Dynamic Elasticity for Distributed Graph Analytics

Kenrick Fernandes, Rami Melhem
Computer Science Department
University of Pittsburgh, Pittsburgh, USA
{kenrick, melhem}@cs.pitt.edu

Mohammad Hammoud
Computer Science Department
Carnegie Mellon University, Doha, Qatar
mhhamoud@cmu.edu

***Abstract*—Graph data analytics has received a great deal of attention in computing theory and systems research over the past decade. This paper proposes, implements and evaluates a distributed graph analytics system that scales resources up and down elastically to address the challenge of dynamic workload changes in analytics jobs. We show that by relying on hash partitioning, a simple and scalable method, and dynamically changing the placement of partitions from the already-partitioned graph, we can improve performance. Hence, the system eliminates the need to burden the user with resource acquisition and management decisions. We compare the system's performance against a static partition placement to set expectations for real-world applications. Early evaluations show our system can achieve up to 2.4x speedup for certain applications.**

***Index Terms*—parallel and distributed systems; cloud computing; graph analytics; data analytics; big data; performance analysis**

I. Introduction and Motivation

Graph or network-structured data topics have seen a marked rise in popularity recently, with a plethora of work directed at both the theoretical and systems aspects of processing graphs [1]. In this work, we focus on addressing a key challenge inherent in running these workloads: the change in workload patterns over the job duration [2]. There are two paths to dealing with this challenge: partition the graph prior to running the job to deal with the issue e.g. [3], or change the partitioning dynamically while the job is running e.g. [4]. The former solution is not guaranteed to perform well, since the specific times and duration of these patterns are not predictable without having already run the job. The latter typically requires an expensive accounting of the current graph partitioning to decide how to change it for the new scenario.

The key to practical feasibility in computing systems is finding a balance between the overhead of data movement and the benefits for computation time and/or cost. We propose a novel means of reaching this balance through a two-stage process: first, we use a randomized partitioning scheme (e.g. hash partitioning) which has the benefits of being simple and scalable in both the graph and resource pool sizes; and second, we change the placement of the partitions formed from the simple initial partitioning in response to changes in workload.

We focus our efforts on the common use case scenarios in real-world analytics with comparatively small cluster sizes. There is often a focus on large clusters in prior work, such as those encountered at large companies; inquiry into the day-to-day efforts in data processing by industry [5] [6] show that smaller clusters of fifty nodes or less are more common and are used to analyze a few terabytes of data. In these scenarios, the Hadoop ecosystem, which has commercial distributions and includes a cluster resource management layer, can be leveraged in combination with resource virtualization to achieve granular use of computing resources. The ability to achieve resource efficiency without sacrificing completion time or adding process complexity is key to smooth operation. The resource management capabilities provided by these systems are well positioned to leverage the popularity of computing-as-a-service through cloud providers, where resources can be cheaply and economically acquired and released. We take advantage of these capabilities to acquire and release resources on-the-fly by designing and implementing a resource-elastic system to solve the problem of dynamic behavior of graph analytics jobs.

The contributions of this short paper are:

- an evidence-based case for leveraging simple, scalable partitioning configurations on real-world workloads (Section III)
- a design, implementation and initial evaluation of a resource-elastic, distributed graph processing system on top of a popular framework (Section IV)

II. Background and Related Work

Cloud-based Graph Processing Systems There are a number of distributed systems for graph processing [1] in which evaluations focus on processing time and scalability alone when evaluating in a cloud computing context. However, changing the system design to enable scalability can lead to resource efficiency issues due to large system overheads [7].

Apache Giraph We build our system on top of Apache Giraph [8], a popular and mature distributed graph processing system that is open-source. Originally created at Yahoo!, it is currently being shepherded by a diverse group of contributors including engineers at Facebook, LinkedIn, Twitter, Hortonworks and academic institutions. Performance improvements have been integrated over multiple releases enabling growth in the size and speed of large scale graph processing [8] [9]. Giraph uses a Bulk Synchronous Parallel (BSP) processing model, where computation proceeds in rounds called *supersteps*. In each round, vertices receive messages sent during the previous superstep, perform computations based on a user-defined computation function, and send messages which are made available to the target vertices in the following superstep.

Static Graph Partitioning Partitioning the graph prior to distributed processing and committing to this partitioning for every superstep relies on methods from static graph partitioning, which has long been an area of research. However, previous work [2] has shown that theoretically optimal algorithms do not necessarily translate to good real-world performance, since the behavior of graph algorithms changes over time and is coupled to the graph structure. Currently, analytics systems employ strategies such as over-partitioning (e.g.: Giraph produces n^2 partitions, for n machines) or sophisticated graph partitioning algorithms [3]. However, the

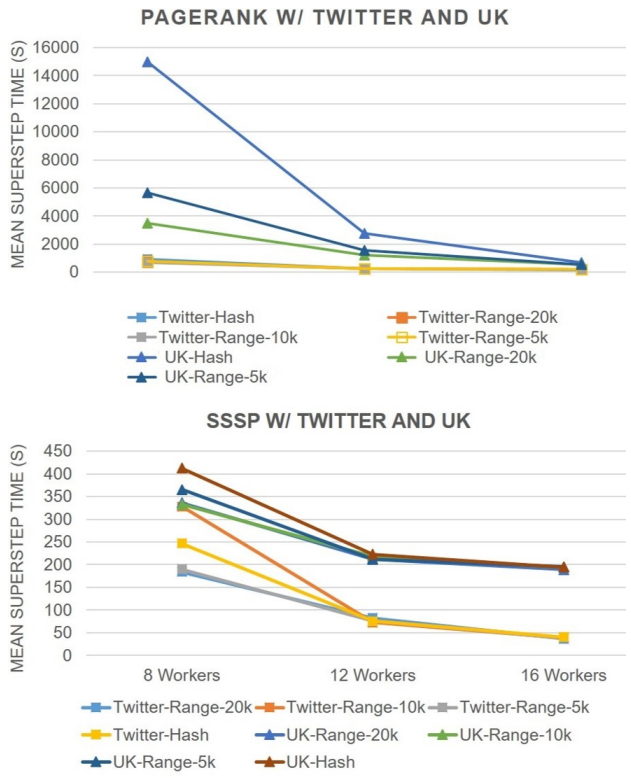

Fig. 1. Selected Performance Comparisons of Hash and Range Partitioning on Large Graphs for PageRank and Single Source Shortest Path Workloads

effects of placement strategies on top of these partitionings have not been thoroughly evaluated in the literature.

Dynamic Graph Re-partitioning Previous works have focused on re-partitioning and migration at the vertex level e.g. [3] [4]. However, these works involve complex re-partitioning schemes and do not consider migration at a coarser granularity than a vertex, which is our focus.

Elastic Graph Processing Systems Some recent works have begun to leverage resource elasticity and adapt graph processing systems to cope with changing load. However, [10] do not perform experimental evaluations in real-world environments but rely on mathematical calculations and depend on recorded behaviors from running jobs apriori. They also rely on METIS, a high quality but challenging to scale partitioning algorithm. [11] focus only vertex data movement.

Additionally, we implement and test our design in a widely used, industrial-strength graph processing system to determine if there are any benefits to be gained when using a system already engineered for performance. The benefits of elasticity have been leveraged for some big data processing frameworks such as Apache Storm [12], indicating the feasibility of resource elasticity in real-world cloud processing.

III. Performance Impact of Partitioning and Cluster Size

A key choice in graph partitioning for real-world applications is between simple, scalable algorithms and complex, expensive ones that are potentially better performing. Our key contribution in this work proposes relying on a simple partitioning algorithm initially and then changing the placement of the resulting partitions. To establish an optimal baseline among different simple and scalable partitionings, we explore here how a label-informed *range* partitioning with different key space sizes performs against a round-robin *hash* partitioning. A traditional hash partitioning relies on hashing the vertex identifiers or label numbers by the number of workers to find the partition. When studying the graph structures of multiple datasets, we observed that the neighbors of a chosen node had labels contiguous or very close to that of the chosen node, indicating a dataset creation process similar to breadth-first-search to create the labels, rather than a depth-first crawling or random assignment. Hence, we can speculate that a range partitioning could perform better due to less communication load between workers, and potentially more balanced computation patterns. While it is often assumed that hash partitioning leads to vertex-balanced partitions, this is dependent on contiguity in the labeling of the underlying graph. We found that over ten percent of the total labels in the contiguous sequence are unassigned in some cases.

The summarized results for the PageRank and Single Source Shortest Path algorithms with two large directed graphs in Figure 1 demonstrate this is indeed the case for the UK-2007-05 graph (web graph with 105M vertices and 3.73B edges) but the performance impact becomes negligible as cluster size increases. Selected results for 8, 12 and 16 workers are shown for visual clarity. For the Twitter-2010 social graph (41.6M vertices and 1.46B edges) and the USA Road network (an undirected graph with 23.9M vertices and 58.3M edges; plot omitted due to a space constraint), the performance of different partitionings also converges as the cluster size increases. Our key takeaways: first, we can rely on a simple and scalable hash partitioning which takes constant time per vertex when using larger cluster sizes without the complexity of tuning range sizes; second, since cluster size has a large impact on performance, finding the right cluster size is critical.

Motivated by these results, we explored the relationship between the number of partitions, the number of threads, and the partitioning scheme. We omit figures due to a page constraint. Our experiments consisted of varying the unit of work i.e. the partition size, the partition count and the processing thread count, using the PageRank, Single Source Shortest Path and Graph Coloring algorithms with the UK-2007-07, Twitter-2010 and USA Road graphs. Disk input/output measurements were not included. Our results show that a larger number of partitions run on a closely matching number of available threads performs significantly better than fewer numbers of large partitions. This reflects design choices wherein partitions are processed by a single-threaded only, and shows we can achieve a significant gain in performance by assigning concurrency parameters in a system-aware manner.

IV. Resource Awareness and Elasticity for Distributed Graph Processing

In this section, we discuss the architecture and implementation of our elastic system on top of Apache Giraph and provide initial experimental results to demonstrate its potential. A resource-elastic system design must address three questions:

- Q1: When to acquire and release resources?
- Q2: What data to move?
- Q3: How to move data?

A. Q1: When to Acquire and Release Resources - Elasticity Policy

We intentionally separate the decision to scale up or down the cluster from the placement of partitions on the scaled cluster. This enables independent decision making and evaluation of the quality of the load balancing algorithms separately from the benefits of elasticity. Based on our analysis, we choose to decrease the cluster size when the ratio of communication to computation is high and increase it in the opposite case. Decisions are made based on the changes across pairs of past supersteps to amortize the cost of data movement observed in our experiments, and also to avoid making potentially expensive decisions hastily. The full algorithm for elastic resource scaling, to be run at every superstep, is parametrized below.

Parameters:

- *history-window* α: The number of past supersteps to consider for resource increment/decrement decisions
- *incr-threshold* μ and *dec-threshold* ν: Cluster size can greatly affect job performance (Section III) and these parameters represent the change threshold to consider for incrementing or decrementing resources respectively. Tuning them allows the user to tradeoff between aggressive and more cautious resource management.
- *step-size* δ: The amount of resources to acquire or release when the thresholds are met i.e. the elasticity step size
- *load-metric* λ_i: The desired load metric to measure for superstep i, for example, the ratio of communication volume to active vertex count

Superstep Algorithm:

$i \leftarrow$ current superstep number (starting from 0)
$\lambda_i \leftarrow$ record active vertex count V_i and communication volume C_i for this superstep
if $i \geq \alpha$ **then**
  if $\forall j, (i-\alpha-1) \leq j \leq i$, $(\lambda_j - \lambda_{j-1}) \geq \mu$ **then**
    increase number of workers by δ
  else if $\forall j, (i-\alpha-1) \leq j \leq i$, $(\lambda_j - \lambda_{j-1}) \leq \nu$ **then**
    decrease number of workers by δ
  end if
end if
{calculate placement using Dynamic Rebalance algorithms below}

B. Q2: What Data to Move - Dynamic Rebalance Algorithms

For a superstep to proceed, two types of data are required: the vertex and edge data and associated values, and the messages sent between vertices. In this subsection, we treat these as a single unit per partition and describe the algorithm we rely on to choose the appropriate placement. Details of the individual data movements are detailed in the next subsection.

As a first approach to tackle the problem, we treated workers as fixed-capacity bins and relied on well-understood approximation algorithms for the bin packing problem such as first fit decreasing packing, observing that this did not provide performance benefits. This also led to the problem of needing to allow creation of new bins when capacity was exceeded, thus nullifying the clean separation of partition placement from the elastic decision. Hence, we decided to lift the capacity constraints on the bins. We did so after observing two key behaviors: first, we did not see any significant straggler effects in our experiments on the cluster; second, we studied the distributions of vertices and edges from these simple partitionings and found that having multiple partitions on the workers greatly reduced disparity, leading to only a small percentage difference in the edge and vertex counts overall between workers. Hence we leverage a simple heuristic algorithm with a different "size" value or "balancing metric" for each partition. The balancing metric can be the communication volume, active vertex count, or another user-chosen metric. Partitions are first sorted by the relevant balancing metric in a decreasing order. The workers are placed in a min-heap based on the balancing metric, and we can initialize constraints based on their load and processing capacity if necessary. For example, if some machines have twice as much processing power as the others, they can be given an appropriately scaled capacity. Then, the partitions are placed one after another on the workers in the heap (we note that some code for this algorithm based on edge counts was present in the codebase).

An advantage of this approach is that metrics can be chosen flexibly and incorporate prior knowledge about job structure e.g. if the algorithm does not alter the graph structure during the job, then we can choose a metric such as vertex/edge count to ensure that migration only happens once if necessary, avoiding unnecessary data movement. To alleviate the issue of needing to move a large number of partitions around when scaling up or down, we also introduce an alternative algorithm which moves the least heavy (in terms of balancing metric) partition from each worker to the new worker.

C. Q3: How to Move Data - Migration and Elasticity Mechanisms

In Giraph, graph structure data including vertices and edges as well as any associated data values are stored in a *Partition* data structure in a *PartitionStore* on the assigned worker. Similarly, messages are stored in a *MessageStore* data structure. At the beginning of each superstep, centralized computation on the master runs our algorithms described above and decides future partition placements. These placements are then shipped to all workers prior to any computation or vertex messaging. After each worker receives these placements, data can be moved from the existing worker to the destination worker.

A naive implementation of data movement would migrate messages and vertex data at the very beginning or end of a superstep. However, we observed that this superstep structure requires the movement of unprocessed messages, which incurs significant overhead. To mitigate this, we made two modifications which are shown alongside Giraph's unaltered system structure in Figure 2. First, after placements are updated at the beginning of the superstep, messages for future supersteps are sent directly to the worker where the partition will reside. Second, partitions are migrated immediately after processing so that data communication can be overlapped with ongoing computations. A key advantage of this approach is that data movement can take place in parallel and independently across workers without requiring specific coordination or signaling.

Giraph measures time and data movement metrics and collects these on the master node after each superstep. Our system then performs additional decision making (including examining additional metrics we add) to alter partition placements and decide whether to add or remove workers. As an artifact of our particular experimental setup in which Giraph runs on Hadoop 1, we require the system to acquire the maximum number of workers that will be used through the job

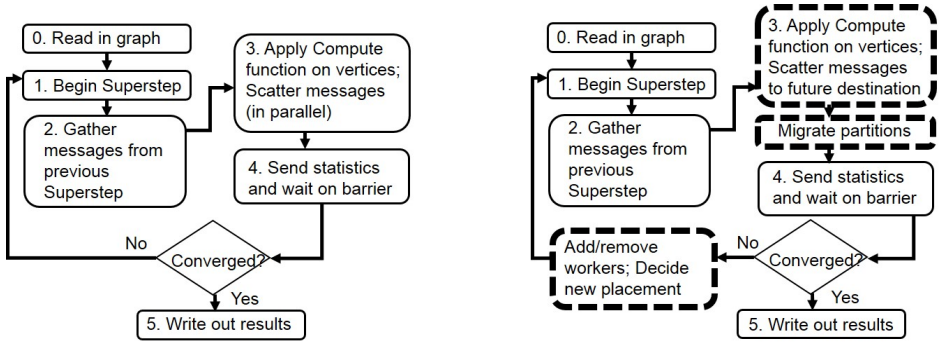

Fig. 2. Our system superstep structure (right) with changes from Giraph's superstep structure (left) shown in dashed boxes

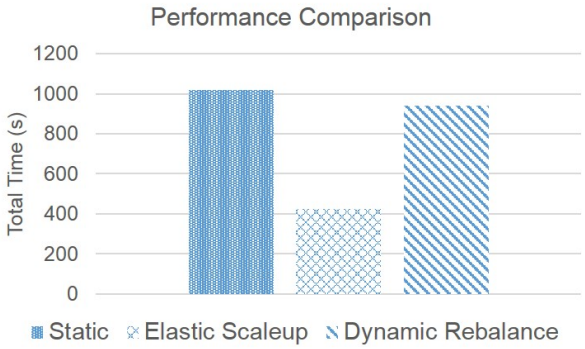

Fig. 3. System Gains on Single Source Shortest Path Algorithm with UK-2007-05, a Large Web Graph with 105M vertices and 3.73B edges

prior to launching, since the underlying resources cannot be acquired and released dynamically in Hadoop 1. We verified in our experiments that these idle acquired resource containers do not affect the running time.

V. Experimental Evaluations

We perform an initial evaluation of our prototype system on a twenty node virtual machine cluster with one Giraph worker to a virtual machine. Results shown are for in-memory processing time over all computation superstep excluding disk input/output time. We evaluate our proposed algorithms for their load-balancing ability with and without elasticity to discover their impact separately. The baseline for comparison (static) is a round-robin placement of partitions onto the workers. Figure 3 shows a performance comparison for the strategies on the Single Source Shortest Path algorithm with the UK-2007-05 graph. This consists of the *static* baseline, a *dynamic rebalance* data migration using edge count as the balancing metric (triggered on superstep 1 deliberately; 1.1x speedup), and an *elastic* migration with a single additional worker (2.4x speedup). Note that the dynamic rebalance strategy provides limited benefit once the cost of data movement is accounted for, whereas our elastic scaleup strategy performs significantly better. For the results described here, we set the following parameter values: history-window=3, incr-threshold & dec-threshold=20%, and step-size=1.

In our early evaluations, we also used other algorithms such as K-Core and smaller graphs such as Twitter-2010 and Arabic-2005 (22.7M vertices, 639M edges) (results omitted due to a page constraint). For the smaller graphs, the cost of performing data migrations required overtakes the benefits accrued over the following supersteps. We also performed initial experiments with vertex count rebalancing, and saw little to no benefit compared to the edge count rebalancing, which can be explained in part by the uneven distribution of edges from the hash partitioning. Our evaluations showed that unless the job in question has an adequate duration to offset the cost of data migration, there is not enough benefit gained for either balancing or elasticity. Hence, the benefits of our work lie primarily in analytics for large scale graphs and medium to long-running jobs.

VI. Conclusions and Future Work

The initial experimental evaluation of our prototype indicates that the most promising applications lie in very large graphs, and most critically that we can realize benefits even for relatively small cluster sizes. In future work, we will perform an evaluation on public clouds to incorporate a cost-benefit evaluation under performance variance. In addition, we will evaluate a broader set of both partitioning and analytics algorithms in our experiments.

Acknowledgments

This publication was made possible by NPRP grant #7-1330-2-483 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

References

[1] R. R. McCune *et al.*, "Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing," *ACM Computing Surveys*, 2015.
[2] Z. Shang *et al.*, "Catch the wind: Graph workload balancing on cloud," in *IEEE 29th International Conference on Data Engineering*, 2013.
[3] C. Martella *et al.*, "Spinner: Scalable graph partitioning in the cloud," *IEEE 33rd International Conference on Data Engineering*, 2017.
[4] N. Xu *et al.*, "Loggp: a log-based dynamic graph partitioning method," *Proceedings of the VLDB Endowment*, 2014.
[5] B. Graham and M. Rangaswami, "Do you hadoop? a survey of big data practitioners," *Sand Hill Group, San Francisco, CA, USA*, 2013.
[6] A. Nadkarni and L. DuBois, "Trends in enterprise hadoop deployments," *Survey, IDC*, 2013.
[7] F. McSherry *et al.*, "Scalability! but at what cost?" in *HotOS*, vol. 15. Citeseer, 2015.
[8] A. Ching *et al.*, "One trillion edges: Graph processing at facebook-scale," *Proceedings of the VLDB Endowment*, 2015.
[9] Facebook Code, "A comparison of state-of-the-art graph processing systems," 2016. [Online]. Available: https://code.facebook.com/posts/319004238457019/a-comparison-of-state-of-the-art-graph-processing-systems/
[10] R. Dindokar *et al.*, "Elastic partition placement for non-stationary graph algorithms," in *16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2016.
[11] M. Pundir *et al.*, "Supporting on-demand elasticity in distributed graph processing," in *IEEE International Conference on Cloud Engineering*, 2016.
[12] N. Katsipoulakis *et al.*, "Ce-storm: Confidential elastic processing of data streams," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2015.