Graph Partitioning for Distributed Shared Memory

Clement Fung
cfung1@cs.ubc.ca

Stewart Grant
sgrant09@cs.ubc.ca

Abstract

The modern internet generates petabytes of data per day. Processing vast amounts of data is an increasingly common task both for scientists and modestly experienced programmers. Often this data is naturally represented as a graph, such as social media networks, webpage links or city networks, and requires clusters of machines to process. Concurrent trends in data centre architecture suggest that the rack is the new server, and shared memory is now a feasible interface between collocated rack servers. These trends made us wonder: *How simple can fast graph processing be on a rack of servers?*. We investigated the tradeoffs of the conventional Pregel *"think like a vertex"* programming model, and found its performance unacceptable. In contrast, we explored the merits of a *"Think like a subgraph"* model, which respects graph locality in common graphs, provides a more holistic programming interface, and runs faster!

1 Introduction

Big data processing is complicated. Scientists and experienced programmers alike struggle with managing and configuring clusters of machines to processes large amounts of data. Frameworks like Hadoop and Pregel have significantly eased the difficulty of big data processing, but they remain intimidating for the layman. We noticed a trend in scientists and researchers, finding that they wanted to *"Just write python code that ran on a bunch of computers"*. For the benefit of science we investigated how to make this dream a reality.

Running all Python code on clusters of machines is impractical would lead to sluggish un-optimal code, due to immense network overhead. Instead, we concentrated our efforts on a common but difficult big data processing task: graph processing. Many frameworks exist for distributed graph processing [18, 6, 15, 17, 24, 9], and many general frameworks exist which are used for graph processing [23, 25, 13, 20]. These frameworks vary in their complexity, but none are *"accessible"* for programmers that lack relevant experience.

With the exception of [15], the aforementioned systems suffer a common pitfall to accessibility; they expose the complexity of a distributed message passing system to the user. Extensive work has been done to hide this complexity in the abstraction of distributed shared memory (DSM) [14, 21, 19, 10, 12]. The benefits of DSM have been ignored in recent years due to its flaws, most notably fate sharing and sub optimal performance. Dismissing DSM may have been a shortsighted mistake. Ultra dense memory and the approach of terabit-level bandwidth within a rack give modern racks the appearance of a single machine, and has lead towards disaggregated architectures [1, 2, 3, 5, 11]. Such futuristic systems lend themselves naturally to DSM which motivates our proposal for a corresponding computation framework.

The largest disadvantage of DSM is performance. Programmers can write terribly performant programs by failing to reason about the location of memory, leading to memory thrashing. In computation where a high degree of consistency between shared resources is needed, DSM is the wrong tool for the job. In contrast, when large amounts of computation can be performed between memory synchronizations, DSM provides a simple and efficient programming model. Graph processing suffers from a lack of locality. In computations such as PageRank, a single iteration may perform edge updates which require the synchronization of all machines in a cluster. This problem can be largely avoided in practice by carefully pre-processing graphs into equally-sized partitions, where the minimum number of graph edges exist across machines. The cost of pre-processing a graph can be large; in some cases, the complexity of finding a good partition is greater than solving the initial problem! Here we demonstrate that the cost of graph partitioning is worth it for the benefits that DSM provides.

In this paper we attack the problem of developing a simple and efficient graph processing interface for DSM. Specifically we make the following contributions.

- A simple graph processing API for DSM
- A graph partitioning scheme optimal for DSM

- An evaluation of processing performance between partitioned DSM processing, and Pregel-style graph processing

The remainder of this paper is organized as follows. In Section 2 we over view related work. In Section 3 we describe our graph processing API. Section 4 we describe our approach to graph partitioning.In Section 5 we evaluate our framework against a Pregel style *think like a vertex model*. Section 6 describes our experiences with our system, and Section 7 concludes the paper.

2 Related work

2.1 Graph Processing

The concept of using DSM in processing graphs has been explored by Shun, who created Ligra: a graph processing framework for shared memory. [22] GraphLab [17] is another system for parallel processing of Graphs, that could also extend to a shared memory model. Both of the aforementioned systems use a vertex-centric level of consistency, but fails to reason about the overall structure of the graph. Placement decisions can be made with a better view of graph structure.

2.2 What about COST?

3 API

4 Partitioning

Our partitioning algorithm aims to provide an optimal environment for the parallel processing of graphs. We leverage the METIS [16] library, an MPI implementation for parallel partitioning of large graphs, to determine an optimal partitioning for the system. Metis is built in and runs in C, and the corresponding Python binding, PyMetis [4] was used to determine the optimal graph partitioning it is read into memory.

Naturally, partitioning a graph will cause edges in the graph to cross partitions. Applying techniques seen in [8] and [7], we use a ”mirroring” technique for storing edges between partitions. For each edge that crosses partitions, only vertices on two separate partitions will be affected. The partition which contains the destination vertex will be considered the master of the given edge. All edges which connect to the master vertex from a different partition will create a ”mirror” vertex. These edges are stored in shared memory, such that they can be written to and read from different machines.

5 Evaluation

6 Discussion

7 Conclusion

References

[1] Facebook disaggregated rack, http://goo.gl/6h2ut.

[2] Hp the machine, http://www.hpl.hp.com/research/systems-research/themachine/.

[3] Intel rsa. https://software.intel.com/en-us/articles/intel-performance-counter-monitoring.

[4] Pymetis, https://mathema.tician.de/software/pymetis/.

[5] Seamicro technology overview, http://seamicro.com/.

[6] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan. One trillion edges: Graph processing at facebook-scale. *Proc. VLDB Endow.*, 8(12):1804–1815, Aug. 2015.

[7] C. et al. From "think like a vertex" to "think like a graph". In *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, 2015.

[8] T. et al. From "think like a vertex" to "think like a graph". In *Proceedings of the VLDB Endowment Volume 7 Issue 3*, VLDB '14, pages 193–204, 2013.

[9] 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*, OSDI'12, pages 17–30, Berkeley, CA, USA, 2012. USENIX Association.

[10] I. F. Haddad and E. Paquin. Mosix: A cluster load-balancing solution for linux. *Linux J.*, 2001(85es), May 2001.

[11] S. Han, N. Egi, A. Panda, S. Ratnasamy, G. Shi, and S. Shenker. Network support for resource disaggregation in next-generation datacenters. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, HotNets-XII, pages 10:1–10:7, New York, NY, USA, 2013. ACM.

[12] Z. Huang, W. Chen, and et al. Vodca: View-oriented, distributed, cluster-based approach to parallel computing. In *DSM WORKSHOP 2006, IN: PROC. OF THE IEEE/ACM SYMPOSIUM ON CLUSTER COMPUTING AND GRID 2006 (CCGRID06), IEEE COMPUTER SOCIETY*, 2006.

[13] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 59–72, New York, NY, USA, 2007. ACM.

[14] P. Keleher, A. L. Cox, S. Dwarkadas, and W. Zwaenepoel. Treadmarks: Distributed shared memory on standard workstations and operating systems. In *Proceedings of the USENIX Winter 1994 Technical Conference on USENIX Winter 1994 Technical Conference*, WTEC'94, pages 10–10, Berkeley, CA, USA, 1994. USENIX Association.

[15] A. Kyrola, G. Blelloch, and C. Guestrin. Graphchi: Large-scale graph computation on just a pc. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 31–46, Berkeley, CA, USA, 2012. USENIX Association.

[16] D. Lasalle and G. Karypis. Multi-threaded graph partitioning. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, IPDPS '13, pages 225–236, Washington, DC, USA, 2013. IEEE Computer Society.

[17] 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 Endow.*, 5(8):716–727, Apr. 2012.

[18] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM.

[19] C. Morin, R. Lottiaux, G. Vallee, P. Gallard, D. Margery, J.-Y. Berthou, and I. D. Scherson. Kerrighed and data parallelism: Cluster computing on single system image operating systems. In *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, CLUSTER '04, pages 277–286, Washington, DC, USA, 2004. IEEE Computer Society.

[20] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi. Naiad: A timely dataflow system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 439–455, New York, NY, USA, 2013. ACM.

[21] R. Power and J. Li. Piccolo: Building fast, distributed programs with partitioned tables. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 293–306, Berkeley, CA, USA, 2010. USENIX Association.

[22] J. Shun and G. Blelloch. Ligra: A lightweight graph processing framework for shared memory. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '13, pages 135–146, 2013.

[23] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 5:1–5:16, New York, NY, USA, 2013. ACM.

[24] 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*, GRADES '13, pages 2:1–2:6, New York, NY, USA, 2013. ACM.

[25] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.