An Elasticity Study of Distributed Graph Processing

Au, Sietse; Uta, Alexandru; Ilyushkin, Alexey; Iosup, Alexandru

DOI
10.1109/CCGRID.2018.00062

Publication date
2018

Document Version
Accepted author manuscript

Published in
2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)

An Elasticity Study of Distributed Graph Processing

Sietse Au*, Alexandru Uta†, Alexey Ilyushkin*‡, and Alexandru Iosup *†
*TU Delft, †Vrije Universiteit Amsterdam, and ‡University of Amsterdam, the Netherlands
{s.au, a.s.ilyushkin}@tudelft.nl, {a.uta, a.iosup}@vu.nl

***Abstract*—Graphs are a natural fit for modeling concepts used in solving diverse problems in science, commerce, engineering, and governance. Responding to the variety of graph data and algorithms, many parallel and distributed graph-processing systems exist. However, until now these platforms use a *static* model of deployment: they only run on a pre-defined set of machines. This raises many conceptual and pragmatic issues, including misfit with the highly dynamic nature of graph processing, and could lead to resource waste and high operational costs. In contrast, in this work we explore a *dynamic*, *elastic* model of deployment. To conduct an in-depth elasticity study of distributed graph processing, we build a prototype, JoyGraph, which is the first such system that implements complex, policy-based, and fine-grained elasticity. Using the state-of-the-art LDBC Graphalytics benchmark and the SPEC Cloud Group's elasticity metrics, we show the benefits of elasticity in graph processing: improved resource utilization, and aligned operation-workload dynamicity. Furthermore, we explore the cost of elasticity in graph processing. We identify a key drawback: although elasticity does not degrade application throughput, graph-processing workloads are sensitive to data movement while leasing or releasing resources.**

I. Problem Statement and Conceptual Contribution

Modern graph-processing systems use sophisticated techniques to exploit the massive parallelism of scale-up machines [1] or the large-scale resources of distributed systems [2]. However, the use of the *static* deployment model has negative consequences. Conceptually, graph applications are a poor fit for static, fixed-size infrastructure, as they have often iterative, but highly irregular workloads [3]. Previous techniques do not focus on a *dynamic model of deployment*, where infrastructure can grow and shrink to match the needs of the graph-processing application. In contrast, in this work we explore the use of *elasticity*, that is, the ability to lease and release machines dynamically, as a key-feature of a new class of graph-processing systems. **Our study is the first to explore the implications of elastic scaling policies, both growing and shrinking resources, on distributed graph processing. We give important performance insights, and quantify the cost and benefits of elasticity in graph processing.**

For public infrastructure, elasticity could reduce operational costs by *reducing resource waste* [4], [5], and improve the ability to meet Quality-of-Service guarantees (such as high performance and availability) by using appropriate autoscaling policies [6]. For private infrastructure, elasticity could *reduce operational costs* by increasing resource utilization [7], or throttle throughput to meet demands across applications [8].

Typically, when making a case for elasticity [9] in graph processing, only graph-related metrics are considered, such as *active vertices*, or *messages exchanged* per super-step. Many complementary studies present an analysis on how the number of active vertices affects the algorithm efficiency: direction-optimizing BFS [10] is a technique entirely motivated by the variable number of active edges per iteration; Mizan [11] analyzes runtime per iteration and addresses the large imbalance; GraphReduce [12] leverages the observed variability in the number of active vertices (frontier size) for two datasets and three algorithms; high workload imbalance appears even between multiple designs and implementations of the same algorithm running the same workload [3]; etc. In contrast, we also analyze the impact of workload imbalance on the consumption of *system-level* resources.

II. Elastic Graph Processing System

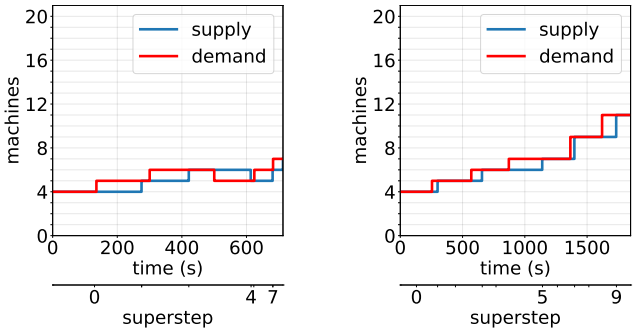
For our exploratory experiments aiming to understand the performance of graph processing under elasticity, we use our prototype, JoyGraph, which adapts the classic mechanisms of elasticity to graph processing. We also propose three new autoscaling policies for deciding when to grow and shrink the infrastructure, and new mechanisms for nested data repartitioning without loss.

JoyGraph consists of three key components: The *Master* node, which orchestrates the execution of the graph processing workloads on the *worker* nodes; *Worker* nodes, which execute super-steps of the graph processing workloads; and *Storage*, a distributed file system which is used to load the input graph and to store the generated results. JoyGraph supports a Pregel-like programming model.

Policy-based elasticity for graph processing has not been explored until now. Recently, we have seen that generic autoscaling policies perform well with scientific workloads, almost on-par with workload-specific policies [6]. Encouraged by such results, we apply autoscaling policies to graph processing.

Therefore, for JoyGraph, we evaluate two types of elastic autoscaling policies: *system-level* policies, and *generic* autoscaling policies. The former are simple autoscaling policies that take into account system level metrics, such as CPU-load, wallclock time, and network-load. The latter are general autoscaling policies, originally used successfully for autoscaling web applications.

We consider three system-level elastic-scaling policies: *CPU Load Policy (CPU)*: The average worker CPU-load is used to generate a new partition function to redistribute graph vertices. *Wallclock Time Policy (WCP)*: Computes the wallclock time of each worker on each super-step. Workers that take more time



(a) BFS + WCP. (b) PR + WCP.

Fig. 1. The JoyGraph elastic policy WCP applied to BFS and PR. The horizontal position of supersteps reflects the duration of elasticity operations.

than the average will distribute part of their vertices to new workers. *Network Load Policy (NP)*: Computes the amount of network data sent by each worker on each super-step.

Moreover, we consider five *g*eneric autoscaling policies (based on the previous study by Ilyushkin et al. [6]): *React* is a simple *reactive* policy, in which the supply follows the changes in demand, and does not predict future demand. *AKTE* is a hybrid *adaptive* policy. It follows the changes in demand, and also predicts future demand based on the change in the arrival rate. *ConPaaS* is a web-application autoscaler that responds to changes in throughput at fixed time intervals. Its predictor uses time series analysis to forecast future demand. *Reg* is a regression-based autoscaler that scales up by using a reactive policy. Scaling down is based on predictions achieved by using a regression model trained on past data. *Hist* is a histogram-based controller that employs a queuing-model to determine how many resources to deploy.

III. Experimental Setup

For our experiments, we used Datagen scale factor 1000 graph (Datagen-1000) as input. The graph was generated using the Datagen tool [13]. As the workload we implemented the following algorithms: breadth-first search (BFS), page-rank (PR), weakly-connected components (WCC) and single-source shortest paths (SSSP). These algorithms were ran statically, on both JoyGraph, our proposed solution, and the well-established Giraph system [14]. To provide evidence of extensive resource variability, during each algorithm run, we measure: the number of active vertices, CPU load, memory utilization, and wallclock time. We ensure that all the results are correct using the Linked Data Benchmark Council (LDBC) Graphalytics [15] validation tool.

IV. Experimental Results

Using the LDBC Graphalytics, and the SPEC Cloud Group's elasticity metrics we assessed the benefits and costs of elasticity. Our results show that even algorithms that do not suffer from active vertices variability are impacted by significant variability in system-level metrics (e.g., CPU load, memory, wallclock time). For example, Figure 1a shows that BFS which exposes significant variability in the number of active vertices, has low variability with WCP. At the same time, Figure 1b shows that PR which does not exhibit variability in the number of active vertices exposes high variability with WCP. The consequences can lead to significant imbalance in the utilization of resources. Therefore, *active vertices* is a metric that cannot be proportionally translated into *system-level* metrics. This indicates there is a need for dynamic mechanisms that can elastically grow and shrink the number of active resources to run graph-processing. We find that elasticity offers an interesting trade-off between performance and fine-grained resource management in comparison to the static state-of-the-art alternatives. We characterize with many elasticity-related metrics the performance of the autoscaling policies and find that they offer distinct trade-offs.

V. Ongoing Work

We are just beginning to understand the potential elasticity holds for graph processing, and for big data processing in general. We are currently extending this work in the following directions: (i) exploring the situation when elasticity becomes particularly challenging when multiple customers compete for the same infrastructure, (ii) re-evaluating and extending JoyGraph for a more extensive set of (emerging) graph-processing algorithms, (iii) addressing property graphs, mutable graphs, and (dynamic) graph-processing workflows.

Acknowledgments

Supported by Vidi MagnaData (NL) and Oracle Labs (USA).

References

[1] J. Zhong and B. He, "Medusa: Simplified graph processing on GPUs," *TPDS*, vol. 25, no. 6, pp. 1543–1552, 2014.
[2] J. E. Gonzalez *et al.*, "GraphX: Graph processing in a distributed dataflow framework." in *OSDI*, vol. 14, 2014, pp. 599–613.
[3] M. Verstraaten, A. L. Varbanescu, and C. de Laat, "Quantifying the performance impact of graph structure on neighbour iteration strategies for pagerank," in *Euro-Par Workshops*, 2015, pp. 528–540.
[4] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *ICDCS*, 2011, pp. 559–570.
[5] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Supercomputing*, 2011.
[6] A. Ilyushkin *et al.*, "An experimental performance evaluation of autoscaling policies for complex workflows," in *ICPE*, 2017, pp. 75–86.
[7] A. Uta, A. Sandu, S. Costache, and T. Kielmann, "MemEFS: an elastic in-memory runtime file system for escience applications," in *e-Science*, 2015, pp. 465–474.
[8] B. Nicolae, P. Riteau, and K. Keahey, "Bursting the cloud data bubble: Towards transparent storage elasticity in iaas clouds," in *IPDPS*, 2014.
[9] S. Heidari, R. N. Calheiros, and R. Buyya, "iGiraph: A cost-efficient framework for processing large-scale graphs on public clouds," in *CCGrid*, 2016, pp. 301–310.
[10] S. Beamer, K. Asanovic, and D. A. Patterson, "Direction-optimizing breadth-first search," in *SC*, 2012, p. 12.
[11] Z. Khayyat *et al.*, "Mizan: a system for dynamic load balancing in large-scale graph processing," in *EuroSys*, 2013, pp. 169–182.
[12] D. Sengupta, S. L. Song, K. Agarwal, and K. Schwan, "Graphreduce: processing large-scale graphs on accelerator-based systems," in *Supercomputing*, 2015, pp. 28:1–28:12.
[13] O. Erling *et al.*, "The LDBC social network benchmark: Interactive workload," in *SIGMOD*. ACM, 2015, pp. 619–630.
[14] A. Ching and C. Kunz, "Giraph: Large-scale graph processing infrastructure on Hadoop," *Hadoop Summit*, vol. 6, no. 29, 2011.
[15] A. Iosup *et al.*, "LDBC Graphalytics: A benchmark for large-scale graph analysis on parallel and distributed platforms," *PVLDB*, vol. 9, no. 13, pp. 1317–1328, 2016.