The Future is Big Graphs! A Community View on Graph Processing Systems

AUTHORS:

Sherif Sakr[1], Angela Bonifati, Hannes Voigt, Alexandru Iosup, and
The other Dagstuhl Seminar participants

1. Introduction

Graphs are by nature 'unifying abstractions' that can leverage interconnectedness to represent, explore, predict, and explain real- and digital-world phenomena. Although real users and consumers of graph instances and graph workloads understand these abstractions, future problems will require new abstractions and systems. What needs to happen in the next decade for big graph processing to continue to succeed?

We are witnessing an unprecedented growth of interconnected data, which underscores the vital role of graph processing in our society. To name only a few remarkable examples of late, the importance of this field for practitioners is evidenced by the large number (over 50,000) of people registered[2] to download the Neo4j book "Graph Algorithms" in just over 1.5 years, and by the enormous interest in the use of graph processing in the Artificial Intelligence and Machine Learning fields[3]. Furthermore, the timely Graphs4Covid-19 initiative[4] provides evidence for the importance of big graph analytics in alleviating the global COVID-19 pandemic.

This article addresses the questions: How do the next-decade big graph processing systems look like from the perspectives of the data management and the large scale systems communities[5]? What can we say today about the guiding design principles of these systems in the next 10 years?

[1] In memoriam.
[2] Cf. https://app.databox.com/datawall/551f309602080e2b2522f7446a20adb705cabbde8
[3] Many highly cited articles support this statement, e.g, Hamilton, W., Zhitao, Y., and Leskovec, J. Inductive representation learning on large graphs. NIPS (2017); Perozzi, B., Al-Rfou, R., Skiena S. DeepWalk: Online Learning of Social Representations. https://arxiv.org/pdf/1403.6652.pdf
[4] https://neo4j.com/graphs4good/covid-19/
[5] The summary of the Dagstuhl seminar: https://www.dagstuhl.de/19491

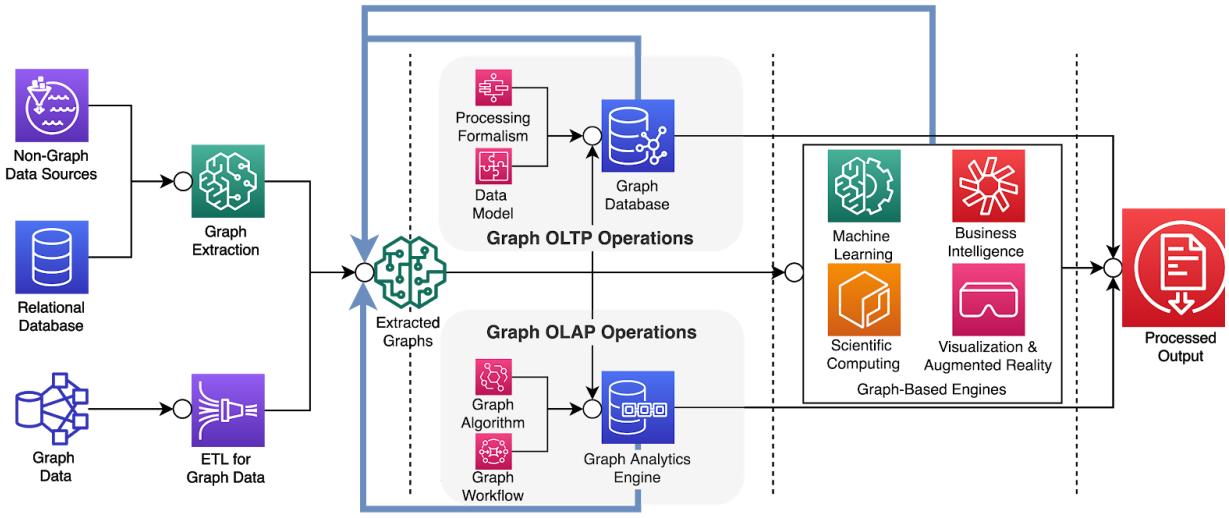

Figure 1. A complex data pipeline.

Figure 1 outlines the complex pipeline of big graph processing systems. Data flows in from diverse sources, is managed and manipulated (OLTP), analysed, enriched, and condensed (OLAP), and gets disseminated and consumed by machine learning, business intelligence, scientific computing, and visualization and augmented reality applications. To build our vision for next-generation, big graph processing systems, we focus on three aspects: abstractions, ecosystem, and performance.

First, we present expected data models and query languages, and inherent relationships among them in lattice structures. We cannot tell at the moment what will be the data model or query language of next-decade graph processing systems. Still, we can start discussing (in Section 2) the fundamental principles and underpinning abstractions upon which we can incrementally build. Any newly designed graph data model and query language can be easily plugged into our lattice of abstractions. This will solidify the understanding of the fundamental principles of graph data extraction, exchange, processing, and analysis, as illustrated in Figure 1.

A second important element is the vision of an ecosystem governing graph data and enabling the tuning of various components and switches, i.e., from graph OLTP operations to graph OLAP operations, from standards for queries and data formats to design choices for workloads, and scale-out versus scale-up and performance needs (Section 3). These aspects make the big processing systems more complicated than we have come to expect from the last decade. Figure 1 provides a high-level perception of this complexity in terms of inputs, outputs, processing needs, and final consumption of graph data.

A third element is how to understand and control performance in these future ecosystems (Section 4). We take a broad view of performance, which we understand as the combination of traditional and emerging non-functional aspects of how well these systems perform. We have important challenges to overcome in performance, from methodological aspects about performing meaningful, tractable, and reproducible experiments, to practical aspects regarding the trade-off of scalability with portability and interoperability, to new ideas about a graph-processing memex for collecting and sharing meaningful performance data.

SIDEBAR A: Dagstuhl Seminar 19491 Big Graph Processing Systems

About: The authors of this article met in December 2019 in Dagstuhl for Seminar 19491 on Big Graph Processing Systems[6]. The seminar gathered a diverse group of 41 high-quality researchers from the data management and large-scale systems communities. It was an excellent opportunity to start the discussion about next-decade opportunities and challenges for graph processing.

Participant list:

Khaled Ammar, Renzo Angles, Walid Aref, Marcelo Arenas, Maciej Besta, Peter A. Boncz, Angela Bonifati (organizer), Khuzaima Daudjee, Emanuele Della Valle, Stefania Dumbrava, Olaf Hartig, Bernhard Haslhofer, Tim Hegeman, Jan Hidders, Katja Hose, Adriana Iamnitchi, Alexandru Iosup (organizer), Vasiliki Kalavri, Hugo Kapp, Wim Martens, M. Tamer Özsu, Eric Peukert, Stefan Plantikow, Mohamed Ragab, Matei R. Ripeanu, Sherif Sakr (organizer), Semih Salihoglu, Christian Schulz, Petra Selmer, Juan F. Sequeda, Joshua Shinavier, Gábor Szárnyas, Riccardo Tommasini, Antonino Tumeo, Alexandru Uta, Ana Lucia Varbanescu, Hannes Voigt (organizer), Hsiang-Yun Wu, Nikolay Yakovets, Da Yan, Eiko Yoneki.

2. Abstractions

High-level abstractions help humans build models that are more intuitive and much closer to the human perception of reality than machine code instructions. Abstractions are widely used in programming languages, computational systems, database systems, etc., to conceal technical aspects in favor of more user-friendly, domain-oriented logical views. Conversely, graphs are conceptual objects by nature and abstractions are already inherent to the **graph data models**. Consequently, users are now called to choose from a large spectrum of graph data models that are similar, but differ in terms of **expressiveness**, cost, and intended use for **querying** and

[6] https://www.dagstuhl.de/en/program/calendar/semhp/?semnr=19491

analytics. In this '**abstraction soup**', new tools are urgently needed to help users make choices amid data and declarative formalisms. Further, reasoning and learning capabilities are needed for the actual use of the chosen graph abstractions.

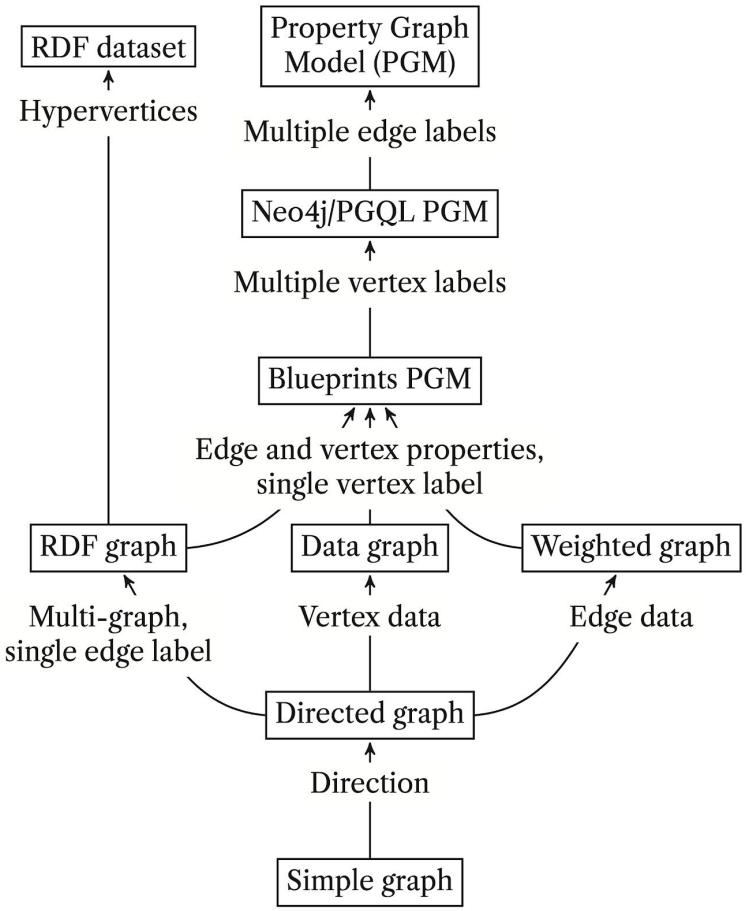

Figure 2. Example lattice showing graph data model variants with their model characteristics. For a more detailed example, see [8].

2.1. Understanding data models

Today, graph data management faces many data models (directed graphs, RDF, several variants of Property Graphs, etc.), with key challenges: (a) deciding which **data model** to choose per **use case**, and (b) mastering **interoperability** of data models where data from different models is combined (as in the left-hand side of Figure 1). Both challenges require deepening our understanding of data models regarding: (1) How do **humans conceptualize** data and data operations? How do data models and their respective operators support or hinder the human thought-process? Can we measure how "natural" or "intuitive" data models and their operators are? (2) How can we quantify and compare the (modelling and operational)

expressive power of data models? Can we (partially) order data models in a lattice regarding their expressiveness? Which costs and benefits of choosing one model over another?

Concretely, Figure 2 illustrates a lattice for a selection of graph data-models. Read bottom-up, this lattice shows which characteristic has to be added to a graph data model to obtain a model of richer expressiveness. The big challenge is to extend this comparative understanding for data models across multiple families, such as graph, relational, document models, etc.
(3) Interoperability between different data models can be achieved by means of **mappings** (semantic assertions across concepts in different data models) or with **direct translations** (e.g. W3C's R2RML). Are there general ways or building blocks for expressing such mappings?

Studying (1) requires foremost investigating people working with data and data models. Such investigations are uncommon in the data management field and should be conducted collaboratively with colleagues from other fields, such as human-computer interaction (HCI). Work on HCI and graphs exists, e.g., in HILDA workshops at Sigmod; however, these are not exploring the search space of graph data models. Studying (2) and (3) can build on existing work in database theory, but can also leverage findings from software engineering on comparison, featurization, and transformation of models.

2.2. Logic-based and declarative formalisms

Logic provides a unifying formalism for *expressing* queries, optimizations, integrity constraints, and integration rules. Starting from Codd's seminal insight relating **logical formulae to relational queries** [3], many First Order (FO) logic fragments have been used to formally define query languages with desirable properties such as decidable evaluation. Graph query languages are essentially a syntactic variant of FO augmented with **recursive** capabilities. Logic provides a yardstick for *reasoning* about graph queries and graph constraints. Indeed, a promising line of research is the application of formal tools, such as model checking, theorem proving [4], and testing, to establishing the *functional correctness* of complex graph-processing systems, in general, and of graph database systems, in particular.

The influence of logic is pivotal not only to database languages, but also as a foundation for combining logical reasoning with statistical learning in AI. Logical reasoning derives categorical notions about a piece of data by logical deduction. Statistical learning derives categorical notions by learning statistical models on known data and applying it to new data. Both leverage the topological structure of graphs (ontologies and knowledge graphs,[7] graph embeddings[3] to produce better insights than on non-connected data. However, both happen to be isolated. Combining both techniques can provide crucial advancements. Open questions here are: How can they be combined? Which underlying formalisms make this possible? How to weigh between the two mechanisms?

[7] A recent practical example it the COVID-19 Knowledge Graph: https://covidgraph.org/

2.3. Algebraic operators for graph processing

Currently, there is no standard property graph query language, nor a standard graph algebra. The outcome of the ISO/IEC standardization project around **GQL** (cf. Section 3.2) could influence the design of a graph algebra alongside existing and emerging use-cases [12].

However, next-generation graph processing systems should address questions about their algebraic components. What are fundamental operators of this **algebra**, compared to other algebras (relation algebra, group algebra, quiver or path algebra, incidence algebra or monadic algebra comprehensions)? What **core graph-algebra** should be supported by graph processing systems? Are there graph analytical operators to include in this algebra? Can this graph algebra be combined and integrated with an algebra of types, to make type-systems more expressive and to facilitate type checking?

A “relational-like” graph algebra able to express all the first-order queries [5] seems like a good starting point. However, the most interesting graph-oriented queries are **navigational** (i.e., reachability queries) and cannot be expressed with limited recursion of relational algebra [7, 8]. Furthermore, relational algebra is a **closed** algebra, i.e., input(s) and output of each operator is a relation, which make relational algebra operators composable. Should we aim for a closed graph algebra that encompasses both relations and graphs?

Current graph query engines combine algebra operators and ad-hoc graph algorithms into **complex workloads** (cf. Section 3.1). This complicates implementation and affects performance. An implementation based on a single algebra also seems utopic. A query language with general Turing Machine capabilities (like a programming language), however, entails tractability and feasibility problems [6]. Algebraic operators that work in both centralized and distributed environments, and that can be exploited by both graph algorithms and machine learning models, including GNNs, graphlets, and graph embeddings, could be highly desirable for the future.

3. Ecosystems

Ecosystems behave differently from mere systems of systems, because they couple many systems developed for different purposes and with different processes. Figure 1 exemplifies the complexity of a graph processing ecosystem through high-performance OLAP and OLTP pipelines working together. What are the ecosystem-related challenges?

3.1. Workloads in graph processing ecosystems

Workloads affect both the functional requirements (what a graph processing ecosystem will be capable of doing) and the non-functional (how well). Survey data [12] point to pipelines as in

Figure 1: complex workflows, combining heterogeneous queries and algorithms, managing and processing diverse datasets.

Presumably, today's graph processing matches traditional data-management workloads: (1) **transactional (OLTP) workloads** maintaining data integrity and up-to-date information in multi-user environments, characterized by many short, discrete, likely atomic transactions; and (2) **analytical (OLAP) workloads** with fewer users (e.g., analysts) submitting fewer but more complex and resource-intensive queries or processing jobs, with longer runtime (e.g., minutes). Also typical are ETL jobs, including data cleaning, and output jobs, including data formatting and visualization. Such pipelines are being linked into **domain-specific processing ecosystems**, including simulation and numerical methods in science and engineering, aggregation and modeling in business analytics, and ranking and recommendation in social media; and into more **general processing**, including machine learning and simulation at-large.

Graph processing workloads are **evolving**, and dynamic and streaming operations are becoming important [12]. However, most current graph processing systems lack such capabilities. **Persistent-query evaluation** over graph streams considering the streaming window-based model, and the arbitrary and simple path semantics of queries [11], go toward realizing **generalized streaming graph processing ecosystems**.

SIDEBAR B: In-Depth: Known properties of graph-processing workloads

Graph workloads may exhibit several properties:

(1) *Graph workloads are useful for many, vastly diverse domains* [12-5]. Notable features include edge orientation, adding properties/timestamps to edges and nodes; graph methods including neighborhood statistics, pathfinding and traversal, and subgraph mining; programming models including think-like-a-vertex, think-like-an-edge, and think-like-a-subgraph; sizes of graphs, i.e., trillion-edge graphs [15]; and selectivities of the queries and processes [1].

(2) *Graph workloads can be highly irregular*, mixing data-intensive and compute-intensive phases [15], where resource demands vary short-term. Consequently, the performance of a graph-processing system depends on the sources of irregularity, e.g., different datasets and algorithms, and computing platforms. The interdependency of these elements forms the Hardware-Platform-Algorithm-Dataset (HPAD) Law [13].

(3) *Graph processing uses a complex pipeline, combining a variety of tasks other than querying and algorithms* [12]. Popular tasks include visualization; loading (building), extracting, and transforming (ETL); cleaning; and debugging and testing, including synthetic graph generation. Figure 1 exemplifies this complex pipeline.

(4) *Scalability, interactivity, and usability* affect how graph users construct their workloads [12].

3.2. Standards for data models and query languages

Graph-processing ecosystem standards, once established and widely adopted, would provide a common technical foundation, thereby increasing the mobility of applications and tooling, and of developers, users, and stakeholders. Standards for both OLTP and OLAP workloads should standardize the data model, the data manipulation and data definition language, and the exchange formats. They should be easily adoptable by existing implementations, and also enable the integration of new implementations into the SQL-based technological landscape. It is important for users that such standards reflect existing industry practice, by following widely used graph query languages. To this end, in 2019, ISO/IEC started a project to define a new graph query language, GQL[8], aiming to standardize the data model, the schema, and the type system.

With an initial focus on transactional workloads, GQL will support composable graph querying using enhanced Regular Path Queries (RPQs) [7], graph transformation (views), and graph updating capabilities. GQL enhances RPQs with pattern quantification, ranking, and path-aggregation using different path-matching semantics. Syntactically, GQL should be SQL-like and combine the visual graph patterns pioneered industrially by Cypher [14] and shared with SQL/PGQ.

Long-term, worthwhile for standardization are building blocks of graph algorithms, analytical APIs and workflow definitions, graph embedding techniques, and benchmarks (cf. [21]). However, broad adoption for these aspects requires maturation.

[8] GQL was inaugurated with the backing of ten national standards bodies that jointly provide representatives from major vendors in the industry. GQL is also supported by the property graph community as represented by the Linked Data Benchmarks Council (LDBC).

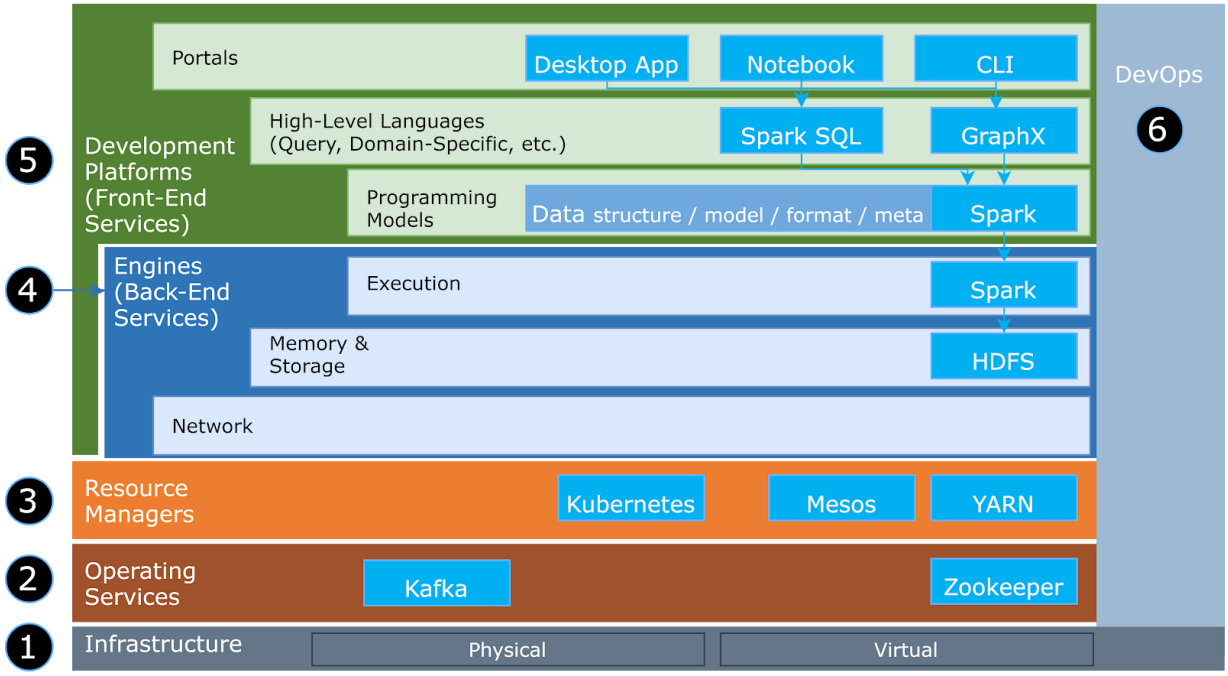

Figure 3. A reference architecture for graph processing ecosystems.

3.3. Reference architecture

We raise here the challenge of defining a reference architecture for big graph processing, to identify the core components used in production ecosystems, and to define their structure (e.g., layers). For large-scale ecosystems, reference architectures are a core tool in addressing complexity. The early definition of a reference architecture has greatly benefitted the discussion around the design, development, and deployment of cloud and grid computing solutions [17].

For big graph processing, our main insight is that graph processing ecosystems match the **common reference architecture of datacenters** [18], from which Figure 3 derives. Big graph processing ecosystems include layers for parallel and distributed algorithms, abstractions and languages (layer 5), libraries and runtimes systems (layer 4), and various software and hardware architectures (layers 1-3). Layer 6, the DevOps, spans all the other layers to facilitate their operation. The Spark ecosystem depicted here is one among thousands of possible instantiations. The challenge is to expand the reference architecture, adding new components and even layers to capture the evolving graph-processing field.

3.4. Beyond scale-up vs. scale-out

In modern graph processing, many solutions focus either on scale-up or on scale-out, each having preferred workload types and relative advantages [15]. Scale-up approaches rely on a tightly integrated system, usually a "**beefy**" multicore machine with large amounts of RAM per core. Scale-out approaches rely on large-scale resources across multiple, relatively "**wimpy**" machines, organized as a distributed system. However, as the history of HPC indicates, these notions merge and relative advantages disappear, as both resources and workloads become larger and more heterogeneous.

Beyond merely reconciling scale-up and scale-out, we envision **a scalability continuum**: given a diverse workload, the ecosystem would decide automatically how to run it, and on what kind of heterogeneous infrastructure, meeting Service Level Agreements. There are numerous mechanisms and techniques to enforce scale-up and scale-out decisions, such as data and work partitioning; and migration, offloading, replication, and elastic scaling. Each decision must take into account the dimensions emphasized by the HPAD Law [13] (see also Section 2.1). All decisions can be taken statically or dynamically, using various optimization and learning techniques.

3.5. Dynamic and streaming aspects

Future graph processing ecosystems should cope with highly evolving graph and streaming graph data. A **dynamic graph** extends the standard notion of a graph to account for updates (e.g., insertions, changes, deletions). Querying a dynamic graph encompasses the current version of the graph and its history of modifications. **Streaming graphs** can grow indefinitely, as new data arrives. Although they can be seen as a special case of dynamic graphs where the updates are limited to insertions, the most relevant aspect of streaming graphs is that they are *unbounded*, so the system is assumed to be unable to store the whole graph state.

The two notions can be seamlessly unified and combined. For instance, one can consider a sliding window over a graph stream [9] where each portion of the graph entering the window is an insertion, and a portion exiting the window is a deletion.

In terms of existing abstractions, most of the continuous analyses which are built on streaming processing technologies are fairly simple, e.g., **aggregations and projections**. In the industrial landscape, where graph processing libraries (e.g., Gelly on Apache Flink) and unified batch and streaming models (e.g., Apache Beam) are available, there is no inherent support for **"complex data" streams** (hierarchical, nested, interconnected data). Whenever users need to use a stream processor for more complex analytics (e.g., ML or graph analysis), they need to build ad-hoc, specialized solutions for offline training or perform snapshot-based analysis.

Another research challenge is to **identify the graph query processing operators** that can be evaluated on dynamic and streaming graphs. While some early attempts have been made

towards supporting recursive views and queries [10, 11], much remains until we can address notable capabilities of standard query languages such as GQL and G-Core [19], such as returning paths and entire subgraphs instead of simple vertices.

Discovering, understanding, and controlling the **dynamic phenomena** that occur in complex graph processing ecosystems is an open challenge. As graph processing ecosystems become more mainstream and are embedded in larger data-processing pipelines, we expect to increasingly observe known systems phenomena such as performance variability, the presence of cascading failures, and flashcrowds. These phenomena might be analogous to the phenomena in astronomy, physics, and even social sciences. But how will they manifest? What new phenomena will emerge? What programming abstractions [16] and systems techniques can respond to them?

4. Performance

Graph processing raises unique performance challenges, from the lack of a widely used performance metric besides response time, to the methodological problem of comparing graph processing systems across architectures and tuning processes, to performance portability and reproducibility. Such challenges become even more daunting for graph processing ecosystems.

4.1. Benchmarks, performance measurement, methodological aspects

Graph processing suffers from methodological issues as other computing disciplines [23,2]. Running comprehensive experiments, especially at scale, lacks tractability, that is, ability to implement, deploy, and experiment within a reasonable amount of time and cost. Moreover, graph systems can be benchmarked until they outperform the baseline. Such practices, encouraged by the lack of registered experiments and similar protocols in computing, threaten to compromise performance-experiments.

Graph processing also raises unique challenges in performance measurement and benchmarking. Because graph processing ecosystems are part of more complex data pipelines (see Figure 1), creating meaningful and tractable benchmarks is challenging. The challenge also includes capturing workload diversity (see Sidebar B), while balancing experiment simplicity and tractability [1]. Graph data can have vastly different structures and properties (see Section 2), and can be partitioned, combined, and transformed significantly while processing. Even seemingly minute variations, such as the degree distribution of the input graphs, can have significant performance implications [24, 25]. Human-system interaction is also important.

Similarly, capturing the essence of the system-under-test is challenging. Graph processing systems rely on a complex runtime that combines software and hardware platforms of high complexity, and thus raise the challenge of balancing representativeness with simplicity and tractability. Capturing parallelism, distribution, streaming vs. batch operation, etc., and testing

the operation of possibly hundreds of libraries, services, and runtime systems present in many real-world deployments of graph processing, are daunting.

We envision a combination of approaches to tackle these aspects: focusing on methodological aspects, including the definition of processes and protocols, and establishing organizations such as the Linked Data Benchmark Council (LDBC) to curate benchmark sharing and to audit their use in practice. Several concrete questions arise: how to define metrics that more faithfully characterize the effort required to execute a graph algorithm, query, program, or workflow? How to define and generate workloads with combined operations, covering temporal, spatial, and streaming aspects? How to integrate machine learning and simulation tasks into broader benchmarking? How to facilitate quick testing that can be easily understood, yet provide representative views into graph performance?

4.2. Specialization vs. portability and interoperability

There is considerable tension between specializing graph processing stacks for performance reasons and exposing the architectural details to the domain scientist for productivity. Simultaneously, these stacks will need to support constantly-evolving data models and interoperability with multi-domain tools.

The HPC domain proposed abstractions and C++ libraries built around them, and led to numerous extensions for high-performance and efficient execution across a variety of hardware platforms. Examples include BGL [21], CombBLAS and GraphBLAS. Approaches with a database focus, such as Neo4j, GEMS [22], and Cray's Urika, focus on convenient query languages such as SPARQL and Cypher to ensure portability.

Specialization, through custom software and especially hardware acceleration, focuses on supporting sparse data structures and irregular processing patterns, and so far have limited the productivity of all but expert users. Portability through reusable components seems promising, but currently there exists no C++ Graph Standard Library or standard query language, and all existing implementations of high-level abstractions make limiting assumptions about the runtime system and even the hardware platform. Lastly, interoperability means integrating graph processing into broader data processing workflows. Integration with machine learning and data mining processes, and with simulation and decision making instruments, seems vital but requires adapting and/or extending existing frameworks.

4.3. A memex for big graph processing systems

Inspired by Vannevar Bush's 1940s concept of personal memex, and by a 2010s specialization into a Distributed Systems Memex [20], we posit it would be both interesting and useful to create a Big Graph Memex for collecting, archiving, and retrieving meaningful operational information about such systems. This could be beneficial for learning about and eradicating performance and related issues, for enabling more creative designs and extending automation,

for meaningful and reproducible testing, as feedback building-block in smart graph processing, etc.

5. Conclusion

Graphs are a mainstay abstraction in today's data processing pipelines. How can future big graph processing and database systems provide highly scalable, efficient and diversified querying and analytical capabilities, as demanded by real-world requirements?

To tackle this question, we have undertaken a community approach. We started through a Dagstuhl Seminar and, shortly after, shaped the structured connections presented here. We have focused in this article on three interrelated elements: abstractions, ecosystems, and performance. For each of these elements, and across them, we have provided a view at what's next.

Only time can tell if our predictions provided worthwhile directions to the community. In the meantime, join us in solving the problems of big graph processing. The future is *big* graphs!

References

[1] Bonifati, A. et al. Graph Generators: State of the Art and Open Challenges. ACM Computing Surveys. April (2020). Online: https://doi.org/10.1145/3379445
[2] Angriman, E. et al. (2019) Guidelines for Experimental Algorithmics: A Case Study in Network Analysis. Algorithms 2019, 12(7), 127.
[3] Codd, E. F. A Relational Model of Data for Large Shared Data Banks". Commun. ACM 13(6): (1970), 377-387.
[4] Gonthier et al. A Machine-Checked Proof of the Odd Order Theorem. ITP (2013), 163-179.
[5] Chandra A. K.. Theory of Database Queries. In Proc. Symposium on Principles of Database Systems, pages 1–9. ACM Press, (1988).
[6] Aho, A. V. and Ullman, J. D. Universality of data retrieval languages. In Proceedings of the 6th ACM SIGACT- SIGPLAN symposium on Principles of programming languages, pages 110–119. ACM Press, (1979).
[7] Angles, R. et al. Foundations of modern query languages for graph databases. ACM Computing Surveys (CSUR), 50(5), Sept. (2017).
[8] Bonifati, A. et al. Querying Graphs. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, (2018).
[9] Babcock, B. et al. Models and Issues in Data Stream Systems. PODS (2002), 1-16.
[10] Pacaci, A., Bonifati, A., and Özsu, M. T. Regular Path Query Evaluation on Streaming Graphs. ACM SIGMOD (2020).
[11] Bonifati, A., Dumbrava, S., Gallego Arias, E. J.: Certified Graph View Maintenance with Regular Datalog. Theory Pract. Log. Program. 18(3-4): 372-389 (2018).

[12] Sahu, S. et al. The ubiquity of large graphs and surprising challenges of graph processing: extended survey. VLDB J. 29(2) (2020), 595-618.
[13] Uta, A. et al., Exploring HPC and big data convergence: A graph processing study on Intel Knights Landing, in CLUSTER (2018).
[14] Francis, N. et al.: Cypher: An Evolving Query Language for Property Graphs, SIGMOD (2019).
[15] Salihoglu, S., Özsu, M. T.: Response to "Scale Up or Scale Out for Graph Processing". IEEE Internet Computing 22(5): 18-24 (2018).
[16] Vasiliki, K., Vlassov, V. and Haridi, S.: "High-level programming abstractions for distributed graph processing." IEEE Transactions on Knowledge and Data Engineering 30.2 (2017), 305-324.
[17] Foster, I. and Kesselman, C.: The Grid 2: Blueprint for a new computing infrastructure. Elsevier, 2003.
[18] Iosup, A. et al.: Massivizing Computer Systems: A Vision to Understand, Design, and Engineer Computer Ecosystems Through and Beyond Modern Distributed Systems. ICDCS (2018), 1224-1237.
[19] Angles, R. et al.: G-CORE: A Core for Future Graph Query Languages. SIGMOD (2018), 1421-1432
[20] Iosup, A. et al.: The AtLarge Vision on the Design of Distributed Systems and Ecosystems. ICDCS (2019), 1765-1776.
[21] Siek, J. G., Lee, L.-Q. and Lumsdaine, A.: The Boost Graph Library: User Guide and Reference Manual. Addison-Wesley, (2002).
[22] Castellana, V. G. et al.: In-Memory Graph Databases for Web-Scale Data. IEEE Computer 48(3), (2015), 24-35.
[23] Papadopoulos, A. V. et al.: Methodological Principles for Reproducible Performance Evaluation in Cloud Computing. IEEE Trans. on Sw. Eng. (2019).
[24] Iosup, A. et al.: LDBC Graphalytics: A Benchmark for Large-Scale Graph Analysis on Parallel and Distributed Platforms. PVLDB 9(13), (2016), 1317-1328.
[25] Saleem, M. et al.: How Representative Is a SPARQL Benchmark? An Analysis of RDF Triplestore Benchmarks. WWW (2019), 1623-1633.