Parallel Graph Computation using a Partition Aware Engine: Review Paper

Mr. S. B. Shirsath[1], Prof. S. R. Durugkar[2]

[1]Department of Comp. Engg., SNDCOE & RC, Maharashtra, India
[2]Head, Department Comp. Engg, SNDCOE & RC, Maharashtra, India

Abstract— *A Partition Aware Engine framework provides a major increase in compression with respect to all currently known techniques, both on web graphs and on social networks. These improvements make it possible to analyse in main memory significantly larger graphs. Graph partition quality affects the overall performance of parallel graph computation systems. The quality of a graph partition is measured by the balance factor and edge cut ratio. A balanced graph partition with small edge cut ratio is generally preferred since it reduces the expensive network communication cost. However, according to an empirical study on Giraph, the performance over well partitioned graph might be even two times worse than simple random partitions. This is because these systems only optimize for the simple partition strategies and cannot efficiently handle the increasing workload of local message processing when a high quality graph partition is used. In this paper, we propose a novel partition aware graph computation engine named PAGE, which equips a new message processor and a dynamic concurrency control model. In this paper, we propose a new paradigm, Partition Aware Engine framework that allows parametric control of asynchrony ranging from completely asynchronous execution to partially asynchronous execution to level-synchronous execution. Partial asynchrony is achieved by generalizing the BSP model to allow each super step to process up to k levels of the algorithm asynchronously. In the model, we studying on two heuristic rules to effectively extract the system characters and generate proper parameters.*

Keywords—Message Processing, Parallel Graph, Graph Partition, Graph Computation.

I. INTRODUCTION

Database Parallel graph algorithms are currently expressed in level synchronous or asynchronous paradigms. Level-synchronous paradigms iteratively process vertices of a graph level by level. This model guarantees the current level's computation to have completed before starting the next one through the use of global synchronizations at the end of each level. Level-synchronous algorithms tend to perform well when the number of levels is small, but suffer from poor scalability when the number of levels is large. Bulk synchronous parallel (BSP) algorithms can naturally be expressed in this paradigm. The asynchronous paradigm replaces global synchronizations with point-to-point synchronizations, which can increase the degree of parallelism, but which may also require the completion of redundant work. For example, an asynchronous breadth-first search (BFS) may re-visit vertices multiple times as shorter paths are discovered. Choosing the right paradigm depends on the system, input graph, and algorithm. This implies different implementations and optimizations for algorithms, with no easy way to switch between them.

II. SURVEY REVIEW

1. Harshvardhan show how common patterns in graph algorithms can be expressed in the KLA paradigm and provide techniques for determining k, the number of asynchronous steps allowed between global synchronizations. Results of an implementation of KLA in the staple Graph Library show excellent scalability on up to 96K cores and improvements of 10x or more over level synchronous and asynchronous versions for graph algorithms such as breadth-first search, PageRank, k-core decomposition and others on certain classes of real-world graphs.
2. Amr Ahmed proposes a framework for large-scale graph decomposition and inference. To resolve the scale, our framework is distributed so that the data are partitioned over a shared nothing set of machines. They propose a novel factorization technique that relies on partitioning a graph so as to minimize the number of neighbouring vertices rather than edges across partitions. System decomposition is based on a streaming algorithm. It is network-aware as it adapts to the network topology of the underlying computational hardware.
3. Nathan Backman present a framework that parallelizes and schedules workflows of stream operators, in real-time, to meet latency objectives. It supports data- and task-parallel processing of all workflow operators, by all computing nodes, while maintaining the ordering properties of sorted data

streams. System show that a latency-oriented operator scheduling policy coupled with the diversification of computing node responsibilities encourages parallelism models that achieve end-to-end latency-minimization goals.

4. Lars Backstrom use decision-tree techniques to identify the most significant structural determinants of these properties. Also develop a novel methodology for measuring movement of individuals between communities, and show how such movements are closely aligned with changes in the topics of interest within the communities.
5. Paolo Boldi presents the compression techniques used in Web Graph, which are centred around referentiation and intervalisation (which in turn are dual to each other). Web Graph can compress the WebBase graph (118 Mnodes, 1 Glinks) in as little as 3.08 bits per link, and its transposed version in as little as 2.89 bits per link.
6. Marco Rosa proposed algorithm with the Web Graph compression framework provides a major increase in compression with respect to all currently known techniques, both on web graphs and on social networks. These improvements make it possible to analyse in main memory significantly larger graphs.

III. PARALELL IMPLEMENTATION

Layered label propagation lends itself naturally to the task-decomposition parallel-programming paradigm, which may dramatically improve performances on modern multicore architectures: since the update order is randomised, there is no obstacle in updating several nodes in parallel. Our implementation breaks the set of nodes into a very small number of tasks (in the order of thousands). A large number of threads picks up the first available task and solves it: as a result, we obtain a performance improvement that is linear in the number of cores. We are helped by Web Graph's facility, which allows us to provide each thread with a lightweight copy of the graph that shares the bit stream and associated information with all other threads.

IV. CHALLENGES

Latent variable modelling is a promising technique for many analytics and predictive inference applications. However, parallelization of such models is difficult since many latent variable models require frequent synchronization of their state. The power law nature of such graphs makes it difficult to use chromatic scheduling. Furthermore, the bulk-synchronous processing paradigm of Map-Reduce does not afford low-enough latency for fast convergence: this has been reported, e.g. in comparisons between bulk-synchronous convergence and asynchronous convergence. Consequently there is a considerable need for algorithms which address the following issues when performing inference on large natural graphs:

Graph Partitioning We need to find a communication efficient partitioning of the graph in such a manner as to ensure that the number of neighbouring vertices rather than the number of edges is minimized. This is relevant since latent variable models and their inference algorithms store and exchange parameters that are associated with vertices rather than edges.

Network Topology In many graph-based applications the cost of communication (and to some extent also computation) the cost of storing data. Hence it is desirable to have an algorithm which is capable to layout data in a network-friendly fashion on the fly once we know the computational resources.

Variable Replication While the problem of variable synchronization for statistical inference with regular structure is by now well understood, the problem for graphs is more complex: The state space is much larger (each vertex holds parts of a state), rendering synchronization much more costly – unlike in aspect models only few variables are global for all partitions.

Asynchronous Communication Finally there is the problem of eliminating the synchronization step in traditional bulk-synchronous systems on graphs. More specifically, uneven load distribution can lead to considerable inefficiencies in the bulk synchronous setting. After all, it is the slowest machine that determines the runtime of each processing round (e.g. Map Reduce). Asynchronous schemes, on the other hand, are nontrivial to implement as they often require elaborate locking and scheduling strategies.

V. DYNAMIC CONCURRENCY CONTROL MODEL

The concurrency control problem can be modelled as a typical producer-consumer scheduling problem, where the computation phase generates messages as a producer, and message process units in the dual concurrent message processor are the consumers. Therefore, the producer-consumer constraints should be satisfied when solving the concurrency control problem.

The concurrency of dual concurrent message processor heavily affects the performance. But it is expensive and also challenging to determine a reasonable concurrency ahead of real execution without any assumption. Therefore, PAGE needs a mechanism to adaptively tune the concurrency of the dual concurrent message processor. The mechanism is named Dynamic Concurrency Control Model, DCCM for short.

For the PAGE situation, the concurrency control problem arises consumer constraints. Since the behaviour of producers is determined by the graph algorithms, PAGE only requires to adjust the consumers to satisfy the constraints (behaviour of graph algorithms), which are stated as follows.

First, PAGE provides sufficient message process units to make sure that new incoming message blocks can be processed immediately and do not block the whole system. Meanwhile, no message process unit is idle.

Second, the assignment strategy of these message process units ensures that each local/remote message process unit has balanced workload since the disparity can seriously destroy the overall performance of parallel processing.

VI. CONCLUSION

Finally we conclude that our study of the partition unaware problem in current graph computation systems and its severe drawbacks for efficient parallel large scale graphs processing. To address this problem, we proposed a partition aware graph computation engine named Parallel Graph Computation using a Partition Aware Engine that monitors three high-level key running metrics and dynamically adjusts the system configurations.

REFERENCES

[1] Yingxia Shao, Bin Cui, Senior, "PAGE: A Partition Aware Engine for Parallel Graph Computation", IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 27, NO. 2, FEBRUARY 2015.

[2] A. Amr, S. Nino, N. Shravan, J. Vanja, and S. J. Alexander, “Distributed large scale natural graph factorization,” in Proc. 22nd Int. Conf. World Wide Web, 2013, pp. 37–48.

[3] N. Backman, R. Fonseca, and U. C¸ et intemel, “Managing parallelism for stream processing in the cloud,” in Proc. 1st Int. Workshop Hot Topics Cloud Data Process., 2012, pp. 1:1–1:5.

[4] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan, “Group formation in large social networks: Membership, growth, and evolution,” in Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2006, pp. 44–54.

[5] P. Boldi and S. Vigna, “The webgraph framework I: Compression techniques,” in Proc. 13th Int. Conf. World Wide Web, 2004, pp. 595– 602.

[6] P. Boldi, M. Rosa, M. Santini, and S. Vigna, “Layered label propagation: A multire solution coordinate-free ordering for compressing social networks,” in Proc. 20th Int. Conf. World Wide Web, 2011, pp. 587–596.

[7] S. Brin and L. Page, “The anatomy of a large-scale hyper textual web search engine,” in Proc. 7th Int. Conf. World Wide Web, 1998, pp. 107–117.

[8] A. Chan, F. Dehne, and R. Taylor, “CGMGRAPH/CGMLIB: Implementing and testing CGM graph algorithms on PC clusters and shared memory machines,” J. High Perform. Comput. Appl., pp. 81–97, 2005.

[9] G. Cong, G. Almasi, and V. Saraswat, “Fast PGAS implementation of distributed graph algorithms,” in Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal., 2010, pp. 1–11.

[10] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” in Proc. Operating Syst. Des. Implementation, 2004, pp. 107–113.

[11] D. Gregor and A. Lumsdaine, “The parallel BGL: A generic library for distributed graph computations,” in Proc. Parallel Object-Oriented Sci. Comput., 2005, pp. 1–18.

[12] C. A. R. Hoare, “Communicating sequential processes,” Commun. ACM, vol. 21, pp. 666–677, 1978.

[13] U. Kang, C. E. Tsourakakis, and C. Faloutsos, “PEGASUS: A petascale graph mining system implementation and observations,” in Proc. IEEE 9th Int. Conf. Data Mining, 2009, pp. 229–238.

[14] G. Karypis and V. Kumar, “Multilevel algorithms for multiconstraint graph partitioning,” in Proc. ACM/IEEE Conf. Supercomput., 1998, pp. 1–13.

[15] A. Roy, I. Mihailovic, and W. Zwaenepoel, “X-stream: Edgecentric graph processing using streaming partitions,” in Proc. 24th ACM Symp. Operating Syst. Principles, 2013, pp. 472–488.

[16] S. Salihoglu and J. Widom, “GPS: A graph processing system,” in Proc. 25th Int. Conf. Sci. Statist. Database Manage., 2013, pp. 22:1–22:12.

[17] B. Shao, H. Wang, and Y. Li, “Trinity: A distributed graph engine on a memory cloud,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2013, pp. 505–516.

[18] H. Simonis and T. Cornelissens, “Modelling producer/consumerconstraints,” in Proc. 1st Int. Conf. Principles Practice Constraint Program.,1995, pp. 449–462.

[19] I. Stanton and G. Kliot, “Streaming graph partitioning for large distributed graphs,” in Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2012, pp. 1222–1230.

[20] C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic, “Fennel: Streaming graph partitioning for massive scale graphs,” in Proc. Int. ACM Conf. Web Search Data Mining, 2014, pp. 333–342.