International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 03 Issue 10
June 2016

Achieving Efficiency in Cloud Data Analysis for Global Social Networks

Meerakori Vijay[1], H. Ateeq Ahmed[2]

[1] M.Tech Student,Department of CSE, Dr.K.V.Subba Reddy Institute of Technology

[2] Assistant Professor, Department of CSE, Dr.K.V.Subba Reddy Institute of Technology

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 cloud, 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 behaviour (e.g., bandwagon effect). Traditional load balancing techniques either require significant effort to rebalance 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.

Keywords: Feature Extraction Process, Straggler, Parallel Execution, Cloud Storage, Load Balancing

I.INTRODUCTION

Social 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 algorithm , Katz Metric, Page Rank are used for proximity search, statistical classification, indexing etc. 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. The key routine of social network analysis namely, the Feature Extraction Process(FEP) suffers from serious computational and

communication skew. The 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.Many computers may become idle in a few iterations, while others are left as stragglers burdened with heavy work loads. One solution to reduce this skew is by decomposition of all the running processes. Current load balancing solutions try to mitigate the load skew either at task level or at worker level.Both do not support the decomposition of straggling processes. 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 subprocesses 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 net-work 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

II.RELATED WORK

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 entire analysis process . The skew resistant parallel processing technique uses scientific user defined data for categorizing load imbalance among several processes. This however adds to large space and time complexity thereby reducing the overall efficiency of the above technique. Another method used is the Skew Reduce technique. This methodology of Skew Reduce has two components. The first component is an API for expressing spatial featureextraction algorithms. The second component of Skew Reduce is a static optimizer that partitions the data to ensure skew-resistant processing if possible. The data partitioning is guided by a user-defined cost function that estimates processing times. However the most widely used existing technique was proposed by katz and is named as Katz Metric algorithm. The existing system can dynamically balance loads at the task level rather than the work level using the previous iterations. It is `an often used link prediction approach via measuring the proximity between two nodes in a graph and is computed as the sum over the collection of paths between two nodes.in a graph and is computed as the sum over the collection of paths between two nodes, exponentially damped by the path length. With this algorithm, we can predict links in the social network, understand the mechanisms by which the social networks evolve and so on.

But the major drawback is even this concept cannot decompose straggling (or) idle processes that contribute to computational skew. Current solutions for this problem either focus on task level load balancing or on worker-level balancing. Task-level load balancing. Skew reduce is a state-of-the-art solution for reducing load imbalance among tasks, in view that in some scientific applications, different partitions of the data object set take vastly different amounts of time to run even if they have an equal size. It proposes to employ user defined cost function to guide the division of the data object set into equallyloaded, rather than equally-sized, data partitions. However, in order to ensure low load imbalance for social network analysis, it has to pay significant overhead to periodically profile load cost for each data object and to divide the whole data set in iterations. Take the widely used data set of Twitter web graph 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. 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. At the task level, these solutions partition the data set according to profiled load cost , or use Power Graph 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 can only statically partition computation for graphs with fixed dependencies and therefore cannot adaptively redistribute sub-processes over nodes to maxi mize the utilization of computation resources. At the worker level, the state-of-the-art solutions, namely persistence-based load balancers (PLB) and retentive work stealing (RWS), can dynamically balance load via tasks redistribution/stealing according to the profiled load from the previous iterations.

III.PROPOSED METHODOLOGY

Usually, a major part, called the decomposable part, of the computation task in an FEP is decomposable, because each feature can be seen as a total contribution from several data objects. Thus, each FEP can be factored into several subprocesses which calculate the value of each data object separately. This allows us to design a general approach to avoid the impact of load skew via a straggler-aware execution method.The remaining non-decomposable part of a straggling FEP is negligible and has little impact on the overall performance. To parallelize the decomposable part of a straggling FEP and speed up the convergence of the feature calculation, the straggling FEP can be factored into several sub-processes.To ensure the correctness of this decomposition, the decomposable part

should satisfy the accumulative property and the independence property. The former property means that the results of this part can be calculated based on results of its sub-processes; the latter suggests that execution can be done in any required order.

IV.ALGORITHM

The proposed Feature Extraction algorithm uses two algorithms .One for the Master and another for the Worker. A master can be a high priority process and a worker can be its feature extracted sub process.

A) Redistribution and Migration Algorithm

Whenever a Straggling feature is identified its value set is divided in to equally spaced blocks corresponding to the nonstraggling processes. After the decomposition, some workers may be more heavily loaded than others. For example, most FEPs have converged during the first several iterations. Then, the workers assigned with these converged FEPs may become idle. Consequently, it needs to identify whether it should redistribute blocks among workers based on the previous load distribution to make the load cost of all workers balanced and to accelerate the convergence of the remaining FEPs. Now, we show the details of deciding when to redistribute blocks according to a given cluster's condition. In reality, whenever the periodically profiled remaining load of workers is received, or a worker becomes idle, it determines whether a block redistribution is needed. It redistributes blocks only when the intuition is as follows. If it decides to redistribute blocks, its gained benefits should be more than its caused cost. In other words, the redistribution makes sense only if the spared processing timeTs is greater than the cost overhead. The processing timeTb and Ta can be approximately evaluated via the straggling worker before and after block redistribution at the previous iteration, respectively. Specifically, Tb can be directly evaluated by the finish time of the slowest worker at the previous iteration. The approximate value of Ta is the average completion time of all workers at the previous iterations. The redistribution time C is mainly determined by the number of redistributed blocks, we can approximately evaluate the redistribution time C as follows:C=A1+A2*N; where constants A1 and A2 can be obtained from the block redistribution of the previous iteration. incrementally redistributes blocks based on the block distribution of the previous iteration. It always migrates the load of the slowest worker to the idle worker or the fastest worker via directly migrating blocks. The migration algorithm always trend to migrate the nonstraggling features and only distributes straggling features blocks over workers when there is no choice. Because the load of straggling workers maybe several times more than the average, redistribution algorithm may need several times to migrate its load to a set of idle workers or fastest workers in an asynchronous way .

b) SAE

To efficiently support the distribution and execution of subprocesses, a system, namely SAE, is realized. 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 communicationcost, 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 to a data object indexed by its key and contains three fields. The first field stores the key value of a data object, the second its value; and the third the index corresponding to its feature recorded in the 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 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. 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 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 number of unprocessed blocks and the number of values in these blocks in an approximate way.

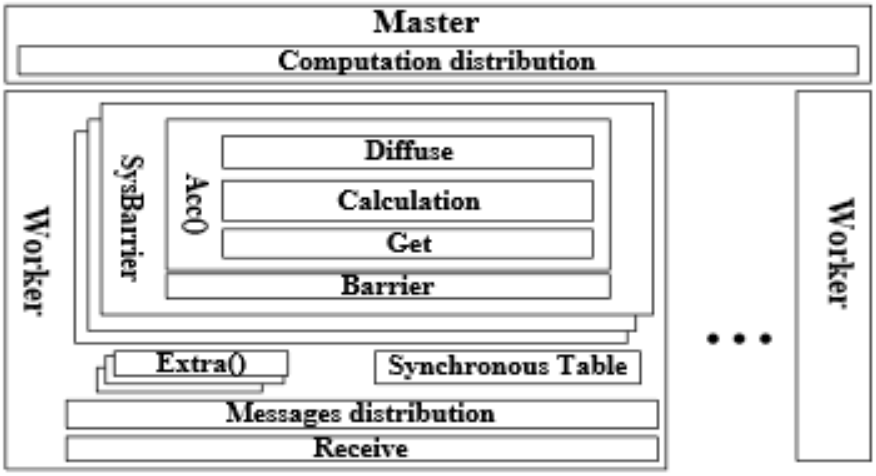
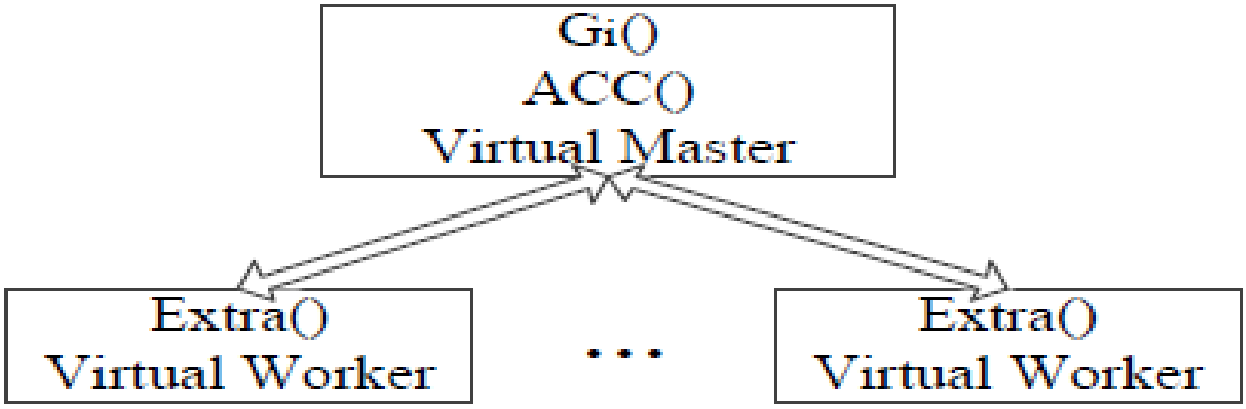


Figure 1: SAE Architecture

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.



V.EXPERIMENTAL RESULTS

In order to evaluate this approach against current solutions, four benchmarks are implemented:

1)Adsorption : It is a graph-based label propagation algorithm, which provides personalized recommendation for contents and is often employed in the recommendation systems.

2)PageRank : It is a popular link analysis algorithm, that assigns a numerical weighting to each element, aiming to measure its relative importance within the set.

International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 03 Issue 10
June 2016

3)KatzMetric : It is a often used link prediction approach via measuring the proximity between two nodes in a graph and is computed as the sum over the collection of paths between two nodes.

4)ConnectedComponents : It is often employed to find connected components in a graph by letting each node propagate its component ID to its neighbours. The following graph indicates the relative computational and communication skew comparison of SAE with other algorithms taking a twitter graph as an example .

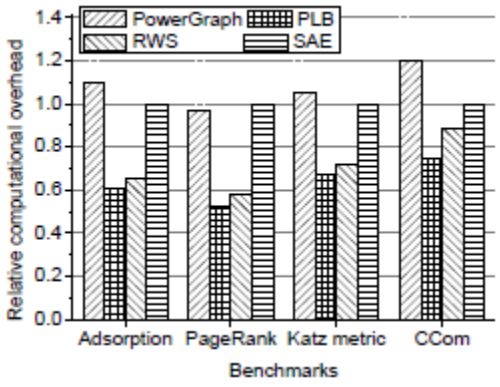


Figure 2:Computational Skew Comparison

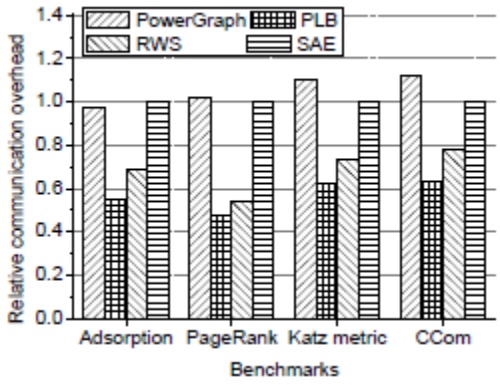


Figure 3: Communicational Skew Comparison

VI. CONCLUSION

For social network analysis, the convergence of straggling FEP may need to experience significant numbers of iterations and also needs very large amounts of computation and communication in each iteration, inducing serious load imbalance. However, for this problem, current solutions either require significant overhead, or cannot exploit underutilized computers when some features converged in early iterations, or perform poorly because of the high load imbalance among initial tasks. 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.

VII.REFERENCES

[1] Z. Song and N. Roussopoulos, "K-nearest neighbor search formoving 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 neighbour 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 labelling 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. Goodet 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