LDBC Graphalytics: A Benchmark for Large-Scale Graph Analysis on Parallel and Distributed Platforms

Alexandru Iosup[△] Tim Hegeman[△] Wing Lung Ngai[△] Stijn Heldens[△] Arnau Prat-Pérez[□]
Thomas Manhardt[◇] Hassan Chafi[◇] Mihai Capotă[⊙] Narayanan Sundaram[⊙]
Michael Anderson[⊙] Ilie Gabriel Tănase[Ψ] Yinglong Xia[⊗] Lifeng Nai[⊕] Peter Boncz[▽]

[◇]Oracle Labs [⊙]Intel Labs [Ψ]IBM Research [⊗]Huawei Research America
[△]Delft University of Technology [□]UPC Barcelona [⊕]Georgia Tech [▽]CWI Amsterdam

ABSTRACT

In this paper we introduce LDBC Graphalytics, a new industrial-grade benchmark for graph analysis platforms. It consists of six deterministic algorithms, standard datasets, synthetic dataset generators, and reference output, that enable the objective comparison of graph analysis platforms. Its test harness produces deep metrics that quantify multiple kinds of system scalability, such as horizontal/vertical and weak/strong, and of robustness, such as failures and performance variability. The benchmark comes with open-source software for generating data and monitoring performance. We describe and analyze six implementations of the benchmark (three from the community, three from the industry), providing insights into the strengths and weaknesses of the platforms. Key to our contribution, vendors perform the tuning and benchmarking of their platforms.

1. INTRODUCTION

Responding to increasingly larger and more diverse graphs, and the need to analyze them, both industry and academia are developing and tuning graph analysis software platforms. Already tens of such platforms exist, among them PowerGraph [21], GraphX [41], and PGX [4], but their performance is often difficult to compare. Moreover, the random, skewed, and correlated access patterns of graph analysis, caused by the complex interaction between input datasets and applications processing them, expose new bottlenecks on the hardware level, as hinted at by the large differences between Top500 and Graph500 rankings. Addressing the need for fair, comprehensive, standardized comparison of graph analysis platforms, in this work we propose the LDBC Graphalytics benchmark.

The Linked Data Benchmark Council (`ldbcouncil.org`, LDBC), is an industry council formed to establish standard benchmark specifications, practices and results for *graph data management systems*. Its goal is to inform IT professionals on the properties of the various solutions available on the market; to stimulate academic research in graph data storage, indexing, and analysis; and to accelerate the maturing process of the graph data management space as a whole. LDBC organizes a Technical User Community (TUC) that gathers benchmark input and feedback, and as such has investigated graph data management use cases across the fields of marketing, sales, telecommunication, production, publishing, law enforcement and bio-informatics. LDBC previously introduced the Social Network Benchmark [17] (SNB), which models a large social network but targets database systems (graph, SQL or SPARQL) that provide interactive updates and query answers. However, the LDBC scope goes beyond such database workloads: it also includes graph analysis frameworks that facilitate complex and holistic graph computations which may not be easily modeled as database queries, but rather as (iterative) graph algorithms, such as global metrics (e.g., diameter, triangle count) or clustering. Algorithmically analyzing large graphs is an important class of problems in "Big Data" processing, with applications such as the analysis of human behavior and preferences in social networks, root cause analysis in large-scale computer and telecommunication networks, and interactions between biological compounds and genetic structures.

In this paper, LDBC introduces Graphalytics, a benchmark for evaluating graph analysis platforms, that builds on the data generators from LDBC SNB and Graph500, making the following original contributions:

1. The first industrial-grade *graph analysis benchmark specification*. We carefully motivate the choice of algorithms in the benchmark, using the LDBC TUC and literature surveys to ensure good coverage of scenarios. Graphalytics consists of six core algorithms: breadth-first search, PageRank, weakly connected components, community detection using label propagation, local clustering coefficient, and single-source shortest paths. The workload includes real and synthetic datasets, which are classified into intuitive "T-shirt" sizes (e.g., XS, S, M, L, XL). The benchmarking process is made future-proof, through a *renewal process*.
2. A detailed *process for running the benchmark*. Our test harness characterizes performance and *scalability* with deep metrics (vertical vs. horizontal and strong vs. weak scaling), and also characterizes *robustness* by measuring SLA compliance, performance variability, and crash points.

Proceedings of the VLDB Endowment, Vol. 9, No. 13

3. A *comprehensive tool-set developed using modern software engineering practices* released as open-source benchmarking software, including a harness capable of supporting many types of target-systems, the scalable LDBC social-network generator **Datagen**, and the versatile **Granula** performance evaluation tool.
4. An extensive *experimental evaluation* of six state-of-the-art graph analysis systems: three community-driven (Giraph, GraphX, and PowerGraph) and three industry-driven (PGX, GraphMat, and OpenG). Benchmarking and tuning of the industry-driven systems in our evaluation has been performed by their respective vendors.

We describe the first three contributions, which combine the conceptual and technical specification of Graphalytics, in Section 2. The experimental evaluation is split among Section 3, which introduces the tested platforms and the benchmarking hardware, and Section 4, which presents and analyzes the real-world benchmarking results. We cover related work in Section 5, before concluding in Section 6.

2. Graphalytics

Graphalytics tests a graph analysis framework, consisting of a software platform and underlying hardware system. Graphalytics models holistic graph analysis workloads, such as computing global statistics and clustering, which run on the entire dataset on behalf of a single user.

2.1 Requirements

A benchmark is always the result of a number of design choices, responding to a set of requirements. In this section we discuss the main requirements addressed by LDBC Graphalytics:

(R1) Target platforms and systems: benchmarks must support any graph analysis platform operating on any hardware system. For platforms, we do not distinguish between programming models and support different models, including vertex-centric, gather-apply-scatter, and sparse matrix operations. For systems, we target the following environments: distributed systems, multi-core single-node systems, many-core GPU systems, hybrid CPU-GPU systems, and distributed hybrid systems. Without R1, a benchmark could not service the diverse industrial following of LDBC.

(R2) Diverse, representative benchmark elements: data model and workload selection must be representative and have good coverage of real-world practice. In particular, the workload selection must not only include datasets or algorithms because experts believe they cover known system bottlenecks (e.g., they can stress real-world systems), but also because they can be shown to be representative of the current and near-future practice. Without representativeness, a benchmark could bias work on platforms and systems towards goals that are simply not useful for improving current practice. Without coverage, a benchmark could push the LDBC community into pursuing cases that are currently interesting for the industry, but not address what could become impassable bottlenecks in the near-future.

(R3) Diverse, representative process: the set of experiments conducted by the benchmark automatically must be broad, covering the main bottlenecks of the target systems. In particular, the target systems are known to raise various scalability issues, and also, because of deployment in real-world clusters, be prone to various kinds of failures, exhibit performance variability, and overall have various robustness problems. The process must also include possibility to validate the algorithm output, thus making sure the processing is done correctly. Without R3, a benchmark could test very few of the diverse capabilities of the target platforms and systems, and benchmarking results could not be trusted.

(R4) Include a renewal process: unlike many other benchmarks, benchmarks in this area must include a renewal process, that is, not only a mechanism to scale up or otherwise change the workload to keep up with increasingly more powerful systems (e.g., the scale parameters of Graph500), but also a process to automatically configure the mechanism, and a way to characterize the reasonable characteristics of the workload for an average platform running on an average system. Without R4, a benchmark could become less relevant for the systems of the future.

(R5) Modern software engineering: benchmarks must include a modern software architecture and run a modern software-engineering process. They must make it possible to support R1, provide easy ways to add new platforms and systems to test, and allow practitioners to easily access the benchmark and compare their platforms and systems against those of others. Without R5, a benchmark could easily become unmaintainable or unusable.

2.2 Specification of Benchmark Elements

Addressing requirement R2, the key benchmarking elements in Graphalytics are the data model, the workload selection process, and the resulting algorithms and datasets.

2.2.1 Data Model

The Graphalytics benchmark uses a typical data model for graphs; a graph consists of a collection of vertices, each identified by a unique integer, and a collection of edges, each consisting of a pair of vertex identifiers. Graphalytics supports both *directed* and *undirected* graphs. Edges in directed graphs are identified by an ordered pair (i.e., the source and destination of the edge). Edges in undirected graphs consist of unordered pairs. Every edge must be unique and connect two distinct vertices. Optionally, vertices and edges have properties, such as timestamps, labels, or weights.

To accommodate requirement R2, Graphalytics does not impose any requirement on the semantics of graphs. That is, any dataset that can be represented as a graph can be used in the Graphalytics benchmark if it is representative of real-world graph-analysis workloads.

2.2.2 Two-Stage Workload Selection Process

To achieve both workload representativeness and workload coverage, we used a *two-stage selection process* to select the workload for Graphalytics. The first stage identifies classes of algorithms and datasets that are representative for real-world usage of graph analysis platforms. In the second stage, algorithms and datasets are selected from the most common classes such that the resulting selection is diverse, i.e., the algorithms cover a variety of computation and communication patterns, and the datasets cover a range of sizes and a variety of graph characteristics.

Table 1: Results of surveys of graph algorithms.

Graph	Class (selected candidates)	#	%
Unweighted	Statistics (PR, LCC)	24	17.0%
	Traversal (BFS)	69	48.9%
	Components (WCC, CDLP)	20	14.2%
	Graph Evolution	6	4.2%
	Other	22	15.6%
Weighted	Distances/Paths (SSSP)	17	34%
	Clustering	7	14%
	Partitioning	5	10%
	Routing	5	10%
	Other	16	32%

Table 2: Mapping of dataset scale ranges to labels ("T-shirt sizes") in Graphalytics.

Scale	< 7	$[7, 7.5)$	$[7.5, 8)$	$[8, 8.5)$	$[8.5, 9)$	$[9, 9.5)$	≥ 9.5
Label	2XS	XS	S	M	L	XL	2XL

2.2.3 Selected Algorithms

Addressing R1, according to which Graphalytics should allow different platforms to compete, the definition of the algorithms of Graphalytics is abstract. For each algorithm, we define its processing task and provide a reference implementation and reference output. Correctness of a platform implementation is defined as output equivalence to the provided reference implementation.

To select algorithms which cover real-world workloads for graph analysis platform, we have conducted two comprehensive surveys of graph analysis articles published in ten representative conferences on databases, high-performance computing, and distributed systems (e.g., VLDB, SIGMOD, SC, PPoPP). The first survey (conducted for our previous paper [24]) focused only on unweighted graphs and resulted in 124 articles. The second survey (conducted for this paper) focused only on weighted graphs and resulted in 44 articles. Table 1 summarizes the results from these surveys. Because one article may contain multiple algorithms, the number of algorithms exceeds the number of articles. In general, we found that a large variety of graph analysis algorithms are used in practice. We have categorized these algorithms into several classes, based on their functionality, and quantified their presence in literature.

Based on the results of these surveys, with expert advice from LDBC TUC we have selected the following five core algorithm for unweighted graphs, and a single core algorithm for weighted graphs, which we consider to be representative for graph analysis in general:

Breadth-first search (BFS): For every vertex, determines the minimum number of hops required to reach the vertex from a given source vertex.

PageRank (PR) [34]: Measures the rank ("popularity") of each vertex by propagating influence between vertices using edges.

Weakly connected components (WCC): Determines the weakly connected component each vertex belongs to.

Community detection using label propagation (CDLP): Finds "communities" in the graph, i.e., non-overlapping densely connected clusters that are weakly connected to each other. We select for community detection the label propagation algorithm [36], modified slightly to be both parallel and deterministic [28].

Table 3: Real-world datasets used by Graphalytics.

ID	Name	$\lvert V\rvert$	$\lvert E\rvert$	Scale	Domain
R1(2XS)	wiki-talk [5]	2.39 M	5.02 M	6.9	Knowledge
R2(XS)	kgs [22]	0.83 M	17.9 M	7.3	Gaming
R3(XS)	cit-patents [5]	3.77 M	16.5 M	7.3	Knowledge
R4(S)	dota-league [22]	0.06 M	50.9 M	7.7	Gaming
R5(XL)	com-friendster [5]	65.6 M	1.81 B	9.3	Social
R6(XL)	twitter_mpi [13]	52.6 M	1.97 B	9.3	Social

Table 4: Synthetic datasets used by Graphalytics.

ID	Name	$\lvert V\rvert$	$\lvert E\rvert$	Scale
D100(M)	datagen-100	1.67 M	102 M	8.0
D100'(M)	datagen-100-cc0.05	1.67 M	103 M	8.0
D100"(M)	datagen-100-cc0.15	1.67 M	103 M	8.0
D300(L)	datagen-300	4.35 M	304 M	8.5
D1000(XL)	datagen-1000	12.8 M	1.01 B	9.0
G22(S)	graph500-22	2.40 M	64.2 M	7.8
G23(M)	graph500-23	4.61 M	129 M	8.1
G24(M)	graph500-24	8.87 M	260 M	8.4
G25(L)	graph500-25	17.1 M	524 M	8.7
G26(XL)	graph500-26	32.8 M	1.05 B	9.0

Local clustering coefficient (LCC): Computes the degree of clustering for each vertex, i.e., the ratio between the number of triangles a vertex closes with its neighbors to the maximum number of triangles it could close.

Single-source shortest paths (SSSP): Determines the length of the shortest paths from a given source vertex to all other vertices in graphs with double-precision floating-point weights.

2.2.4 Selected Datasets

Graphalytics uses both graphs from real-world applications and synthetic graphs which are generated using data generators. Table 3 summarizes the six selected real-world graphs. By including real-world graphs from a variety of domains, Graphalytics covers users from different communities. Our two-stage selection process led to the inclusion of graphs from the knowledge, gaming, and social network domains. Within the selected domains, graphs were chosen for their variety in sizes, densities, and characteristics.

The real-world graphs in Graphalytics are complemented by two synthetic dataset generators, to enable performance comparison between different graph scales. The synthetic dataset generators are selected to cover two commonly used graphs: power-law graphs generated by **Graph500**, and social network graphs generated using **LDBC Datagen** (see Section 2.5.1). The graphs generated for the experiments are listed in Table 4.

To facilitate performance comparisons across datasets, we define the *scale* of a graph in Graphalytics as a function of the number of vertices ($|V|$) *and* the number of edges ($|E|$) in a graph: $s(V, E) = \log_{10}(|V| + |E|)$, rounded to one decimal place. To give its users an intuition of what the scale of a graph means in practice, Graphalytics groups dataset scales into *classes*. We group scales in classes spanning 0.5 *scale units*, e.g., graphs in scale from 7.0 to 7.5 belong to the same class. The classes are labelled according to the familiar system of "T-shirt sizes": small (S), medium (M), and large (L), with extra (X) prepended to indicate smaller and larger classes to make extremes such as 2XS and 3XL possible.

The reference point is class L, which is intuitively defined by the Graphalytics team to be the largest class such that the BFS algorithm completes within an hour on any graph from that class in the Graphalytics benchmark using a state-of-the-art graph analysis platform on a single common-off-the-shelf machine. The resulting classes used by Graphalytics are summarized in Table 2.

2.3 Process

Addressing R3, the goal of the Graphalytics benchmark is to objectively compare different graph analysis platforms, facilitating the process of finding their strengths and weaknesses, and understanding how the performance of a platform is affected by aspects such as dataset, algorithm, and environment. To achieve this, the benchmark consists of a number of different *experiments*. In this section, we introduce these experiments, which we detail and conduct in Section 4.

The **baseline** experiments measure how well a platform performs for different workloads on a single machine. The core metric for measuring the performance of platforms is run-time. Graphalytics breaks down the total run-time into several components:

- **Upload time**: Time required to preprocess and convert the graph into a suitable format for a platform.
- **Makespan**: Time required to execute an algorithm, from the start of a job until termination.
- **Processing time** (T_{proc}): Time required to execute an actual algorithm as reported by the Graphalytics performance monitoring tool, Granula (Section 2.5.2). This does not include platform-specific overhead, such as allocating resources, loading the graph from the file system, or graph partitioning.

In our experiments we focus on T_{proc} as a primary indication of the performance of a platform. We complement this metric with two user-level throughput metrics:

- **Edges per second (EPS)**: Number of edges in a graph divided by T_{proc} in seconds. EPS is used in other benchmarks, such as Graph500.
- **Edges and vertices per second (EVPS)**: Number of edges plus number of vertices (i.e., 10^{scale}, see Section 2.2.4), divided by T_{proc} in seconds. EVPS is closely related to the scale of a graph, as defined by Graphalytics.

To investigate how well a platform performs when scaling the amount of available resources, the size of the input, or both, Graphalytics includes **scalability** experiments. We distinguish between two orthogonal types of scalability: strong vs. weak scalability, and horizontal vs. vertical scalability. The first category determines whether the size of the dataset is increased when increasing the amount of resources. For **strong scaling**, the dataset is kept constant, whereas for **weak scaling**, the dataset is scaled. The second category determine *how* the amount of resources is increased. For **horizontal scaling**, resources are added as additional computing machines, whereas for **vertical scaling** the added resources are cores within a single machine. Graphalytics expresses scalability using a single metric:

- **Speedup (S)**: The ratio between T_{proc} for scaled and baseline resources. We define the baseline for each platform and workload as the minimum amount of resources needed by the platform to successfully complete the workload.

Finally, Graphalytics assesses the **robustness** of graph analysis platforms using two metrics:

- **Stress-test limit**: The scale and label of the smallest dataset defined by Graphalytics that the system cannot process.
- **Performance variability**: The coefficient of variation (CV) of the processing time, i.e., the ratio between the standard deviation and the mean of the repeatedly measured performance. The main advantage of this metric is its independence of the scale of the results.

For all experiments, Graphalytics defines a service-level agreement (SLA): generate the output for a given algorithm and dataset with a makespan of up to 1 hour. A job breaks this SLA, and thus does not complete successfully, if its makespan exceeds 1 hour or if it crashes (e.g., due to insufficient resources).

Auto-validation: After each job, its output is validated by comparing it against the reference output. The output does not have to be exactly identical, but is must be equivalent under an algorithm-specific comparison rule (for example, for PageRank we allow a 0.01% error).

2.4 Renewal Process

Addressing requirement R4, we include in Graphalytics a renewal process which leads to a new version of the benchmark every two years. This renewal process updates the workload of the benchmark to keep it relevant for increasingly powerful systems and developments in the graph analysis community. This results in a benchmark which is future-proof. Renewing the benchmark means renewing the algorithms as well as the datasets. For every new version of Graphalytics, we follow the same two-stage workload selection process as presented in Section 2.2.2.

The algorithms of Graphalytics have been selected based on their representativeness. However, over time, graph algorithms might lose or gain popularity in the community. For example, community detection is an active field of graph research nowadays, even though our label propagation algorithm [36] was only introduced less than a decade ago. To ensure that algorithms stay relevant, for every version of the benchmark, we will select a new set of core algorithms using the same process as presented in Section 2.2.3. We will perform a new comprehensive survey on graph analysis in practice to determine new algorithm classes and select new algorithms from these classes using expert advice from LDBC TUC. If a new algorithm is found to be relevant which was not part of the set of core algorithms, it will be added. If an older core algorithm is found to be no longer relevant, it is marked as obsolete and will be removed from the specification in the next version.

The datasets of Graphalytics have been selected based on their variety in size, domain, and characteristics [24, 5]. Using the same process as described for algorithms, the Graphalytics team will introduce additional real-world datasets and synthetic dataset generators as they become relevant to the community. This may include graphs from new application domains if they are not yet represented by similar graphs from other domains. In addition, with every new version of the specification the notion of a "large" graph is reevaluated. In particular, class L is redefined as the largest class of graphs such that at a state-of-the-art platform can complete the BFS algorithm within one hour on all graphs in class L using a single common-off-the-shelf machine. The selection of platforms used to determine class L is limited

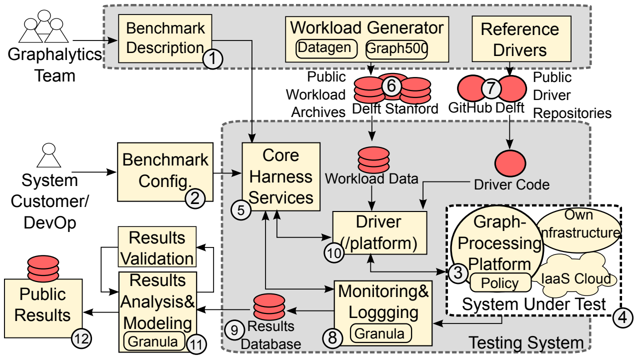

Figure 1: Graphalytics architecture, overview.

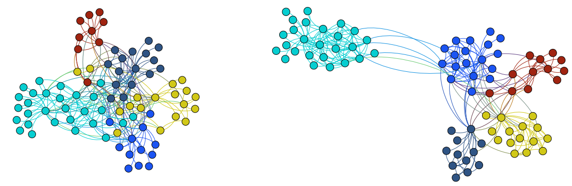

Figure 2: Datagen graphs with a target average clustering coefficient of 0.05 (l) and 0.3 (r). Communities (colors) detected using the Louvain algorithm.

to platforms implementing Graphalytics that are available to the Graphalytics team when the new specification is formalized.

2.5 Design of the Graphalytics Architecture

The Graphalytics architecture, depicted in Figure 1, consists of a number of components, including the system under test and the testing system.

As input for the benchmark, the Graphalytics team provides a benchmark description (1). This description includes definitions of the algorithms, the datasets, and the algorithm parameters for each graph (e.g., the root for BFS or number of iterations for PR). In addition, the system customer, developer, or operator can configure the benchmark (2). The benchmark user may select a subset of the Graphalytics workload to run, or they may tune components of the system under test for a particular execution of the benchmark.

The workload of Graphalytics is executed on a specific graph analysis platform (3), as provided by the user. This platform is deployed on user-provided infrastructure, e.g., on machines in a self-owned cluster or on virtual machines leased from IaaS clouds. The graph analysis platform and the infrastructure it runs on form the system under test (4). The graph analysis platform may optionally include policies to automatically tune the system under test for different parts of the benchmark workload.

At the core of the testing system are the Graphalytics harness services (5). The harness processes the benchmark description and configuration, and orchestrates the benchmarking process. Two components of the workload, datasets (6) and algorithm implementations (i.e., driver code (10)), must be provided by the benchmark user. Datasets can be obtained through public workload archives, or generated using a workload generator, such as LDBC Datagen. Reference drivers can be provided by platform vendors or obtained from public repository. The Graphalytics team also offers the drivers for a number of platforms (7).

A platform can be integrated with the Graphalytics harness through a platform-specific driver (10). The driver must implement a well-defined API consisting of several operations, including uploading a graph to the system under test (this may include pre-processing to transform the provided dataset into a format compatible with the target platform), executing an algorithm with a specific set of parameters on an uploaded graph, and returning the output of an algorithm to the harness for validation. Additionally, the driver may provide a detailed performance model of the platform to enable detailed performance analysis using Granula (described in Section 2.5.2).

The final component of the testing system is responsible for monitoring and logging (8) the system under test, and storing the obtained information in a results database (9). Raw monitoring information is gathered using Granula (see Section 2.5.2), and can be analyzed after each run or offline to extract rich information about the performance of the system under test (11). Finally, the results are validated, the SLA is checked and the results are stored in a repository to track benchmark results across platforms.

To address the requirement for modern software engineering practices (R5), all components of the Graphalytics architecture provided by the Graphalytics team are developed online as open source software. To maintain the quality of the Graphalytics software, continuous integration is used and contributions are peer-reviewed by the Graphalytics maintainers. Through its development process, Graphalytics also invites collaboration with platform vendors, as evidenced by the contributions already made to Graphalytics drivers.

2.5.1 LDBC Datagen: Graph Generation

Graphalytics relies not only on real but also on synthetically generated graphs. Synthetic graph generators provide a means of testing data configurations not always available in the form of real datasets (e.g., due to privacy concerns). Thus, Graphalytics adopts the LDBC Social Network Benchmark Data generator (Datagen) [17][1], a scalable, synthetic social network generator, whose output preserves many realistic graph features: correlated data (i.e., persons with similar characteristics are more likely to be connected), skewed degree distribution (it generates a Facebook-like friendship distribution), non-uniform activity volume, etc. However, the static nature of Datagen did not allow for the generation of graphs with different degree distributions or structural characteristics. Thus, as envisioned previously [12], we have extended for this work Datagen to generate graphs with these characteristics. Moreover, we have optimized the critical execution path of Datagen, to improve its performance and scalability. We summarize this two-fold contribution as follows:

Tunable Clustering Coefficient: Besides supporting different degree distributions [12], we now also allow changing the friendship generation algorithm. With the goal of generating realistic yet diverse graphs, we have implemented an edge generator which allows tuning the average clustering coefficient of the resulting friendship graph. The method relies on constructing a graph with a core-periphery community structure. Such communities are ubiquitous in social networks and their presence is strongly related to other

[1]Available at `github.com/ldbc/ldbc_snb_datagen`

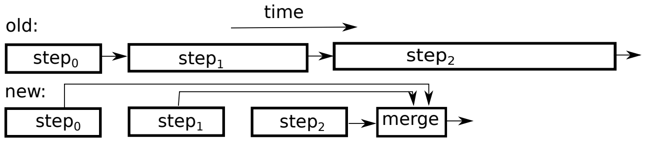

Figure 3: Datagen: old vs new execution flow.

real-world graph properties, such as a small diameter and a large connected component. Figure 2 shows two small graphs generated with Datagen, with two different target average clustering coefficients of 0.05 (left) and 0.3 (right). Both graphs exhibit a community structure (shown in different colors, detected by the Louvain method), but we see that the right one is clearly better defined than the one in the left, a consequence of the larger average clustering coefficient.

Datagen generates friendships between persons falling in the same block. For details on how blocks are constructed please refer to LDBC SNB [17]. Given a block of persons and their expected degree, *the new goal set for this work is to build communities including these persons*, while maintaining the correlated nature of the produced graph (consecutive persons in a block must have a larger probability to connect) and achieving a given average clustering coefficient.

Optimization of execution flow: Figure 3 shows the old versus the new execution flow implemented for the Person-Person graph generation in Datagen. In the old flow, the output produced by step i (Persons and all edges generated in steps from 0 to i) is read by $i+1$, which sorts it by the corresponding correlation dimension and produces new edges. Thus, the cost of running a step grows as more edges are produced in previous steps, because more data needs to be sorted. This is exemplified in the figure with the lengths of the steps. This design guarantees that no duplicate edges are generated. In the new flow, each edge generation step is independent of the rest, and its output is written into a different file. Later, all files are merged to remove the duplicates. This approach is more efficient because the cost of executing a step remains constant, as does the amount of I/O required for the sorting. As we will see in Section 4.8, the performance improvements are significant.

2.5.2 Granula: Fine-grained Evaluation

Performance evaluation is a critical part in developing a graph analysis platform, as it helps developers gain a better understanding of the platform's performance. However, the comprehensive evaluation of graph-analysis platforms still faces many challenges: using a coarse-grained "black-box" approach does not provide sufficient insight into the platform performance; using a fine-grained approach is too time-consuming and requires in-depth knowledge of the platform architecture; and finally it is difficult for users to apply the results of empirical performance studies for their specific use cases.

To extend Graphalytics with fine-grained performance evaluation, we developed *Granula* [33][2], a performance evaluation framework consisting of three main modules: the modeler, the archiver, and the visualizer.

Modeler: Fine-grained evaluation of graph-analysis platforms requires domain-specific expertise and can be time-consuming. The Granula modeler allows experts to explicitly define once their evaluation method for a graph analysis platform, such that the evaluation process can be fully automated. This includes defining phases in the execution of a job (e.g., graph loading), and recursively defining phases as a collection of smaller, lower-level phases (e.g., graph loading includes reading and partitioning), up to the required level of granularity. The performance model may also include other information, such as the number of vertices processed in a phase.

Archiver: The *Granula* archiver uses the performance model of a graph analysis platform to collect and archive detailed performance information for a job running on the platform. Such information is either gathered from log files produced by the platform, or derived using rules defined in the performance model. The archiver produces a performance archive which encapsulates the comprehensive set of performance information captured for each job. The archive is complete (i.e., all observed and derived results are included), descriptive (i.e., all results are described to non-experts) and examinable (i.e., all results are derived from a traceable source).

Visualizer: While a performance archive is sufficiently informative, it is not the most natural way of examining performance results. The Granula visualizer presents the performance archive in a human-readable manner and allows efficient navigation through the performance results at varying levels of granularity using an interactive Web interface. Results presented using the Granula visualizer can be easily communicated and shared among performance analysts with different levels of expertise.

In this work, we use Granula to build comprehensive performance models for several graph processing platforms, e.g., Apache Giraph (see detailed analysis in our technical report [28]). And for each platform, we have developed a basic performance model which allow us to define, capture, and report fine-grained performance breakdown metrics, e.g., *processing time* (See Section 2.3).

3. EXPERIMENTAL SETUP

A major contribution of this work is the evaluation and comparison of graph analysis platforms whose development is *industry-driven*, and of other, *community-driven*, platforms. In this section, we present the setup of our experiments.

3.1 Selected Platforms

We evaluate and compare in this work six different graph analysis platforms, three *community-driven* (C) and three *industry-driven* (I), see Table 5. These platforms are based on six different programming models, spanning an important design space for real-world graph analysis.

Table 5: Selected graph analysis platforms. Acronyms: C, community-driven; I, industry-driven; D, distributed; S, non-distributed.

Type	Name	Vendor	Lang.	Model	Vers.
C, D	Giraph [1]	Apache	Java	Pregel	1.1.0
C, D	GraphX [41]	Apache	Scala	Spark	1.6.0
C, D	PowerGraph [21]	CMU	C++	GAS	2.2
I, S/D	GraphMat [39, 8]	Intel	C++	SpMV	May '16
I, S	OpenG [32]	G.Tech	C++	Native code	May '16
I, S/D	PGX [4, 27]	Oracle	Java/C++	Push-pull	May '16

[2]Available at `github.com/tudelft-atlarge/granula`

Table 6: Experiments used for benchmarks.

Category	Sec.	Experiment	Algorithms	Datasets	#nodes	#threads	Metric
Baseline	4.1	Dataset variety	BFS, PR	All, up to L	1	-	T_{proc}, E(V)PS
	4.2	Algorithm variety	All	R4(S), D300(L)	1	-	T_{proc}
Scalability	4.3	Vertical	BFS, PR	D300(L)	1	1-32	T_{proc}, S
	4.4	Strong/Horizontal	BFS, PR	D1000(XL)	1-16	-	T_{proc}
	4.5	Weak/Horizontal	BFS, PR	G22(S)-26(XL)	1-16	-	T_{proc}
Robustness	4.6	Stress test	BFS	All	1	-	SLA
	4.7	Variability	BFS	D300(L), D1000(XL)	1, 16	-	CV
Self-Test	4.8	Data Generation	-	-	-	-	T_{gen}

The platforms can be categorized into two classes: **distributed** (D) and **non-distributed** (S) platforms. Distributed platforms use when analyzing graphs multiple machines connected using a network, whereas non-distributed platforms can only use a single machine. Distributed systems suffer from a performance penalty because of network communication, but can scale to handle graphs that do not fit into the memory of a single machine. Non-distributed systems cannot scale as well because of the limited amount of resources of a single machine.

Apache Giraph [1] uses an iterative vertex-centric programming model similarly to Google's Pregel. Giraph is open source and built on top of Apache Hadoop's MapReduce.

Apache GraphX [41] is an extension of Apache Spark, a general platform for big data processing. GraphX extends Spark with graphs based on Spark's Resilient Distributed Datasets (RDDs).

PowerGraph [21], developed by Carnegie Mellon University, is designed for real-world graphs which have a skewed power-law degree distribution. PowerGraph uses a programming model known as Gather-Apply-Scatter (GAS).

GraphMat [39, 8], developed by Intel, maps Pregel-like vertex programs to high-performance sparse matrix operations, a well-developed area of HPC. GraphMat supports two different backends which need to be selected manually: a single-machine shared-memory backend [39] and a distributed MPI-based backend [8].

OpenG [32] consists of handwritten implementations for many graph algorithms. OpenG is used by *GraphBIG*, a benchmarking effort initiated by Georgia Tech and inspired by IBM System G.

PGX [4], developed by Oracle, is designed to analyze large scale graphs on modern hardware systems. PGX has two different runtimes: a single-machine shared memory runtime implemented in Java and a distributed runtime [27] implemented in C++. Both runtimes share the same user facing API and are part of the Oracle Big Data Spatial and Graph product [3].

For GraphMat and PGX, we report single-machine results using the single-machine backend, and horizontal scalability results using the distributed backend. For the single-machine horizontal scalability experiments we report results for both backends. Processing times reported for PGX shared memory exclude its integrated warm up procedure. Cold processing times are additionally reported in our technical report [28].

Table 7: Hardware specifications.

Component	Name
CPU	2 × Intel Xeon E5-2630 @ 2.40 GHz
Cores	16 (32 threads with Hyper-Threading)
Memory	64 GiB
Disk	2 × 4 TB
Network	1 Gbit/s Ethernet, FDR InfiniBand

3.2 Environment

Experiments have been performed on the DAS-5 [2] (Distributed ASCII Supercomputer), consisting of 6 clusters with over 200 dual 8-core compute nodes. DAS-5 is funded by a number of organizations and universities from the Netherlands and is actively used as a tool for computer science research in the Netherlands. We use individual clusters for the experiments and we test all platforms on the same hardware. The hardware specifications of the machines in the clusters are listed in Table 7.

4. EXPERIMENTAL RESULTS

Graphalytics conducts automatically the complex set of experiments summarized in Table 6. The experiments are divided into four categories: **baseline**, **scalability**, and **robustness** (all introduced in Section 2.3); and **self-test**. Each category consists of a number of experiments, for which Table 6 lists the parameters used for the benchmarks (algorithm, dataset, number of machines, and number of threads) and the metrics used to quantify the results.

4.1 Dataset Variety

For this experiment, Graphalytics reports the processing time of all platforms executing BFS and PageRank on a variety of datasets using a single node. Key findings:

- GraphMat and PGX significantly outperform their competitors in most cases.
- PowerGraph and OpenG are roughly an order of magnitude slower than the fastest platforms.
- Giraph and GraphX are consistently two orders of magnitude slower than the fastest platforms.
- Across datasets, all platforms show significant variability in performance normalized by input size.

The workload consists of two selected algorithms and all datasets up to class L. We present the processing time (T_{proc}) in Figure 4, and the processed edges per second (EPS) and processed edges plus vertices per second (EVPS) in Figure 5. The vertical axis in both figures lists datasets, ordered by scale (results for missing datasets are available in our technical report [28]).

Figure 4 depicts the processing time of BFS and PageRank for all platforms on a variety of datasets. For both algo-

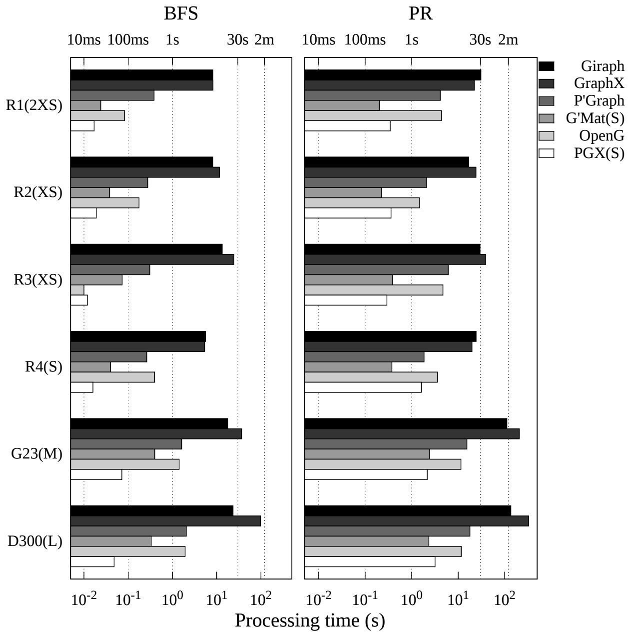

Figure 4: Dataset variety: T_{proc} for BFS and PR.

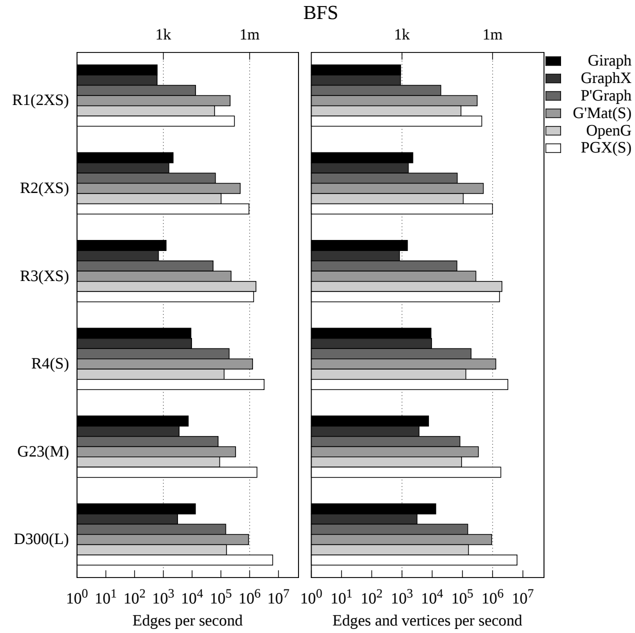

Figure 5: Dataset variety: EPS and EVPS for BFS.

rithms, GraphMat and PGX are consistently fast, although PGX has significantly better performance on BFS. Giraph and GraphX are the slowest platforms and both are two orders of magnitude slower than GraphMat and PGX for most datasets. Finally, OpenG and PowerGraph are generally slower than both PGX and GraphMat, but still significantly faster than Giraph and GraphX. A notable exception is OpenG's performance for BFS on dataset R3(XS). The BFS on this graph covers approximately 10% of the vertices in the graph, so OpenG's queue-based BFS implementation results in a large performance gain over platforms that process all vertices using an iterative algorithm.

To better understand the sensitivity of the tested platforms to the datasets, we present normalized processing times for the BFS algorithm in Figure 5. The left and right subfigures depict EPS and EVPS, respectively. Ideally, a platform's performance should be proportional to graph size, thus the normalized performance should be constant. As evident from the figure, all platforms show signs of dataset sensitivity, as EPS and EVPS vary between datasets.

Besides T_{proc}, it is also interesting to look at the makespan (i.e., time spent on the complete job for one algorithm). This includes platform-specific overhead such as resource allocation and graph loading. Table 8 lists the makespan, T_{proc}, and their ratio for BFS on D300(L). The percentages show that the overhead varies widely for the different platforms and ranges from 66% to over 99% of the makespan. However, we note that the platforms have not been tuned to minimize this overhead and in many cases it could be significantly reduced by optimizing the configuration. In addition, we observe that the majority of the runtime for all platforms is spent in loading the input graph, indicating that algorithms could be executed in succession with little overhead.

Table 8: T_{proc} and makespan for BFS on D300(L).

Time	Giraph	GraphX	P'Graph	G'Mat(S)	OpenG	PGX(S)
Makespan	277.9 s	278.4 s	216.5 s	23.3 s	5.7 s	14.3 s
T_{proc}	23.4 s	97.9 s	2.1 s	0.3 s	1.9 s	0.05 s
Ratio	8.4%	35.2%	1.0%	1.3%	33.3%	0.3%

4.2 Algorithm Variety

The second set of baseline experiments focuses on the algorithm variety in the Graphalytics benchmark, and on how the performance gap between platforms varies between workloads.

- Relative performance between platforms is similar for BFS, WCC, PR, and SSSP.
- LCC is significantly more demanding than the other algorithms, Giraph and GraphX are unable to complete it without breaking the SLA.
- GraphX is unable to complete CDLP. The performance gap for the remaining platforms for CDLP is smaller than for the other algorithms.

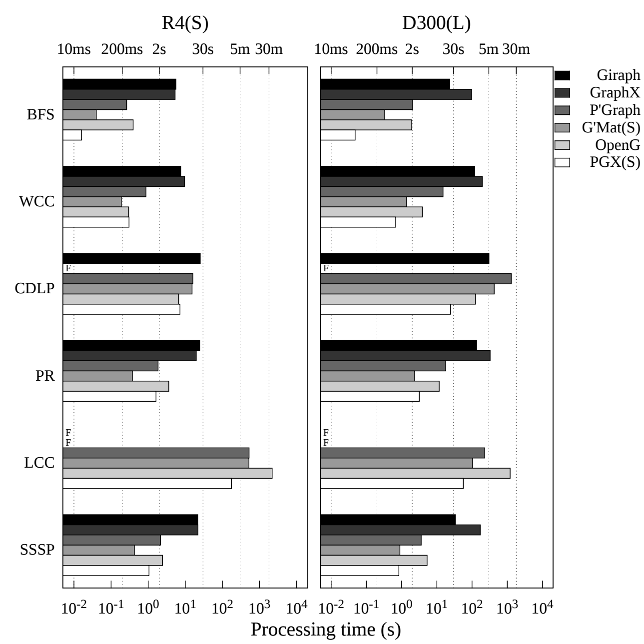

Figure 6: Algorithm variety: T_{proc}.

Figure 6 depicts T_{proc} for the core algorithms in Graphalytics on two graphs with edge weights: R4(S), the largest real-world graph in Graphalytics with edge-weights; and D300(L). BFS, WCC, PR, and SSSP all show similar results. PGX and GraphMat are the fastest platforms. Giraph, GraphX and PowerGraph are much slower, with GraphX showing the worst performance, especially on D300(L). OpenG's performance is close to that of PGX and GraphMat on WCC, and up to an order of magnitude worse for BFS, PR, and SSSP. CDLP requires more complex computation which results in longer processing times for all platforms, reducing the performance impact of the chosen platform, especially on smaller graphs like R4(S). GraphX is unable to complete CDLP on both graphs. LCC is also very demanding; Giraph and GraphX break the SLA for both graphs. The complexity of the LCC algorithm depends on the degrees of vertices, so longer processing times are expected on dense graphs. Because of the high density of R4(S), processing times are larger on this graph than on D300(L), despite it being an order of magnitude smaller.

4.3 Vertical Scalability

To analyze the effect of adding additional resources in a single machine, we use Graphalytics to run the BFS and PageRank algorithms on D300(L) with 1 up to 32 threads on a single machine. Key findings:

- All platforms benefit from using additional cores, but only PowerGraph exceeds a speedup of 10 on 16 cores.
- Most platforms experience minor or no performance gains from Hyper-Threading.

Figure 7 depicts the processing time for this experiment. The majority of tested platforms show increasing performance as threads are added up to 16, the number of cores. Adding additional threads up to 32, the number of threads with hardware support through Hyper-Threading, does not appear to improve the performance of GraphX, GraphMat, or PGX. PowerGraph benefits most from the additional 16 threads; it achieves an additional 1.5x speedup on BFS. The maximum speedup obtained by each platform is summarized in Table 9. Overall, PowerGraph scales best with a maximum speedup of 12.5.

4.4 Strong Horizontal Scalability

We use Graphalytics to run BFS and PR for all distributed platforms on D1000(XL) while increasing the number of machines from 1 to 16 in powers of 2 to measure strong scalability. Key findings:

- PGX and GraphMat show a reasonable speedup.
- Giraph's performance degrades significantly when switching from 1 machine to 2 machines, but improves significantly with additional resources.
- PowerGraph and GraphX scale poorly; GraphX shows no performance increase past 4 machines.

The processing times for this experiment are depicted in Figure 8. Ideally, T_{proc} halves when the amount of resources (i.e., the number of machines) is doubled given a constant workload. Giraph suffers a large performance hit

Table 9: Vertical scalability: speedup on D300(L) for 1–32 threads on 1 machine.)

Alg.	Giraph	GraphX	P'Graph	G'Mat(S)	OpenG	PGX(S)
BFS	5.6	4.6	12.5	7.2	6.9	9.5
PR	8.5	3.1	10.5	11.2	6.3	7.7

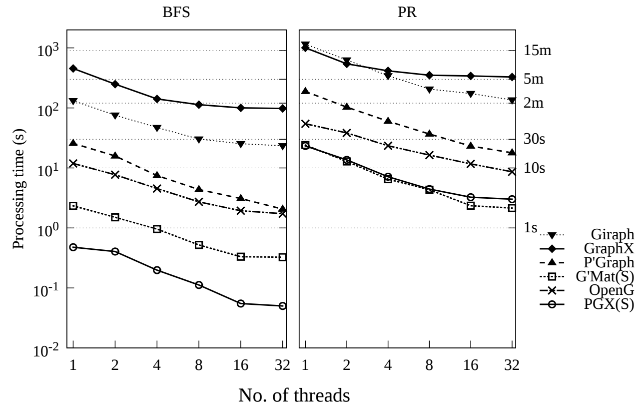

Figure 7: Vertical scalability: T_{proc} vs. #threads.

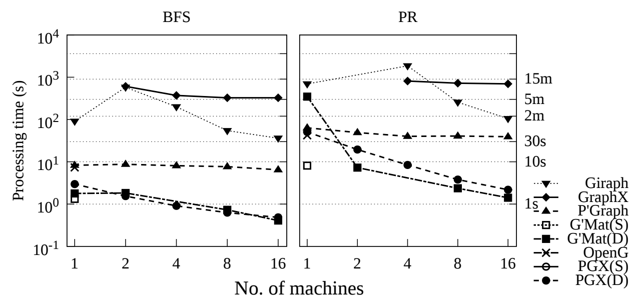

Figure 8: Strong scalability: T_{proc} vs. #machines.

when switching from 1 machine to a distributed setup with 2 machines. For PR, this results in an SLA failure on 2 machines, even though it succeeds on 1 machine. At least 4–8 machines are required for Giraph to improve in performance over the single-machine setup. GraphX also scales poorly for the given workload. It requires 2 machines to complete BFS, and 4 machines to complete PR. GraphX achieves a speedup of 1.9 using 8 times as many resources on BFS, and a speedup of 1.2 with 4 times as many resources on PR. PowerGraph is able to process the D1000(XL) graph on any number of nodes, but scales poorly for both BFS and PR. Both PGX and GraphMat show significant speedup. However, for PR both platforms show super-linear speedup when using 2 machines, possibly due to resource limitations on a single machine. In our environment, GraphMat crashed on 4 machines due to an unresolved issue in the used MPI implementation.

For comparison, results for the single-machine backends are included. Distributed GraphMat on two machines performs on-par with the single-machine backend. PGX shared memory was unable to complete either algorithm due to memory limitations.

4.5 Weak Horizontal Scalability

To measure weak scalability, Graphalytics runs BFS and PR for all distributed platforms on Graph500 G22(S) through G26(XL) while increasing the number of machines from 1 to 16 in powers of 2. The amount of work per machine is approximately constant, as each graph in the series generated using Graph500 is twice as large as the previous graph. As the workload per machine is constant, T_{proc} is ideally constant. Key findings:

- None of the tested platforms achieve optimal weak scalability.

Table 10: Stress Test: the smallest dataset that failed to complete BFS successfully on one machine.

Platform	Giraph	GraphX	P'graph	G'Mat(S)	OpenG	PGX(S)
Dataset	G26(XL)	G25(L)	R5(XL)	G26(XL)	R5(XL)	G25(L)
Scale	9.0	8.7	9.3	9.0	9.3	8.7

- Giraph's performance degrades significantly on 2 machines, but scales well from 4 to 16 machines.
- GraphX and PowerGraph scale poorly, whereas GraphMat scales best.

In Figure 9, GraphX and PowerGraph show increasing processing times as the number of machines increases, peaking at a maximum slowdown (i.e., inverse of speedup) of 15.5 and 8.2, respectively. Similar to the strong scalability experiments, Giraph's performance is worst with two machines and shows a slowdown of 15.5 on PR. Performance improves slightly as more machines are added, for a slowdown of 4.7 with 16 machines on PR. GraphMat shows the best scalability with a maximum slowdown of 2.1. Although PGX outperforms GraphMat on a single machine, GraphMat's better scalability allows it outperform PGX when using 2 or more machines.

4.6 Stress Test

To test the maximum processing capacity of each platform, we use Graphalytics to run the BFS algorithm on all datasets, and report the scale of the smallest dataset that breaks the SLA (Section 6) on a single machine. Key findings:

- GraphX and PGX fail to process the largest class L graph on a single machine.
- Most platforms fail on a Graph500 graph, but succeed on a Datagen graph of comparable scale. This indicates sensitivity to graph characteristics other than graph size.
- PowerGraph and OpenG can process the largest graphs on a single machine, up to scale 9.0.

Table 10 lists the smallest graph, by scale, for which each platform fails to complete. The results show that both GraphX and PGX are unable to complete the BFS algorithm on Graph500 scale 25, a class L graph. PGX is specifically optimized for machines with large amount of cores and memory, and thus exceeds the memory capacity of our machines. Like GraphX and PGX, Giraph and GraphMat fail on a Graph500 graph. Both platforms successfully process D1000 with scale 9.0, but fail on G26 of the same scale. This suggests that characteristics of the graphs affect the performance of graph analysis platforms, an issue not revealed by the Graph500 benchmark. Finally, PowerGraph and OpenG fail to complete BFS on the Friendster graph, a

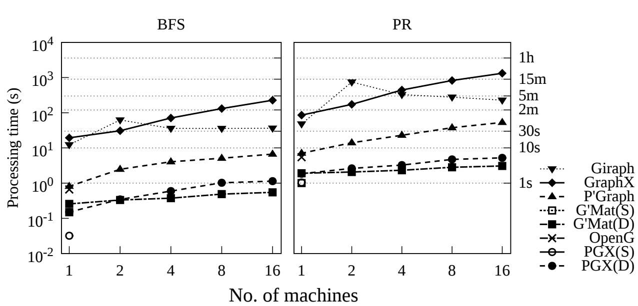

Figure 9: Weak scalability: T_{proc} vs. #machines.

Table 11: Variablity: T_{proc} mean and coefficient of variation. BFS on 1 (S) and 16 (D) nodes, $n = 10$.

		Giraph	GraphX	P'graph	G'Mat	OpenG	PGX
S	Mean	22.3s	101.5s	2.1s	0.4s	2.0s	49ms
	CV	5.0%	2.6%	1.5%	13.6%	5.8%	8.6%
D	Mean	36.4s	326.9s	6.6s	0.4s	-	0.6s
	CV	8.0%	4.3%	3.3%	4.2%	-	28.5%

scale 9.3 graph and among the largest graphs currently used by Graphalytics.

4.7 Variability

To test the variability in performance of each platform, Graphalytics runs BFS 10 times on D300(L) with 1 machine for all platforms, and on D1000(XL) with 16 machines for the distributed platforms.

- Most platforms have a CV of at most 10%, i.e., their standard deviation is at most 10% of the mean T_{proc}.
- GraphMat (S) and PGX show higher than average variability. However, due to their much smaller mean, the absolute variability is small.

The mean and CV for T_{proc} are reported in Table 11. In both S and D configurations, PowerGraph shows the least variability in performance. GraphX has similarly low variability, but due to its significantly longer mean processing time it can deviate by tens of seconds between runs. Conversely, GraphMat and PGX show much larger variability between runs, but in absolute values their deviation is limited to tens of milliseconds.

4.8 Data Generation

We also evaluate the performance and scalability of Datagen with the new execution flow presented in Section 2.5.1. We compare the new version of Datagen (v0.2.6) against the latest version not including these performance optimization (v0.2.1). For these experiments, we used Hadoop 2.4.1 on the DAS-4 (dual Intel Xeon E5620, 24 GiB RAM, spinning disks, 1 Gbit/s Ethernet) to perform the experiments, which leads to conservative results. For each run, one machine is reserved as a master while the others are workers. The number of mappers is controlled by Hadoop, and depends on the size of the input files. The number of reducers per worker is set to 16 (one per core).

Figure 10 (l) shows the execution time (T_{gen}) of the two versions, running on 16 machines, for five different scale factors. The scale factor reflects the approximate number of edges in millions. For the five scale factors (30, 50, 300, 500, 3000), the new version improves the execution time by a factor of between 1.16 and 2.9, where the speedup increases with the scale factor. This indicates an increase in scalabil-

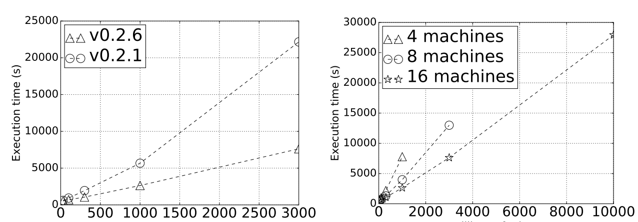

Figure 10: Execution time vs. #edges in the generated graph for Datagen: (left) v0.2.6 (new in this this work) vs v0.2.1 (old), for 16 machines; (right) 4 vs. 8 vs. 16 machines on v0.2.6.

Table 12: Summary of related work. (Acronyms: *Reference type*: S, study, B, benchmark. *Target system, structure*: D, distributed system; P, parallel system; MC, single-node multi-core system; GPU, using GPUs. *Input*: 0, no parameters; S, parameters define scale; E, parameters define edge properties; +, parameters define other graph properties, e.g., clustering coefficient. *Datasets/Algorithms*: Rnd, reason for selection not explained; Exp, selection guided by expertise; 1-stage, data-driven selection; 2-stage, 2-stage data- and expertise-driven process. *Scalability tests*: W, weak, S, strong, V, vertical, H, horizontal.)

Reference (chronological order)		Target System (R1)		Design (R2)				Tests (R3)		(R4)
	Name [Publication]	Structure	Programming	Input	Datasets	Algo.	Scalable?	Scalability	Robustness	Renewal
B	CloudSuite [19], only graph elements	D/MC	PowerGraph	S	Rnd	Exp	—	No	No	No
S	Montresor et al. [16]	D/MC	3 classes	0	Rnd	Exp	—	No	No	No
B	HPC-SGAB [10]	P	—	S	Exp	Exp	—	No	No	No
B	Graph500	P/MC/GPU	—	S	Exp	Exp	—	No	No	No
B	GreenGraph500	P/MC/GPU	—	S	Exp	Exp	—	No	No	No
B	WGB [7]	D	—	SE+	Exp	Exp	1B Edges	No	No	No
S	**Own prior work** [24, 25, 12]	**D/MC/GPU**	**10 classes**	**S**	**Exp**	**1-stage**	**1B Edges**	**W/S/V/H**	**No**	**No**
S	Özsu et al. [26]	D	Pregel	0	Exp,Rnd	Exp	—	W/S/V/H	No	No
B	BigDataBench [31, 40], only graph elements	D/MC	Hadoop	S	Rnd	Rnd	—	S	No	No
S	Satish et al. [37]	D/MC	6 classes	S	Exp,Rnd	Exp	—	W	No	No
S	Lu et al. [30]	D	4 classes	S	Exp,Rnd	Exp	—	S	No	No
B	GraphBIG [32]	P/MC/GPU	System G	S	Exp	Exp	—	No	No	No
S	Cherkasova et al. [15]	MC	Galois	0	Rnd	Exp	—	No	No	No
B	**LDBC Graphalytics** (this work)	**D/MC/GPU**	**10+ classes**	**SE+**	**2-stage**	**2-stage**	**Process**	**W/S/V/H**	**Yes**	**Yes**

ity. Overall, the new version takes just 44 minutes to generate a billion-edge graph using 16 machines, a significant improvement over the 95 minutes of the old version.

Figure 10 (r) shows the execution time of the new version for different cluster sizes and scale factors. We see that Datagen scales very well. For example, increasing the scale factor by a factor of ten (from 1000 to 10000) increases the execution time by 10.6. This means a 10 billion edge graph can be generated in less than 8 hours, using commodity hardware from 2010. The poor scalability observed for smaller scales is due to the constant overhead incurred by Hadoop, which is negligible for large scale factors. We also observe very good horizontal scalability. For example, the speedup from 4 to 16 machines is 3.0 for scale factor 1000. This means more hardware can be added to generate larger datasets faster. We conclude that Datagen can generate large and complex graphs on small-sized clusters of commodity hardware in reasonable amounts of time.

5. RELATED WORK

Table 12 summarizes and compares Graphalytics with previous studies and benchmarks for graph analysis systems. R1–R5 are the requirements formulated in Section 2.1. As Table 12 indicates, there is no alternative to Graphalytics in covering requirements R1–R4. We also could not find evidence of requirement R5 being covered by other systems than LDBC. While there have been a few related benchmark proposals (marked "B"), these either do not *focus* on graph analysis, or are much narrower in scope (e.g., only BFS for Graph500). There have been comparable studies (marked "S") but these have not attempted to define—let alone maintain—a benchmark, its specification, software, testing tools and practices, or results. Graphalytics is not only industry-backed but also has industrial strength, through its detailed execution process, its metrics that characterize robustness in addition to scalability, and a renewal process that promises longevity. Graphalytics is being proposed to SPEC as well, and BigBench [20, 35] explicitly refers to Graphalytics as its option for future benchmarking of graph analysis platforms.

Previous studies typically tested the open-source platforms Giraph [1], GraphX [41], and PowerGraph [21], but our contribution here is that vendors (Oracle, Intel, IBM) in our evaluation have themselves tuned and tested their implementations for PGX [27], GraphMat [39] and OpenG [32]. We are aware that the database community has started to realize that with some enhancements, RDBMS technology could also be a contender in this area [18, 29], and we hope that such systems will soon get tested with Graphalytics.

Graphalytics complements the many existing efforts focusing on graph databases, such as LinkedBench [9], XGDBench [14], and LDBC SNB [17]; efforts focusing on RDF graph processing, such as LUBM [23], the Berlin SPARQL Benchmark [11], SP^2Bench [38], and WatDiv [6] (targeting also graph databases); and community efforts such as the TPC benchmarks. Whereas all these prior efforts are interactive database query benchmarks, Graphalytics focuses on algorithmic graph analysis and on different platforms which are not necessarily database systems, whose distributed and highly parallel aspects lead to different design trade-offs.

6. CONCLUSION

Responding to an increasing use of large-scale graphs, industry and academia have proposed a variety of distributed and highly-parallel graph analysis platforms. To compare these platforms, but also to tune them and to enable future designs, the Linked Data Benchmark Council (LDBC) has been tasked by its industrial constituency to develop an offline (batch) graph analysis workload—the LDBC Graphalytics benchmark, which is the focus of this work. Graphalytics[3] brings both conceptual and technical contributions, and is used to compare three main community-driven and three vendor-tuned graph analysis platforms.

[3]Acknowledgements: This research is supported by the Dutch NWO through projects Vidi MagnaData, KIEM KIESA, COMMIT COMMIssioner, and by LDBC (ldbcouncil.org), originally funded by EU project FP7-317548. This work is also partially supported by a donation from Oracle Research Labs.

The Graphalytics workload was designed through a two-stage selection incorporating the concept of choke-point (expertise-driven) design, and a data-driven selection of relevant algorithms and input datasets. The specification of Graphalytics is innovative: its metrics go beyond the traditional performance metrics, and in particular enable deep studies of two key features of distributed and highly-parallel systems, scalability and robustness. Graphalytics is the first graph benchmark to cover stress-testing and performance variability. The benchmarking process is managed by an advanced harness, which includes flexible and scalable tools for data collection, analysis, and sharing, and for distributed generation of synthetic yet realistic graph datasets. In particular, the data generator tool Datagen is the first to generate graphs with a pre-specified clustering coefficient for benchmarking. Graphalytics also specifies a novel process for renewing its core parameters, to withstand the test of time while still being understandable for non-experts.

We present here the open-source implementation of the harness[4], which is able to conduct over ten different experiments, collect in-depth data that can be further used for tuning, and then extract the relevant benchmarking metrics. We also provide reference implementations of the drivers and algorithms for six target systems, which differ widely in distribution and programming model. Three of the systems originate from industry: PGX from Oracle, GraphMat from Intel, and OpenG from IBM. We hope and believe Graphalytics to be interesting for academics, IT practitioners, industry engineers, and system designers.

7. REFERENCES

[1] Apache Giraph. `http://giraph.apache.org`.

[2] DAS-5: Distributed ASCI Supercomputer 5. `http://cs.vu.nl/das5`.

[3] Oracle Big Data Spatial and Graph. `http://oracle.com/database/big-data-spatial-and-graph`.

[4] Oracle Labs PGX: Parallel Graph Analytics Overview. `http://oracle.com/technetwork/oracle-labs/parallel-graph-analytics`.

[5] SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`.

[6] G. Aluç et al. Diversified stress testing of RDF data management systems. In *ISWC*, pages 197–212, 2014.

[7] K. Ammar and M. T. Özsu. WGB: towards a universal graph benchmark. In *WBDB*, pages 58–72, 2013.

[8] M. Anderson et al. GraphPad: optimized graph primitives for parallel and distr. platforms. In *IPDPS*, 2016.

[9] T. Armstrong et al. LinkBench: a database benchmark based on the Facebook social graph. In *SIGMOD*, pages 1185–1196, 2013.

[10] D. Bader and K. Madduri. Design and implementation of the HPCS graph analysis benchmark on symmetric multiprocessors. In *HiPC*, pages 465–476, 2005.

[11] C. Bizer and A. Schultz. The Berlin SPARQL benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.

[12] M. Capota et al. Graphalytics: A big data benchmark for graph-processing platforms. In *GRADES*, pages 7:1–7:6, 2015.

[13] M. Cha et al. Measuring User Influence in Twitter: The Million Follower Fallacy. In *ICWSM*, page 30, 2010.

[14] M. Dayarathna and T. Suzumura. Graph database benchmarking on cloud environments with XGDBench. *Autom. Softw. Eng.*, 21(4):509–533, 2014.

[15] A. Eisenman et al. Parallel graph processing: Prejudice and state of the art. In *ICPE*, 2016.

[16] B. Elser and A. Montresor. An evaluation study of bigdata frameworks for graph processing. In *Big Data*, pages 60–67, 2013.

[17] O. Erling et al. The LDBC social network benchmark: Interactive workload. In *SIGMOD*, pages 619–630, 2015.

[18] J. Fan et al. The case against specialized graph analytics engines. In *CIDR*, 2015.

[19] M. Ferdman et al. Clearing the clouds: a study of emerging scaleout workloads on modern hardware. In *ASPLOS*, pages 37–48, 2012.

[20] A. Ghazal et al. BigBench: towards an industry standard benchmark for big data analytics. In *SIGMOD*, pages 1197–1208, 2013.

[21] J. E. Gonzalez et al. PowerGraph: Distributed graph parallel computation on natural graphs. In *OSDI*, pages 17–30, 2012.

[22] Y. Guo and A. Iosup. The game trace archive. In *NETGAMES*, page 4. IEEE Press, 2012.

[23] Y. Guo et al. LUBM: A benchmark for OWL knowledge base systems. *J. Web Sem.*, 3(2-3):158–182, 2005.

[24] Y. Guo et al. How well do graph-processing platforms perform? In *IPDPS*, pages 395–404, 2014.

[25] Y. Guo et al. An empirical performance evaluation of gpu-enabled graph-processing systems. In *CCGrid*, pages 423–432, 2015.

[26] M. Han et al. An experimental comparison of pregel-like graph processing systems. *PVLDB*, 7(12):1047–1058, 2014.

[27] S. Hong et al. PGX.D: a fast distributed graph processing engine. In *SC*, pages 58:1–58:12, 2015.

[28] A. Iosup, et al. LDBC Graphalytics: A Benchmark for Large-Scale Graph Analysis on Parallel and Distributed Platforms. Technical Report DS-2016-001, Delft University of Technology, 2016. `http://ds.ewi.tudelft.nl/research-publications/technical-reports/2016/`.

[29] A. Jindal et al. Vertexica: your relational friend for graph analytics! *PVLDB*, 7(13):1669–1672, 2014.

[30] Y. Lu et al. Large-scale distributed graph computing systems: An experimental evaluation. *PVLDB*, 8(3):281–292, 2014.

[31] Z. Ming et al. BDGS: A scalable big data generator suite in big data benchmarking. In *WBDB*, pages 138–154, 2013.

[32] L. Nai et al. GraphBIG: understanding graph computing in the context of industrial solutions. In *SC*, pages 69:1–69:12, 2015.

[33] W. L. Ngai. Fine-grained Performance Evaluation of Large-scale Graph Processing Systems. Master's thesis, Delft University of Technology, the Netherlands, 2015.

[34] L. Page et al. The pagerank citation ranking: bringing order to the web. 1999.

[35] T. Rabl et al. The vision of BigBench 2.0. In *DanaC*, pages 3:1–3:4, 2015.

[36] U. Raghavan et al. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3):036106, 2007.

[37] N. Satish et al. Navigating the maze of graph analytics frameworks using massive datasets. In *SIGMOD*, pages 979–990, 2014.

[38] M. Schmidt et al. Spˆ2 bench: a SPARQL performance benchmark. In *ICDE*, pages 222–233, 2009.

[39] N. Sundaram et al. Graphmat: High performance graph analytics made productive. *PVLDB*, 8(11):1214–1225, 2015.

[40] L. Wang et al. BigDataBench: a big data benchmark suite from internet services. In *HPCA*, pages 488–499, 2014.

[41] R. Xin et al. GraphX: A resilient distr. graph system on Spark. In *GRADES*, page 2, 2013.

[4] Available at `github.com/tudelft-atlarge/graphalytics`