Balanced Distributed Graph Processing

Anil Yelam, Audrey Randall

Abstract

In big data analytics, balanced execution of a computation is important for maximizing utilization of available resources in the cluster like CPU, Memory and Network. Various works in the past have looked at optimized execution of big data platforms like Apache Hadoop MapReduce, Apache Spark and other sort and shuffle based workloads. In this paper, we go on a similar quest for graph processing frameworks, which are equally commonly used in big data processing and has not been hitherto well studied. We discuss various factors that affect the balanced execution of a graph processing algorithms on a given cluster. Specifically, we focus on the effect of different graph partitioning methods on performance of Pagerank algorithm on a wide range of natural graphs. By both studying the properties of these graphs and results from the pagerank runs on Apache Giraph, we show that even the simplest of the partitioning schemes can have a significant effect on the performance.

1 Introduction

The growing popularity of technologies such as Internet of Things (IoT), mobile devices, smartphones, and social networks has led toward the emergence of "big data". Such applications produce not just gigabytes or terabytes of data, but soon petabytes of data that need to be actively processed. A large percentage of this growing dataset exists in the form of linked data, more generally, graphs, of unprecedented sizes. Graphs from today's social networks contain billions of edges while inter-connected data from millions of IoT sensors can generate graphs that are exponentially larger. This requires large-scale graph processing to analyze the data and generate useful statistics. Frequently applied algorithms to this end include shortest paths computations, different flavors of clustering, and different variations of page rank.

Traditional big data analytics frameworks like Hadoop MapReduce[6], Apache Spark[17], etc does not perform well for graph procesing. Previous studies [2], [9] have repeatedly shown that these frameworks are too general and does not make use of properties of graph algorithms that often exhibit poor locality of memory access, very little work per vertex, and a changing degree of parallelism over the course of execution. As a consequence, iterative graph processing systems started to emerge in 2010, starting with Google's Pregel[12] that uses Valiant's Bulk Synchronous Parallel (BSP) processing model for its computation, and promoted a "think like a vertex" notion for processing large graphs. Following that, there has been an explosion in distributed graph processing frameworks like Apache Giraph[7], GraphLab[11], PowerGraph[8], Stratosphese[1], Blogel[16], etc (to name a few) that offer different programming and computational models.

The importance of balanced execution of running large workloads like Sort on big clusters has been studied before (TritonSort[15]), including studies of big data frameworks like Spark (Osterhout et al.[13] and MapReduce (Themis[14]), however there hasn't been a lot of similar work for graph algortihms. Distributed graph processing platforms, like the ones above, make a lot of design choices like the programming model (vertex or edge centric), distributed coordination (synchronous or asynchronous), partitioning schemes (vertex or edge cuts), specializing for certain kinds of graphs, etc (which are discussed in detail in the next section) that influence the performance of the frameworks.

In this paper, we pick one of these design choices i.e., partitioning schemes that are used to divide input graph among workers, and study how different choices can affect the performance. Specifically, we focus on real-world graphs like social networks and web domains (made available Laboratory for Web Algorithmics[4][3]) and show some properties these (power law) graphs exhibit under two very simple but different paritioning schemes called Hash and Range Partitioning. Later, we run pagerank on these same graphs using a distributed graph processing framework called Apache Giraph[7] using the afforementioned partioning schemes and present our results.

The rest of the paper is structured as follows. In sections 2 and 3, we present background of some design choices made by graph processing platforms and rationale for choices we make for this paper. In sections 4 and 6, we present our partitioning schemes in detail, and theoretically study the effect of these schemes on selected input graphs. In section 5, we present results from our pagerank runs of Giraph.

2 Background

All about the graph processing frameworks and their computational models, basically a brief summary of [10]. And a bunch of related papers we have read, like comparison papers [9, 2] and some earlier graph processing systems these papers point to - such as Pregel[12], PowerGraph [**?**] and others.

Notes from Heidari et al. [10] Look at the paper for citations of all the statements below.

Q. Which algorithm to choose? Needs to be global, traversing the whole graph. Examples include page rank and connected components, degree distribution.

Q. Which programming model? Programming models used in existing frameworks (Section 3) - distributed vs shared memory architecture. While shared mem architectures work well for gigabtes of data (Scalability but at what cost paper), they don't scale to terabytes. There's also GPU and FPGA based acceleration proposed in some works, but we want to restrict ourselves to frameworks that run on commodity clusters.

Q. Which Programming abstraction?: 1. Vertex Centric: how it works and the shortcomings. Discuss some of the example frameworks. 2. Edge centric: how it works and the shortcomings. Discuss some of the example frameworks. 3. Component centric: how it works and the shortcomings. Discuss some of the example frameworks. 4. Others: Like MapReduce and other general purpose big-data processing frameworks.

Q. What sort of distributed coordination: 1. Synchronous - using barriers between iterations, makes life easier for programmer but sensitive to stragglers. 2. Asynchronous - better performance, but much complex to program and ensure error recovery. 3. Hybrid

Q. Input partitioning: 1. Static vs Dynamic: Do we reparition graph at runtime depending on algorithm or straggler issues. 2. Edge cut vs Vertex cut: How do we partition a non-uniform graph? What's best for the algorithm? This is one of the key design choices that network performance could affect. Usually frameworks make one of these choices to minimize number of edges or vertices between to nodes to save on network, but is there a better way assuming infinite network bandwidth? (Think more about this!). This is something we could vary in our experiments.

Q. Others: Disk vs Memory based, fault tolerance with detection or recovery where choices are either clear or not important for our purposes.

3 Discussion

Here goes the discussion on what choices we made for our implementation w.r.t all the choices we discussed in the previous section. We will go with global algorithm like pagerank, on input graphs with uniform vertex and edge distribution to start with. We will go with synchronous model inspite of some performance disadvantages as it makes programming easier and is widely used. Since the kind of paritioning (vertex centric, edge centric or graph centric) should not matter that much for very uniform graphs, we will start with the most popular one i.e., vertex centric model. Choosing uniform graphs should help alleviate the skewed straggler problems affecting the synchronous model as well. We will go with static partitioning since page rank keeps all the vertices active throughout the run.

Then we do a theoretical analysis on the kind of network performance we could get. Given nodes of CPU c, memory m and network n for an input graph g = (v, e) what would a perfectly written distributed page rank program with above characteristics give us in terms of network performance?

Argue that while all of the design choices we discussed in background and in the first paragraph affects how balanced the resource usage of the run is going to be, the thing that seems to matter significantly is how input graph is partitioned. For the purposes of this project, we will look at how partitioning choices will affect the resource usage balance of a graph algorithm, particularly pagerank. More details on the partitioning is provided in the following section.

Instead of implementing distributed page rank from scratch, we decided to pick a one of the many graph processing frameworks available in the wild. We pickled giraph [7] for these reasons: very much in use, open sourced, matches much of the design choices we picked in the first paragraph, seen many recent optimizations from facebook [5], easier to plug in the partitioners, and more? Downside: takes too much time and tuning to setup, only allows vertex partitioning, and more?

4 Input Partitioning

Here goes the partitioning choices we chose to implement, how and why for each of those - and what we expect in terms of resource usage.

Giraph only supports vertex partitioning, so we are limited to that for now.

- Random Vertex
- Random edge
-

5 Apache Giraph

Description of Apache Giraph[**?**]. And bunch of optimizations made by Facebook[**?**].

6 Results

6.1 Input Graph Characterisitcs

Detailed description of characteristics of the input graphs that we picked, such as its vertex degree distribution, and what we expect for this particular input.

6.2 Setting up Giraph

1. Look at all the settings file and decribe important settings 2. JNI error due to Java version mismatch 3. Lot of effort spent in adjusting getting the resource limits right - too many files to set configurations - no facilities for fine-tuned resource allocation - lot of out of memory exceptions due to tuning errors - incomprehensible error messages or errors hidden in remote corners of unknown log files - super-fragile, one change in setting causes an error in a completely unrelated place. - 8 hours from getting giraph built to getting the first job running without excepts 4.

6.3 Effect of different partition schemes

References

[1] ALEXANDROV, A., BERGMANN, R., EWEN, S., FREYTAG, J.-C., HUESKE, F., HEISE, A., KAO, O., LEICH, M., LESER, U., MARKL, V., NAUMANN, F., PETERS, M., RHEINLÄNDER, A., SAX, M. J., SCHELTER, S., HÖGER, M., TZOUMAS, K., AND WARNEKE, D. The stratosphere platform for big data analytics. *The VLDB Journal 23*, 6 (Dec. 2014), 939–964.

[2] AMMAR, K., AND ÖZSU, M. T. Experimental analysis of distributed graph systems. *Proc. VLDB Endow. 11*, 10 (June 2018), 1151–1164.

[3] Boldi, P., Rosa, M., Santini, M., and Vigna, S. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th international conference on World Wide Web* (2011), S. Srinivasan, K. Ramamritham, A. Kumar, M. P. Ravindra, E. Bertino, and R. Kumar, Eds., ACM Press, pp. 587–596.

[4] Boldi, P., and Vigna, S. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)* (Manhattan, USA, 2004), ACM Press, pp. 595–601.

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

[6] Dean, J., and Ghemawat, S. MapReduce: Simplified data processing on large clusters. In *Symposium on Operating System Design and Implementation (OSDI)* (San Francisco, CA, USA, December 2004).

[7] GIRAPH. http://giraph.apache.org/. accessed: 2019-05-03.

[8] Gonzalez, J. E., Low, Y., Gu, H., Bickson, D., and Guestrin, C. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2012), OSDI'12, USENIX Association, pp. 17–30.

[9] Guo, Y., Biczak, M., Varbanescu, A. L., Iosup, A., Martella, C., and Willke, T. L. How well do graph-processing platforms perform? an empirical performance evaluation and analysis. In *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium* (Washington, DC, USA, 2014), IPDPS '14, IEEE Computer Society, pp. 395–404.

[10] Heidari, S., Simmhan, Y., Calheiros, R. N., and Buyya, R. Scalable graph processing frameworks: A taxonomy and open challenges. *ACM Comput. Surv. 51*, 3 (June 2018), 60:1–60:53.

[11] Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., and Hellerstein, J. M. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow. 5*, 8 (Apr. 2012), 716–727.

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

[13] Ousterhout, K., Rasti, R., Ratnasamy, S., Shenker, S., and Chun, B.-G. Making sense of performance in data analytics frameworks. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2015), NSDI'15, USENIX Association, pp. 293–307.

[14] Rasmussen, A., Lam, V. T., Conley, M., Porter, G., Kapoor, R., and Vahdat, A. Themis: An i/o-efficient mapreduce. In *Proceedings of the Third ACM Symposium on Cloud Computing* (New York, NY, USA, 2012), SoCC '12, ACM, pp. 13:1–13:14.

[15] Rasmussen, A., Porter, G., Conley, M., Madhyastha, H. V., Mysore, R. N., Pucher, A., and Vahdat, A. Tritonsort: A balanced and energy-efficient large-scale sorting system. *ACM Trans. Comput. Syst. 31*, 1 (Feb. 2013), 3:1–3:28.

[16] Yan, D., Cheng, J., Lu, Y., and Ng, W. Blogel: A block-centric framework for distributed computation on real-world graphs. *Proc. VLDB Endow. 7*, 14 (Oct. 2014), 1981–1992.

[17] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., and Stoica, I. 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* (Berkeley, CA, USA, 2012), NSDI'12, USENIX Association, pp. 2–2.