Big Graph Analytics Platforms

Da Yan
The University of Alabama at Birmingham
yanda@uab.edu

Yingyi Bu
Couchbase, Inc.
yingyi@couchbase.com

Yuanyuan Tian
IBM Almaden Research Center, USA
ytian@us.ibm.com

Amol Deshpande
University of Maryland
amol@cs.umd.edu

the essence of knowledge
Boston — Delft

Foundations and Trends® in Databases

Published, sold and distributed by:
now Publishers Inc.
PO Box 1024
Hanover, MA 02339
United States
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:
now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is

D. Yan, Y. Bu, Y. Tian, and A. Deshpande. *Big Graph Analytics Platforms.* Foundations and Trends® in Databases, vol. 7, no. 1-2, pp. 1–195, 2015.

This Foundations and Trends® issue was typeset in LaTeX using a class file designed by Neal Parikh. Printed on acid-free paper.

ISBN: 978-1-68083-242-6

Foundations and Trends® in Databases
Volume 7, Issue 1-2, 2015

Editorial Scope

Topics

Foundations and Trends® in Databases covers a breadth of topics relating to the management of large volumes of data. The journal targets the full scope of issues in data management, from theoretical foundations, to languages and modeling, to algorithms, system architecture, and applications. The list of topics below illustrates some of the intended coverage, though it is by no means exhaustive:

- Data models and query languages
- Query processing and optimization
- Storage, access methods, and indexing
- Transaction management, concurrency control, and recovery
- Deductive databases
- Parallel and distributed database systems
- Database design and tuning
- Metadata management
- Object management
- Trigger processing and active databases
- Data mining and OLAP
- Approximate and interactive query processing
- Data warehousing
- Adaptive query processing
- Data stream management
- Search and query integration
- XML and semi-structured data
- Web services and middleware
- Data integration and exchange
- Private and secure data management
- Peer-to-peer, sensornet, and mobile data management
- Scientific and spatial data management
- Data brokering and publish/subscribe
- Data cleaning and information extraction
- Probabilistic data management

Information for Librarians

Foundations and Trends® in Databases, 2015, Volume 7, 4 issues. ISSN paper version 1931-7883. ISSN online version 1931-7891. Also available as a combined paper and online subscription.

Foundations and Trends® in Databases
Vol. 7, No. 1-2 (2015) 1–195
© 2017 D. Yan, Y. Bu, Y. Tian, and A. Deshpande
DOI: 10.1561/1900000056

Big Graph Analytics Platforms

Da Yan
The University of Alabama at Birmingham
yanda@uab.edu

Yingyi Bu
Couchbase, Inc.
yingyi@couchbase.com

Yuanyuan Tian
IBM Almaden Research Center, USA
ytian@us.ibm.com

Amol Deshpande
University of Maryland
amol@cs.umd.edu

Contents

Abstract

Due to the growing need to process large graph and network datasets created by modern applications, recent years have witnessed a surging interest in developing big graph platforms. Tens of such big graph systems have already been developed, but there lacks a systematic categorization and comparison of these systems. This article provides a timely and comprehensive survey of existing big graph systems, and summarizes their key ideas and technical contributions from various aspects. In addition to the popular vertex-centric systems which espouse a think-like-a-vertex paradigm for developing parallel graph applications, this survey also covers other programming and computation models, contrasts those against each other, and provides a vision for the future research on big graph analytics platforms. This survey aims to help readers get a systematic picture of the landscape of recent big graph systems, focusing not just on the systems themselves, but also on the key innovations and design philosophies underlying them.

D. Yan, Y. Bu, Y. Tian, and A. Deshpande. *Big Graph Analytics Platforms*. Foundations and Trends® in Databases, vol. 7, no. 1-2, pp. 1–195, 2015.

DOI: 10.1561/1900000056.

1

Introduction

The growing need to deal with massive graphs in real-life applications has led to a surge in the development of big graph analytics platforms. Tens of big graph systems have already been developed, and more are expected to emerge in the near future. Researchers new to this young field can easily get overwhelmed and lost by the large amount of literature. Although several experimental studies have been conducted in recent years that compare the performance of several big graph systems [Lu et al., 2014, Han et al., 2014a, Satish et al., 2014, Guo et al., 2014], there lacks a comprehensive survey that clearly summarizes the key features and techniques developed in existing big graph systems. A recent survey [McCune et al., 2015] attempts to cover the landscape as well, but primarily focuses on vertex-centric systems; it omits most of the work on other programming models and also several crucial optimization and programmability issues with vertex-centric systems. In addition to describing the various systems, this survey puts more emphasis on the innovations and technical contributions of existing systems, in order to help readers quickly obtain a systemic view of the key ideas and concepts. We hope this will help big graph system researchers avoid reinventing the wheel, apply useful existing techniques to their own systems, and come up with new innovations.

In the rest of this chapter, we first review the history of research on Big Graph systems, and then overview some important features of existing Big Graph systems. Finally, we present the organization of this survey. Many contents of this survey are covered by our tutorial in SIGMOD 2016 [Yan et al., 2016a], the slides of which are available online[1] and contain animations to illustrate the various techniques used by existing systems.

1.1 History of Big Graph Systems Research

Although graph analytics has always been an important research topic throughout the history of computation, the research on *big* graph processing only flourished in recent years as part of the big data movement, which has seen increased use of advanced analytics on large volumes of unstructured or semi-structured data. A hallmark of this movement has been the MapReduce distributed data processing framework, introduced by Google [Dean and Ghemawat, 2004], and the companion Google File System (GFS) [Ghemawat et al., 2003]. Subsequently, the Apache Hadoop project[2] implemented the open-source counterparts, the Hadoop Distributed File System (HDFS) and the Hadoop MapReduce framework in 2006. Since then, a huge body of research has focused on designing novel MapReduce algorithms as well as on improving the framework for particular workloads. A large body of work in that space focused on big graph analytics, and many tailor-made MapReduce algorithms were proposed for solving specific graph problems [Lin and Schatz, 2010]. As an early MapReduce-based framework designed for general-purpose graph processing, PEGASUS [Kang et al., 2009] models graph computation by a generalization of matrix-vector multiplication. However, the reliance on the disk-based Hadoop MapReduce runtime, which requires repeated reads and writes of large files from HDFS, fundamentally limits its performance.

[1]`http://www.cse.cuhk.edu.hk/systems/gsys_tutorial/`
[2]`https://hadoop.apache.org/`

Later, Malewicz et al. [2010] proposed the Pregel framework specially designed for large-scale big graph processing. Since many graph algorithms are iterative, Pregel keeps the graph data in the main memory and adopts an iterative, message-passing computation model (inspired by the well-known Bulk Synchronous Parallel model for parallel computation), and is thus much more efficient than MapReduce. Pregel also adopts a "think-like-a-vertex" programming model which is more intuitive and user-friendly for average programmers and a natural fit for a range of graph analysis tasks. The vertex-centric programming model of Pregel is also very expressive since a vertex can communicate with any other vertex by passing messages. Since the introduction of Pregel, it has sparked a large number of research works on extending the basic Pregel framework in different aspects to improve the graph processing performance [Tian et al., 2013, Yan et al., 2014a, Zhang et al., 2014, Han and Daudjee, 2015, Yan et al., 2016b].

Independent of Pregel, Low et al. [2010] developed a multi-core, shared-memory graph-based computation model, called GraphLab. Then, Low et al. [2012] extended it to work in a distributed environment, while keeping the shared memory programming abstraction, in which a vertex can directly access the states of its adjacent vertices and edges. Later, GraphLab switched to the GAS (Gatter-Apply-Scatter) computation model to further improve the system performance [Gonzalez et al., 2012]. Although the GAS model covers a large number of graph algorithms, it is less expressive than the Pregel model, since a vertex can only access the data of its adjacent vertices and edges; we call this a *neighborhood-based shared memory* abstraction. This programming abstraction is especially popular among recent big graph systems designed to run on a single machine, such as GraphChi [Kyrola et al., 2012].

While Pregel and GraphLab are designed specially for graph processing, a number of systems, such as GraphX [Gonzalez et al., 2014] and Pregelix [Bu et al., 2014], rely on a general-purpose data processing engine for execution, at the same time providing graph-specific programming interfaces similar to those in Pregel and GraphLab.

Vertex-centric systems are ideally suited for graph analysis tasks like PageRank computation where the overall computation can be broken down into individual tasks, each involving a specific vertex (i.e., its local state, and the states of its adjacent edges). Many machine learning tasks (e.g., belief propagation, matrix factorization, stochastic gradient descent) are also a natural fit for those systems. However, many complex graph analysis tasks cannot be easily decomposed in such fashion. For example, a class of graph problems termed "ego-centric analysis" [Quamar et al., 2016] require analyzing the neighborhoods of the vertices in their entirety. Also, graph problems such as graph matching or graph mining may have intermediate or output results with size superlinear or even exponential in the input graph size. Complex graph algorithms, e.g., the Hungarian algorithm for maximum bipartite matching, even require random access to the entire graph. Solving these problems using vertex-centric processing leads to substantial communication and memory overheads, since each vertex needs to collect the relevant neighborhood subgraph (if not the entire graph) to its local state before processing the subgraph.

This has led to the development of many alternative programming frameworks, examples of which include Socialite [Seo et al., 2013b], Arabesque [Teixeira et al., 2015], NScale [Quamar et al., 2016], among others. In addition, several systems including Ligra [Shun and Blelloch, 2013], Galois [Nguyen et al., 2013], Green-Marl DSL [Hong et al., 2012], etc., provide low-level graph programming frameworks that can handle nearly arbitrary graph computations. These frameworks often focus on specific classes of graph problems, and make a range of different assumptions about the computing environment, making them incomparable in many cases. Arabesque tackles problems like graph matching and graph mining where the intermediate result can be very large, while assuming that the entire graph can be held in a single machine memory. NScale is a strict generalization of the vertex-centric framework, and can handle tasks that require access to multi-hop neighborhoods of vertices; but it does not support the other classes of problems discussed above. Socialite uses a Datalog-inspired programming model which is most suitable for graph problems that can be expressed as recursive

Datalog queries. Ligra, Galois, and other similar systems require random access to the graph and focus on large-memory multi-core environments. Thus, developing a sufficiently expressive, yet easy-to-use and easy-to-parallelize graph programming model, remains a critical and open challenge in this field.

The majority of existing big graph systems are designed for processing static graphs (or with small topology mutations). However, real-world graphs often evolve over time, with vertices and edges continually being added or deleted, and their attributes being frequently updated. A new class of big graph systems, such as KineoGraph [Cheng et al., 2012], TIDE [Xie et al., 2015b], DeltaGraph [Khurana and Deshpande, 2013], and Chronos [Han et al., 2014b], have emerged to process and analyze temporal and streaming graph data. This area is however still in its infancy and there are many open problems that need to be addressed to effectively handle continuous and/or temporal analytics on big graphs.

There is also a large body of work on executing queries related to a specific vertex (or a small subset of vertices) against large volumes of graph data, which has developed a range of specialized indexes and search algorithms. This survey does not cover that body of work.

1.2 Features of Big Graph Systems

We can categorize the big graph platforms along various dimensions. Since an important feature of the modern big graph systems is user-friendliness in programming parallel graph algorithms, we first summarize the programming abstractions (languages and models) of existing systems. While most systems adopt existing programming languages that are familiar to users (e.g., C/C++ and Java), some systems require users to learn a new domain-specific language dedicated to programming parallel graph algorithms (e.g., Green-Marl [Hong et al., 2012] and Trinity Specification Language [Shao et al., 2013]).

1.2.1 Programming Model

Most big graph systems adopt the vertex-centric model where a programmer only needs to specify the behavior of one vertex. The vertex-centric model can be further divided into two types: (1) message passing (e.g., in Pregel), where vertices communicate with each other by sending messages; and (2) shared-memory abstraction (e.g., in GraphLab), where vertices directly access the states of other vertices and edges.

Message passing is a natural model in a distributed environment, since users can explicitly dictate message passing behavior in their programs. In contrast, the shared-memory abstraction allows programmers to directly access data as if operating on a single machine, and most single-machine vertex-centric systems adopt this model. However, distributed GraphLab adopts the shared-memory abstraction and there are also single-machine systems that adopt message passing (e.g., Flash-Graph [Zheng et al., 2015]).

The vertex-centric framework can be further extended with a block-centric model (e.g., Giraph++ [Tian et al., 2013] and Blogel [Yan et al., 2014a]), which partitions the vertices into multiple disjoint subgraphs, so that value propagation within each subgraph could bypass network communication. The block-centric model often improves the performance of graph computation by orders of magnitude.

Besides the vertex-centric systems, some big graph systems adopt a matrix-based programming model; these include PEGASUS [Kang et al., 2009], GBASE [Kang et al., 2011], and SystemML [Ghoting et al., 2011]. These systems represent a graph algorithm by a sequence of generalized matrix-vector multiplications, which can be efficiently processed since sparse matrix algebra has been studied for decades in the High Performance Computing (HPC) field. However, users who are not familiar with matrix algebra might prefer vertex-centric programming to matrix-based programming. Recently, Sundaram et al. [2015] helped bridge the gap for these users: their GraphMat system translates a vertex-centric program to high performance sparse matrix operations to be run on the backend.

Another important class of programming models is subgraph-centric models, where users write programs to process a subgraph instead of a single vertex. These models target graph problems whose output size can be exponential to the graph size (e.g., graph matching and finding motifs), or problems that require analyzing entire neighborhoods in a holistic manner. Since vertices in a subgraph can be randomly accessed by a user program, a critical issue for a subgraph-centric model is how to efficiently construct the relevant subgraphs. Arabesque [Teixeira et al., 2015] and NScale [Quamar et al., 2016] are two systems that use a subgraph-centric model, although there are significant differences in the models they adopt.

There are also systems that require users to write graph algorithms using a domain specific language (DSL), e.g., Green-Marl [Hong et al., 2012, 2014], Galois [Nguyen et al., 2013], and Ligra [Shun and Blelloch, 2013]. The language constructs of those DSLs expose opportunities for parallelism, which can be utilized by the system for efficient parallel execution. Of course, users have to learn a new language or programming paradigm in order to use such a system.

Finally, several recent systems have been built to bring in declarative query languages for big graph analytics. First, since many graph algorithms can be expressed as recursive Datalog [Bancilhon and Ramakrishnan, 1986] queries, a number of research projects are inventing new-generation Datalog systems for scalable big graph analytics. Second, often times, a graph analytics job is only one part of a gigantic, end-to-end SQL[3]-dominated data analysis pipeline which includes constructing graphs dynamically from tabular data sources and converting graph computation results back into tabular reports; therefore, several systems have integrated vertex-centric programming models into declarative query languages to make those end-to-end data analysis tasks easier [Simmen et al., 2014, Gonzalez et al., 2014].

1.2.2 Expressiveness

Most big graph systems aim at solving a broad class of graph problems using a unified programming framework. Therefore, it is meaningless to study big graph systems without studying the algorithms and applications that can be implemented in these systems. However, many

[3]SQL. https://en.wikipedia.org/wiki/SQL

papers just introduce API simplicity and performance advantages of their systems in order to promote their work, but these benefits may come at a cost of additional assumptions and narrower expressiveness that were understated, which should be made clear to avoid blind or even wrong system choice. We now discuss the expressiveness of the various programming models described before, and provide some advice on how to choose an appropriate framework for an application at hand.

Many graph algorithms only require each vertex to communicate with its neighbors, such as PageRank and other more complicated random walk algorithms (e.g., [Zhang et al., 2016]). In these algorithms, intermediate data are only exchanged along edges, and so the volume of intermediate data is comparable to the data size. We say that these algorithms require *edge-based communication*. In some of these algorithms, a vertex only needs the aggregated value of the received values, which provides opportunities for further optimization. For example, MOCgraph [Zhou et al., 2014], GraphD [Yan et al., 2016d], and the superstep-splitting technique of Giraph [Ching et al., 2015] all propose aggregating messages earlier instead of buffering them for later processing, in order to save memory space; while PowerGraph [Gonzalez et al., 2012], GraphChi [Kyrola et al., 2012] and X-Stream [Roy et al., 2013] assume that data values are aggregated at each vertex from its incoming edges, in their model design. We, however, would like to indicate that not all algorithms with edge-based communication allow its vertices to aggregate received values, such as the attribute broadcast algorithm of Yan et al. [2015].

Edge-based communication implies that any information can be propagated for just one hop at a time, which leads to poor performance if a vertex u needs to transmit a value to another vertex v far away from u in a large-diameter graph. Pointer jumping (aka path doubling), a technique from PRAM algorithm design, solves this problem by doubling the propagation length from u to v, until v is reached. This requires a vertex to be able to send data to any other vertex, not just its neighbors. We say that these algorithms require *ID-based communication*, where a vertex u can send messages to another vertex w as long as w's ID is known. Pregel [Malewicz et al., 2010] adopts ID-based communication and thus can implement pointer-jumping algorithms such as those to be described in Chapter 3.2, while GraphLab [Gonzalez et al., 2012] only allows each vertex to access its neighbors' data, and thus cannot support these algorithms. In fact, Pregel has probably the most expressive programming model in theory, and it is known how to write a large number of graph algorithms efficiently in that model. The Bulk Synchronous Parallel (BSP) model, on which Pregel is based, has been very well-studied, but as a synchronous model, the number of iterations must be kept low in a distributed setting, which can be achieved with the help of pointer jumping.

Another solution to avoid slow value propagation is to use a block-centric model, where nearby vertices are grouped into a block for processing together each time. In a distributed environment, since a block is assigned to a unique machine, only blocks need to communicate with each other, and computation over vertices inside a block does not generate communication. In a single-machine environment, each block usually fits in a CPU cache, and thus block-based processing improves cache locality in its execution. In addition to faster value propagation (i.e., block-wise), the block-centric model also significantly reduces the communication workload. Representative block-centric system include Giraph++ [Tian et al., 2013] and Blogel [Yan et al., 2014a].

Some graph algorithms (e.g., k-core finding [Salihoglu and Widom, 2014]) need to mutate the graph topology during computation, and thus, support for deletion and addition of edges and vertices is also an important aspect of system expressiveness. For example, VENUS [Cheng et al., 2015] streams immutable graph structure and thus does not support algorithms that require graph mutations.

The models discussed so far are mostly vertex-centric. However, many graph mining problems define constraints on subgraphs, e.g., graph matching and motif mining. Subgraph-based models are proposed to solve these problems by writing user-friendly programs, where computation is directly performed on subgraphs. Such systems include NScale [Quamar et al., 2016] and Arabesque [Teixeira et al., 2015], which we discuss in more detail in Chapter 7.

We remark that there are other models that could be more appropriate for a specific application at hand. For example, if one is viewing a graph as a matrix, and solving a machine learning problem that uses matrix operations, then matrix-based systems like SystemML [Ghoting et al., 2011] could be a better choice. Also, if graph processing is just part of a dataflow program, then dataflow-based systems could provide more flexibility, e.g., GraphX [Gonzalez et al., 2014] can interoperate with other dataflow operators in Spark [Zaharia et al., 2012] to avoid data import/export.

1.2.3 Execution Mode

Most big graph systems target iterative graph computation, where vertex values are repeatedly updated until the computation converges. There are two typical execution modes: synchronous and asynchronous. The synchronous mode is also called bulk synchronous parallel (BSP), exemplified by Pregel, while the asynchronous mode is adopted by GraphLab and several other systems (especially those targeting machine learning workloads). The difference between these two modes is that, in the synchronous mode, there is a global barrier from one iteration to another, and out-going messages or updates of one iteration are only accessible in the next iteration; in the asynchronous mode, a vertex has immediate access to its in-bound messages or updates.

Asynchronous parallel computation incurs race conditions and thus requires additional effort to enforce data consistency (e.g., by using locks). Moreover, in a distributed environment, asynchronous execution tends to transmit a lot of small messages, since the update to a vertex value should be reflected in time. In contrast, BSP only requires updates to be synchronized at the end of each iteration, and messages can be sent in large batches. In fact, GraphLab has a synchronous mode that simulates the BSP mode of Pregel, and both Lu et al. [2014] and Han et al. [2014a] found that the synchronous mode is generally faster than the asynchronous mode. Further, for many algorithms, asynchronous execution is not an option because the indeterministic execution may lead to incorrect answers.

However, for some problems like PageRank computation, vertex values converge asymmetrically: most vertices converge quickly after a few iterations, but some vertices take a large number of iterations to converge. In that case, asynchronous execution can schedule those vertices that converge more slowly to compute for more iterations, while synchronous execution processes every vertex once in each iteration even if most vertices are converged. Therefore, asynchronous mode is much faster for such algorithms and is thus preferred. Moreover, asynchronous computation is always preferred in a single-machine system since data access no longer incurs network communication, and accessing the latest vertex value leads to faster convergence.

It is, however, worth noting that some asynchronous frameworks may not converge to the exact results (e.g., PageRank values), but the approximate results are often good enough while the significant improvement in performance (compared with synchronous execution) is highly attractive. More discussion can be found in Section 4.2.

Recently, PowerSwitch [Xie et al., 2015a] showed how to support mode switch between asynchronous execution and synchronous execution in GraphLab. They found that when the workload is low, asynchronous execution is faster due to the faster convergence rate provided by accessing the latest values. Race conditions (e.g., updates to the same vertex) are unlikely to occur since only a small portion of vertices participate in computation, and the number of messages is too small to benefit from sending in large batches. In contrast, when the workload is high, synchronous execution is faster since there is no need to handle race conditions (i.e., it avoids the expensive locking/unlocking cost required by asynchronous execution), and messages are sent in large batches. Thus, PowerSwitch constantly collects execution statistics on-the-fly, which are used to predict future performance and determine the timing of a profitable mode switch.

1.2.4 Other Features

There are also many other dimensions to categorize big graph systems. As for the **execution environment**, there are systems developed to process graphs in a single machine, or using a cluster of machines.

The single-machine environment can be further divided into two types, commodity PCs and high-end servers. The former targets processing big graphs efficiently using readily available resources; since the available memory on a commodity PC is limited, the graph is usually disk-resident, and loaded into memory for processing part-by-part or in a streaming fashion. The latter aims at beating distributed systems by eliminating the cost of network communication, and the graph is usually memory-resident. As for the **graph placement**, distributed systems usually keep the graph in main memory, since there are many machines and the total RAM size is sufficient, while single-PC systems tend to process disk-resident or SSD-resident graphs. There are also distributed systems that process disk-resident graphs in order to scale to giant graphs whose size is much larger than the total RAM size in a cluster, such as Pregelix [Bu et al., 2014], GraphD [Yan et al., 2016d] and Chaos [Roy et al., 2015].

There are also many design techniques that may significantly influence the system performance for specific algorithms. For example, disk-based single-machine systems like GraphChi [Kyrola et al., 2012] are designed for iterative batch processing, while TurboGraph [Han et al., 2013] maintains an in-memory page ID table for directly locating the disk page of any vertex. Given these differences in system design, a reader will not be surprised to see a claim like TurboGraph "significantly outperforms GraphChi by up to four orders of magnitude", for a query that is to find the neighbors of a particular vertex.

1.3 Organization of the Survey

The diverse features supported by different big graph systems, and the cross-cutting nature of many of the key designs, make it challenging to organize such a survey. As an example, the popular vertex-centric programming model is easy to support on top of a wide range of different underlying implementations, including distributed frameworks like Hadoop MapReduce, matrix-based systems, and relational databases. However, each of those implementations raises unique and different challenges despite their use of the vertex-centric model on top.

In this survey, we focus on presenting the key designs and features of the various graph processing systems, while endeavoring to place related systems together to summarize the common ideas underlying their designs. For quick reference, Table 1.1 presents a list of all the systems that we discuss in each chapter.

We broadly divide the survey into three parts. In Part I, we discuss the big graph systems that primarily use the vertex-centric programming model, which has been widely studied recently due to its simplicity in programming parallel graph algorithms. Specifically, Chapter 3 reviews the framework of Pregel, and introduces how to develop algorithms with performance guarantees in Pregel; it then discusses existing open-source Pregel-like systems with improvements in communication mechanism, load balancing, out-of-core support and fault tolerance. Chapter 4 walks through the various extensions to the basic framework of Pregel that could significantly improve the performance of graph computations. Chapter 5 covers a few important big graph systems that adopt shared memory abstraction, including the pioneering GraphLab system.

In Part II, we review other systems that attempt to provide support for more general graph programming models; most of these are motivated by the observation that complex graph algorithms or analysis tasks are often difficult to program using the simple vertex-centric programming framework. Chapter 6 describes several matrix-based big graph systems, including the pioneering MapReduce-based systems PEGASUS and GBASE, and the more powerful SystemML system. Chapter 7 explains why the vertex-centric and matrix-based frameworks are not sufficient for graph problems like graph matching and motif mining, and introduces two subgraph-centric systems, NScale and Arabesque, to process such graph problems efficiently. Chapter 8 reviews several systems that either offer database-style declarative query languages or leverage database-style query processing techniques.

Finally, in Part III, we discuss some miscelleneous issues. While some vertex-centric single-machine big graph systems are also introduced in Chapter 5, Chapter 9 surveys more single-machine systems that adopt a computation model beyond a pure vertex-centric one.

Table 1.1

Section	System
3.1	Pregel
3.3	Giraph, Pregel+, GPS, MOCgraph
3.4	WindCatch, PAGE
3.5	GraphD
4.1	Giraph++, Blogel
4.2	Maiter, GiraphUC
4.3	Quegel
5.1	GraphLab/PowerGraph
5.2	GraphChi, X-Stream, Chaos, VENUS, GridGraph
6.1	PEGASUS
6.2	GBASE
6.2	SystemML
7.1.1	Trinity
7.2	NScale
7.3	Arabesque
8.1	SociaLite, DeALS, Myria, Yedalog
8.2	GraphX, Pregelix, Vertexica
8.3	REX, Maiter
8.4	Aster Data
9.1	GraphMat, GraphTwist
9.2	Green-Marl, Ligra, GRACE, Galois
10.1	TurboGraph, FlashGraph
10.2	Medusa, MapGraph, CuSha
11.2	Chronos, DeltaGraph, LLAMA
11.3	Kineograph, TIDE

Chapter 10 discusses a few systems that utilize new hardware technologies to significantly boost the performance of big graph analytics. Then, in Chapter 11, we discuss the issues of managing time-evolving graphs and supporting real-time analytics over streaming graph data, and discuss several recent systems that focus on providing those capabilities. Finally, we conclude the survey in Chapter 12 and provide a discussion on future research in big graph analytics platforms.

References

Charu C. Aggarwal, Yao Li, Philip S. Yu, and Ruoming Jin. On dense pattern mining in graph streams. *VLDB*, 2010.

Jae-wook Ahn, Catherine Plaisant, and Ben Shneiderman. A task taxonomy for network evolution analysis. *IEEE Transactions on Visualization and Computer Graphics*, 2014.

Sattam Alsubaiee, Yasser Altowim, Hotham Altwaijry, Alexander Behm, Vinayak R. Borkar, Yingyi Bu, Michael J. Carey, Inci Cetindil, Madhusudan Cheelangi, Khurram Faraaz, Eugenia Gabrielova, Raman Grover, Zachary Heilbron, Young-Seok Kim, Chen Li, Guangqiang Li, Ji Mahn Ok, Nicola Onose, Pouria Pirzadeh, Vassilis J. Tsotras, Rares Vernica, Jian Wen, and Till Westmann. Asterixdb: A scalable, open source BDMS. *PVLDB*, 7(14):1905–1916, 2014.

Darko Anicic, Paul Fodor, Sebastian Rudolph, and Nenad Stojanovic. EP-SPARQL: a unified language for event processing and stream reasoning. In *WWW*, 2011.

Lars Backstrom and Jure Leskovec. Supervised random walks: predicting and recommending links in social networks. In *WSDM*, pages 635–644, 2011.

Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized pagerank. *VLDB*, 2010.

Isaac Balbin and Kotagiri Ramamohanarao. A generalization of the differential approach to recursive query evaluation. *J. Log. Program.*, 4(3):259–262, 1987.

François Bancilhon and Raghu Ramakrishnan. An amateur's introduction to recursive query processing strategies. In *SIGMOD*, pages 16–52, 1986.

François Bancilhon, David Maier, Yehoshua Sagiv, and Jeffrey D. Ullman. Magic sets and other strange ways to implement logic programs. In *VLDB*, pages 1–15, 1986.

Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, and Michael Grossniklaus. An execution environment for C-SPARQL queries. In *EDBT*, 2010.

Alain Barrat, Marc Barthelemy, and Alessandro Vespignani. *Dynamical processes on complex networks*. Cambridge University Press Cambridge, 2008.

Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *KDD*, pages 16–24, 2008.

Catriel Beeri, Shamim A. Naqvi, Raghu Ramakrishnan, Oded Shmueli, and Shalom Tsur. Sets and negation in a logic database language (LDL1). In *PODS*, pages 21–37, 1987.

Tanya Y Berger-Wolf and Jared Saia. A framework for analysis of dynamic social networks. In *SIGKDD*, 2006.

Matthias Boehm, Shirish Tatikonda, Berthold Reinwald, Prithviraj Sen, Yuanyuan Tian, Douglas R. Burdick, and Shivakumar Vaithyanathan. Hybrid parallelization strategies for large-scale machine learning in systemml. *PVLDB*, 7(7):553–564, 2014.

Matthias Boehm, Michael W. Dusenberry, Deron Eriksson, Alexandre V. Evfimievski, Faraz Makari Manshadi, Niketan Pansare, Berthold Reinwald, Frederick R. Reiss, Prithviraj Sen, Arvind C. Surve, and Shirish Tatikonda. Systemml: Declarative machine learning on spark. *PVLDB*, 9(13):1425 – 1436, 2016.

Vinayak R. Borkar, Michael J. Carey, Raman Grover, Nicola Onose, and Rares Vernica. Hyracks: A flexible and extensible foundation for data-intensive computing. In *ICDE*, 2011.

Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh International World-Wide Web Conference (WWW)*, pages 107–117, 1998.

Yingyi Bu. *On Software Infrastructure for Scalable Graph Analytics*. PhD thesis, Computer Science Department, University of California, Irvine, August 2015.

Yingyi Bu, Vinayak R. Borkar, Jianfeng Jia, Michael J. Carey, and Tyson Condie. Pregelix: Big(ger) graph analytics on a dataflow engine. *PVLDB*, 8(2):161–172, 2014.

Craig Chambers, Ashish Raniwala, Frances Perry, Stephen Adams, Robert R. Henry, Robert Bradshaw, and Nathan Weizenbaum. FlumeJava: easy, efficient data-parallel pipelines. In *PLDI*, pages 363–375, 2010.

K. Mani Chandy and Leslie Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, 1985.

Surajit Chaudhuri. An overview of query optimization in relational systems. In *PODS*, pages 34–43, 1998.

Jiefeng Cheng, Qin Liu, Zhenguo Li, Wei Fan, John C. S. Lui, and Cheng He. VENUS: vertex-centric streamlined graph computation on a single PC. In *ICDE*, pages 1131–1142, 2015.

Raymond Cheng, Ji Hong, Aapo Kyrola, Youshan Miao, Xuetian Weng, Ming Wu, Fan Yang, Lidong Zhou, Feng Zhao, and Enhong Chen. Kineograph: taking the pulse of a fast-changing and connected world. In *EuroSys*, pages 85–98, 2012.

Brian Chin, Daniel von Dincklage, Vuk Ercegovac, Peter Hawkins, Mark S. Miller, Franz Josef Och, Christopher Olston, and Fernando Pereira. Yedalog: Exploring knowledge at scale. In *SNAPL*, pages 63–78, 2015.

Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. One trillion edges: Graph processing at facebook-scale. *PVLDB*, 8(12):1804–1815, 2015.

Sutanay Choudhury, Lawrence Holder, George Chin, Khushbu Agarwal, and John Feo. A selectivity based approach to continuous pattern detection in streaming graphs. *EDBT*, 2015.

Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. Estimating pagerank on graph streams. In *PODS*, 2008.

Ankur Dave, Alekh Jindal, Li Erran Li, Reynold Xin, Joseph Gonzalez, and Matei Zaharia. GraphFrames: An integrated api for mixing graph and relational queries. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems*, GRADES '16, pages 2:1–2:8, 2016.

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.

Camil Demetrescu, Irene Finocchi, and Andrea Ribichini. Trading off space for passes in graph streaming problems. *ACM Trans. Algorithms*, 2009.

David J. DeWitt and Jim Gray. Parallel database systems: The future of high performance database systems. *Commun. ACM*, 35(6):85–98, 1992.

Jason Eisner and Nathaniel Wesley Filardo. Dyna: Extending datalog for modern AI. In *Datalog*, pages 181–220, 2010.

Ahmed Elgohary, Matthias Boehm, Peter J. Haas, Frederick R. Reiss, and Berthold Reinwald. Compressed linear algebra for large-scale machine learning. *PVLDB*, 9(12):960–971, 2016.

E. N. (Mootaz) Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, September 2002.

David Eppstein, Zvi Galil, and Giuseppe F. Italiano. *Dynamic Graph Algorithms*. CRC Press, 1999.

Shimon Even. *Graph Algorithms*. Cambridge University Press, New York, NY, USA, 2nd edition, 2011.

Stephan Ewen, Kostas Tzoumas, Moritz Kaufmann, and Volker Markl. Spinning fast iterative data flows. *PVLDB*, 5(11):1268–1279, 2012.

Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. In *ICALP*, 2004.

Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the streaming model: the value of space. In *SODA*, 2005.

Zhisong Fu, Bryan B. Thompson, and Michael Personick. Mapgraph: A high level API for fast development of high performance graph analytics on gpus. In *GRADES*, pages 2:1–2:6, 2014.

Jun Gao, Chang Zhou, Jiashuai Zhou, and Jeffrey Xu Yu. Continuous pattern detection over billion-edge graph using distributed framework. In *ICDE*, pages 556–567, 2014.

B. Gedik and R. Bordawekar. Disk-based management of interaction graphs. *TKDE*, 2014.

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *SOSP*, pages 29–43, 2003.

Amol Ghoting, Rajasekar Krishnamurthy, Edwin Pednault, Berthold Reinwald, Vikas Sindhwani, Shirish Tatikonda, Yuanyuan Tian, and Shivakumar Vaithyanathan. Systemml: Declarative machine learning on mapreduce. In *ICDE*, pages 231–242, 2011.

A. Ghrab, S. Skhiri, S. Jouili, and E. Zimányi. An analytics-aware conceptual model for evolving graphs. In *Data Warehousing and Knowledge Discovery*. Springer, 2013.

Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *OSDI*, pages 17–30, 2012.

Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. Graphx: Graph processing in a distributed dataflow framework. In *OSDI*, pages 599–613, 2014.

Goetz Graefe. Query evaluation techniques for large databases. *ACM Comput. Surv.*, 25(2):73–170, 1993.

D. Greene, D. Doyle, and P. Cunningham. Tracking the evolution of communities in dynamic social networks. In *ASONAM*, 2010.

Yong Guo, Marcin Biczak, Ana Lucia Varbanescu, Alexandru Iosup, Claudio Martella, and Theodore L. Willke. How well do graph-processing platforms perform? an empirical performance evaluation and analysis. In *IPDPS*, pages 395–404, 2014.

Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. WTF: the who to follow service at twitter. In *WWW*, pages 505–514, 2013.

Minyang Han and Khuzaima Daudjee. Giraph unchained: Barrierless asynchronous parallel execution in pregel-like graph processing systems. *PVLDB*, 8(9):950–961, 2015.

Minyang Han, Khuzaima Daudjee, Khaled Ammar, M Tamer Özsu, Xingfang Wang, and Tianqi Jin. An experimental comparison of Pregel-like graph processing systems. *PVLDB*, 7(12):1047–1058, 2014a.

Wentao Han, Youshan Miao, Kaiwei Li, Ming Wu, Fan Yang, Lidong Zhou, Vijayan Prabhakaran, Wenguang Chen, and Enhong Chen. Chronos: a graph engine for temporal graph analysis. In *EuroSys*, pages 1:1–1:14, 2014b.

Wook-Shin Han, Sangyeon Lee, Kyungyeol Park, Jeong-Hoon Lee, Min-Soo Kim, Jinha Kim, and Hwanjo Yu. TurboGraph: a fast parallel graph engine handling billion-scale graphs in a single PC. In *KDD*, pages 77–85, 2013.

Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B. Gibbons, Garth A. Gibson, Gregory R. Ganger, and Eric P. Xing. More effective distributed ML via a stale synchronous parallel parameter server. In *NIPS*, pages 1223–1231, 2013.

Sungpack Hong, Hassan Chafi, Eric Sedlar, and Kunle Olukotun. Green-marl: a DSL for easy and efficient graph analysis. In *ASPLOS*, pages 349–362, 2012.

Sungpack Hong, Semih Salihoglu, Jennifer Widom, and Kunle Olukotun. Simplifying scalable graph processing with a domain-specific language. In *CGO*, page 208, 2014.

Botong Huang, Matthias Boehm, Yuanyuan Tian, Berthold Reinwald, Shirish Tatikonda, and Frederick R. Reiss. Resource elasticity for large-scale machine learning. In *SIGMOD*, pages 137–152, 2015.

W. Huo and V. Tsotras. Efficient temporal shortest path queries on evolving social graphs. In *SSDBM*, 2014.

Dawei Jiang, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, and Sai Wu. epic: an extensible and scalable system for processing big data. *PVLDB*, 7(7): 541–552, 2014.

Alekh Jindal, Samuel Madden, Malú Castellanos, and Meichun Hsu. Graph analytics using the Vertica relational database. *CoRR*, abs/1412.5263, 2014a.

Alekh Jindal, Praynaa Rawlani, Eugene Wu, Samuel Madden, Amol Deshpande, and Mike Stonebraker. VERTEXICA: your relational friend for graph analytics! *PVLDB*, 7(13):1669–1672, 2014b.

Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In Lusheng Wang, editor, *Computing and Combinatorics*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2005.

U. Kang, Charalampos E. Tsourakakis, and Christos Faloutsos. PEGASUS: A peta-scale graph mining system. In *ICDM*, pages 229–238, 2009.

U. Kang, Hanghang Tong, Jimeng Sun, Ching-Yung Lin, and Christos Faloutsos. GBASE: a scalable and general graph management system. In *SIGKDD*, pages 1091–1099, 2011.

George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. *J. Parallel Distrib. Comput.*, 48(1):96–129, 1998.

Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

Jeremy Kepner and John Gilbert. *Graph algorithms in the language of linear algebra*, volume 22. SIAM, 2011.

Zuhair Khayyat, Karim Awara, Amani Alonazi, Hani Jamjoom, Dan Williams, and Panos Kalnis. Mizan: a system for dynamic load balancing in large-scale graph processing. In *EuroSys*, pages 169–182, 2013.

Farzad Khorasani, Keval Vora, Rajiv Gupta, and Laxmi N. Bhuyan. Cusha: vertex-centric graph processing on gpus. In *HPDC*, pages 239–252, 2014.

Udayan Khurana and Amol Deshpande. Efficient snapshot retrieval over historical graph data. In *ICDE*, pages 997–1008, 2013.

Udayan Khurana and Amol Deshpande. Storing and analyzing historical graph data at scale. In *EDBT*, pages 65–76, 2016.

Eric D Kolaczyk. *Statistical analysis of network data*. Springer, 2009.

G. Koloniari and E. Pitoura. Partial view selection for evolving social graphs. In *GRADES workshop*, 2013.

M. Kornacker, A. Behm, V. Bittorf, T. Bobrovytsky, C. Ching, A. Choi, J. Erickson, M. Grund, D. Hecht, M. Jacobs, I. Joshi, L. Kuff, D. Kumar, A. Leblang, N. Li, I. Pandis, H. Robinson, D. Rorke, S. Rus, J. Russell, D. Tsirogiannis, S. Wanderman-Milne, and M. Yoder. Impala: A Modern, Open-Source SQL Engine for Hadoop. In *CIDR*, 2015.

Aapo Kyrola, Guy E. Blelloch, and Carlos Guestrin. GraphChi: Large-scale graph computation on just a PC. In *OSDI*, pages 31–46, 2012.

A. Labouseur, J. Birnbaum, Jr. Olsen, P., S. Spillane, J. Vijayan, J. Hwang, and W. Han. The G* graph database: efficiently managing large distributed dynamic graphs. *Distributed and Parallel Databases*, 2014.

Longbin Lai, Lu Qin, Xuemin Lin, and Lijun Chang. Scalable subgraph enumeration in mapreduce. *PVLDB*, 8(10):974–985, 2015.

Jure Leskovec and Julian J. Mcauley. Learning to discover social circles in ego networks. In *NIPS*, pages 548–556, 2012.

Jimmy Lin and Michael Schatz. Design patterns for efficient graph algorithms in mapreduce. In *MLG*, pages 78–85. ACM, 2010.

Guimei Liu and Limsoon Wong. Effective pruning techniques for mining quasi-cliques. In *ECML/PKDD Part II*, pages 33–49, 2008.

Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Graphlab: A new framework for parallel machine learning. In *UAI*, pages 340–349, 2010.

Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Distributed GraphLab: A framework for machine learning in the cloud. *PVLDB*, 5(8):716–727, 2012.

Yi Lu, James Cheng, Da Yan, and Huanhuan Wu. Large-scale distributed graph computing systems: An experimental evaluation. *PVLDB*, 8(3):281–292, 2014.

Peter Macko, Virendra J. Marathe, Daniel W. Margo, and Margo I. Seltzer. LLAMA: efficient graph analytics using large multiversioned arrays. In *ICDE*, pages 363–374, 2015.

Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD Conference*, pages 135–146, 2010.

Mirjana Mazuran, Edoardo Serra, and Carlo Zaniolo. Extending the power of datalog recursion. *VLDB J.*, 22(4):471–493, 2013.

Robert Ryan McCune, Tim Weninger, and Greg Madey. Thinking like a vertex: A survey of vertex-centric frameworks for large-scale distributed graph processing. *ACM Comput. Surv.*, 48(2):25, 2015.

Frank McSherry, Michael Isard, and Derek G Murray. Scalability! but at what cost. In *HotOS*. USENIX Association, 2015.

Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: Interactive analysis of web-scale datasets. *PVLDB*, 3(1):330–339, 2010.

Youshan Miao, Wentao Han, Kaiwei Li, Ming Wu, Fan Yang, Lidong Zhou, Vijayan Prabhakaran, Enhong Chen, and Wenguang Chen. Immortalgraph: A system for storage and analysis of temporal graphs. *ACM TOS*, July 2015. URL http://research.microsoft.com/apps/pubs/default.aspx?id=242176.

Svilen R. Mihaylov, Zachary G. Ives, and Sudipto Guha. REX: Recursive, delta-based data-centric computation. *PVLDB*, 5(11):1280–1291, 2012.

Jayanta Mondal and Amol Deshpande. Managing large dynamic graphs efficiently. In *SIGMOD*, pages 145–156, 2012.

Jayanta Mondal and Amol Deshpande. Eagr: supporting continuous egocentric aggregate queries over large dynamic graphs. In *SIGMOD*, pages 1335–1346, 2014.

Inderpal Singh Mumick, Hamid Pirahesh, and Raghu Ramakrishnan. The magic of duplicates and aggregates. In *VLDB*, pages 264–277, 1990.

Derek G. Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. Naiad: A timely dataflow system. In *SOSP*, pages 439–455, 2013.

Donald Nguyen, Andrew Lenharth, and Keshav Pingali. A lightweight infrastructure for graph analytics. In *SOSP*, pages 456–471, 2013.

Raj Kumar Pan and Jari Saramäki. Path lengths, correlations, and centrality in temporal networks. *Physical Review E*, 2011.

Keshav Pingali, Donald Nguyen, Milind Kulkarni, Martin Burtscher, Muhammad Amber Hassaan, Rashid Kaleem, Tsung-Hsien Lee, Andrew Lenharth, Roman Manevich, Mario Méndez-Lojo, Dimitrios Prountzos, and Xin Sui. The tao of parallelism in algorithms. In *PLDI*, pages 12–25, 2011.

Abdul Quamar and Amol Deshpande. NScaleSpark: Subgraph-centric graph analytics on Apache Spark. In *Proceedings of the SIGMOD Workshop on Network Data Analytics (NDA)*, pages 5:1–5:8, 2016.

Abdul Quamar, Amol Deshpande, and Jimmy Lin. NScale: neighborhood-centric large-scale graph analytics in the cloud. *VLDB Journal*, 25(2): 125–150, 2016.

Louise Quick, Paul Wilkinson, and David Hardcastle. Using pregel-like large scale graph processing frameworks for social network analysis. In *ASONAM*, pages 457–463, 2012.

Chenghui Ren, Eric Lo, Ben Kao, Xinjie Zhu, and Reynold Cheng. On querying historical evolving graph sequences. *PVLDB*, 4(11):726–737, 2011.

Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011. .

Kenneth A. Ross and Yehoshua Sagiv. Monotonic aggregation in deductive databases. In *PODS*, pages 114–126, 1992.

Maayan Roth, Assaf Ben-David, David Deutscher, Guy Flysher, Ilan Horn, Ari Leichtberg, Naty Leiser, Yossi Matias, and Ron Merom. Suggesting friends using the implicit social graph. In *KDD*, pages 233–242, 2010.

Amitabha Roy, Ivo Mihailovic, and Willy Zwaenepoel. X-stream: edge-centric graph processing using streaming partitions. In *SOSP*, pages 472–488, 2013.

Amitabha Roy, Laurent Bindschaedler, Jasmina Malicevic, and Willy Zwaenepoel. Chaos: scale-out graph processing from secondary storage. In *SOSP*, pages 410–424, 2015.

Semih Salihoglu and Jennifer Widom. GPS: a graph processing system. In *SSDBM*, pages 22:1–22:12, 2013.

Semih Salihoglu and Jennifer Widom. Optimizing graph algorithms on pregel-like systems. *PVLDB*, 7(7):577–588, 2014.

Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Jiwon Seo, Jongsoo Park, M. Amber Hassaan, Shubho Sengupta, Zhaoming Yin, and Pradeep Dubey. Navigating the maze of graph analytics frameworks using massive graph datasets. In *SIGMOD*, pages 979–990, 2014.

Sebastian Schelter, Stephan Ewen, Kostas Tzoumas, and Volker Markl. "All roads lead to rome": optimistic recovery for distributed iterative data processing. In *CIKM*, pages 1919–1928, 2013.

Jiwon Seo, Stephen Guo, and Monica S. Lam. Socialite: Datalog extensions for efficient social network analysis. In *ICDE*, pages 278–289, 2013a.

Jiwon Seo, Jongsoo Park, Jaeho Shin, and Monica S. Lam. Distributed socialite: A datalog-based language for large-scale graph analysis. *PVLDB*, 6 (14):1906–1917, 2013b.

Zechao Shang and Jeffrey Xu Yu. Catch the wind: Graph workload balancing on cloud. In *ICDE*, pages 553–564, 2013.

Bin Shao, Haixun Wang, and Yatao Li. Trinity: a distributed graph engine on a memory cloud. In *SIGMOD*, pages 505–516, 2013.

Yingxia Shao, Bin Cui, and Lin Ma. PAGE: A partition aware engine for parallel graph computation. *IEEE Trans. Knowl. Data Eng.*, 27(2):518–530, 2015.

Yanyan Shen, Gang Chen, H. V. Jagadish, Wei Lu, Beng Chin Ooi, and Bogdan Marius Tudor. Fast failure recovery in distributed graph processing systems. *PVLDB*, 8(4):437–448, 2014.

Yossi Shiloach and Uzi Vishkin. An o(log n) parallel connectivity algorithm. *J. Algorithms*, 3(1):57–67, 1982.

Alexander Shkapsky, Mohan Yang, and Carlo Zaniolo. Optimizing recursive queries with monotonic aggregates in deals. In *ICDE*, pages 867–878, 2015.

Julian Shun and Guy E. Blelloch. Ligra: a lightweight graph processing framework for shared memory. In *ACM SIGPLAN Notices*, pages 135–146, 2013.

David E. Simmen, Karl Schnaitter, Jeff Davis, Yingjie He, Sangeet Lohariwala, Ajay Mysore, Vinayak Shenoi, Mingfeng Tan, and Yu Xiao. Large-scale graph analytics in aster 6: Bringing context to big data discovery. *PVLDB*, 7(13):1405–1416, 2014.

Yogesh Simmhan, Alok Gautam Kumbhare, Charith Wickramaarachchi, Soonil Nagarkar, Santosh Ravi, Cauligi S. Raghavendra, and Viktor K. Prasanna. Goffish: A sub-graph centric framework for large-scale graph analytics. In *Euro-Par*, pages 451–462, 2014.

Chunyao Song, Tingjian Ge, Cindy Chen, and Jie Wang. Event pattern matching over graph streams. *VLDB*, 2014.

Isabelle Stanton and Gabriel Kliot. Streaming graph partitioning for large distributed graphs. In *SIGKDD*, pages 1222–1230, 2012.

Zhao Sun, Hongzhi Wang, Haixun Wang, Bin Shao, and Jianzhong Li. Efficient subgraph matching on billion node graphs. *PVLDB*, 5(9):788–799, 2012.

Narayanan Sundaram, Nadathur Satish, Md. Mostofa Ali Patwary, Subramanya Dulloor, Michael J. Anderson, Satya Gautam Vadlamudi, Dipankar Das, and Pradeep Dubey. Graphmat: High performance graph analytics made productive. *PVLDB*, 8(11):1214–1225, 2015.

Siddharth Suri and Sergei Vassilvitskii. Counting triangles and the curse of the last reducer. In *WWW*, pages 607–614, 2011.

Aubrey Tatarowicz, Carlo Curino, Evan P. C. Jones, and Sam Madden. Lookup tables: Fine-grained partitioning for distributed databases. In *ICDE*, pages 102–113, 2012.

Carlos H. C. Teixeira, Alexandre J. Fonseca, Marco Serafini, Georgos Siganos, Mohammed J. Zaki, and Ashraf Aboulnaga. Arabesque: a system for distributed graph mining. In *SOSP*, pages 425–440, 2015.

Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Anthony, Hao Liu, and Raghotham Murthy. Hive - a petabyte scale data warehouse using hadoop. In *ICDE*, pages 996–1005, 2010.

Yuanyuan Tian, Shirish Tatikonda, and Berthold Reinwald. Scalable and numerically stable descriptive statistics in systemml. In *ICDE*, pages 1351–1359, 2012.

Yuanyuan Tian, Andrey Balmin, Severin Andreas Corsten, Shirish Tatikonda, and John McPherson. From ”think like a vertex” to “think like a graph”. *PVLDB*, 7(3):193–204, 2013.

Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O’Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *SOCC*, pages 5:1–5:16, 2013.

Changliang Wang and Lei Chen. Continuous subgraph pattern search over graph streams. In *ICDE*, 2009.

Jingjing Wang, Magdalena Balazinska, and Daniel Halperin. Asynchronous and fault-tolerant recursive datalog evaluation in shared-nothing engines. *PVLDB*, 8(12):1542–1553, 2015.

Peng Wang, Kaiyuan Zhang, Rong Chen, Haibo Chen, and Haibing Guan. Replication-based fault-tolerance for large-scale graph processing. In *DSN*, pages 562–573, 2014.

Ming Wu, Fan Yang, Jilong Xue, Wencong Xiao, Youshan Miao, Lan Wei, Haoxiang Lin, Yafei Dai, and Lidong Zhou. Gram: Scaling graph computation to the trillions. In *SoCC*, pages 408–421, 2015.

Jingen Xiang, Cong Guo, and Ashraf Aboulnaga. Scalable maximum clique computation using mapreduce. In *ICDE*, pages 74–85, 2013.

Chenning Xie, Rong Chen, Haibing Guan, Binyu Zang, and Haibo Chen. SYNC or ASYNC: time to fuse for distributed graph-parallel computation. In *PPoPP*, pages 194–204, 2015a.

Wenlei Xie, Guozhang Wang, David Bindel, Alan J. Demers, and Johannes Gehrke. Fast iterative graph computation with block updates. *PVLDB*, 6 (14):2014–2025, 2013.

Wenlei Xie, Yuanyuan Tian, Yannis Sismanis, Andrey Balmin, and Peter J. Haas. Dynamic interaction graphs with probabilistic edge decay. In *ICDE*, pages 1143–1154, 2015b.

Konstantinos Xirogiannopoulos, Udayan Khurana, and Amol Deshpande. Graphgen: Exploring interesting graphs in relational data. *PVLDB*, 8(12), 2015.

Da Yan, James Cheng, Yi Lu, and Wilfred Ng. Blogel: A block-centric framework for distributed computation on real-world graphs. *PVLDB*, 7(14): 1981–1992, 2014a.

Da Yan, James Cheng, Kai Xing, Yi Lu, Wilfred Ng, and Yingyi Bu. Pregel algorithms for graph connectivity problems with performance guarantees. *PVLDB*, 7(14):1821–1832, 2014b.

Da Yan, James Cheng, Yi Lu, and Wilfred Ng. Effective techniques for message reduction and load balancing in distributed graph computation. In *WWW*, pages 1307–1317, 2015.

Da Yan, Yingyi Bu, Yuanyuan Tian, Amol Deshpande, and James Cheng. Big graph analytics systems. In *SIGMOD*, pages 2241–2243, 2016a.

Da Yan, James Cheng, M. Tamer Özsu, Fan Yang, Yi Lu, John C. S. Lui, Qizhen Zhang, and Wilfred Ng. A general-purpose query-centric framework for querying big graphs. *PVLDB*, 9(7):564–575, 2016b.

Da Yan, James Cheng, and Fan Yang. Lightweight fault tolerance in large-scale distributed graph processing. *CoRR*, abs/1601.06496, 2016c.

Da Yan, Yuzhen Huang, James Cheng, and Huanhuan Wu. Efficient processing of very large graphs in a small cluster. *CoRR*, abs/1601.05590, 2016d.

Mohan Yang, Alexander Shkapsky, and Carlo Zaniolo. Parallel bottom-up evaluation of logic programs: Deals on shared-memory multicore machines. In *ICLP*, 2015.

Philip S. Yu, Xin Li, and Bing Liu. On the temporal dimension of search. In *WWW Alt*, pages 448–449, 2004.

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, pages 15–28, 2012.

Honglei Zhang, Jenni Raitoharju, Serkan Kiranyaz, and Moncef Gabbouj. Limited random walk algorithm for big graph data clustering. *CoRR*, abs/1606.06450, 2016.

Yanfeng Zhang, Qixin Gao, Lixin Gao, and Cuirong Wang. Maiter: An asynchronous graph processing framework for delta-based accumulative iterative computation. *IEEE Trans. Parallel Distrib. Syst.*, 25(8):2091–2100, 2014.

Peixiang Zhao, Charu C. Aggarwal, and Min Wang. gSketch: on query estimation in graph streams. *VLDB*, 2011.

Da Zheng, Disa Mhembere, Randal C. Burns, Joshua T. Vogelstein, Carey E. Priebe, and Alexander S. Szalay. Flashgraph: Processing billion-node graphs on an array of commodity ssds. In *FAST*, pages 45–58, 2015.

Li Zheng, Chao Shen, Liang Tang, Tao Li, Steve Luis, and Shu-Ching Chen. Applying data mining techniques to address disaster information management challenges on mobile devices. In *KDD*, pages 283–291, 2011.

Jianlong Zhong and Bingsheng He. Medusa: Simplified graph processing on gpus. *IEEE Trans. Parallel Distrib. Syst.*, 25(6):1543–1552, 2014.

Chang Zhou, Jun Gao, Binbin Sun, and Jeffrey Xu Yu. Mocgraph: Scalable distributed graph processing using message online computing. *PVLDB*, 8 (4):377–388, 2014.

Yang Zhou, Ling Liu, Kisung Lee, and Qi Zhang. Graphtwist: Fast iterative graph computation with two-tier optimizations. *PVLDB*, 8(11):1262–1273, 2015.

Xiaowei Zhu, Wentao Han, and Wenguang Chen. Gridgraph: Large-scale graph processing on a single machine using 2-level hierarchical partitioning. In *USENIX ATC*, pages 375–386, 2015.

Xiaowei Zhu, Wenguang Chen, Weimin Zheng, and Xiaosong Ma. Gemini: A computation-centric distributed graph processing system. In *OSDI*, 2016.