YT: In real-world applications, workloads are rarely homogeneous. Graph analytics are often intertwined with SQL queries, machine learning (ML), and other forms of data analysis. Users don't want fragmented silos—they seek a unified platform that can seamlessly support various types of analytics.

In this context, polystores offer a significant advantage by providing a flexible environment where different data models and analytics can coexist and interact within a single system [11]. However, graph-only stores aren't necessarily limited in this regard. They can also be closely integrated with other data engines within a unified platform, enabling them to support diverse analytics needs effectively. For example, Neo4j recently announced a deep integration with Microsoft Fabric [21].
HV: Graph platforms are crucial for future graph analytics, enabling organizations to quickly derive business value by representing their business domain as a graph. This provides an intuitive way to explore relationships between entities, a common focus of high-value questions. A robust graph platform must also support various data types—temporal, spatial, nested, and vectors—to meet diverse use case needs. Vector support is vital for generative AI (GenAI) applications, especially in retrieval-augmented generation (RAG), which contextualizes large language models (LLMs) with structured knowledge. Despite vector importance, graphs remain the most natural way to represent interconnected data.
WY: We believe that a "one-size-fits-all" approach is insufficient, as the choice of storage solutions depends on specific scenarios. Given the diversity of storage options, including both graph stores and polystores, it is crucial to focus on how to share graph data between them and make it accessible across various systems. At GraphScope, we have taken two key initiatives to address this challenge. First, to manage exchange graphs and graphs on data lakes, we developed GraphAr [2], an open-source, standard data file format for efficient graph data storage and retrieval. Second, we proposed GRIN [3], a common graph retrieval interface aimed at unifying graph data access across different systems. These efforts make it easier to share graph data and ensure accessibility across different storage systems, whether they are graph-only or polystores.
WZ: Graph-only stores may not be sufficient for the future of graph analytics. While they excel at graph-specific tasks like graph traversals and graph algorithms, many applications require integrating graph data with relational, document-based, or other unstructured data. Polystores enable seamless interaction between these data types, providing more comprehensive analytics for various applications. Additionally, legacy systems and existing data sources pose challenges that polystores can address. As graph analytics evolves, polystores will become crucial for managing complex, multi-modal data.

Q5 – DSL & API

MTO: We have witnessed a surge of APIs and DSLs for graphs. For instance, Blueprints is an API for the property graph model, underlying many graph databases, e.g. Neo4j and Titan. As DSLs, several graph databases and parallel graph analytics started with their own, such Green-Marl [14] and PGQL [35] – as front-ends with which users describe their custom graph algorithms or pattern-matching queries in Oracle PGX. This fragmented landscape has been unified with the first versions of standard graph query languages, such as SQL/PGQ and GQL, respectively in 2023 and 2024, concerning pattern matching in the DSL. A similar effort is highly desirable for the graph analytics counterparts.
YT: In my opinion, the focus shouldn't be solely on developing new DSLs or APIs. Instead, the key is unification. To avoid creating isolated graph silos, we need to lower the barriers between graph databases and other data systems, making it easier to integrate graph analytics with broader data science, ML, and LLM tasks. As mentioned before, it's essential to meet users where they are. This means providing seamless integration with the tools and languages that data scientists and engineers are already using, such as SQL, Python, or even natural language interfaces.
HV: To enable analytics on graph structures for

data science, machine learning (ML), and large language model (LLM) tasks, three factors are key. First, ease of use through simple APIs and tools that enhance developer productivity. Second, seamless cloud integration to support scalable workflows. Third, enabling ML/AI applications requires advanced graph algorithms like embeddings and model training to be easily accessible. This involves packaging powerful algorithms behind simple APIs, with tools for data ingestion, orchestration, and result integration. DSLs and APIs must also integrate smoothly with the generative AI (GenAI) stack to fully unlock the potential of graph analytics.

WY: Specific DSLs and APIs may be needed to enable seamless analytics on graph structures for data science, ML, and LLM tasks. ML and LLMs can naturally interact with graphs in various ways, such as representing them as sparse tensors for GNNs or using them as knowledge bases through knowledge graphs or GraphRAG [4] in LLMs. Furthermore, an "ETL"-like DSL may assist in aligning and transforming existing graph data into formats that are more compatible with ML and LLM inputs, potentially enabling more effective integration and utilization of graph data in these advanced tasks.

Q6 – ANALYTICAL OPERATORS

MTO: There are few real static graphs, despite an inordinate amount of research efforts devoted to them. Most real graphs are dynamic – they are entirely availability to the system (at least the vertex set), but see updates, usually in the form of edge additions and (sometimes) deletions. Theoreticians usually call these *semi-streaming* graphs [37].

Majority of the focus has been on dealing with topological dynamicity, i.e., changes graph topology with updates to structure. However, in the domain of property graphs, updates to vertex and edge attributes are also important. The languages need to support predicates on both topology and attributes, and execution strategies need to optimize those queries that include both predicates.

More challenging are streaming graphs that emerge over time and are fundamentally unbounded. They are more challenging since the graph is never fully available to the system [27, 28].

The features that are needed for addressing dynamic and streaming graphs are, at a minimum, the following: incremental computation (to avoid batch computation from scratch every time), nonblocking operators for streaming graphs since the graphs as operands are unbounded, windowed operators if we want to do analytic computation requiring multiple passes over data (windows provide a means of batching), and support for temporality (somehow keeping track of evolution of the graph) if we wish to do time-travel (temporal) queries to determine trends over time. If temporality is desired, it needs to be added as a first-class object to the language.

YT: In most of today's graph databases, time is treated as just another property of vertices and edges. To effectively support dynamic, incremental, and streaming graphs, time needs to be elevated to a first-class concept within graph structures. Another critical requirement is the seamless integration with streaming ingestion systems like Kafka [17] and Event Hubs [22]. Finally, in a world where data streams in continuously, we want to avoid recomputing everything from scratch each time new data arrives. Therefore, better support for incremental updates of analytical results is crucial.

HV: In terms of analytical operators for dynamic, incremental, and streaming graphs [45], it is important to recognize that most organizations' analytical needs are not limited to pure streaming but also involve historical and master data. Event-driven processing plays a crucial role, especially in coordinating multiple subsystems within a data ecosystem—Kafka [17, 18] is a prime example of a tool used for this purpose. One essential capability in this context is enabling the ecosystem around a graph to react to changes in real time. Change Data Capture (CDC) allows organizations to track and capture modifications to the graph database as they happen. By combining CDC, event-streaming connectors like Kafka, and high-throughput scalability, organizations can meet many of their analytical requirements. Purer forms of streaming are currently less prevalent and remain an evolving area for the future.

WY: The requirements will likely depend on specific scenarios, encompassing a broad range of existing dynamic, incremental, and streaming processing capabilities found in relational data systems. Key considerations may include stream processing, materialized views, time-series graphs, incrementalization of existing graph algorithms, and designing a dynamic, incremental, and streaming graph store to handle these complex needs effectively.

WZ: Incremental updates are crucial for handling evolving graphs to avoid recomputation from scratch. However, efficiently identifying the affected subgraphs and limiting the computation to them is challenging, calling for a trade-off between recomputation and incremental updates. In evolving graphs, such as dynamic or streaming graphs, effectively managing and analyzing temporal information is also crucial. The operators need to support temporal queries in addition to structure-based queries to ensure comprehensive analytics.

1 References

[1] The LDBC Benchmark Overview. https://ldbcouncil.org/benchmarks/overview.

[2] Apache GraphAr. https://graphar.apache.org/, 2024.

[3] GraphScope GRIN. https://github.com/GraphScope/GRIN, 2024.

[4] Microsoft GraphRAG. https://github.com/microsoft/graphrag, 2024.

[5] D. Allen, A. Hodler, M. Hunger, M. Knobloch, W. Lyon, M. Needham, and H. Voigt. Understanding Trolls with Efficient Analytics of Large Graphs in Neo4j. In *Datenbanksysteme für Business, Technologie und Web (BTW 2019), 18. Fachtagung des GI-Fachbereichs,Datenbanken und Informationssysteme (DBIS), 4.-8. März 2019, Rostock, Germany, Proceedings*, volume P-289 of *LNI*, pages 377–396. Gesellschaft für Informatik, Bonn, 2019.

[6] R. Angles, A. Bonifati, R. García, and D. Vrgoč. Path-based Algebraic Foundations of Graph Query Languages. In *Proceedings of the 28th International Conference on Extending Database Technology (to appear)*, 2025.

[7] A. Bonifati, F. Murlak, and Y. Ramusat. Transforming property graphs. *Proc. VLDB Endowment*, 17(12):3224–3238, 2024.

[8] R. Chen, J. Shi, Y. Chen, and H. Chen. PowerLyra: Differentiated graph computation and partitioning on skewed graphs. In *Proc. 10th ACM SIGOPS/EuroSys European Conf. on Comp. Syst.*, pages 1:1–1:15, 2015.

[9] A. Deutsch, N. Francis, A. Green, K. Hare, B. Li, L. Libkin, T. Lindaaker, V. Marsault, W. Martens, J. Michels, F. Murlak, S. Plantikow, P. Selmer, O. van Rest, H. Voigt, D. Vrgoc, M. Wu, and F. Zemke. Graph Pattern Matching in GQL and SQL/PGQ. In *Proceedings of the 2022 International Conference on Management of Data*, volume 46, pages 2246–2258, New York, NY, USA, jun 2022. ACM.

[10] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 1433–1445, New York, NY, USA, 2018. Association for Computing Machinery.

[11] V. Gadepally, P. Chen, J. Duggan, A. J. Elmore, B. Haynes, J. Kepner, S. Madden, T. Mattson, and M. Stonebraker. The bigdawg polystore system and architecture. In *2016 IEEE High Performance Extreme Computing Conference, HPEC 2016, Waltham, MA, USA, September 13-15, 2016*, pages 1–6, 2016.

[12] L. Gómez, B. Kuijpers, and A. Vaisman. Performing OLAP over graph data: Query language, implementation, and a case study. In *Proc. Intl. Workshop on Real-Time Business Intelligence and Analytics*, New York, NY, USA, 2017. Association for Computing Machinery.

[13] T. He and et al. Graphscope flex: Lego-like graph computing stack. In *SIGMOD 2024*, page 386–399, 2024.

[14] S. Hong, H. Chafi, E. Sedlar, and K. Olukotun. Green-marl: a DSL for easy and efficient graph analysis. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2012, London, UK, March 3-7, 2012*, pages 349–362, 2012.

[15] ISO. ISO/IEC 9075-16:2023 Information technology — Database languages SQL - Part 16: Property Graph Queries (SQL/PGQ). https://www.iso.org/standard/79473.html, June 2023.

[16] ISO. ISO/IEC 39075:2024 Information technology — Database languages — GQL. https://www.iso.org/standard/76120.html, April 2024.

[17] A. Kafka. Apache kafka. https://kafka.apache.org/, 2024.

[18] J. Kreps, N. Narkhede, J. Rao, et al. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, volume 11, pages 1–7. Athens, Greece, 2011.

[19] J. Lin. Scale up or scale out for graph processing? *IEEE Internet Comput.*, 22(3):72–78, 2018.

[20] M. Lissandrini, M. Brugnara, and Y. Velegrakis. Beyond macrobenchmarks: Microbenchmark-based graph database evaluation. *Proc. VLDB Endowment*, 12(4):390–403, 2018.

[21] B. Lück. Neo4j integration with microsoft fabric and azure openai service. https://www.bigdata-insider.de/neo4j-integration-in-microsoft-fabric-und-azure-openai-service-a-32c0d7a24675d919b0f430313fb5abb3/, 2024.

[22] Microsoft. Event hubs—real-time data ingestion. https://azure.microsoft.com/

en-us/products/event-hubs, 2024.
[23] Neo4j. BT Group Keeps Customers Connected with Lightning-Fast Inventory Management. https://neo4j.com/customer-stories/bt-group/.
[24] Neo4j. iuvity Increases Fraud Detection Rates by 200 Percent with Neo4j. https://neo4j.com/customer-stories/iuvity-fka-todo1/.
[25] Neo4j. Transport For London Cuts Congestion by 10% with a Digital Twin Powered by Neo4j. https://neo4j.com/customer-stories/transport-for-london/.
[26] Oracle. 17 Use Cases for Graph Databases and Graph Analytics. https://www.oracle.com/a/ocom/docs/graph-db-use-cases-ebook.pdf, 2021.
[27] A. Pacaci, A. Bonifati, and M. T. Özsu. Regular path query evaluation on streaming graphs. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 1415–1430, 2020.
[28] A. Pacaci, A. Bonifati, and M. T. Özsu. Evaluating complex queries on streaming graphs. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*, pages 272–285, 2022.
[29] J. Z. Pan, S. Razniewski, J. Kalo, S. Singhania, J. Chen, S. Dietze, H. Jabeen, J. Omeliyanenko, W. Zhang, M. Lissandrini, R. Biswas, G. de Melo, A. Bonifati, E. Vakaj, M. Dragoni, and D. Graux. Large language models and knowledge graphs: Opportunities and challenges. *TGDK*, 1(1):2:1–2:38, 2023.
[30] M. A. Rodriguez. The gremlin graph traversal machine and language. In *Proceedings of the 15th Symposium on Database Programming Languages*, DBPL 2015, page 1–10, New York, NY, USA, 2015. Association for Computing Machinery.
[31] S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin, and M. T. Özsu. The ubiquity of large graphs and surprising challenges of graph processing. *Proc. VLDB Endowment*, 11(4):420–431, 2017.
[32] S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin, and M. T. Özsu. The ubiquity of large graphs and surprising challenges of graph processing: extended survey. *VLDB J.*, 29(2-3):595–618, 2020.
[33] S. Sakr, A. Bonifati, H. Voigt, A. Iosup, K. Ammar, R. Angles, W. G. Aref, M. Arenas, M. Besta, P. A. Boncz, K. Daudjee, E. D. Valle, S. Dumbrava, O. Hartig, B. Haslhofer, T. Hegeman, J. Hidders, K. Hose, A. Iamnitchi, V. Kalavri, H. Kapp, W. Martens, M. T. Özsu, E. Peukert, S. Plantikow, M. Ragab, M. Ripeanu, S. Salihoglu, C. Schulz, P. Selmer, J. F. Sequeda, J. Shinavier, G. Szárnyas, R. Tommasini, A. Tumeo, A. Uta, A. L. Varbanescu, H. Wu, N. Yakovets, D. Yan, and E. Yoneki. The future is big graphs: a community view on graph processing systems. *Commun. ACM*, 64(9):62–71, 2021.
[34] S. Salihoglu and M. T. Özsu. Response to 'Scale up or scale out for graph processing?'. *IEEE Internet Comput.*, 22(5):18–24, 2018.
[35] M. Sevenich, S. Hong, O. van Rest, Z. Wu, J. Banerjee, and H. Chafi. Using domain-specific languages for analytic graph databases. *Proc. VLDB Endow.*, 9(13):1257–1268, 2016.
[36] S. Shen, Z. Yao, L. Shi, L. Wang, L. Lai, Q. Tao, L. Su, R. Chen, W. Yu, H. Chen, B. Zang, and J. Zhou. Bridging the gap between relational OLTP and graph-based OLAP. In *2023 USENIX Annual Technical Conference, USENIX ATC 2023, Boston, MA, USA, July 10-12, 2023*, pages 181–196, 2023.
[37] J. Thaler. Semi-streaming algorithms for annotated graph streams. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 59:1–59:14, 2016.
[38] Y. Tian. The world of graph databases from an industry perspective. *SIGMOD Rec.*, 51(4):60–67, 2022.
[39] Y. Tian, E. L. Xu, W. Zhao, M. H. Pirahesh, S. J. Tong, W. Sun, T. Kolanko, M. S. H. Apu, and H. Peng. Ibm db2 graph: Supporting synergistic and retrofittable graph queries inside ibm db2. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 345–359, 2020.
[40] TigerGraph. Customer Success Stories: Why Our Customers Choose TigerGraph. https://www.tigergraph.com/customers/.
[41] W. W. W. C. (W3C). Sparql 1.1 overview. https://www.w3.org/TR/sparql11-overview/, 2013.
[42] Y. Wang, L. Yuan, Z. Chen, W. Zhang, X. Lin, and Q. Liu. Towards efficient shortest path counting on billion-scale graphs. In *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7,*

2023, pages 2579–2592, 2023.
[43] J. Webber. The Top 10 Use Cases of Graph Database Technology: Unlock New Possibilities with Connected Data. Technical report, Neo4j, 2021.
[44] D. Yan, Y. Bu, Y. Tian, and A. Deshpande. Big graph analytics platforms. *Found. Trends Databases*, 7(1-2):1–195, 2017.
[45] C. Zhang, A. Bonifati, and M. T. Özsu. Incremental Sliding Window Connectivity over Streaming Graphs. *Proc. VLDB Endow.*, 17(10):2473–2486, 2024.