International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

COPY RIGHT

IJIEMR Transactions, online available on 17 April 2019.

Link : http://www.ijiemr.org

Title:- A Survey on SAE Large-Scale Social Networks Using Cloud Data Analysis Service.

Volume 08, Issue 04, Pages: 239 - 244.

Paper Authors

NUKALA BINDU, K.N.VENKATESWARA RAO.

Department of MCA, SKBR PG College.

USE THIS BARCODE TO ACCESS YOUR ONLINE PAPER

To Secure Your Paper As Per UGC Approvals We Are Providing A Electronic Bar Code

International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

A SURVEY ON SAE LARGE-SCALE SOCIAL NETWORKS USING CLOUD DATA ANALYSIS SERVICE

[1]NUKALA BINDU, [2]K.N.VENKATESWARA RAO
[1]PG Scholar, Department of MCA, SKBR PG College, Amalapuram
[2]Assistant Professor, Department of MCA, SKBR PG College, Amalapuram

ABSTRACT: Social network analysis is used to extract features of human communities and proves to be very instrumental in a variety of scientific domains. The dataset of a social network is often so large that a cloud data analysis service, in which the computation is performed on a parallel platform in the could, becomes a good choice for researchers not experienced in parallel programming. In the cloud, a primary challenge to efficient data analysis is the computation and communication skew (i.e., load imbalance) among computers caused by humanity's group behavior (e.g., bandwagon effect). Traditional load balancing techniques either require significant effort to re-balance loads on the nodes, or cannot well cope with stragglers. In this paper, we propose a general straggler-aware execution approach, SAE, to support the analysis service in the cloud. It offers a novel computational decomposition method that factors straggling feature extraction processes into more fine-grained sub-processes, which are then distributed over clusters of computers for parallel execution. Experimental results show that SAE can speed up the analysis by up to 1.77 times compared with state-of-the-art solutions.

KEY WORDS: Cloud service, Social network analysis, Computational skew, Communication skew, Computation decomposition.

I.INTRODUCTION

OCIAL network analysis is used to extract features, such as neighbors and ranking scores, from social network datasets, which help understand human societies. With the emergence and rapid development of social applications and models, such as disease modeling, marketing, recommender systems, search engines and propagation of influence in social network, social network analysis is becoming an increasingly important service

In the cloud. For example, k-NN [1], [2] is employed in proximity search, statistical classification, recommendation systems, internet marketing and so on. Another example is k-means [3], which is widely used in market segmentation, decision support and so on.

Other algorithms include connected component [4], [5], katz metric [6], [7], adsorption [8], SSSP [13] and so on. These algorithms often need to repeat the same process round by round until the computing satisfies a convergence or stopping condition. In order to accelerate the execution, the data objects are distributed over clusters to achieve parallelism. However, because of the humanity's group behavior [14], [15], [16], [17], the key routine of social net-work analysis, namely feature extraction process (FEP), suffers from serious computational and communication skew in the cloud. Specifically, some FEPs need much

more computation and communication in each iteration than others. Take the widely used data set of Twitter web graph [18] as an example, less than one percent of the vertices are adjacent to nearly half of all edges. It means that tasks hosting this small fraction of vertices may require many times more computation and communication than an average task does. Moreover, the involved data dependency graph of FEPs may be known only at execution time and changes dynamically. It not only makes it hard to evaluate each task's load, but also leaves some computers underutilized after the convergence of most features in early iterations.

In the PageRank algorithm running on a Twitter web graph, for example, the majority of the vertices require only a single update to get their ranking scores, while about 20% of the vertices require more than 10 updates to converge. This implies that many computers may become idle in a few iterations, while others are left as stragglers burdened with heavy workloads. Current load balancing solutions try to mitigate load skew either at task level or at worker level. At the task level, these solutions partition the data set according to profiled load cost [19], or use PowerGraph [20] for static graph, which partitions edges of each vertex to get balance among tasks. The former method is quite expensive, as it has to periodically profile load cost of each data object. PowerGraph [20] can only statically partition computation for graphs with fixed dependencies and therefore cannot adaptively redistribute sub-processes over nodes to maximize the utilization of computation resources.

At the worker level, the state-of-the-art solutions, namely persistence-based load balancers (PLB) [21] and retentive work stealing (RWS) [21], can dynamically balance load via tasks redistribution/stealing according to the profiled load from the previous iterations. However, they cannot support the computation decomposition of straggling FEPs. The task partitioning for them mainly considers evenness of data size, and so the corresponding tasks may not be balanced in load. This may cause serious computational and communication skew during the execution of program. In practice, we observe that a straggling FEP is largely decomposable, because each feature is an aggregated result from individual data objects.

As such, it can be factored into several sub-processes which perform calculation on the data objects in parallel. Based on this observation, we propose a general straggler-aware computational partition and distribution approach, named SAE, for social network analysis. It not only parallelizes the major part of straggling FEPs to accelerate the convergence of feature calculation, but also effectively uses the idle time of computers when available. Meanwhile, the remaining non-decomposable part of a straggling FEP is negligible which minimizes the straggling effect. We have implemented a programming model and a runtime system for SAE. Experimental results show that it can speed up social network analysis by a factor of 1.77 compared with PowerGraph. Besides, it also produces a speedup of 2.23 against PUC [19], which is a state-of-the-art task-level load balancing scheme. SAE achieves speedups of 2.69 and 2.38 against PLB [21] and RWS [21], respectively, which are two state-of-the-art worker level load balancing schemes. In

summary, we make the following three contributions:

1) A general approach to supporting efficient social network analysis, using the fact the FEP is largely decomposable. The approach includes a method to identify straggling FEPs, a technique to factor FEP into sub-processes and to adaptively distribute these sub-processes over computers.

2) A programming model along with an implementation of the runtime system, which efficiently supports such an approach.

3) Extensive experimental results showing the advantages of SAE over the existing solutions.

II. MOTIVATION

Social network analysis is used to analyze the behavior of human communities. However, because of human's group behavior, some FEPs may need large amounts of computation and communication in each iteration, and may take many more iterations to converge than others. This may generate stragglers which slow down the analysis process. Consider a Twitter follower graph [18], [22] containing 41.7 million vertices and 1.47 billion edges.It can be seen that less than one percent of the vertices in the graph are adjacent to nearly half of the edges. Also, most vertices have less than ten neighbors, and most of them choose to follow the same vertices because of the collective behavior of humanity. This implies that tasks hosting these small percentage of vertices will assume enormous computation load while others are left underloaded. With persistence-based load balancers [21] and retentive work stealing [21], which cannot support the computation decomposition of straggling FEPs, high load imbalance will be generated for the analysis program. For some applications, the data dependency graph of FEPs may be known only at run time and may change dynamically.

This not only makes it hard to evaluate each task's load, but also leaves some computers underutilized after the convergence of mostfeatures in early iterations. the distribution of update counts after a dynamic PageRank algorithm converges on the twitter follower graph. From this figure, we can observe that the majority of the vertices require only a single update, while about 20% of the vertices require more than 10 updates to reach convergence. Again, this means that some computers will undertake much more workload than others. However, to tackle this challenge, the tasklevel load balancing approach based on profiled load cost [19] has to pay significant overhead to periodically profile load cost for each data object and to divide the whole data set in iterations. For PowerGraph [20], which tends to statically factor computation tasks according to the edge distribution of graphs, this challenge also renders this static task planning ineffective and leaves some computers underutilized, especially when most features have converged in early iteration.

III. PROPOSED SYSTEM

In reality, the computation decomposition approach proposed for the processing of a feature can be mainly abstracted as follows:

1) receive the values of related features and calculate several needed local attributes for this feature according to received value;

2) when all the local attributes needed by this feature are calculated and available, gather and accumulate these attributes for this feature;

3) calculate the new value of this feature based on the above accumulated attributes, then

International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

diffuse the new value of this feature to related features for the next iteration.

The whole execution progress of a feature can be depicted in Fig. 1. To support such a data-centric programming model, several interfaces are provided. The interfaces that require application developers to instantiate are summarized. The decomposable part is factored into several sub-processes, which are abstracted by Extra(). These sub-processes are distributed and executed over workers by runtime. The Barrier() contained in system code SysBarrier().When all those attribute values needed by a set of features are available, Acc() contained in SysBarrier() will be triggered and executed on a worker to accumulate the results of related Extra() for this set of features. Then it calculates and outputs the new value of these features for the next iteration. In reality, Acc() is the nondecomposable part of FEP.

In the following part, we take the PageRank algorithm as an example to show how to use the above programming model to meet the challenges of straggling feature. PageRank is a popular algorithm proposed for ranking web pages. Formally, the web linkage graph is a graph where the node set V is the set of web pages, and there is an edge from node i to node j if there is a hyperlink from page i to page j. To efficiently support the distribution and execution of sub-processes, a system, namely SAE, is realized. It is implemented in the Piccolo [23] programming model.

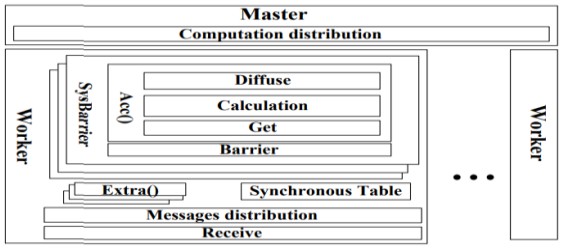

Fig. 1: ARCHITECTUREOF SAE

The architecture of SAE is presented in Fig. 5. It contains a master and multiple workers. The master monitors status of workers and detects the termination condition for applications. Each worker receives messages, triggers related Extra() operations to process these messages and calculates new value for features as well. In order to reduce communication cost, SAE also aggregates these messages that are sent to the same node. Each worker loads a subset of data objects into memory for processing. All data objects on a worker are maintained in a local in-memory key-value store, namely state table. Each table entry corresponds toa data object indexed by its key and contains three fields. The first field stores the key value j of a data object, the second its value; and the third the index corresponding to its feature recorded in the following table.

To store the value of features, a feature table is also needed, which is indexed by the key of features. Each table entry of this table contains four fields. The first field stores the key value j of a feature, the second its iteration number, the third its value in the current iteration; and the fourth the attribute list. At the first iteration, SAE only divides all data objects into equally-sized partitions. Then it can get the load of each FEP from the finished iteration. With this information, in the subsequent iterations, each worker can identify straggling features and partition their related value set into a proper number of blocks according to the ability of each worker.

In this way, it can create more chances for the straggling FEPs to be executed and achieve rough load balance among tasks. At the same time, the master detects whether there is necessity to redistribute blocks according to its gained benefits and the related cost, after

International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

receiving the profiled remaining load of each worker, or when some workers become idle. Note that the remaining load of each worker can be easily obtained by scanning the number of unprocessed blocks and the number of values in these blocks in an approximate way. While the new iteration proceeds as follows in an asynchronously way without the finish of block redistribution, because only the unprocessed blocks are migrated. When a diffused message is received by a worker, it triggers an Extra() operation and makes it process a block of values contained in this message. After the completion of each Extra(), it sends its results to the worker w, where the feature's original information is recorded on this worker's feature table.

After receiving this message, worker w records the availability of this block on its synchronization table and stores the results, where these records will be used by Barrier() in SysBarrier() to determine whether all needed attributes are available for related features. Then SysBarrier() is triggered on this worker. When all needed attributes are available for a specified feature, the related Acc() contained in SysBarrier() is triggered and used to accumulate all calculated results of distributed decomposable parts for this feature. Then Acc() is employed to calculate a new value of this feature for the next iteration. After the end of this iteration, this feature's new value is diffused to specified other features for the next iteration to process. At the same time, to eliminate the communication skew occurred at the value diffusion stage, these new values are diffused in a hierarchical way. In this way, the communication cost is also evenly distributed over clusters at the value diffusion stage.

IV. RESULTS

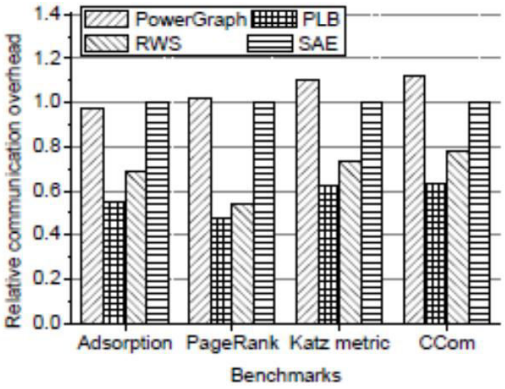

fig. 2: communicational skew comparison

V. CONCLUSION

This paper identifies that the most computational part of straggling FEP is decomposable. Based on this observation, it proposes a general approach to factor straggling FEP into several sub-processes along with a method to adaptively distribute these sub-processes over workers in order to accelerate its convergence. Later, this paper also provides a programming model along with an efficient runtime to support this approach. Experimental results show that it can greatly improve the performance of social network analysis against state-of-the-art approaches.

VI. REFERENCES

[1] Z. Song and N. Roussopoulos, "K-nearest neighbor search for moving query point," Lecture Notes in Computer Science, vol. 2121, pp. 79–96, July 2001.

[2] X. Yu, K. Q. Pu, and N. Koudas, "Monitoring k-nearest neighbor queries over moving objects," in Proceedings of the 21st International Conference on Data Engineering. IEEE, 2005, pp. 631–642.

[3] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering

algorithm: Analysis and implementation," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 7, pp. 881–892, July 2002.

[4] L. Di Stefano and A. Bulgarelli, "A simple and efficient connected components labeling algorithm," in Proceedings of the International Conference on Image Analysis and Processing. IEEE, 1999, pp. 322–327.

[5] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good et al., "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," Scientific Programming, vol. 13, no. 3, pp. 219–237, January 2006.

[6] L. Katz, "A new status index derived from sociometric analysis," Psychometrika, vol. 18, no. 1, pp. 39–43, March 1953.

[7] D. Liben-Nowell and J. Kleinberg, "The link prediction problem for social networks," in Proceedings of the 12th international conference on Information and knowledge management. ACM, 2003, pp. 556–559.

[8] S. Baluja, R. Seth, D. Sivakumar, Y. Jing, J. Yagnik, S. Kumar, D. Ravichandran, and M. Aly, "Video suggestion and discovery for youtube: taking random walks through the view graph," in Proceedings of the 17th international conference on World Wide Web. ACM, 2008, pp. 895–904.

[9] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," Computer networks and ISDN systems, vol. 30, no. 1, pp. 107–117, April 1998.

[10] S. Baluja, R. Seth, D. Sivakumar, Y. Jing, J. Yagnik, S. Kumar, D. Ravichandran, and M. Aly, "Video suggestion and discovery for youtube: taking random walks through the view graph," in Proceedings of the 17th international conference on World Wide Web. ACM, 2008, pp. 895–904.