GraphLib: A Parallel Graph Mining Library for Joint Cloud Computing

Kai Zhang[1,2], Yange Fang[1,2], Yingying Zheng[1,2], Hongbin Zeng[1,2], Lijie Xu[1,*] , Wei Wang[1,2]
[1]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
[2]University of Chinese Academy of Sciences, Beijing, China
{zhangkai18,fangyange18,zhengyingying,xulijie,wangwei}@otcaix.iscas.ac.cn; zenghongbin16@mails.ucas.ac.cn

***Abstract*—Graph algorithms are widely applied in social networks, computational biology, Internet security and a broad range of complexity science. Although there are many state-of-the-art graph frameworks, few frameworks support parallel graph mining in joint cloud computing environment. In this paper, we propose GraphLib, a parallel graph mining library, based on a BSP (Bulk Synchronous Parallel) service over joint cloud computing which was proposed in our prior work. We first summarize the features of commonly-used graph mining algorithms, and present our approaches for parallelizing typical graph mining algorithms. GraphLib includes 17 parallel graph mining algorithms that can be used in 3 scenarios. We evaluate the performance of 4 typical parallel graph algorithms in GraphLib on three real-world datasets. Our parallelized algorithms can achieve sub-linear scalability.**

Keywords—Graph algorithm, BSP, JointCloud

I. INTRODUCTION

With the rapid development of graph data, single-thread algorithms are getting overwhelmed to process TB or even PB-scale data in reasonable time. To overcome this challenge, cloud storage and cloud computation platforms with distributed system arise. While more and more applications are deployed on various cloud platforms, there are many real-world scenarios that require several applications cooperating, such as complexity network analysis [1, 2, 20]. In complexity network analysis, graph computation is a vital topic, including graph structure analysis [3, 4], community detection [3, 5], ranking [11, 14], and centralities [6, 10]. Most of these applications consume enormous computation resources when processing large scale data, running far out of a single machine's capacity. Although some distributed graph processing frameworks provide several graph algorithms to deal with complexity network analysis[7, 8, 9], it takes great learning cost for cross-platform migration due to highly platform-coupled implementations. In addition, the parallel graph algorithms are needed for distributed graph processing frameworks to improve their performance. As such, achieving efficient graph computation in joint cloud environment is challenging.

To overcome this challenge, in this paper, we develop a parallel graph mining library, named GraphLib. Through analyzing and summarizing different graph algorithms, we supply a general parallel design approach to direct the design of parallel graph algorithms. According to this approach, we have implemented 17 parallel graph algorithms in typical scenarios, such as graph partition, vertex analysis, and community detection. GraphLib executes these algorithms based on the bulk synchronous parallel (BSP) service over joint cloud computation proposed in [12]. Due to the space limitation, we only introduce the parallel design and implementation of 4 typical graph algorithms, namely GraphColoring, TrustRank [11], KCore [10] and SCAN [3]. We evaluate the scalability of GraphLib on three real-world datasets. The experiments show that results indicate that our algorithms can achieve sub-linear scalability. The main contributions of this paper can be summarized as follows:

(1) We design BSP-based approaches for parallelizing the graph algorithms in Joint Cloud.

(2) We evaluate the performance and scalability of four typical algorithms in Joint Cloud.

The remainder of this paper is organized as follows. We introduce the proposed BSP service based on joint cloud computation in section II. Four typical algorithms implementations are illustrated in section III. Scalability evaluation is conveyed in section IV. Then, section V introduces some related work. We conclude this paper in section V.

II. BSP SERVICE FOR JOINT CLOUD COMPUTING

As shown in Figure 1, we design BSP service based on JointCloud [13] to realize applications in the cross-cloud environment such as graph mining algorithms. The primary infrastructure of the BSP service is composed of the BSP service management component in JointCloud Cooperation Environment (JCCE) and the BSP service controller component in Peer Cooperation Mechanism (PCM).

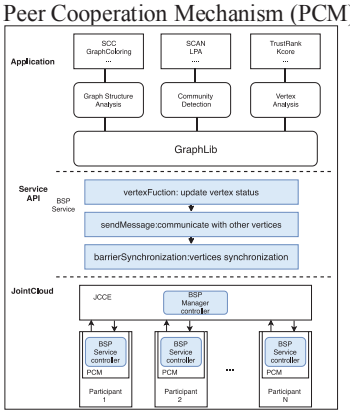

Fig.1. Design of BSP service over JonitCloud.

The control panel in PCM is responsible for managing the stored data, computing and message generation processes performed by the participants. After message is generated, the PCM passes the message to JCCE. JCCE provides services to participants via PCM.

* Corresponding author

978-1-7281-6981-1/20/$31.00 ©2020 IEEE
DOI 10.1109/JCC49151.2020.00010

TABLE I. ALGORITHM LIST

Algorithm class	Sub-class	Algorithm	Description
Graph Structure Analysis	Graph Partition	Strongly Connected Components	Finding components of which vertices are connected strongly
		Graph Coloring	Find independent sets of graphs
Community Detection	Modularity-based Methods	CNM Louvain Semi-Clustering	Merging nodes into clusters by maximizing modularity or modularity-liked metrics
	Propagation and Overlapping Methods	BMLPA SLPA Label Propagation	Spreading vertex's label to neighbors to make cluster expansion, either overlapped or non-overlapped
	Density-based Methods	SCAN	Detecting graph structural community
Vertex Analysis	Ranking	PageRank TrustRank LeaderRank	Ranking vertices by their locations in the graph using different metrics
	Centrality	Betweenness Centrality Closeness Centrality K-Core	Measuring importance of vertices and analyzing vertex features
	Neighborhood	K-Neighbor	Neighbors in k-hubs
	Similarity	SimRank	Measuring similarity between vertices

Our BSP service design is inspired by programming thinking of vertex-centric. Similar to the vertex-centric programming model, we also divide the computation process based on BSP services into a series of supersteps. In each superstep, participants perform local computation, generate messages, and interact with JCCE through the local PCM. The BSP service controller in PCM and the BSP service manager in JCCE work together to establish the synchronization barrier. The service controller in PCM passes the message to the BSP service manager in JCCE to let the service manager in JCCE control the synchronization signal. After that, PCM passes the synchronization signal to the participants. Thus, BSP service has completed three important elements of parallel computing, message passing and synchronization.

III. GRAPHLIB: PARALLEL GRAPH MINING LIBRARY

A. Parallel Approach

We design an approach to parallelize graph algorithms based on BSP model. In our approach, *vertex-cut* based partition and *vertex-centric* programming model are adopted. *vertex-cut* based partition creates multiple replicas of vertices on different machines to store graph data on different machines. *Vertex-centric* programming model allows users to abstract computational processes into vertex-based computing and edge-based message-passing. vertex acts as an independent computing unit. It communicates with other vertices using *sendMessage* API in BSP service, and updates the vertex status according to the received messages using vertexFuction API in BSP service. We design messages that vertex sends to other vertices and vertex status update function based on the characteristics of each algorithm.

B. Algorithm Library Summary

Based on the parallel design approach proposed in *Section III-A*, we have implemented 17 parallel graph algorithms, which can be classified into 3 categories: (1) **Graph Structure Analysis**. The algorithms in this category divide graph into several subgraphs or shards by rules, such as GraphColoring. (2) **Vertex Analysis**. The algorithms in this category score the vertices through locations in the graph, such as PageRank, TrustRank. (3) **Community Detection**. The algorithms in this category find clusters or subgraphs with mass intra-links but few inner-links, such as Louvain, SCAN. These algorithms are typical and popular ones among the traditional graph algorithms, and we summarize them in Table I.

Due to the space limitation, we only select four typical graph algorithms as examples to illustrate our parallel design approach.

C. GraphColoring

Description: The graph coloring problem is defined as follows. Given an undirected graph $G = (V, E)$, where V is vertex set and E is edge set, assign color to all vertices in V and ensure that the two vertices of any edge in E are different in color. Finding optimum solution for graph coloring problem is one of the most well-known NP-complete problem. There is currently no definitive algorithm to solve this problem in polynomial time. Therefore, the speed of graph coloring is far more important than finding optimum solution on big graph analytics.

Parallel approaches: As with most parallel graph coloring algorithms, the implemention in GraphLib is also based on greedy strategies. In our implementation, vertices with local maximum degree will be colored preferentially. The detail of the parallel algorithm design is as follows:

Each vertex has three variables: *VertexId*, *Degree,* and *Color*. At initial stage, all vertices set their *Color* as -1, which means all vertices are uncolored. All vertices send their *VertexId* and *Degree* to their neighbors. Only uncolored vertices participate in the computation. In each superstep, each uncolored vertex compares its own *Degree* to the *Degree* in the messages to check if it contains the largest *Degree* between neighbors, and set its value as current superstep number if yes. When the *Degree* of the vertex is the same as the maximum *Degree* in the messages, it will be colored only if it has a higher *VertexId*. If the vertex is uncolored, then it will send its *VertexId* and *Degree* to its neighbors. The algorithm is terminated when all vertices are colored.

D. TrustRank

Description: PageRank estimates the importance of a website by counting the number and quality of page links. However, a fraudulent website can cheat to get high PageRank by introducing links to other pages. TrustRank can further evaluate whether the vertex is really important based

on the results of PageRank or the set of important vertices which is manually evaluated. The basic idea of TrustRank is that the really important vertices are rarely connected to the fraud vertices, but the fraud vertices will often actively connect to the fraud vertices to enhance their credibility. Therefore, TrustRank can be used to identify fraud vertices in fraud networks.

Parallel approaches: The three major steps of the TrustRank algorithm is seed selection, trust propagation and trust attenuation. The detail of the parallel algorithm design is as follows:

Each vertex has three variables: *VertexId*, *Degree* and *TrustRank*. At initial stage, Vertices in the set of seed vertices set their *TrustRank* as 1 while the remaining vertices set their *TrustRank* as 0.5. There are two ways to select the set of see d vertices: manually or according to PageRank. Then all vertices calculate their own *TrustRank* divided by *Degree* and send it as message to neighbors. In each superstep, each vertex sums the values in the received messages, then multiplies the sum by attenuation coefficient as new *TrustRank*. Then each vertex calculates its own *TrustRank* divided by *Degree* and send it as message to neighbors. The algorithm is terminated when the number of iterations reaches the given value.

E. K-Core

Description: K-Core is a method to find a maximal connected subgraph in which all vertices have no less than k degrees. Deleting vertices of less than k degrees repeatedly is a common way in traditional implementations.

Parallel approaches: Each vertex has one variable *Deg*, recording the number of neighbors, with an initial value of the vertex's degree. In each superstep, a vertex sends messages to its neighbors if its *Deg* falls below *K*, and turns into inactive statue. Meanwhile, the number of messages that a vertex receives stands that some neighbors of the vertex are deleted, the vertex should update *Deg* minus the count of messages. The algorithm ends when no message is sent by any vertex.

F. SCAN

Description: As a density-based clustering approach, SCAN can find more than communities, as well as hubs and outliers in the network. The paper supposes that densely connected nodes belong to the same cluster, through extension of the DBSCAN. It is intuitive in community detection, that two persons share larger common social circle are more likely friends.

The algorithm is defined as follow. Assume an undirected and un-weighted graph $G = \{V, E\}$, the structural similarity between node v and w, denoted by $\sigma(v,w)$, is defined as $\sigma(v,w) = |\Gamma(v) \cap \Gamma(w)|/\sqrt{|\Gamma(v)||\Gamma(w)|}$. $\Gamma(v)$ contains node v and its neighbors. The ε-neighbors are adjacent nodes with structural similarity larger than ε. A node with more than μ ε-neighbors is defined as a core. When all the ε-neighbors and core nodes are detected, each core node expands as a cluster through ε-neighbors. Further detection examines nodes not in any clusters and distinguish them as outliers or hubs.

Parallel approaches: The e-neighbors computation is over each edge, between each two vertices, through counting common neighbors, and then whether a vertex is core can be determined by the number of e-neighbors. After this, each vertex records a variable *ClusterId*, and each core vertex possesses a unique *Id* at the initial state. In each superstep, core vertices send the unique *ClusterId* to e-neighbors and sleep. Vertices update *ClusterID* with the minimum *Id* in the received messages. In the next supersteps, only the core vertices that receive messages and change *ClusterId* are entitled to send new messages. The algorithm finishes when no message is generated.

IV. Evaluation

We use three real-world graph sets from SNAP [27] for the scalability evaluation experiments. In detail, the three are Skitter (1,696,415 nodes, 11,095,298 edges), Pokec (1,632,803 nodes, 30,622,564 edges) and LiveJournal (3,997,962 nodes, 34,681,189 edges). The cluster we use contains eight slaves, with 64 GB main memory and 4 cores per node.

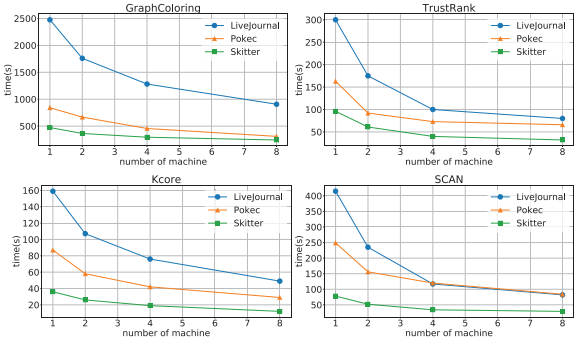

Fig.2. Performance results of GraphColoring, TrustRank, Kcore and Scan

The result (see figure 2) shows that the execution time decreases as numbers of machines increase, but sub-linear. On the one hand, the communication time of machines increases with number of machines. On the other hand, the graphs need to be saved on each machine, and the uneven partition will also affect the execution time of the experiments. The speedup result (see figure 3) indicate that the speedup of our parallel implementation can reach up to 3.5× compared with performance on single machine and our parallel implementations have shorter execution time compared to single-machine implementations. All in all, our parallel implementations have great scalability and high performance comparing to other parallel implementation.. We also test the output quality on graphs and the results stay the same with the single-machine implementations. Although our implementations are baselines of algorithms, it is comparable to the state-of-the-art versions, with great possible improvement.

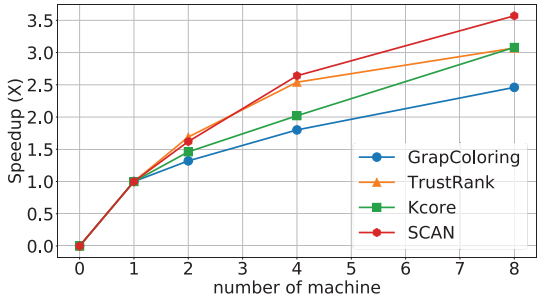

Fig.3. Average speedup of GraphColoring, TrustRank, Kcore and Scan

V. Related Work

A. Graph Computation Algorithms

Graph computation algorithms are based on graph theory, using the relationships between vertices to infer the structure of complex systems. Graph computation algorithms can be roughly divided into three categories: community detection algorithms, vertex analysis algorithms and graph structure analysis algorithms. Community detection algorithms are used to discover community structure in networks. Newman and Girvan [15] proposed modularity which can be used to measure quality of community structures. Blondel et al. [5] proposed Louvain algorithm based on modularity which is considered as one of the most powerful community detection algorithms. There are also many algorithms which is not based on modularity such LPA [16] and SCAN. Vertex analysis algorithms and graph structure analysis algorithms are used to analyze vertex feature and graph structure. Ranking algorithm originates from PageRank proposed by Google. Gy¨ongyi, Garcia-Molina and Pedersen propose TrustRank on the basis of PageRank to evaluate the credibility of website. Lu¨, Zhang, Yeung and Zhou [17] devise LeaderRank algorithm to utilize the leadership network. In [18], SimRank was proposed to measure similarity of vertices. However, these algorithms are all serial algorithms, which may have poor performance on large graph. Therefore, we design parallel implementations of these algorithms in GraphLib.

B. Graph Processing Systems

With the rapid development of big data, the scale of the graph has increased dramatically, exceeding the memory capacity of a single machine. Graph data must be stored outside memory or in distributed memory by partitions. However, for serial graph algorithms that require random access to all graph data, poor locality and graph data partitioning can lead to inefficient computation.

In order to overcome this drawback, Google introduced the Pregel systems as a distributed graph processing system based on BSP model. Since then, a large number of distributed processing systems have sprung up. Some Pregel-based graph processing systems have been developed such asApache Giraph [19]. Distributed GraphLab [8] was proposed by Low et al. in 2012. GraphX [7] is a distributed graph computing framework based on the Spark platform. However, these systems don't support services and applications based on JointCloud and lack graph computation algorithm library.

VI. Conclusion

We propose a parallel graph mining library, GraphLib, over joint cloud computation based on BSP service. The algorithms in GraphLib are widely used both in academy and industry. More algorithms will be appended to GraphLib, and optimization work of exist algorithms is under way. We believe GraphLib is a pioneer of graph computation over JCC, which is crucial for practical application.

Acknowledgment

This work is supported by the National Key R&D Program of China (2016YFB1000103), National Natural Science Foundation of China (61802377), and Youth Innovation Promotion Association at CAS.

References

[1] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: simple building blocks of complex networks," Science, vol. 298, no. 5594, pp. 824–827, 2002.

[2] C. C. Aggarwal, Ed., ''An introduction to social network data analytics,'' in Social Network Data Analytics. New York, NY, USA: Springer, 2011, pp. 1–15

[3] X. Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger, "SCAN: A structural clustering algorithm for networks," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2007, pp. 824–833.

[4] J. Huang, H. Sun, J. Han, H. Deng, Y. Sun, and Y. Liu, "Shrink: a structural clustering algorithm for detecting hierarchical communities in networks," in Proceedings of the 19th ACM international conference on Information and knowledge management. ACM, 2010, pp. 219–228.

[5] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," Journal of statistical mechanics: theory and experiment, vol. 2008, no. 10, p. P10008, 2008.

[6] S. P. Borgatti, "Centrality and network flow," Social Networks, vol. 27, no. 1, pp. 55–71, 2005.

[7] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: A resilient distributed graph system on spark," in First International Workshop on Graph Data Management Experiences and Systems. ACM, 2013, p. 2.

[8] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "PowerGraph: Distributed graph-parallel computation on natural graphs," in Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation, ser. OSDI '12. USENIX Association, Oct. 2012, pp. 17–30.

[9] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: a framework for machine learning and data mining in the cloud," Proc. VLDB Endowment, vol. 5, no. 8, pp. 716–727, 2012.

[10] S. B. Seidman, ''Network structure and minimum degree,'' Social Netw., vol. 5, no. 3, pp. 269–287, Sep. 1983.

[11] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen, "Combating web ¨ spam with trustrank," in Proceedings of the Thirtieth international conference on Very large data bases-Volume 30. VLDB Endowment, 2004, pp. 576–587.

[12] X. Sun, X. Ye, K. Kang, L. Xu, W. Wang and L. Lv, "BSP-Based Strongly Connected Component Algorithm in Joint Cloud Computing", in 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE), 2019, pp. 287-2875.

[13] H. Wang, P. Shi and Y. Zhang, "Jointcloud: A cross-cloud cooperation architecture for integrated internet service customization," in 2017 IEEE 37th international conference on distributed computing systems (ICDCS), 2017, pp. 1846-1855.

[14] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," Computer Networks, vol. 30, no. 1-7, pp. 107–117, 1998.

[15] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," Proc. of the National Academy of Sciences, vol. 99, no. 12, pp. 7821–7826, 2002.

[16] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in largescale networks," Physical review E, vol. 76, no. 3, p. 036106, 2007.

[17] L. Lü, Y.-C. Zhang, C. H. Yeung, and T. Zhou, ''Leaders in social networks, the delicious case,'' PloS ONE, vol. 6, no. 6, p. e21202, Jun. 2011, doi: 10. 1371/journal.pone.0021202.

[18] G. Jeh and J. Widom, ''SimRank: A measure of structural-context similarity,'' in Proc. KDD, Edmonton, AB, Canada, Jul. 2002, pp. 538–543.

[19] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan, "One trillion edges: graph processing at Facebook-scale," Proc. VLDB Endowment, vol. 8, no. 12, pp. 1804–1815, 2015.

[20] R. Pastor-Satorras, C. Castellano, P. Van Mieghem, and A. Vespignani, ''Epidemic processes in complex networks,'' Rev. Mod. Phys., vol. 87, no. 3, p. 925, 2015.

[21] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.