Summary of Large Scale Graph Computing Models

Sheng Yang

State Key Laboratory of Mathematical Engineering and Advanced Computing, No. 62 Science Avenue, High-tech Zone, Zhengzhou City, Henan Province, China

15638573620@163.com

Keywords: Large-scale graph computing; graph processing; distributed computing.

Abstract: Large-scale graph data is a research hotspot in the processing of big data and is generally used in traffic management, analysis of social network and semantic web, etc., resulting in the emergence of a variety of large-scale graph computing platforms. How to select a more suitable graph computing platform for different tasks has become one of the hot topics in the field of graph computing. The related knowledge of graph computing models is systematically reviewed in this paper. First of all, the concept of graph computing model is outlined, including the definition and characteristics of graph computing and the development of graph computing models. Then, three kinds of graph computing models are introduced. Finally, the development process of the graph computing models is summarized, and the future research direction is prospected.

1. Introduction

Graph computing is an important branch in the field of big data processing. It has extensive theoretical research and practical application value and has been applied to solve practical problems. Graph computing, as one of the most commonly used types of abstract data structures in computer science, can effectively express the widely existing relationships between objects, especially in large-scale social networks. If we regard a user as a vertex and a friend relationship between users as a directed edge, even if we merely count the friend relationships, the Facebook social network is a gigantic graph with over one billion vertices and one hundred billion edges. Considering that Facebook is building all the physical data vertices into a mesh structure, the resulting massive network data scale has become impotent in the traditional single-machine processing, which must be processed in parallel using distributed network.

This paper systematically reviews the research progress of the large-scale graph computing platforms, which lays a foundation for further studying the theory of large-scale graph computing platform and expanding its application field. According to the computing object, they are divided into three types: vertex-centered computing model, edge-centered computing model, and subgraph-centered computing model. The vertex-centered model is further divided into three categories according to computing task scheduling methods: synchronous computing model, asynchronous computing model and hybrid computing model. The classification of computing models is shown in Figure 1. In this paper, we select the classical computing models from the graph computing systems using various computing models, introduce its iterative operation characteristics and performance, and finally summarize the development process of the graph computing model, and look forward to the future developing direction of the graph computing models.

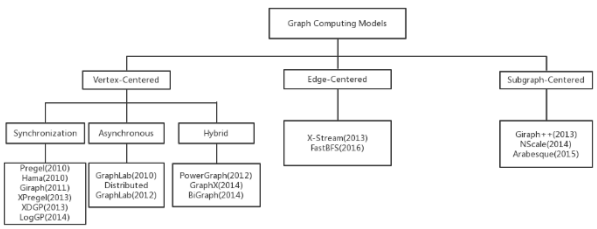

Figure 1. The classification of graph computing models

2. Overview of Large-Scale Graph Computing Models

Google proposed the earliest graph computing model based on the BSP (bulk synchronous processing) framework [1], which converts graph computations into fine-grained vertex computations. Since then, the BSP model has become the basis for various types of graph computing systems. The BSP model is shown in Figure 2. The iterative computing is divided into multiple super-step operations. One super-step operation completes one round of iteration computing. The three tasks are executed in parallel in the super-step and the data synchronization between the tasks is completed after each super-step operation. The BSP model provides a solution for the segmentation and fine-grained parallel computing of the iterative graph algorithm faced by the computing model.

Therefore, the three-step strategy (computation-communication-data synchronization), which is abstracted from BSP model, has become the classical framework of the graph computing models.

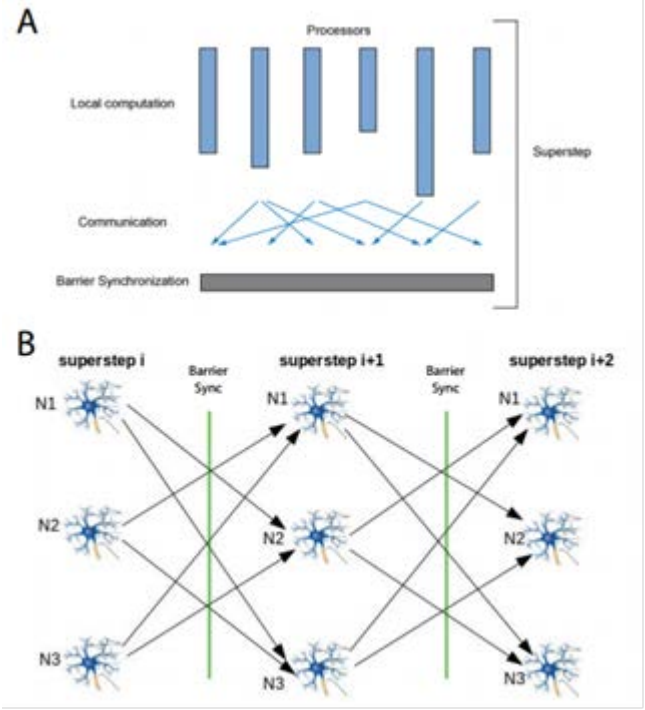

Figure 2. Three-step strategy of BSP

3. Vertex-Centered Model

Google The basic graph structure is expressed as:

$$G = (V, E, D)$$.

The vertex-centered computing model considers the in the graph as the computing center. The user can customize the update function to perform related operations on the vertices.

Before the graph computing models were proposed, the graph data analysis systems generally used the MapReduce framework, but MapReduce could not solve the problems such as frequent data partitioning and reorganization, excessive communication overhead, and limited computational parallelism caused by the frequent iterative operations of the graph computing.

In order to solve these problems, Google first proposed a vertex-centered computing model based on the BSP model in 2010. It converts the frequently iterated global calculations into multiple super-steps, and all vertices independently execute operations in parallel. The data dependencies only exist in two adjacent super-steps.

In this chapter, the vertex-centered computing models are divided into synchronous computing model, asynchronous computing model, and hybrid computing model according to the data synchronization strategy.

3.1 Synchronous Vertex-centered Model

The graph computing often needs to go through several iterations. The synchronous vertex-centered model converts each round of iterations of the graph algorithm into one super-step operation for each vertex in the graph. One super-step operation includes three steps: a) receiving the information sent by the in-neighbors of the current vertex in the last super-step; b) performing the computing function which is defined by programmers to compute the new value of the current vertex according to the received information; c) sending update information to all out-neighbors of the current vertex. After all vertices complete one super-step operation, they update the vertex information synchronously, and then enter the next super-step operation.

The synchronous vertex-centered model converts the computations in Figure 3 to the parallel computations at each vertex in Figure 4. For example, Vertex 2 first executes Step 1 to collect the update message sent by the in-neighbor Vertex 1; secondly executes Step 2 to run the computing function; and finally executes Step 3 to send the result to the out-neighbor Vertices 3 and 4. After waiting for the longest-running Vertex 4 to complete this super-step operation, global data synchronization is performed, and all vertices then enter the next round of super-step operations. The global synchronization operation allows all data to be simultaneously updated at the beginning of each super-step operation, ensuring the consistency of the calculated data.

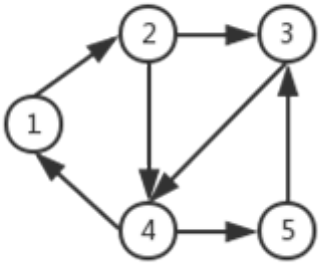

Figure 3. An example

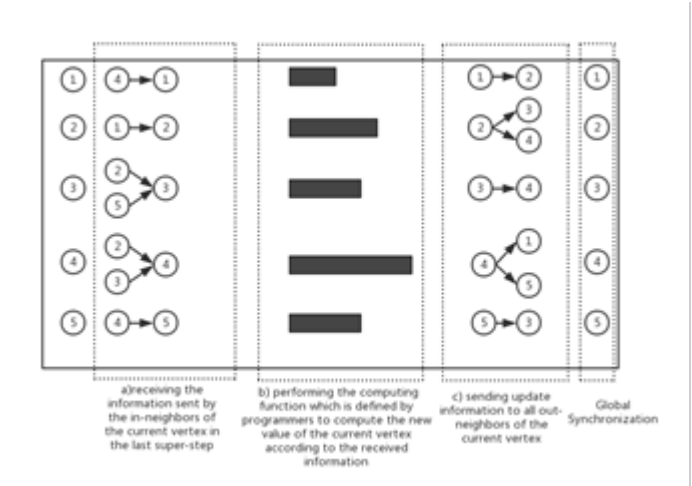

Figure 4. An example of synchronous computation

3.2 Asynchronous Vertex-centered Model

The synchronous model requires vertices that perform parallel computing to wait for the end of the slowest vertex. When the degrees of the vertices in the graph are very different, for example, a small number of vertices have a large number of edges and neighbors, which requires longer computing time and more communication overhead. The system will be limited to the slower vertices and cannot obtain the best computing efficiency. In 2010, researchers at Carnegie Mellon

University proposed GraphLab, an asynchronous graph computing system, and in 2012 released the improved system Distributed GraphLab [3].

The asynchronous computing model is the same as the iterative design of the synchronous computing, and is still the BSP three-step operation model. However, when receiving the message of the previous round of super-step computing, the updated data is no longer pushed by the neighbor vertices. It is up to the computing vertices to selectively read messages from the neighbor vertices.

In the asynchronous model, each vertex does not perform global data synchronization when performing a super-step operation, but asynchronously reads or updates the neighbor vertex's edges and information. After completing the three steps of the super-step operation, each vertex executes the next super-step operation independently.

In the asynchronous model, vertices may generate read and write accesses to vertices or edges at the same time. Therefore, in order to ensure data consistency, three schemes of consistency are proposed in the asynchronous model according to the ranges of the associated edges and neighbor vertices that each vertex can read and update asynchronously. The three schemes are: vertex consistency, edge consistency, and global consistency. The correlation edges and neighbor vertices (namely domain) that each vertex can operate in three consistency types are shown in Figure 5.

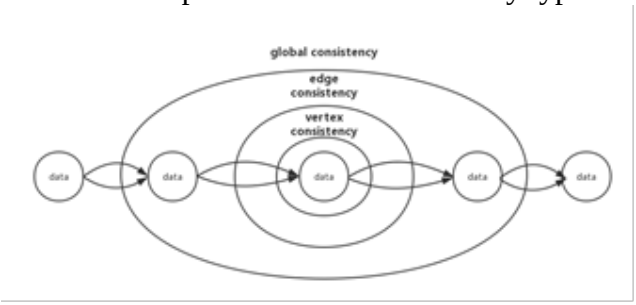

Figure 5. The operation domain of consistency schemes of asynchronous vertex-centered model.

3.3 Hybrid Vertex-centered Model

On Both synchronous and asynchronous computing models use vertices as the computing centers and edges as the information delivery paths. Therefore, the computing capacity of the two models is limited by the distribution characteristics of the vertices and edges in the graph data. The two main problems are: 1) when the number of edges is much larger than that of vertices, the communication overhead of the vertex-centered computing model will be much larger than computation overhead; 2) when the difference in the degrees of the vertices increases, the vertex with more degrees has more neighbor vertices. Larger vertices in the asynchronous model will maintain a large number of locks in order to maintain data consistency, and their neighbor vertices will have frequent locking request conflicts due to accesses to the vertex. Gonzalez et al. [4] proposed a hybrid vertex-centered model GAS to solve the above problem of graph computing. The GAS computing model follows the concept of super-step in the synchronous vertex-centered computing model, and achieves parallel computing within a single computing vertex by dividing the large vertices. As shown in the figure, GAS divides the vertex into two computing units: the right master vertex and the left mirror vertex. Each time the super-step operation is divided into three steps:

1) Gathering and summing, that is, to collect neighboring vertices and edge information of a computing vertex, and executing a user-defined function to summarize the obtained information to the master vertex;

2) Applying and updating, the master vertex performs a user-defined computing operation, and updates the mirror vertex to the computed new value.

3) Scattering, the master vertex and the mirror vertex push the update information to their respective neighbor vertices and edges.

Compared with the asynchronous vertex-centered computing model, the GAS computing model has significant improvements in both the computational parallelism and the computational speed when analyzing densely distributed graph data.

3.4 Summary

The vertex-centered model converts the common iterations in the graph algorithm into vertex computing that can be conducted in parallel, and solves the limitations of the MapReduce distributed framework. The three models have their advantages in simplicity, parallelism, and computation speed.

Compared with the asynchronous vertex-centered model and the GAS computing model, the synchronous vertex-centered model brings a long time of data synchronization waiting overhead, the computing parallelism and the computing speed are limited, but the implementation is simple and there is no need to maintain the complexity of data consistency. The strategy is therefore applied to a number of graph computing systems.

Asynchronous vertex-centered model improves the utilization of computing resources. As the scale of graph data increases, its computing advantage becomes more significant. However, the asynchronous vertex-centered model also introduces the overhead of maintaining data consistency. The data consistency strategy is complicated to implement and is prone to conflicts and errors. Therefore, most graph computing systems choose to implement synchronous and asynchronous computing models at the same time.

The GAS computing model achieves parallel computing within one vertex by dividing the vertex, and its computational parallelism is better than that of the asynchronous vertex-centered model. When there are significant differences in the degrees between vertices in the graph data of the analysis, the computational advantage of the GAS computing model is also more significant. Therefore, the GAS computing model is applied to and improved by many graph computing systems, such as BiGraph [12] and PowerLyra [13]. However, the implementation of the GAS computing model is more complex than that of the asynchronous and the synchronous models.

4. Edge-Centered Model

The vertex-centered model improves the ability of graph computing system to implement graph computing and analysis. However, it still faces some problems in practical applications: a) in order to increase the speed of random access to neighbors of computing vertices, graph computing systems generally load complete graph data into memory. Therefore, when the graph data size is too large, the vertex-centered computing model is usually implemented in a distributed system; b) when the number of edges in the graph data is much larger than that of vertices, the time overhead for information updating and distributing operations in each super-step operation will be much greater than the vertex computing time. Communication becomes the major bottleneck for graph computing, limiting the speed at which major computations are performed.

To solve the computing problems of graph data when equipment resources are limited and the number of edges is much larger than that of vertices, the EPFL instituted the edge-centered model in 2013 and applied it to the graph computing system X-Stream [5]. The edge-centered computing model constructs the graph algorithm as a flow-iteration computing on the edge list of the graph data. It completes the computing, sorting and updating in three steps at each round of iteration: a) reads the edge list stream to complete the user-defined computing operation, outputs updated information to the destination vertex list; b) the applications shuffle the list of destination vertices as the updated message flow; c) reads the updated message flow and source vertex list, and updates the source vertex value.

Three-step operations are performed sequentially in an iterative computing. The edge-centered computing model takes the edge data of the graph data as the core data structure and maintains the source vertex list. Each round of iteration of the computing operation updates the destination vertex list, which is recorded on the edge list and is calculated for each edge. The destination vertex updates the message sequence.

The edge-centered computing model converts the iterative computing of the graph algorithm into sequential execution on the edge list, avoiding the high requirements of random read-write data on memory resources, thereby solving the problem of resource limitation and communication overhead

faced by the vertex-centered computing model. The computing characteristics of the flow-sequencing algorithm in the edge-centered computing model make it possible to perform block-by-block computing on the global graph data and sequentially access the data stored on the hard disk, reducing the requirement for memory capacity of graph data analysis and computing. That means conducting large-scale graph data analysis and processing on a stand-alone machine can be achieved. Therefore, the edge-centered computing model meets the computing needs of analyzing graph data whose number of edges is much larger than that of vertices on a stand-alone system [13].

5. Subgraph-Centered Model

The vertex-centered computing model and the edge-centered computing model transform the graph algorithm into iterative computing that can be performed on vertices and edges. But at the same time, the parallelism of the graph computing is limited to the vertex and edge levels.

In order to solve the above problems, the IBM Almaden Research Center proposed a subgraph-centered computing model in the graph computing system Giraph + + [6] in 2013. In this model, the computations on the complete graph structure are transformed into iterative super-steps on multiple subgraphs. The subgraph-centered computing model forms each subgraph of each vertex and its associated edges and neighbor vertices in the original graph.

After the subgraph-centered computing model completes the graph division, iterative graph computations are performed on multiple subgraphs in parallel, and one super-step operation performs two steps: 1) the subgraph executes user-defined computing operations in parallel, and outputs the computing results; 2) subgraphs containing the same vertices update vertex information. Step b) can be performed synchronously after the operation of step a) of all subgraphs or asynchronously while preserving data consistency.

In order to prove the above analysis results, algorithms such as Unicom subgraph, PageRank, and graph clustering were tested on a data set whose graph vertex size was over a million and edge scale was over 100 million. The experimental results showed that the computing speed of the subgraph-centered computing model is 63 times faster than that of the vertex-centered computing model, and the number of iterations and communication overhead are reduced by 70% and 90% respectively compared with the vertex-centered computing model.

The subgraph-centered computing model converts the graph algorithm into iterative computations on multiple subgraphs through the subgraph division method, which successfully reduces the communication overhead and the number of iterations. Therefore, in the short time after the subgraph-centered computing model is proposed, many graph computing systems adopted the subgraph-centered computing model and made improvements to subgraph division problems, such as N Scale [11] and Arabesque [15].

The heading for subsubsections should be in Times New Roman 11-point italic with initial letters capitalized.

6. Conclusion

In order to cope with the challenges posed by the complex and ever-changing large-scale data analysis, the industry and academia have successively proposed various types of graph computing models. This paper summarizes the three types of graph calculation models, each with its own characteristics, as shown in Table 1 and Table 2.

Table 1. Table captions should be placed above the table

Graphics	Top	In-between	Bottom
Tables	End	Last	First
Figures	Good	Similar	Very well

Table 2. Comparison of graph computing models (1)

Graph Computing Systems	Task scheduling	Data partitioning	Parallelism performance	Systems
Vertex-centered	Synchronous/ Asynchronous	Vertex sequence subsets	High	Distributed/ Stand-alone
Edge-centered	Synchronous/ Asynchronous	Edge sequence subsets	Medium	Stand-alone
Subgraph-centered	Synchronous/ Asynchronous	Subgraphs	Low	Distributed/ Stand-alone

Table 2. Comparison of graph computing models (2)

Graph Computing Systems	Advantages	Disadvantages
Vertex-centered	The model is easy to implement and easy for algorithm migration. High computational parallelism can be used for synchronous or asynchronous scheduling. It is applicable to all types of algorithms.	The communication overhead between computing vertices is large, and the number of iterations required to complete the graph computation is large. The computational parallelism is limited by the data consistency.
Edge-centered	Resources required by the model are low. Data storage, chunking and read-write access are simpler. Data access is performed sequentially, making it easy to maintain data consistency.	Computational parallelism is limited by the edge list block. Graph algorithm migration is complex and the application scope is small.
Subgraph-centered	The communication overhead between the super-step operations is small, and the number of iterations to complete the graph algorithm is small.	Computational parallelism is limited. Subgraph-based data partitioning is difficult. The user-definable subgraph division is complicated.

The algorithms of various types of graphs on the vertex-centered computing model are simple to implement and are applicable to stand-alone or distributed systems. However, the system is faced with the problems of large communication overhead and difficulty in data division. The data division of the edge-centered computing model is simple and it is applicable to stand-alone systems, but the parallelism of computation is limited, and the general computing time is very long. The subgraph-centered computing model proposes the shortest time, its communication overhead is small and the number of iterative operations performed is small, but the computational parallelism is limited by the divided sub-graphs.

Although various types of computing models have been improved in terms of graph data analysis in many aspects, communication overhead, data consistency, and computational parallelism are still important issues that need to be considered in the design of computing models in the future. Data consistency and computational parallelism are mutually restrictive, and the higher the computational parallelism, the greater the overhead of maintaining data consistency, and the higher the computational parallelism, the greater the communication overhead. Therefore, the graph computing models should be gradually refined to meet different computational needs.

In addition, the real graph data is often in a state of rapid updating, which requires that the graph computing systems can quickly complete the computation of the updating data, otherwise the computing results cannot meet the timeliness requirements. Therefore, designing a real-time graph computing model has become the research direction of the graph computing models.

References

[1] Valiant L G. A bridging model for parallel computation[J]. Communications of the ACM, 1990, 33(8): 103-111.

[2] Malewicz G, Austern M H, Bik A J C, et al. Pregel: a system for large-scale graph processing[C]// Proc of ACM SIGMOD International Conference on Management of data. New York: ACM Press. 2010: 135-146.

[3] Low Y, Bickson D, Conzalez J, et al. Distributed Graphlab: a framework for machine learning and data mining in the cloud [J]. Proceedings of the VLDB Endowment, 2012, 5(8): 716-727.

[4] Gonzalez J E, Low Y, Gu Haijie, et al. Powergraph: distributed graph parallel computation on natural graphs[C]//Proc of the 10th USENIX Conference on Operating Systems Design and Implementation. Berkely: USENIX Association, 2012: 17-30.

[5] Roy A, Mihailovic I, Zwaenepoel W. X-Stream: edge-centric graph processing using streaming partitions[C]//Proc of the 24th ACM Symposium on Operating Systems Principles. New York: ACM Press, 2013: 472-488.

[6] Tian Yuanyuan, Balmin A, Corsten S A, et al. From “think like a vertex” to “think like a graph”[J]. Proceedings of the VLDB Endowment, 2013, 7(3): 193-204.

[7] Avery C. Giraph: large-scale graph processing infrastructure on Hadoop[EB/OL]. (2011-06-29).https://github.com/aching/giraph.

[8] Salihoglu S, Widom J. GPS: a graph processing system[C]//Proc of the 25th International Conference on Scientific and Statistical Database Management. New York: ACM Press, 2013: 22.

[9] Vaquero L, Cuadrado F, Logothetis D, et al. xDGP: a dynamic graph processing system with adaptive partitioning[J]. airXive: 1309.1049, 2013.

[10] Xu Ning, Chen Lei, Cui Bin. LogGP: a log-based dynamic graph partitioning method[J]. Proceedings of the VLDB Endowment, 2014, 7(14): 1917-1928.

[11] Quamar A, Deshpande A, Lin J. NScale: neighborhood-centric analytics on large graphs[J]. Proceedings of the VLDB Endowment, 2014, 7(13): 1673-1676.

[12] Chen Rong, Shi Jiaxin, Zhang Binyu, et al. Bipartite-oriented distributed graph partitioning for big learning[C]//Proc of the 5th Asia-Pacific Workshop on Systems. New York: ACM Press, 2014.

[13] Chen Rong, Shi Jiaxin, Chen Yanzhe, et al. Powerlyra: differentiated graph computation and partitioning on skewed graphs[C]//Proc of the 10th European Conference on Computer Systems. New York: ACM Press, 2015.

[14] Teixeira C H C, Fonseca A J, Serafini M, et al. Arabesque: a system for distributed graph mining[C]//Proc of the 25th Symposium on Operating Systems Principles. New York: ACM Press, 2015: 425-440.

[15] Michael Lesniak. "Palovca: Describing and Executing Graph Algorithms in Haskell", Lecture Notes in Computer Science, 2012.