Techniques, Tools and Applications of Graph Analytic

Faiza Ameer[1], Muhammad Kashif Hanif[2], Ramzan Talib[3], Muhammad Umer Sarwar[4],
Zahid Khan[5], Khawar Zulfiqar[6], Ahmad Riasat[7]
Department of Computer Science,
Government College University,
Faisalabad, Pakistan

***Abstract*—Graphs have acute significance because of polytropic nature and have wide spread real world big data applications, e.g., search engines, social media, knowledge discovery, network systems, etc. Major challenge is to develop efficient systems to store, process and analyze large graphs generated by these applications. Graph analytic is important research area in big data graphs dealing with efficient extraction of useful knowledge and interesting patterns from rapidly growing big data streams. Tremendously huge and complex data of graph applications requires specially designed graph databases having special data structures and effective features for data modeling and querying. The manipulation of large size of data requires effective scalable and distributed computational techniques for efficient graph partitioning and communication. Researchers have proposed different analytical techniques, storage structures, and processing models. This study provides insight of different graph analytical techniques and compares existing graph storage and computational technologies. This work also assesses the performance, strengths and limitations of various graph databases and processing models.**

***Keywords*—Graph; graph analytic; big data; graph tools; analytical techniques**

I. Introduction

Graphs have astute magnitude due to their versatile and expressive nature. Real world big data problems like weather forecasting, geographical changes, large network systems, social networks, semantic search and knowledge discovery, text mining, IOT, cyber security, etc. all can be viewed and modeled as graphs.

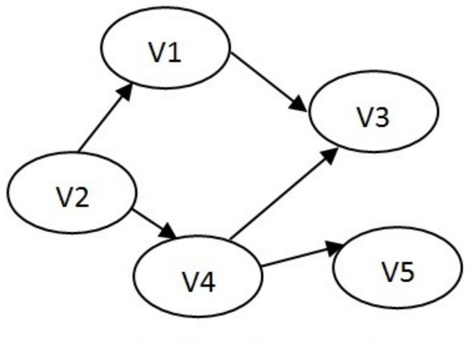
Graphs can be used to represent and analyze big data. Big data is huge volume, high velocity and a large variety of information asset that demands cost effective, and innovative forms of information processing for enhanced insight and decision making [1]. The term huge volume refers to the large size of static or continuously growing data like Facebook, Twitter, Google, etc. High Velocity represents the required speed of data generation, and analysis. Large variety means the use of various types of structured (e.g., data from relational databases), semi-structured (e.g., XML and JSON documents), and unstructured data (e.g., video, audio, images etc.) [1].

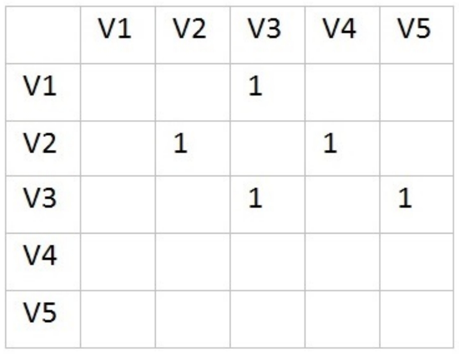
Graph analytic is based on graph theory (a branch of Mathematics). Graph theory was born out of a very practical urban planning problem. The problem started in Konigsberg (old city in Russia). The city had two large islands, connected by seven bridges. Back in 1736, the problem was to device a walkway from one part of city to another by traversing all seven bridges only once. A mathematician, named Euler, proved mathematically that this problem had no solution due to odd number of bridges. He reformulated the problem and solved it by eliminating all features except land masses (termed as vertex) and the connecting bridges (termed as edge). The resulting mathematical structure was called a graph (Fig. 1). By solving this problem, Euler laid down the foundations of whole field of graph theory [2]. Different operations which can be performed on a graph are add/remove a vertex, add/remove an edge, or find the nearest neighbors (i.e., finding nodes connected to the vertex), etc.

Graph analytic models large and complex data problems as a set of graphs. It expresses relationship patterns of objects by exploiting the mathematical properties of data and statistical modeling techniques to provide efficient algorithmic solutions and discover meaningful patterns [2]. Many organizations are competing with their peers in market using graph analytic by making accurate and timely decisions [2]. There exists variant techniques for graph analytic like path analytic,connectivity analytic, centrality analytic, and community analytic based on solution to different types of problems [2]. Each one of them use different principles and methods to answer divergent analytical questions.

To handle large graphs, new systems incorporate efficient storage and processing technologies. The new database technologies fulfill the growing requirements of current applications and cover the limitations of traditional database models. Modern graph related database management system includes graph databases and graph stores [3]. These databases provide general features for data storage, data modeling, and support for graph queries and query languages. Use of graphs in big data have also generated much interest in the field of large-scale graph data processing. Modern parallel processing systems are based on four different processing models: MPI-Like, Map-Reduce, BSP, and Vertex-Centric Graph Processing. Pregel, Giraph, GPS, Mizan, and GraphLab are important parallel processing models.

The remainder of the paper is organized in different sections. Section II gives a brief overview of the applications of graphs. Section III describes different graph analytic techniques. In Section IV, graph storage techniques are discussed. Sections V provides different processing models for large graphs. At the end, we conclude the outcomes.





	V1	V2	V3	V4	V5
V1			1		
V2		1		1	
V3			1		1
V4					
V5					

Fig. 1. Sample graph and its representation using adjacency matrix.

II. Some Applications/Use Cases of Graphs

A. Social Media

A Facebook page contains some elements, e.g., primary users, friends, groups, and posts. The posts may contain text, tags, and media such as images and videos. Some people comment and like posts. The commenter must be a Facebook user. Other people may also like or respond to some of the comments. Many posts have locations associated with them. If all of these are considered together, graph can be made to visualize almost the same types of information. In graphs, everything can be organized in terms of objects and relationship of these objects (Fig. 2).

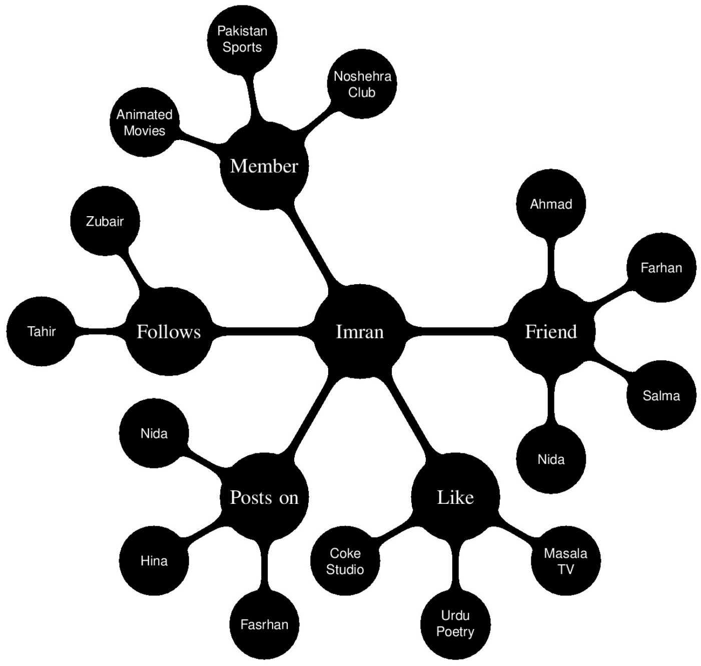


Fig. 2. An example of social media graph.

Graphs can helps to answer different questions [2]. For example,

- Behavioral psychologists, want to know, that a war game user or any other violent game user, also shows violent behavior in online fighting or not?
- Following somebody's post over time and applying text mining techniques, it can be investigated that person is addicted to the game or not.
- Are there groups? What are these groups? Who are the most influential users in group? Which people are referred to, listened by everybody? Are the players interacting with other players? Where they belong to? Who are they? Do they form a close community? (Fig. 3).

B. Analysis and Planning of Smart Cities

Cities have multiple interacting networks including transportation networks, water and sewage networks, power transmission networks, broadband IP, and M2M networks. Each network has multiple subtypes, e.g., transportation networks include the bus networks, the subway network, and the railway network, etc. These networks can be represented as graphs where each node has a geographical coordinate (Fig. 4).

A city planner would like to make sure that he covers the entire city with optimized traversal time and plan traffic congestion. To accomplish this, he would need to create a network model. During planning, he needs to make sure that all the right things happen at the same place. For example, people coming out of metro stations will find nearby businesses, IP network, and water supply. Network models also represent congestion, people's behavior, materials behavior, and energy use patterns for network.

C. Fraud Detection

Traditional fraud prevention measures focused on discrete data points such as specific accounts, individuals, devices or IP addresses. Sophisticated fraudsters use stolen and synthetic identities to escape detection. Graph databases (e.g., Neoj4) are used beyond individual data points to the connections that link these fraud rings. They uncover difficult to detect patterns in real time as well as with high accuracy [6]. Taking example of bank, following are some common fraud patterns:

- A group containing two or more people organize a fraud ring.

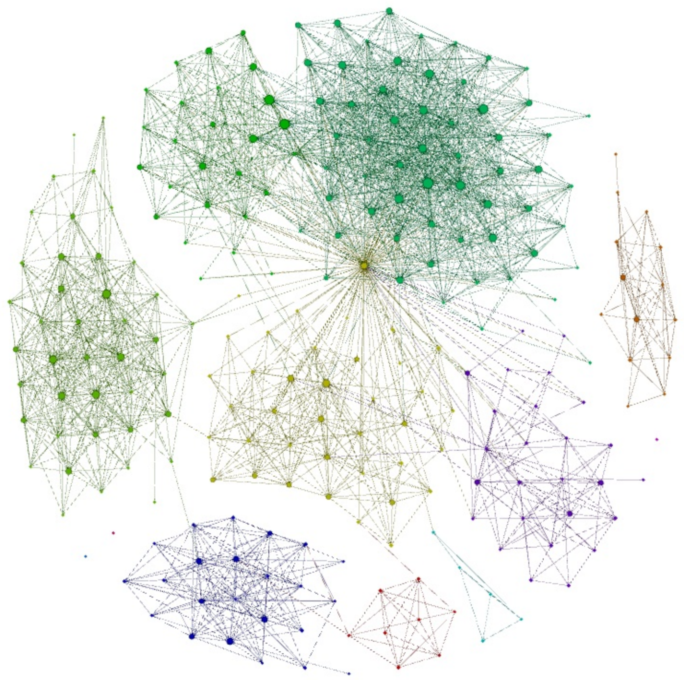


Fig. 3. Visualizing groups in graph [4].

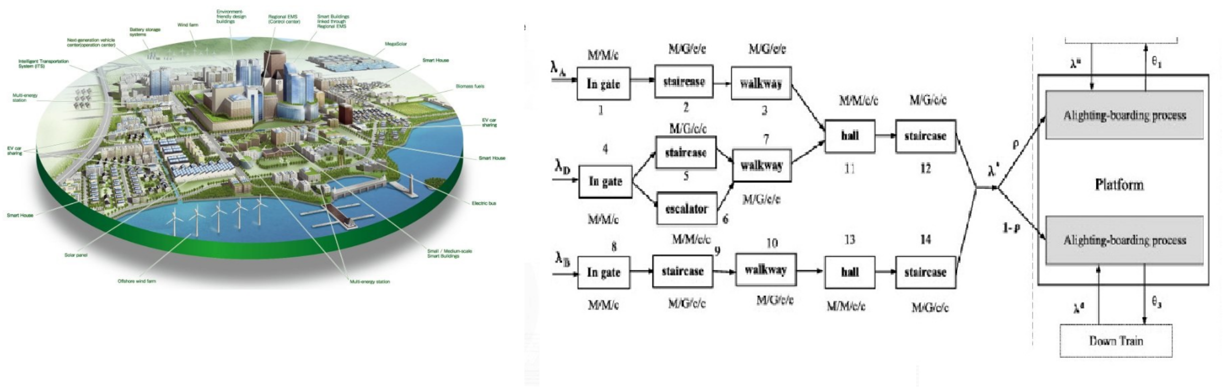


Fig. 4. Graph in planning a smart city [5].

- These people open accounts using fictional identities with subset of shared contact information, i.e., phone numbers and addresses.
- New accounts are added to the original unsecured credit lines, credit cards, overdraft protection, and personal loans.
- Whenbanks increase the revolving credit lines, the ring breaks in, coordinate their activity, max out all of their credit lines, and disappear.
- Sometimes, fraud rings bring all of their balances to zero using fake cheques immediately before the prior step, doubling the damage.
- Even after identification of fraud ring, agents are never able to reach the fraudster in real time.
- The due debt is written off without paying due amount.

Through graph database, bank's existing fraud detection infrastructure can be implied to support ring detection during key stages in the customer and account life cycle. Real time graph analysis can help banks identify probable fraud rings (Fig. 5).

III. Graph Analytic Techniques

The techniques for graph analytic include path analytic, connectivity analytic, community analytic and centrality analytic.

A. Path Analytic

Path analytic deals with finding the best path between two given nodes. Specification of "best" may include optimization of specific function, traversal of certain nodes/edges, avoidance of nodes/edges, and satisfaction of some preferences. For example,

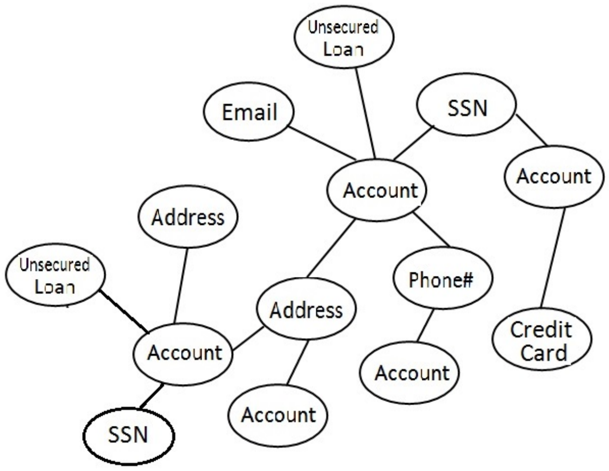


Fig. 5. Graphs in bank fraud detection.

In Google maps, shortest route would change based on weather, traffic, and road conditions.

A standard method for finding least weighted path is Dijkstra's algorithm. Complexity for applying Dijkstra's algorithm for big data is very high. The worst case complexity of Dijkstra is proportional to the number of edges times log (number of nodes). For 1 million nodes and 10 million edges, the worst case complexity is proportional to approximately 14 million [2]. There are several modification of Dijkstra's algorithm to enhance the performance, e.g., Bi-directional Dijkstra Algorithm, Goal-directed Dijkstra Algorithm, etc.

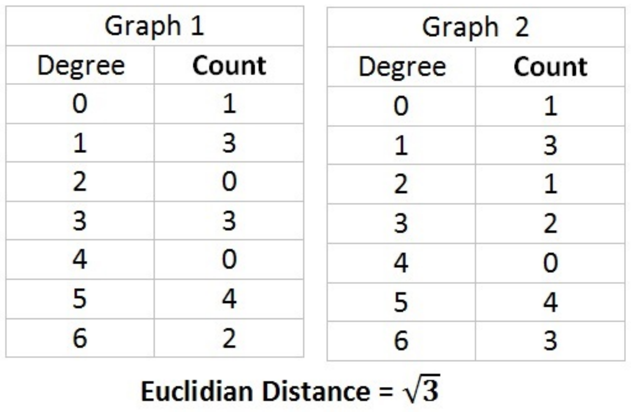
B. Connectivity Analytic

Connectivity analytic explores the connectivity pattern and similarity between structures of graphs based on different features. It also discovers robustness, i.e., which node should be the next target of attacker.

Degree of a node is the number of edges connected to a node. Through degree of nodes, we can specify if a node is more connected than another. Degree of the node can be calculated by adding in-degree (incoming edges) and out-degree (outgoing edges) of the node. Similarity of the graph can be found by comparing degree histogram of the graphs and calculating vector distance of histogram. Degree histogram contains number of nodes against degree value of node. Graph similarity can be calculated by Euclidean distance (Fig. 6) using the vector distance function given in Equation (1).

$$D = \sqrt{\sum_{i=0}^{k} (h_{1i} - h_{2i})^2} \tag{1}$$

There are many other sophisticated methods/formulas available to compare graph similarities, e.g., Hellinger distance, Histogram intersection etc. A joint, two dimensional colorful histogram of the graph, provides more insight about the graph [2]. For example, in a social networking graph, the nodes with more incident edges than outgoing edges represent members who are listeners.



Graph 1	
Degree	Count
0	1
1	3
2	0
3	3
4	0
5	4
6	2

Graph 2	
Degree	Count
0	1
1	3
2	1
3	2
4	0
5	4
6	3

Euclidian Distance = $\sqrt{3}$

Fig. 6. Using histogram for finding graphs similarity.

For measuring robustness of the network, we have to evaluate how much a structure is affected by an attack. It can be done using Weighted Spectral Distribution (WSD). WSD emphasizes the contribution of eigenvalues believed to be important [7].

C. Community Analytic

A cluster of nodes which are more connected to inside of the cluster than outside of the cluster is called community (Fig. 7). Community analytic deals with detection and behavior pattern of communities. For example, who are member of community? From where they belong to? Is the community stable? Dominant members in the community? Is community evolving, growing, splitting or going to be dead? Internal degree of a sub graph is the summation of edges of all vertices within the cluster. Summation of edges of all vertices outside the sub graph is called external degree of cluster.

Communities are found by comparing internal and external degrees of clusters. A cluster having more internal density (Equation (2)) than external density (Equation (3)) is called community.

$$\delta_{int} = \frac{\#\ of\ internal\ edges\ in\ C}{n_c(nC-1)/2} \tag{2}$$

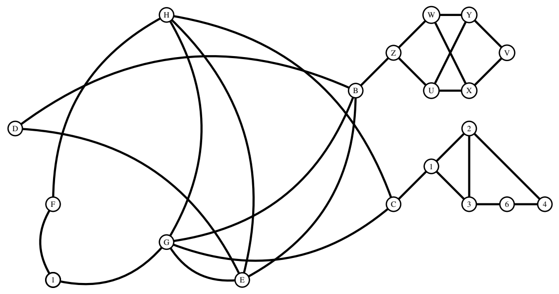


Fig. 7. Communities

$$\delta_{ext} = \frac{\#\ of\ inter\ cluster\ edges\ of\ C}{n_c(n - nC)} \quad (3)$$

A community always have higher internal density than external density. A community have different local properties (i.e., clique, n-clique, n-clan, and k-core) and a global property which is called modularity. Modularity is a global property of graph. It estimates the quality of the cluster. Louvain community detection is one of methods of detecting community and finding modularity of network/community. Clique is a perfect community where every vertex is connected to every other vertex in the cluster. Finding cliques is computationally very hard problem as compared to finding k-size cliques (Fig. 8). n-clique and n-clan are distance based measures while k-core is a density based method of finding community. n-clique is a sub graph having distance of length not more than n between each member. n-clan is an n-clique having distance between all nodes not more than n and without involving outsider nodes. k-core is a cluster in which each vertex is directly connected to at least k other vertices of the cluster.

D. Centrality Analytic

Centrality analytic characterizes important nodes (influencers) of a network with respect to a specific analysis problem. Their significance is detected by looking at how central they are in a community. We know that every node does not have equal importance in a network. Some nodes are more important than others in specific perspectives. For example, a central server in a computer network, a junction station in a transport network etc. We need to find the nodes which are maximally connected to other nodes and the nodes if removed would maximally disturb the communication of other nodes. Network centrality has centrality and centralization characteristics. Centrality is the measure of importance of a node (or edge) based on its position in the network. While, centralization is the measure for a network (not just a single node). If more nodes start having higher centrality, then there is less variation in the centrality values of the network. As a result, the centralization of the network drops (Fig. 9).

Network centralization is the sum of the difference between the maximum centrality and the centrality of the node divided by the maximum centrality (Equation (4)).

$$Centralization = \frac{\sum(c_{max} - c(v_i))}{c_{max}} \quad (4)$$

There are many types of centrality including degree centrality, closeness centrality, betweenness centrality, eigenvector centrality, and katz centrality. There are different principles and methods to calculate centrality values.

IV. Graph Storage Techniques

As the size increases, more edges stream into graphs adding more data into database and making real time analysis a challenge. These graphs may be stored both in RDBMS or NoSQL for efficient query processing using supportive query languages. Currently, NoSQL databases are schema less databases and are getting more attention due to high scalability and fault-tolerance. NoSQL databases can have key/value stores (e.g., Apache Cassandra), document stores (e.g. MongoDB), and graph databases (e.g., AllegroGraph, Neo4J, OpenLink Virtuoso) [8]. There are two types of database management system for graphs, i.e., graph stores and graph databases.

Graph stores provide facilities for storing and querying graphs. G-Store is a storage manager for large vertex-labeled graphs. Redis graph is a Python implementation for storing graphs. VertexDB implements a graph store on top of TokyoCabinet (a B-tree key/value disk store). Filament is a graph storage library with default support for SQL through JDB. Additionally, we can mention CloudGraph, Horton, and Trinity as prototypes of graph stores [3].

Graph databases can have better speedups over relational databases for selected problems. For example, graph queries formulated in terms of paths can be concise and intuitive [8]. Graph databases must provide database languages (for data definition, manipulation and querying), query optimizer, database engine, external interfaces, storage engine transaction engine, management, and operation features (tuning, backup, recovery, etc.). Some databases available according to above criteria are DEX (Sparksee), AllegroGraph, InfiniteGraph, HypergraphDB, Neo4J, and Sones.

DEX (Spakrsee) is a high performance graph database management system based on bitmaps and other secondary structures. DEX works efficient for the manipulation of very large graphs. DEX uses a java library for management of graphs [9]. AllegroGraph was initially developed as graph database and meets the semantic web standards (i.e., RDF/S, SPARQL and OW) [3]. AllegroGraph also provides special features for geo-temporal reasoning and social network analysis.

InfiniteGraph is an object-oriented database to support large-scale graphs in a distributed environment [3]. It provides the efficient traversal of relations across massive and distributed data stores. HyperGraphDB is highly customizable system based on hypergraph database model. It represents complex and large scale domain specific knowledge within uniform conceptual framework. It removes usual difficulties while dealing with higher order relationships [10]. It is useful for modeling data of complex applications like artificial intelligence, bio-informatics, and natural language processing. Most graph database models supports different features like graph structures, data definition and manipulation, storage, essential graph queries, basic integrity constraints, and representation of entities and relationships [3].

V. Computational Models for Graph Processing

A parallel computational model abstractly specifies the way a parallel program will run. In parallel computational model, a program is divided into multiple concurrently running processes. It decides how and when these processes communicate and exchange data. It also observes how parallelism is actually achieved. In shared memory architecture, the memory of multiple machines is virtually considered as a single large memory. While in message passing model, the processes can directly pass messages to and from other processes or they can use a common message carrying pipe. There are also multiple ways for achieving parallelism, e.g., task parallelism,

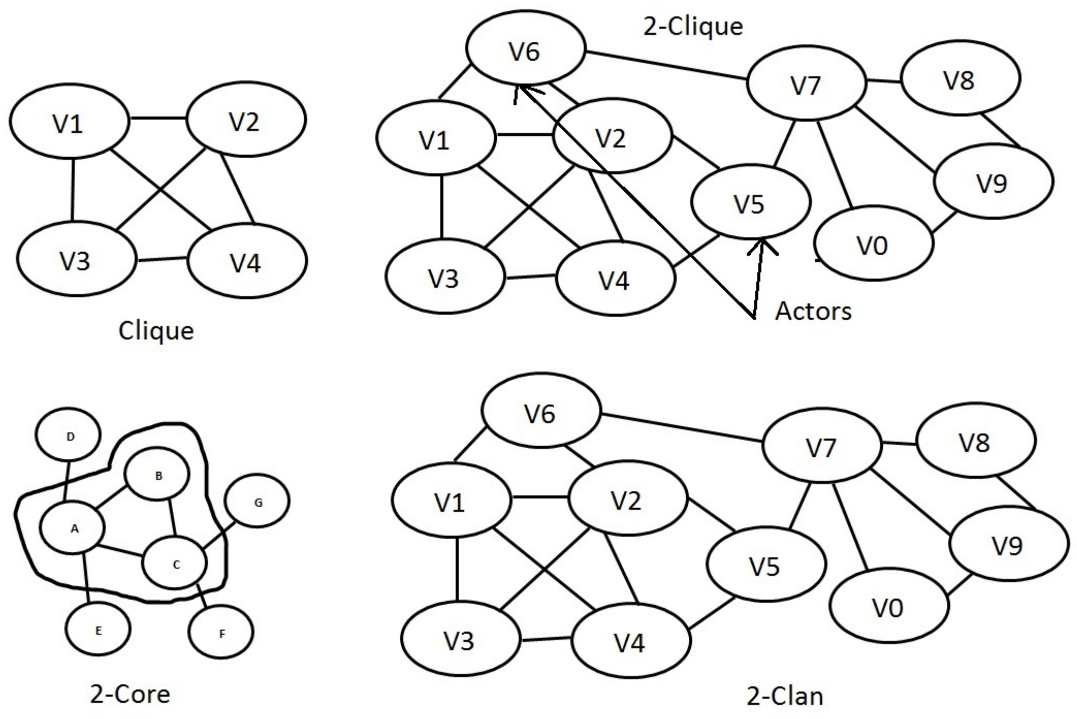


Fig. 8. Clique, n-clique, n-clan, k-core

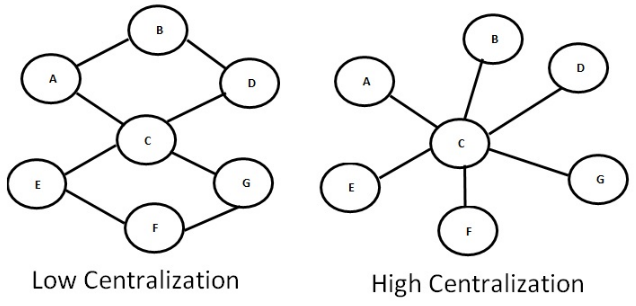


Fig. 9. Centralization of a network

data parallelism. In data parallelism, the data is divided into partitions. While in task parallelism, a single task is divided into multiple sub-tasks. In both cases, the partitions are synchronized and can run independently of each other. A computational model is independent of the programming languages.

Some of widely used techniques for implementation of big graph processing algorithms are:

- A customized distributed infrastructure can be developed [11]. It requires different implementation for different graph and its processing algorithm.
- An existing distributed computing platform can be utilized, like Map-Reduce, or SQL-like queries. However, these structures are often ill suited and lead to suboptimal performance and usability issues for graph processing [11].
- A single-computer graph algorithm library can also be utilized. JDSL [12], BGL [13], NetworkX [14], LEDA [15], Stanford GraphBase [16], or FGL [17] are example of such libraries. Major issue with libraries is scalability [11].
- An existing parallel graph system such as Parallel BGL [18] and CGMgraph [19] can be deployed. One major issue is of fault tolerance.

Next, some of the most important distributed computational models are described.

A. MPI-like

There are many libraries available that use MPI for distributed graph processing. Some important libraries are Parallel BGL [18] and CGMgraph [19]. As mentioned above, there are some important distributed graph processing issues like no support for fault tolerance [20].

B. MapReduce

MapReduce was developed by Google for processing of big data. MapReduce consists of two phases: map phase and reduce phase. The system have a master node and several of workers. In the map phase, input is divided into smaller jobs by the master node and then these are assigned to all the workers which work independently. In the reduce phase, the workers combine the result for the original problem [20]. MapReduce model did not turn out to be ideal for many different graph algorithms, e.g.

parallel BFS. It can lead to a lot of computational overhead, sub-optimal performance, and poor usability [11].

C. Bulk Synchronous Parallel

Bulk Synchronous Parallel (BSP) is a general parallel processing model which consists of three main attributes, i.e., several concurrently running processes, communication layer, and a synchronization barrier. In communication layer, pairs of processes exchange messages. BSP works in series of super steps. Processes perform computation during supersteps. Before going to next superstep, synchronization barrier maintains consistency by synchronizing processes. Synchronization may either happen periodically after a specific time interval or there may be another specific way. After completing computation assigned in each super step, every process waits for other processes to finish the processing and to receive all the messages destined to it. Once all processes complete, one superstep they get synchronized and then next superstep is executed. When a processor is not needed to compute or exchange data, it becomes inactive. It is activated when it is needed to perform computation or to exchange message. This model is terminated when all the processors become inactive. BSP is a simple, efficient, and scalable model for parallel algorithm design and analysis. It does not take into account the case of heterogeneous clusters [21].

D. Vertex-Centric Graph Processing

To apply BSP on big graphs, Google proposed a model of vertex-centric graph processing [11]. It is especially designed for big graphs. Each vertex is considered as a processor. After each superstep, the messages are synchronized, i.e., each vertex waits for other vertices to finish the processing and also to receive all the messages destined to it. Similarly, next superstep is continued and the process keeps repeating until the algorithm finishes.

The first implementation of this idea is Pregel, which is not available to the general public. Introduction of Pregel inspired many other BSP based graph processing systems like GPS (Graph Processing System) [23], Apache Giraph [24], and GraphLab [24].

1) Pregel: Pregel is flexible, scalable, and fault tolerant platform for big graph computation [11]. Similar to BSP, computation is divided into sequence of supersteps, separated by global synchronization points. This system mutates graph topology. Vertex can receive messages from its previous iteration, send/receive messages to/from other vertices, and modify its state and also the state of its outgoing edges.

Pregel system consists of one master and multiple workers. The master coordinates and oversees the activities of worker. The master divides the input directed graph. It uses an abstract API of `C++` to uniformly distribute these partitions to the workers. Then signals all the workers to start execution. The worker nodes calls the compute() method for each active vertex with the messages received from the last superstep. After completing its computation; the vertex votes for halt and becomes inactive, and is not called in the next supersteps. If this vertex gets new message then it is activated again. When all vertices become inactive, master marks the end of current job. If a worker fails, Pregel dynamically reassigns the job to other worker [11]. In this way, Pregel offers a fault tolerant system. In addition to fault tolerance, Pregel offers scalability, efficiency, ease, and simplicity. It also offers additional performance boosters like combiners and aggregators. Combiners are used during message passing (especially to a vertex on another machine). Aggregators are mechanism for global communication and monitoring of data.

2) Apache Giraph: Apache Giraph [24] is an open-source system and extends basic BSP model. Yahoo implemented using Java and built it on Hadoop echo system (Fig. 10). Giraph follows message passing model and performs global synchronization without using semaphores.

It uses Hadoop for running workers, HDFS for input and output data storage. Apache ZooKeeper is used for coordination, check pointing, and failure recovery schemes. Usability of Giraph is excellent due to Hadoop web monitoring interface. Giraph also provides shared aggregators to avoid bottlenecks at the master, yield substantial memory savings, and reduced Java garbage collection overheads. It supports different data structures for vertex adjacency lists. By default, it uses byte array for faster input loading times but graph mutation is inefficient. On the other hand, hash map edges efficient for mutation but inefficient for memory. Giraph is massively parallelizable and uses multi-threading which boosts up the speed of graph loading, dumping, and computation. Additionally, global synchronization, debugging, and monitoring progress is easy in Giraph.

3) Graph Processing System: A Graph Processing System (GPS) is open source system which is implemented in Java for computation of extremely large graphs. It offers scalability, fault tolerance, and ease of programing to express complex graphs efficiently [23]. It is faster than Giraph [24]. It has extension of the Pregel API, LALP, and DP for performance boosting [26]. It introduced master.compute() function to access, update, and store global aggregated values that are transparent to vertices. LALP divides the adjacency lists of high degree vertices among workers along with a mirror of the vertex. When such vertex broadcasts a message to its neighbors, one copy of message is sent to its mirror at each machine and all neighbors in the partition. During execution, DP dynamically repartitions the graph to balance the workload across workers. GPS minimize the thread synchronization using message buffers per-worker rather than per-vertex. However, overhead of reassigning vertices among workers can exceed the overall benefits.

4) GraphLab/PowerGraph: GraphLab is an open source project which is popular for its maturity for graph analytical tasks. Its recent version is called PowerGraph. It is implemented in `C/C++`. GraphLab adopts a gather, apply, scatter (GAS) data-pulling model and shared memory abstraction [26]. For each vertex a user-defined GAS function is implemented. In gather phase, each active vertex collects information from its neighboring vertices and edges. It then performs a generalized accumulation operation over them. In apply phase, the vertex updates its value based on old value and resultant accumulation. In scatter phase, each active vertex activates its adjacent vertices. In contrast to BSP model, vertices do not receive messages for its neighbors but can directly pull their neighbors data (via Gather). Due to this feature, the communication barriers are no longer required and execution is completely asynchronous. This

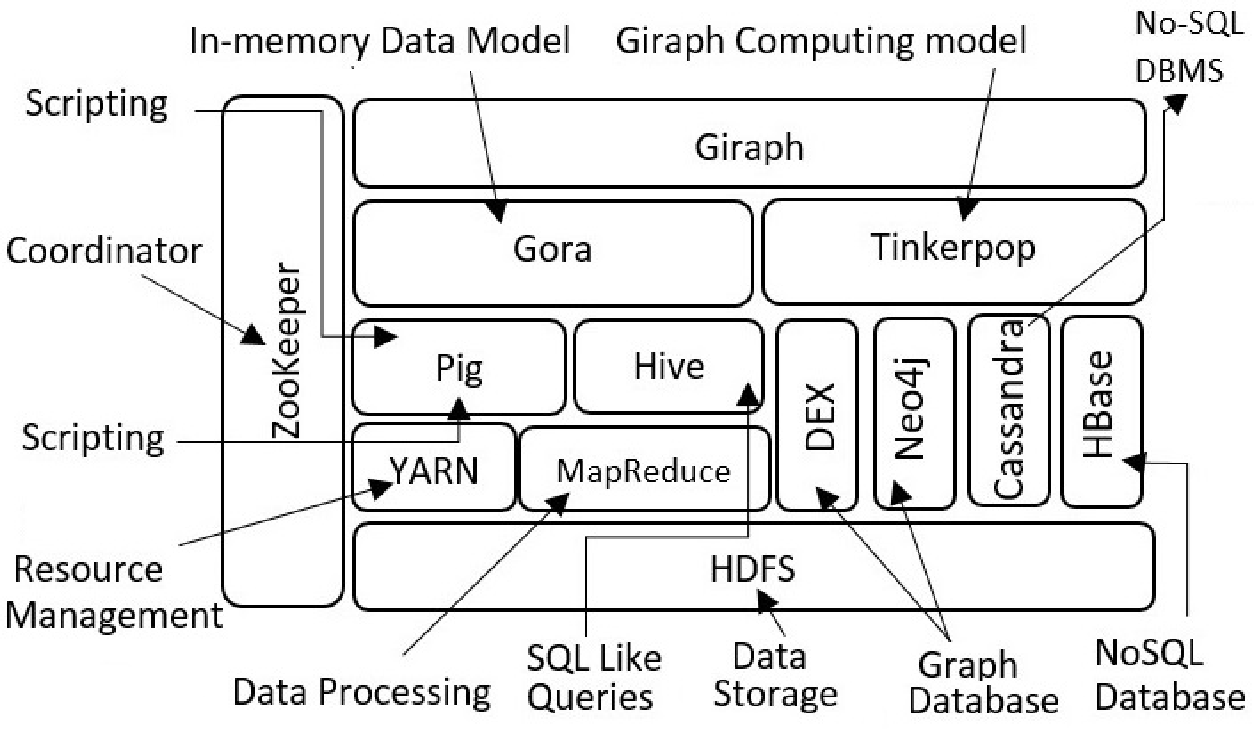


Fig. 10. Giraph and Hadoop Echo System [22].

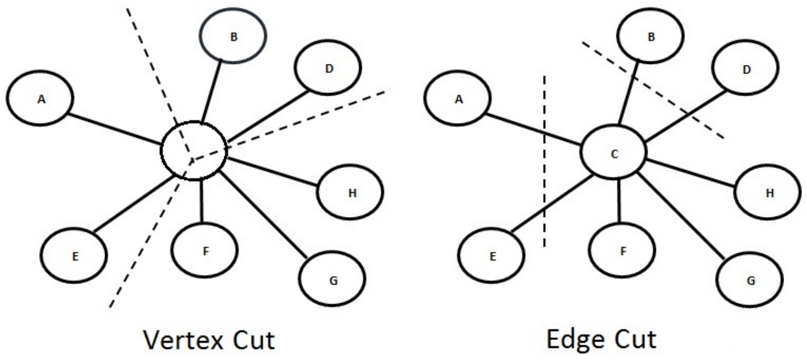


Fig. 11. Vertex Cut vs. Edge Cut.

avoids wasting CPU and network resource [24]. GraphLab also offers an optional synchronous execution mode which requires communication barriers.

GraphLab performs vertex cuts for partitioning graph while Giraph, GPS, and Mizan all perform edge cuts (Fig. 11). It replicates vertices on remote machines and each edge is assigned to a unique machine. This feature results in more balanced workloads for asymmetric degree distributions graphs [24]. In GraphLab graph mutation is partially supported. Vertices and edges can be added but cannot be removed from graphs.

5) Mizan: Mizan is a C++ optimized and open source system [26]. It monitors vertices and optimize the computation by dynamic load balancing and complex vertex migration. Mizan requires separate pre-partitioning of graph. Due to this overhead, Mizan is not suitable for large graph because it can exceed the overall benefits of the system [24]. Moreover, it has few bugs and many useful features are missing.

6) GraphX: GraphX is an embedded graph processing system which supports GraphLab and Pregel abstractions [26]. GraphX framework is built on top of Apache Spark which is a widely used distributed dataflow framework (Fig. 12).

It does not require changes to Spark. It enforces graph computation through a specific join, map, and group by dataflow pattern [27]. GraphX API is modified version of Pregel abstraction and a range of common graph operations. Through this API, graphs can be composed with unstructured and tabular data. The physical data can be represented as a

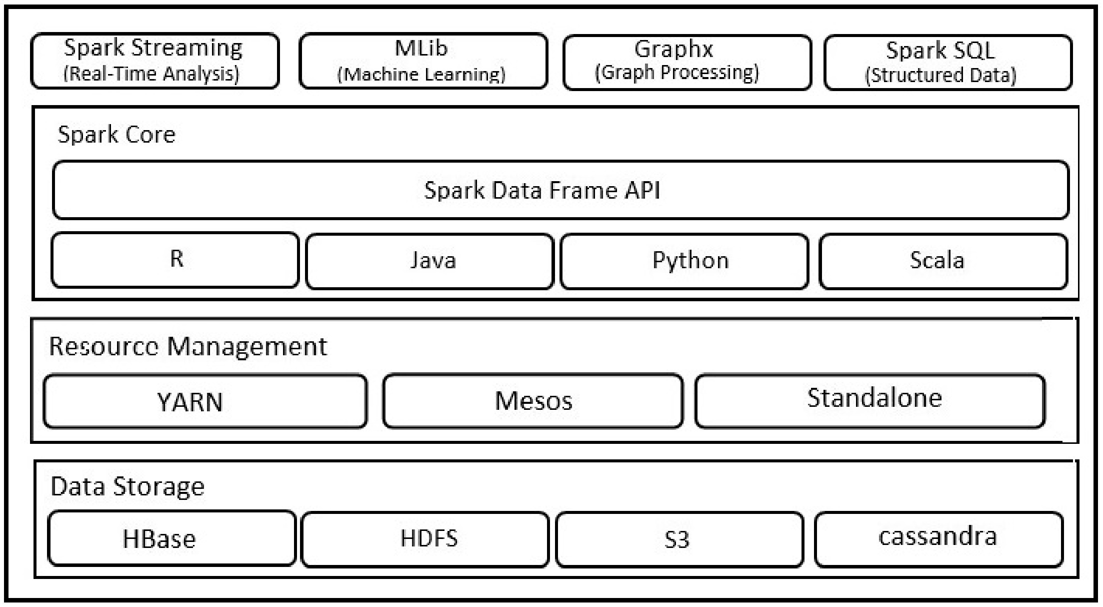


Fig. 12. Apache Spark and Graphx [25].

graph as well as collections without data movement, duplication, and sacrificing performance or flexibility [27].

GraphX optimizes distributed join and materialized view maintenance for better graph processing performance. It provides low-cost fault tolerance [27]. Its performance is comparable with specialized graph processing systems. It provides better end-to-end performance for pipelined jobs. However, it takes longer than GraphLab for actual graph computation [26].

VI. Conclusion

This work describes a broad overview of the storage techniques and algorithmic strategies for graph analytics. Moreover, we point out the critical challenges that need to be addressed. Graph databases provide an intrinsic support to data structures, data modeling, querying, and some integrity constraints features that results in faster computation. However, there is still need for further development in the definition of standard graph database languages for computation and querying of graph databases. Moreover, new query languages are needed which should be capable of expressing graph analytical operations. Big graphs requires special and faster parallel computation models which offers different features like fault tolerance, flexibility, simplicity, better usability, optimal performance, scalability, computational efficiency, and better resource utilization. We are working on security requirements of graph analytics.

References

[1] A. Gandomi and M. Haider, “Beyond the hype: Big data concepts, methods, and analytics,” *International Journal of Information Management*, vol. 35, no. 2, pp. 137–144, 2015.

[2] A. Gupta, “Graph Analytics for Big Data,” https://www.coursera.org/learn/big-data-graph-analytics, [Accessed: 2017-05-22].

[3] R. Angles, “A comparison of current graph database models,” in *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*. IEEE, 2012, pp. 171–177.

[4] “Visualizing groups in graph,” http://allthingsgraphed.com/2014/08/28/facebook-friends-network/, accessed: 2017-04-15.

[5] “smart cities,” https://futurism.com/heres-a-look-at-the-smart-cities-of-the-future/, accessed: 2017-02-12.

[6] “Financial services and neo4j: fraud detection,” https://neo4j.com/blog/financial-services-neo4j-fraud-detection/?ref=solutions, accessed: 2017-06-10.

[7] D. Fay, H. Haddadi, S. Uhlig, A. W. Moore, R. Mortier, and A. Jamakovic, “Weighted spectral distribution,” *University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-729*, 2008.

[8] C. T. Have and L. J. Jensen, “Are graph databases ready for bioinformatics?” *Bioinformatics*, vol. 29, no. 24, pp. 3107–3108, 2013.

[9] N. Martinez-Bazan, S. Gomez-Villamor, and F. Escale-Claveras, “Dex: A high-performance graph database management system,” in *Data Engineering Workshops (ICDEW), 2011 IEEE 27th International Conference on*. IEEE, 2011, pp. 124–127.

[10] B. Iordanov, “Hypergraphdb: a generalized graph database,” in *International Conference on Web-Age Information Management*. Springer, 2010, pp. 25–36.

[11] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, “Pregel: a system for large-scale graph processing,” in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 135–146.

[12] M. T. Goodrich and R. Tamassia, *Data structures and algorithms in Java*. John Wiley & Sons, 2008.

[13] J. G. Siek, L.-Q. Lee, and A. Lumsdaine, *The Boost Graph Library: User Guide and Reference Manual, Portable Documents*. Pearson Education, 2001.

[14] D. A. Schult and P. Swart, “Exploring network structure, dynamics, and function using networkx,” in *Proceedings of the 7th Python in Science Conferences (SciPy 2008)*, vol. 2008, 2008, pp. 11–16.

[15] K. Mehlhorn and S. Näher, *LEDA: a platform for combinatorial and geometric computing*. Cambridge university press, 1999.

[16] D. E. Knuth, *The Stanford GraphBase: a platform for combinatorial computing*. Addison-Wesley Reading, 1993, vol. 37.

[17] M. Erwig, “Inductive graphs and functional graph algorithms,” *Journal of Functional Programming*, vol. 11, no. 05, pp. 467–492, 2001.

[18] D. Gregor and A. Lumsdaine, “The parallel bgl: A generic library for distributed graph computations,” *Parallel Object-Oriented Scientific Computing (POOSC)*, vol. 2, pp. 1–18, 2005.

[19] A. Chan, F. Dehne, and R. Taylor, “Cgmgraph/cgmlib: Implementing and testing cgm graph algorithms on pc clusters and shared memory machines,” *The international Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 81–97, 2005.

[20] M. U. Nisar, *A Comparison of Techniques for Graph Analytics on Big Data*. uga, 2013.

[21] L. G. Valiant, “A bridging model for parallel computation,” *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[22] “Giraph and Hadoop Echo System,” https://image.slidesharecdn.com/hadoopsummit2014-140403113102-phpapp02/95/giraph-at-hadoop-summit-2014-19-638.jpg?cb=1396526440, accessed: 2017-03-12.

[23] S. Salihoglu and J. Widom, “Gps: A graph processing system,” in *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*. ACM, 2013, p. 22.

[24] M. Han, K. Daudjee, K. Ammar, M. T. Özsu, X. Wang, and T. Jin, “An experimental comparison of pregel-like graph processing systems,” *Proceedings of the VLDB Endowment*, vol. 7, no. 12, pp. 1047–1058, 2014.

[25] “Apache spark and graphx,” https://www.safaribooksonline.com/library/view/learning-spark/9781449359034/ch01.html, accessed: 2017-06-15.

[26] Y. Lu, J. Cheng, D. Yan, and H. Wu, “Large-scale distributed graph computing systems: An experimental evaluation,” *Proceedings of the VLDB Endowment*, vol. 8, no. 3, pp. 281–292, 2014.

[27] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, “Graphx: Graph processing in a distributed dataflow framework.”