In-Memory Big Graph: A Future Research Agenda

Deepali Jain, Ripon Patgiri(✉), and Sabuzima Nayak

National Institute of Technology Silchar, Silchar, India
jaindeepali010@gmail.com, ripon@cse.nits.ac.in, sabuzimanayak@gmail.com

Abstract. With the growth of the inter-connectivity of the world, Big Graph has become a popular emerging technology. For instance, social media (Facebook, Twitter). Prominent examples of Big Graph include social networks, biological network, graph mining, big knowledge graph, big web graphs and scholarly citation networks. A Big Graph consists of millions of nodes and trillion of edges. Big Graphs are growing exponentially and requires large computing machinery. Big Graph is posing many issues such as storage, scalability, processing and many more. This paper gives a brief overview of in-memory Big Graph Systems and some key challenges. Also, sheds some light on future research agendas of in-memory systems.

Keywords: Big Graph · Big Data · In-memory Big Graph · Large graph · Semi-structured data · Social networks

1 Introduction

Today, every domain ranging from social networks to web graphs implements Big Graph. There are diverse graph data that are growing rapidly. Big Graph has found its application in many domains, in particular, computer networks [19], social networks [23,39], mobile call networks [40], and biological networks [13]. A prominent example of Big Graph in the field of computer network is Mobile Opportunistic Networks (MONs) [19]. It is a challenging task to understand and characterize the properties of time-varying graph, for instance, MONs. Big Graph has major applications in social networking sites, for example, Facebook friends [39], and Twitter tweets [23]. In Facebook, there are millions of nodes which represent people and billion of edges which represent the relationships between these people. In Twitter, "who is following whom" is represented by Big Graphs. In mobile call networks, Wang et al. [40] uses Big Graph to understand the similarity of two individual relationships over mobile phones and in the social network. In Bioinformatic, Big Graph is used to represent DNA and other protein molecular structure. Moreover, the graph theory is used for analysis and calculation of molecular topology [13].

Handling Big Graph is a complex task. Moreover, there are many challenges associated with large graphs as they require huge computation. Therefore, there is a great need for parallel Big Graph systems. Most of the Big Graph frameworks are implemented based on HDD. A few in-memory Big Graph frameworks are available. Because, RAM is volatile storage media, small in size and costly. However, the cost of the hardware is dropping sharply. Therefore, in the future, RAM will be given prime focus in designing a Big Graph framework. Currently, Flash/SSD-based Big Graph framework is developing. Flash/SSD based frameworks are faster than HDD. Naturally, Flash/SSD based Big Graph frameworks are slower than in-memory Big Graph frameworks. Thus, in coming future, more RAM-based Big Graph frameworks will be designed for storing Big Graph. In-memory Big Graph is a key research to be focused on. A few research questions (RQ) on in-memory Big Graph are outlined as follows: (a) **RQ1**: Can In-memory Big Graph able to process more than trillions of nodes or edges? (b) **RQ2**: Can In-memory Big Graph able to handle the Big Graph size beyond terabytes? (c) **RQ3**: Is there any alternative to HDD, SSD or Flash memory (NAND)? (d) **RQ4**: What are the real-time Big Graph processing engines available without using HDD, SSD, or Flash?

The research questions **RQ1**, **RQ2**, **RQ3**, and **RQ4** motivate us to examine the insight on massively scalable in-memory Big Graph processing engine. Besides, implementing in-memory Big Graph Database is a prominent research challenge. Thus, in this paper, we present a deep insight on scalable in-memory Big Graph processing engine as well as a database for future research.

2 Big Graph

A Big Graph comprises of billions of vertices and hundreds of billion of edges. Big Graph is applied in diverse areas [40], namely, biological networks, social networks, information networks and technological network. Big Graph is unstructured and irregular that makes the graph processing more complex. In real world cases, Big graph is dynamic, i.e., there are some temporal graphs which changes with time [29]. Particularly, new nodes are inserted and deleted frequently. Handling the frequent changes in edges and vertices are truly a research challenge.

2.1 In-Memory Big Graph

Big Graph represents a huge volume of data. This huge sized data are usually stored in secondary memory. However, storing graphs in RAM improves the performance of graph processing and analysis. But, in-memory Big Graph system requires a huge amount of resources [34]. Numerous Big Graph processing systems are designed based on in-memory Big Graph with the backing of secondary storage. For example, PowerGraph [15], GraphX [16], and Pregelix [6]. The ability of continuous holding data in RAM in a fault-tolerant manner makes in-memory big graph systems more suitable for many data analytic applications. For instance, Spark [44] uses the elastic persistence model to keep the dataset in memory, or disk or both. However, state-of-the-art Big Graphs do not provide intrinsic in-memory Big Graph without using secondary storage.

3 In-Memory Big Graph Framework

As the data keeps on growing, there should be a system which can efficiently work with the incremental data and has a large memory to hold the data. As the data is growing rapidly, processing of large graph becomes the key barrier. There is also scalability issue associated with in-memory Big Graph systems. So, there is a great need for Big Graph processing systems that can overcome the issues. There are many existing in-memory Big Graph systems like Power Graph [15], GraphX [16]. These systems handle the issues like storage, scalability, fault tolerance, communication costs, workload. In this section some in-memory Big Graph engines are discussed. Table 1 illustrates the evaluation of in-memory Big Graph on the basis of various parameters. Moreover, Table 2 exposes the various sizes of nodes and edges with data sources.

The power-law degree distribution graphs are challenging task to partition. Because, it causes work imbalance. Work imbalance leads to communication and storage issue. Hence, PowerGraph [15] uses Gather-Apply Scatter (GAS) model.

Table 1. Evaluation of existing framework

Name	In-memory	Hybrid	Scalability	Fault tolerance issue	Communication overhead	Framework
PowerGraph [15]	✓	×	✓	×	×	Vertex-centric
GraphX [16]	✓	×	✓	×	×	RDD [16]
Ringo [31]	✓	×	×	×	×	SNAP [22]
Trinity [33]	✓	×	✓	✓	×	Distributed graph engine
Pregelix [6]	✓	×	✓	×	×	Distributed graph system
GraphBig [24]	✓	×	✓	×	-	Vertex-centric
GraphMP [45]	×	✓	✓	×	×	VSW [35]
GraphH [34]	✓	×	✓	×	×	Vertex-centric

Table 2. Evaluation of existing framework. M = Million, B = Billion

Name	Nodes	Edges	Data source
PowerGraph [15]	40M	1.5B	Twitter [20]
GraphX [16]	4.8M	69M	twitter-2010 [4], uk-2007-05 [3]
Ringo [31]	4.8M,42M	69M,1.5B	Twitter [20], LiveJournal [1]
Trinity [33]	1B	13B	Social Graph, Web Graph
Pregelix [6]	6.9M	6 B	BTC [10], Webmap [37]
GraphBig [24]	1.9M	2.8M	CA Road Network [22]
GraphMP [45]	1.1B		Twitter Graph
GraphH [34]	788M	47.6B	Twitter Graph

It uses vertices for computation over edges. In this way, PowerGraph maintains the 'think like a vertex' [38] philosophy. And, it helps in processing trillion of nodes. It exploits parallelism, to achieve less communication and storage costs. PowerGraph supports both asynchronous and synchronous execution. It provides fault-tolerance by vertex replication and data-dependency method.

There is another system, called GraphX [16] which is built on Spark. GraphX supports in-memory system by using Spark storage abstraction, called Resilient Distributed Dataset (RDD) which is essential for iterative graph algorithms. RDD helps in handling trillions of nodes. It also has enough in-memory replication to reduce the re-computation in case of any failure. GraphX retains low-cost fault tolerance by using distributed dataflow networks. GraphX provides ease of analyzing unstructured and tabular data.

Ringo [31] is an in-memory interactive graph analytic system that provides graph manipulation and analysis. Working data set is stored in RAM, and non-working data set are stored in HDD. The prime objective is to provide faster execution rather than scalability. Also, Ringo needs a dynamic graph representation. For efficient graph representation, Ringo uses a Compressed Sparse Row format [17]. Ringo builds on the Stanford Network Analysis Platform (SNAP) [21]. Ringo is easily adaptable due to the integrated processing of graphs and tables. It uses an easy-to-use Python interface and execution on a single machine. However, scalability is a major concern in Ringo. In addition, Ringo is unable to support more than trillions of edges or higher sized Big Graph.

Trinity [33] is a distributed graph engine build over a memory cloud. Memory cloud is a globally addressable, distributed key-value store over a cluster of machines. Data sets can be accessed quickly through distributed in-memory storage. It also supports online query processing as well as offline analysis large graphs. The basic philosophy behind designing the Trinity is (a) high-speed network is readily available today, and (b) DRAM prices will go down in the long run. Trinity is an all-in-memory system, thus, the graph data are loaded in RAM before computation. Trinity has its own language called Trinity specification language (TSL) that minimizes the gap between graph model and data storage. Trinity tries to avoid memory gaps between large numbers of key-value pairs by implementing circular memory management mechanism. It uses heartbeat messages to proactively detect machine failures. Trinity uses Random access data of distributed RAM storage, therefore, it can support trillions of nodes in future. There are many advantages of Trinity, specifically, (1) object-oriented data manipulation of data in the memory cloud, (2) data integration, and (3) TSL facilitates system extension.

Preglix [6] is an open source distributed graph processing system. It supports bulk-synchronous vertex-oriented programming model for analysis of large scale graphs. Pregelix is an iterative dataflow design, and it can effectively handle both in-memory and out-of-core workloads. Pregelix uses Hyracks [5] engine for execution purpose. It is a general-purpose shared-nothing dataflow engine. Pregelix employs both B-Tree and LSM (log-structured merge-tree) B-Tree index structures. These index structures are used to store partitions of vertices on worker machines and these are imported from the Hyracks storage library. The level of fault tolerance in Pregelix is same as other Pregel-like systems. Pregelix system supports larger datasets, and also strengthen multi-user workloads. Pregelix explores more flexible scheduling mechanisms. It gives various data redistribution (allowed by Hyracks) techniques for the optimization of given Pregel algorithm's computation time. It is the only open source system that supports multi-user workloads, has out-of-core support, and allows runtime flexibility. Accommodation of nodes in memory is based on the available memory.

GraphBig [24] is a benchmark suite inspired by IBM System G project. It is a toolkit for computing industrial graphs used by many commercial clients. It is used for performing graph computations and data sources. GraphBig utilizes a dynamic, vertex-centric data representation, which can be oftenly seen in real-world graph systems. GraphBig uses compact format of CSR (Compressed Sparse Row) to save memory space and simplify the graph build complexity. The memory of graph computing shows high cache miss rates on CPUs and also high branch/memory divergence on GPUs.

GraphMP is a semi-external-memory Big Graph processing system. In SEM [45] all vertices of the graph are stored in RAM and edges are accessed from the disk. GraphMP uses vertex-centric sliding window (VSW) computation model. It initially separates the vertices into disjoint intervals. Each interval has a shard. The shard contains the edges that have destination vertices within the interval. During computation, GraphMP slides a window on every vertex and the edges are processed shard by shard. The shard is loaded into RAM for processing. At the end of the program the updates are written to the disk. GraphMP uses a Bloom Filter for selective scheduling to avoid inactive shards. A shard cache mechanism is implemented for complete usage of the memory compressed. GraphMP does not store the edges in memory to handle Big Graph efficiently with limited memory. However, it requires more memory to store all vertices. In addition, it does not use logical locks to improve the performance. It is unable to support trillions of nodes since it is a single machine semi-external memory graph processing system.

GraphH [34] is a memory-disk hybrid approach which maximizes the amount of in-memory data. Initially, the Big Graph is partitioned using two stages. In first stage, the Big Graph is divided into a set of tiles. Each set of tiles uses a compact data structure to store the assigned edges. In the second stage, GraphH assigns the tiles uniformly to computational servers. These servers run the vertex-centric programs. Each vertex maintains a replica of all servers during computation. GraphH implements GAB (Gather-Apply-Broadcast) Computation Model for updating the vertex. Along the in-edges, the data are gathered from local memory to compute the accumulator. GraphH implements Edge Cache Mechanism to reduce the disk access overhead. It is efficient in small cluster or single commodity server. However, it can support trillions of nodes due to GAB model implementation.

4 Key Issues

Power-Law Graph: Power-law graph can be defined as the graph with vertex degree distribution follows a power-law function. It creates many difficulties in the analysis and processing of Big Graph. For example, imbalanced workload in the Big Graph processing systems.

Graph Partitioning: Big Graph uses graph parallel processing technology. The processing requires partitioning of Big Graph into subgraphs. However, the real world graph is highly skewed and have a power-law degree distribution. Hence, Big Graph needs to efficiently partition the graph.

Distributed Graph Placement: Big Graph is stored in cluster of systems. Hence, issues of distributed system are also applicable to the storage and processing of Big Graph. In-memory Big Graph system suffers from scalability issues because RAM is costly and small sized. In addition, in-memory Big Graph must ensure consistency. The data are replicated in several nodes to retain the data even if there is a system fault. In-memory Big Graph systems also has hotspot issue due to the skewed nature of Big Graph.

Cost: In-memory Big Graph processing systems store the graph in RAM. Hence, it requires good quality computing infrastructure to handle such huge size graph. For example, GraphX requires 16 TB memory to process 10 billion edges [16]. During computation the systems require to store the whole graph and also the network-transmitted messages in RAM [34].

Incompetent to Execute Inexact Algorithm: Big Graphs are incapable to execute the inexact algorithms due to graph matching. Most of the inexact algorithms take a very long computational time for processing [8].

5 Key Challenges

Dynamic Graph: Analysis of dynamic graph is an arduous process in which the graph structure changes frequently. In this type of graph, vertices and edges are inserted and deleted frequently [42]. As the dynamic graphs keep on changing, data management and graph analytic takes the responsibility for the sequence of large graph snapshots as well as for streaming data.

Graph-Based Data Integration and Knowledge Graphs: For the analysis purpose, Big Graph data is extracted from original data sources. However, data extraction from the data source is full of obstacles. One key challenge is knowledge graph [12,27]. The knowledge graph provides a huge volume of interrelated information regarding real-world entities. Moreover, key issues with the knowledge graph is integration of low quality, highly diverse and large volume of data.

Graph Data Allocation and Partitioning: Effectiveness of graph processing highly depends on efficient data partitioning of Big Graph. Along with partitioning, load balancing is required for efficient utilization of nodes. Hence, the challenge is to find a graph partitioning technique that balances the distribution of vertices and their edges such that each subgraph have minimum and same number of vertices and vertex cut. But, graph partitioning problem is NP-hard [7].

Interactive Graph Analytic: Interactive graphs with proper visualization is highly desirable for exploration and analysis of graph data. But sometimes this visualization becomes a major challenge to analyze. For example, $k - SNAP$ [38] generates summarized graphs having k vertices. The parameter change k in $k - SNAP$ activates an OLAP-like roll-up and drill-down within a dimension hierarchy [9]. However, due to its dependency on pre-determined parameter, this approach is not fully interactive.

6 Future Research Agenda

An 'in-memory' system requires backing of the secondary storage for consistency due to volatility of RAM. The in-memory system stores data in RAM as well as in HDD/SSD. However, the data of intrinsic 'in-memory' system is stored entirely in RAM. HDD/SSD is used to recover data upon failure of a machine. A few hybrid system stores working datasets in RAM and non-working dataset in secondary storage. Read/write cost is high in secondary storage. Nevertheless, hybrid system becomes more scalable. Hence, there is a trade-off between performance and scalability.

Today, everyone is connected globally through internet. Hence, data is growing exponentially. But, the size of RAM is fixed. Thus, the challenge starts with maintaining large scale data in RAM. Similarly, Big Graph size is also growing. For instance, Twitter and Facebook. Big Graph analytic requires a real-time processing engine which demands in-memory Big Graph system. It is a grand challenge to design a pure in-memory Big Graph database and also a future research agenda. Intrinsic in-memory Big Graph database can be implemented through Dr. Hadoop framework [11]. In this paper, future research agenda is categorized into two categories, namely, data-intensive Big Graph, and compute-intensive Big Graph.

6.1 Data-Intensive Big Graph

Dr. Hadoop: A Future Scope for Big Graph. Dr. Hadoop is a framework of purely in-memory systems [11]. However, Dr. Hadoop backups the data in secondary storage periodically [26]. Even though, Dr. Hadoop is developed for massive scalability of metadata, it can be adapted in various purposes [30]. Dr. Hadoop stores all data in RAM and replicates in other two neighbor RAM, say, left and right node. Figure 1 demonstrates the replication of data in RAM in Dr. Hadoop framework. Dr. Hadoop forms 3-node cluster at the very beginning [30]. Each node must have left and right node for replication of data from RAM.

Any node can leave or join the Dr. Hadoop cluster. Figure 1 illustrates the insertion of a new node in Dr. Hadoop. Dr. Hadoop implements circular doubly linked list where a node failure breaks the ring. But, Dr. Hadoop is unaffected by the failure of any one node at a given time [30]. There are left node and right

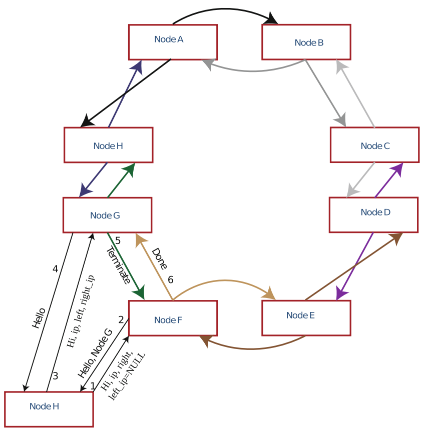

Fig. 1. Insertion of a node in Dr. Hadoop.

node to serve the data. Moreover, Dr. Hadoop can tolerate many non-contiguous node failure at a given time. However, Dr. Hadoop is unable to tolerate consecutive three-node failure at a given time. Three contiguous node failure causes loss of data of a node. Even, Dr. Hadoop can tolerate consecutive two-node failure at a given time. Big Graph can be stored in the RAM and also replicated to two neighboring nodes. The backup is stored in HDD/SSD. The key merits of Dr. Hadoop are- (a) purely in-memory database system, (b) incremental scalability, (c) fine-grained fault-tolerance, (d) efficient load-balancing, and (e) requires least administration. Incorporating Dr. Hadoop with Big Graph can help in overcoming the issues like scalability, fault tolerance, communication overhead associated with existing in-memory system. Hence, implementing Big Graph in Dr. Hadoop framework is future research agendas.

Bloom Filter. Bloom Filter [2] is a probabilistic data structure for approximate query. Bloom Filter requires a tiny on-chip memory to store information of a large set of data. A few modern Big Graph engine deploys Bloom Filter to reduce the on-chip memory requirement. For instance, ABySS [18]. The DNA assembly requires very large size of RAM. Therefore, DNA assembler deploys Bloom Filter for faster processing with low sized RAM. In biological graph such as a de Bruijn graph, Bloom Filter is commonly used to increase it efficiency, for instance, deBGR [28]. [32] uses cascading Bloom Filters to store the nodes of the graph. It reduces construction time of the graph. Gollapudi et al. [14] proposed a Bloom Filter based HITS-like ranking algorithm. Bloom Filter helps in reducing the query time. Similarly, Najork et al. [25] proposed a Bloom Filter based query reduction technique to increase the performance of SALSA. Bloom Filter is used to approximate the neighborhood graph. Bloom Filter has great potential for its implementation in the Big Graph. Bloom Filter is a simple and dumb data structure. However, these two are the parameters for its efficiency.

Its simple data structure makes it space and time efficient. It's dumbness makes its applicable in any field. Regardless, new variants of Bloom Filter are required which are able to store the relationship among the nodes. Such variants help in checking the relation among the nodes and reduces the complexity of Big Graph processing.

6.2 Compute-Intensive Big Graph

In contrast to memory-intensive computation, compute-intensive tasks require more processing capabilities. Nowadays, data size is growing exponentially. However, the decline in the growth rate is expected in near future. Therefore, the future Big Graph will become compute-intensive task. A few research work has been carried out on Big Graph learning. There are numerous machine learning algorithms to evaluate the learning capability on Big Graph data. Hierarchical Anchor Graph Regularization (HAGR) [41] for instance. Deep learning is another example of extreme learning which requires a huge computation capability [36]. Also, machine learning is deployed in spatio-temporal networks [43].

7 Conclusion

Big Graphs are interdependent among each subgraphs. Whole Big Graph cannot be stored in RAM. However, storing whole Big Graph in RAM extremely boosts up the performance. There is future research scope in building in-memory Big Graph database without using HDD/SSD. Big Graph helps in representing the relationship between entities in Big Data. Furthermore, in-memory Big Graph boosts up the system performance. Because, RAM is about 100$\times$ faster than SSD/Flash-based Big Graph representation. Moreover, in-memory Big Graphs are nearly 1000$\times$ faster than HDD-based representations. In-memory Big Graphs are capable of storing trillions of nodes and edges by using other techniques such as Bloom Filter. Bloom Filter helps in eliminating the duplication in Big Graph. In addition, in-memory Big Graphs have to use scalable framework to increase its storage capacity beyond terabytes. For instance, Dr. Hadoop has infinite scalability and many merits including fault-tolerant, and load balancing. Many in-memory Big Graph techniques are proposed which are discussed in the paper. However, one big challenge in in-memory Big Graph is RAM. RAM is very costly and using a large size is impractical in current scenario. But high performance can be achieved through in-memory representations of Big Graph. The trade-off between cost and performance creates difference among HDD-based, SSD/Flashed-based and in-memory based representation of Big Graph. The choice depends on the priority of the applications. Most of the applications require high performance. Hence, in-memory Big Graph is near future, since RAM cost is dropping.

References

1. Backstrom, L., Huttenlocher, D., Kleinberg, J., Lan, X.: Group formation in large social networks: membership, growth, and evolution. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 44–54. ACM (2006)
2. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
3. Boldi, P., Rosa, M., Santini, M., Vigna, S.: Layered label propagation: a multiresolution coordinate-free ordering for compressing social networks. In: Proceedings of the 20th International Conference on World Wide Web, pp. 587–596. ACM (2011)
4. Boldi, P., Vigna, S.: The webgraph framework I: compression techniques. In: Proceedings of the 13th International Conference on World Wide Web, pp. 595–602. ACM (2004)
5. Borkar, V., Carey, M., Grover, R., Onose, N., Vernica, R.: Hyracks: a flexible and extensible foundation for data-intensive computing. In: Proceedings of the 2011 IEEE 27th International Conference on Data Engineering, ICDE 2011, pp. 1151–1162. IEEE Computer Society (2011)
6. Bu, Y., Borkar, V., Jia, J., Carey, M.J., Condie, T.: Pregelix: Big(ger) graph analytics on a dataflow engine. Proc. VLDB Endow. **8**(2), 161–172 (2014). https://doi.org/10.14778/2735471.2735477
7. Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., Schulz, C.: Recent advances in graph partitioning. In: Kliemann, L., Sanders, P. (eds.) Algorithm Engineering. LNCS, vol. 9220, pp. 117–158. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49487-6_4
8. Carletti, V., Foggia, P., Greco, A., Saggese, A., Vento, M.: Comparing performance of graph matching algorithms on huge graphs. Pattern Recognit. Lett. (2018)
9. Chen, C., Yan, X., Zhu, F., Han, J., Philip, S.Y.: Graph OLAP: towards online analytical processing on graphs. In: Eighth IEEE International Conference on Data Mining, ICDM 2008, pp. 103–112. IEEE (2008)
10. Cheng, J., Ke, Y., Chu, S., Cheng, C.: Efficient processing of distance queries in large graphs: a vertex cover approach. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp. 457–468. ACM (2012)
11. Dev, D., Patgiri, R.: Dr. Hadoop: an infinite scalable metadata management for Hadoop–How the baby elephant becomes immortal. Front. Inf. Technol. Electron. Eng. **17**(1), 15–31 (2016). https://doi.org/10.1631/FITEE.1500015
12. Dong, X., et al.: Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2014, pp. 601–610. ACM (2014)
13. Gao, W., Wu, H., Siddiqui, M.K., Baig, A.Q.: Study of biological networks using graph theory. Saudi J. Biol. Sci. **25**, 1212–1219 (2017)
14. Gollapudi, S., Najork, M., Panigrahy, R.: Using bloom filters to speed up HITS-like ranking algorithms. In: Bonato, A., Chung, F.R.K. (eds.) WAW 2007. LNCS, vol. 4863, pp. 195–201. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77004-6_16
15. Gonzalez, J.E., Low, Y., Gu, H., Bickson, D., Guestrin, C.: PowerGraph: distributed graph-parallel computation on natural graphs. In: Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation, OSDI 2012, pp. 17–30. USENIX Association (2012)

16. Gonzalez, J.E., Xin, R.S., Dave, A., Crankshaw, D., Franklin, M.J., Stoica, I.: Graphx: graph processing in a distributed dataflow framework. In: OSDI, vol. 14, pp. 599–613 (2014)
17. Gregor, D., Willcock, J., Lumsdaine, A.: Compressed sparse row graph. https://www.boost.org/doc/libs/1_57_0/libs/graph/doc/compressed_sparse_row.html. Accessed 21 June 2018
18. Jackman, S.D., et al.: Abyss 2.0: resource-efficient assembly of large genomes using a Bloom filter. Genome Res. **27**, 768–777 (2017). https://doi.org/10.1101/gr.214346.116
19. Kui, X., Samanta, A., Zhu, X., Li, Y., Zhang, S., Hui, P.: Energy-aware temporal reachability graphs for time-varying mobile opportunistic networks. IEEE Trans. Veh. Technol. **67**, 9831–9844 (2018). https://doi.org/10.1109/TVT.2018.2854832
20. Kwak, H., Lee, C., Park, H., Moon, S.: What is Twitter, a social network or a news media? In: Proceedings of the 19th International Conference on World Wide Web, pp. 591–600. ACM (2010)
21. Leskovec, J.: Stanford network analysis project. http://snap.stanford.edu/. Accessed 22 June 2018
22. Leskovec, J., Perez, Y., Sosic, R.: Snap datasets. http://snap.stanford.edu/ringo/. Accessed 20 June 2018
23. Myers, S.A., Sharma, A., Gupta, P., Lin, J.: Information network or social network?: the structure of the Twitter follow graph. In: Proceedings of the 23rd International Conference on World Wide Web, pp. 493–498. ACM (2014)
24. Nai, L., Xia, Y., Tanase, I.G., Kim, H., Lin, C.Y.: GraphBIG: understanding graph computing in the context of industrial solutions. In: SC15: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–12 (2015). https://doi.org/10.1145/2807591.2807626
25. Najork, M., Gollapudi, S., Panigrahy, R.: Less is more: sampling the neighborhood graph makes salsa better and faster. In: Proceedings of the Second ACM International Conference on Web Search and Data Mining, pp. 242–251. ACM (2009)
26. Nayak, S., Patgiri, R.: Dr. Hadoop: in search of a needle in a Haystack. In: Fahrnberger, G., Gopinathan, S., Parida, L. (eds.) ICDCIT 2019. LNCS, vol. 11319, pp. 99–107. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-05366-6_8
27. Nickel, M., Murphy, K., Tresp, V., Gabrilovich, E.: A review of relational machine learning for knowledge graphs. Proc. IEEE **104**(1), 11–33 (2016)
28. Pandey, P., Bender, M.A., Johnson, R., et al.: deBGR: an efficient and near-exact representation of the weighted de Bruijn graph. Bioinformatics **33**(14), i133–i141 (2017)
29. Paranjape, A., Benson, A.R., Leskovec, J.: Motifs in temporal networks. In: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, pp. 601–610. ACM (2017)
30. Patgiri, R., Nayak, S., Dev, D., Borgohain, S.K.: Dr. Hadoop cures in-memory data replication system. In: 6th International Conference on Advanced Computing, Networking, and Informatics, 04–06 June 2018 (2018)
31. Perez, Y., et al.: Ringo: interactive graph analytics on big-memory machines. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD 2015, pp. 1105–1110. ACM (2015). https://doi.org/10.1145/2723372.2735369
32. Salikhov, K., Sacomoto, G., Kucherov, G.: Using cascading bloom filters to improve the memory usage for de Brujin graphs. Algorithms Mol. Biol. **9**(1), 2 (2014)

33. Shao, B., Wang, H., Li, Y.: Trinity: a distributed graph engine on a memory cloud. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, pp. 505–516. ACM (2013). https://doi.org/10.1145/2463676.2467799
34. Sun, P., Wen, Y., Duong, T.N.B., Xiao, X.: GraphH: high performance big graph analytics in small clusters. In: 2017 IEEE International Conference on Cluster Computing (CLUSTER), pp. 256–266. IEEE (2017)
35. Sun, P., Wen, Y., Duong, T.N.B., Xiao, X.: GraphMP: an efficient semi-external-memory big graph processing system on a single machine. In: 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS), pp. 276–283. IEEE (2017)
36. Sun, Y., Li, B., Yuan, Y., Bi, X., Zhao, X., Wang, G.: Big graph classification frameworks based on extreme learning machine. Neurocomputing **330**, 317–327 (2019). https://doi.org/10.1016/j.neucom.2018.11.035
37. Tabaja, A.: Yahoo!webscope program. https://webscope.sandbox.yahoo.com/. Accessed 20 June 2018
38. Tian, Y., Balmin, A., Corsten, S.A., Tatikonda, S., McPherson, J.: From "think like a vertex" to "think like a graph". Proc. VLDB Endow. **7**(3), 193–204 (2013). https://doi.org/10.14778/2732232.2732238
39. Ugander, J., Karrer, B., Backstrom, L., Marlow, C.: The anatomy of the facebook social graph. arXiv preprint arXiv:1111.4503 (2011)
40. Wang, D., Pedreschi, D., Song, C., Giannotti, F., Barabasi, A.L.: Human mobility, social ties, and link prediction. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1100–1108. ACM (2011)
41. Wang, M., Fu, W., Hao, S., Liu, H., Wu, X.: Learning on big graph: label inference and regularization with anchor hierarchy. IEEE Trans. Knowl. Data Eng. **29**(5), 1101–1114 (2017). https://doi.org/10.1109/TKDE.2017.2654445
42. Yan, D., Bu, Y., Tian, Y., Deshpande, A., Cheng, J.: Big graph analytics systems. In: Proceedings of the 2016 International Conference on Management of Data, pp. 2241–2243. ACM (2016)
43. Yu, B., Yin, H., Zhu, Z.: Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18), pp. 3634–3640 (2017)
44. Zaharia, M., et al.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, p. 2. USENIX Association (2012)
45. Zheng, D., Mhembere, D., Lyzinski, V., Vogelstein, J.T., Priebe, C.E., Burns, R.: Semi-external memory sparse matrix multiplication for billion-node graphs. IEEE Trans. Parallel Distrib. Syst. **28**(5), 1470–1483 (2017)