Investigating and Modeling Performance Scalability for Distributed Graph Analytics

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

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

***Abstract*—Distributed systems for graph processing continue to attract attention in the literature, driven by the power of the underlying computational model and the availability of open-source software. Among the biggest barriers to achieving high resource efficiency are time and money-consuming system parameter tuning for cluster size. In this work, we undertake a comprehensive experimental study to characterize and then model the scaling behaviors of multiple graph analytics benchmarks in a production-grade distributed system, Apache Giraph. We show that our theoretical model can leverage existing domain knowledge in an interpretable fashion and accurately predict optimal choices even for previously unseen clusters. Our primary conclusions are that despite using randomized hash partitioning for work distribution, two factors - dynamic algorithm behavior and graph dataset skew - are the most important, respectively, for predicting scalability. Under the same regime, different numbers of partitions and memory optimizations have little impact on performance.**

***Keywords*-parallel and distributed systems; cloud computing; graph analytics; data analytics; big data; performance analysis; predictive modeling;**

I. Introduction

Graph data management and processing systems have been proliferating for almost a decade now and the abundance of specialized work has led to the production of meta-analyses comparing systems along dimensions such as communication, execution models and partitioning[1][2][3]. The goal of such meta-analyses is often to guide a user/developer in making the right choice that best suits their particular needs. These systems require user-specified parameters including cluster size, memory capacity, threads devoted to networking and computation, and software optimization settings. The ambiguity inherent in choosing such parameters limits the accessibility of these systems, especially in distributed settings for analyzing big data where significant parameter-tuning efforts are already required. The end result: making the right choices of resource unit size (cores, memory, and network bandwidth) and cluster size are challenging, and seemingly only admit solutions requiring costly up-front exploration and tuning.

For popular graph processing systems, the evaluation of system scalability on commodity clusters or cloud environments is practically de-facto[4][5][6]. The accessibility of cloud computing resources through large providers like Amazon EC2 has helped drive the democratization of cloud computing. Popular systems such as Apache Giraph[7] are built on top of cluster resource management frameworks such as Hadoop YARN[8] which provide convenient programming abstractions for cluster resources. Careful management of these resources is essential to control monetary costs when executing on pay-as-you-go cloud infrastructure. Since small clusters running on virtualized resources represent a large fraction of real-world use cases[9][10], it is essential to understand the impacts of parameter tuning in this environment. In this work we conduct and analyze the results of extensive experiments on Apache Giraph, one of the few production-grade systems that also operates as part of the widely-used Hadoop ecosystem, running on a cloud infrastructure. Using rigorous statistical methodology, we analyze evidence leading to a number of conclusions regarding the impact of system parameters. In addition, we develop a theoretical model using domain knowledge to predict performance and test it on real-world data.

In summary, the primary contributions of this paper are:

- Characterization of scaling behaviors of graph analytics workloads, distributed using randomized hash partitioning, on a production-grade distributed graph processing system, Apache Giraph (Section III)
- A domain-inspired, interpretable model to set expectations for the potential benefits of optimal resource choices, along with recommendations for system parameter choices (Section IV)
- Demonstration of the model's ability to generalize and accurately predict optimal cluster size choice, using previously unseen scenarios (Section V)

II. Background and Related Work

Graph Processing in the Wild: Much work in the data analytics literature focuses on solving large-scale problems, which only large technology companies like Google, Facebook, and Yahoo! face on a regular basis. However, recent industry surveys[9][10] show that the much more common use cases involve relatively small cluster sizes (typically less than 50 nodes) and a few terabytes of data. In such situations with tight resource constraints, efficient resource usage is critical. Resource usage and fungibility for a group of computing resources are typically achieved by using a layer of software abstraction such as a cluster management framework like Hadoop YARN[8], which provide feature-rich abstractions for programming distributed clusters.

Background on Apache Giraph: Pregel[11] was among the first works to provide a simple programming model for enabling computation on large graph data. Pregel-like systems apply the Bulk Synchronous Parallel (BSP) computation model to individual vertices of an input graph and are often called Think Like A Vertex (TLAV) systems. Apache Giraph[7] is a popular open-source, Pregel-like graph processing system engineered for performance over multiple releases, leading to a number of advances in the size and speed of graph processing at scale[7]. We chose to focus on Apache Giraph in this work for two main reasons

despite other works' claims of surpassing it in absolute performance. First, other prototype-quality systems do not provide the programming flexibility or stability of Giraph which was built and tested in a production environment. Second, other systems also do not play well or at all in the widely used Hadoop ecosystem which Giraph was designed to work in. We aim to provide answers in the context of a real-world environment similar to a number of use cases rather than in a constrained setting with the aim of avoiding pitfalls of applicability[12]. Since Giraph is an open-source member of a small group of industrially engineered and production tested systems, reflecting real-world integration and flexibility constraints, it is an ideal candidate for study.

Giraph stores vertices in a partition data structure - each vertex is stored with its outgoing edges along with data associated with the vertex/edge. By default, *hash partitioning* by graph vertex ID (a discrete numeric value) is used to place vertices into partitions, and then assign partitions to workers by hashing partition IDs. In this work, we use the default hash partitioning due to its relative simplicity at scale. Each iteration or *superstep* consists of some maintenance work such as initializing data structures, followed by computation which is overlapped with communication between workers. At the end of the superstep, a worker flushes its outgoing message queue and waits for the barrier signal from the ZooKeeper coordination system. Giraph interfaces with Netty, an open-source library for asynchronous, event-driven Java networking library which provides two customizable options - Pooled Buffers and Direct Memory - which are Java Virtual Machine (JVM) software optimizations. The former enables buffer pooling to minimize the allocation and deallocation overhead, and the latter enables finer-grained memory management to avoid wasting resources on "zeroing" out memory - both are recommended as performance optimizations[13]. Lastly, Giraph exposes other configurable parameters such as preference for data locality and thread counts for input-output, computation and networking.

Related Work on Experimental Analysis of Cloud-based Graph Processing Systems: As mentioned, there are previous works proposing distributed graph processing systems where evaluations focus on processing time and scalability alone. Many works in both systems and partitioning[14][6][15] do not explicitly address the questions of choosing the right cluster size and number of partitions.

Related Work in Performance Modeling for Parallel and Distributed Systems: [2] compares a number of graph processing systems, including an older release of Giraph, to study both performance properties and advantages and disadvantages. [2],[3] also propose graph algorithm behavior classifications. [16],[17] undertake performance comparisons of graph processing systems. However, none of these works attempt to model and predict performance scalability.

III. Understanding Performance Impact of Multiple System Parameters

We investigate the impact of resource allocation on the performance and energy consumption of a production-grade, industrial graph processing system, across a number of benchmarks in this work. "Resources" includes hardware and software resources such as working memory, CPU threads, network bandwidth and JVM optimizations. The questions we seek to answer are of the form: "How important is resource X for distributed graph analytics on a production system (Giraph) in a virtual cloud-based deployment? To answer such questions, we collect and analyze performance measurements from Apache Giraph 1.1.0 across a number of configurations of the resource variables. We assemble a large, multi-dimensional ***measurement set*** from these performance measurements on a VMware private research cloud provisioned on 20 physical Dell PowerEdge R710 servers, each having 12 cores, 144 GB memory and dual 10 Gbps Ethernet. Based on this measurement set, we provide readers with a nuanced understanding of real-world behaviors for optimal decision making from a statistical standpoint.

Table I
Graph Datasets Used

Graph	$\|V\|, \|E\|$	Avg. Out-Degree	Skew
USA-Road	23.9M, 57.7M	2.4	6.6
Arabic-2005	22.7M, 639M	28	9872
Twitter-2010	41.6M, 1.46B	35	2899965
UK-2007-05	105M, 3.73B	35	14965
Weibo	58.6M, 261M	4.5	3995
CCHost-12	101M, 2.04B	20	3898540

A. Data Collection and Algorithm Classification

Collecting sufficient performance data is essential to build a suitable measurement set of samples that can capture the variety of complex behaviors encountered in real-world workloads. We collect nearly 9500 measurements or data points for three categories of variables described below: benchmark, hardware environment and software. We analyze experimental data obtained from running algorithms on large directed (Arabic-2005, Twitter-2010, UK-2007-05, Weibo, CCHost-12) and undirected (USA-Road) graph datasets (Table I) of different scales and degree distributions, obtained from a number of online sources[18][19][20] (e.g. Twitter and USARoad are a large social network and road network respectively). In our experiments with various algorithms, we observed the following classes of behavior based on the load patterns in superstep time seen in our measurements:
-**Identical Iterations**: Algorithms in which the computation and messaging loads of vertices are identical (e.g. PageRank, Diameter Estimation, Semi-clustering) or nearly identical across supersteps which could lead to near-identical superstep times without resource performance variations
-**Clustered Iterations**: Algorithms in which the computation and messaging loads vary across supersteps (explained in more detail in the next section). We group these into:

1) Graph traversals (e.g. Single and Multiple Source Shortest Path, Breadth First Search), where peak superstep time is many times the average, and the peak is concentrated in a few contiguous iterations
2) Repeated rounds of sequences of different supersteps (e.g. Graph Coloring, Maximal Matching), where the peak superstep time is not as large a multiple of the average, and is spread across a longer period. In this group, shorter peaks occur more frequently.

We use seven analytics algorithms for our measurement set - PageRank (PR), Diameter Estimation (DE), Single Source Shortest Path (SSSP), Multiple Source Shortest Path

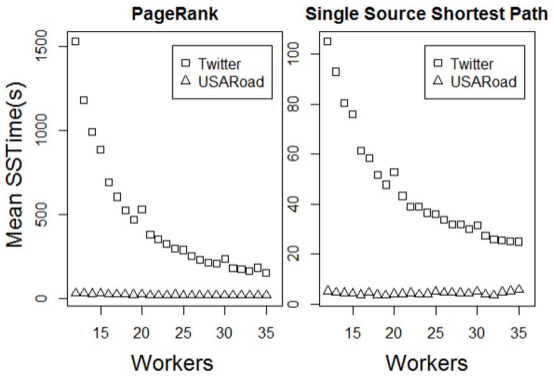

Figure 1. Superstep time for different cluster sizes, showing impact of graph dataset features on the benefits of scaling

(MSSP), Breadth First Search (BFS), Connected Components (CC) and Graph Coloring (GC) - as representatives of the above classes of behavior and set values for the following variables/parameters before running jobs:

-*Cluster Used*: We collect measurements on 2 virtual-machine clusters created on our physical hardware - one contains 36 nodes and the other 20. The 36-node cluster allocates 8 GB per node for the JVM and has 1 Gbps network bandwidth, while the 20-node cluster allocates 45 GB per node for the JVM and has 0.5 Gbps network bandwidth. A majority of our samples ($\sim$8770) were collected on the 36-node cluster. We do so with the aim of training a model on these samples and using the 20-node cluster measurements mainly to test our model's ability to generalize.

-*Workers*: This is the number of Giraph workers processing the graph dataset, with one worker assigned to each virtual machine. The range of workers used for each graph dataset is upper-bounded by the available cluster size and lower-bounded by the aggregate memory needed to hold the graph dataset in-memory.

-*Algorithms & Datasets*: These two variables describe the workload for a job, with 1 algorithm (GC) run on the undirected USARoad graph dataset, 4 (PR, DE, SSSP, MSSP) on the other directed graph datasets, and 2 (BFS, CC) on bi-directional versions of some directed graph datasets where an edge in the reverse direction was added for every existing edge (27 benchmarks in total).

-*Vertices, Edges and Skew*: These variables have unique values for each graph dataset (see Table I) - the Dataset variable above is simply a string representation of the graph dataset name. The value of Skew is calculated as the difference between average and maximum out-degree for directed graphs (simple maximum degree for undirected).

-*Compute Threads*: This is a tunable Giraph parameter for which we assign values of 2, 4, 6 and 8 - the latter two settings are used only on the 20-node cluster for scaling and testing purposes.

-*Network Bandwidth* between the workers: 1Gbps on the 36-node cluster, 0.5Gbps on the 20-node cluster.

-*Netty Software Optimizations*: We group both optimizations described in Section II, Pooled Buffers and Direct Memory, and name this binary variable NettyOn, indicating whether both optimizations are enabled or not.

-*Partition Count*: This is the total number of partitions created from the input graph, which can be modified in the hash partitioning code in Giraph. We use either n^2 or $4n$ partitions - where n is the number of workers - as they are two different asymptotic growths. Choosing 4 as a multiplier enables us to test the impact of multiple *waves*[21] or rounds of computation since each partition is assigned to a single compute thread.

All variables except the following have discrete numerical values: Algorithms & Datasets, Cluster Used, Netty Optimizations and Partition Count. After running the jobs, we measure the superstep time (**SSTime**) as a dependent/response variable, which is calcualted from the metrics in the logs in seconds, calculated as the sum of the time taken for all supersteps to complete i.e. the total time required for in-memory processing. For four experiments (in particular PR & SSSP on Twitter and USARoad with n^2 partitions), a larger number of samples was collected to determine the effect on the confidence intervals and variance of the measured/response variables. In real-world scenarios, it is likely that a few benchmarks will be run more frequently, hence more data will be available for those which must be leveraged effectively without limiting a potential model's ability to generalize. A key limiting factor in real-world modeling is the paucity of data for less-used workloads, something we attempt to account for in our analysis by collecting more samples for these four experiments.

B. Analyzing Resource Impact on Performance

Performance can be quantified in two different ways for a computation running on a distributed cluster: using either the **SSTime** which is the end-to-end in-memory processing (excluding Input-Output) time of a job measured in seconds (or ms), or Resource Minutes which is the equivalent of cost or energy consumed and defined as (**SSTime**$\times workers$), thus accounting for the quantity of resources engaged over the period of the job. Our scalability study includes keeping the problem size constant for each chosen benchmark and varying the number of processors (weak scalability). Since the benchmarks themselves involve a range of workloads per processor, we are also able to study the strong scalability aspect (keeping the number of processors constant while increasing problem size), although that is not our focus. A key challenge in analyzing data collected in a virtualized environment is the addition of a noise component to results that is non-trivial to analyze and model (e.g. non-Gaussian behavior), which might be similar or worse compared to what one would expect in a public cloud environment[22].

Prior to modeling, we explore the measurements collected to build our modeling hypothesis and leverage domain insights in the process. Figure 1 shows the scaling behavior of **SSTime** for both PR (representing algorithms with identical iterations) and SSSP (representing traversal algorithms with clustered iterations) on the Twitter graph, which apparently follows a trend of decreasing returns to scale. This demonstrates that the benefits of increasing the available resources (workers) can provide large benefits initially, but this benefit reduces rapidly to the point where adding more workers has little benefit but significantly increases the resource minutes. We observed that these benefits only apply for Twitter and other highly skewed graphs, and not for USARoad which has a much lower skew. Our initial hypothesis is that the

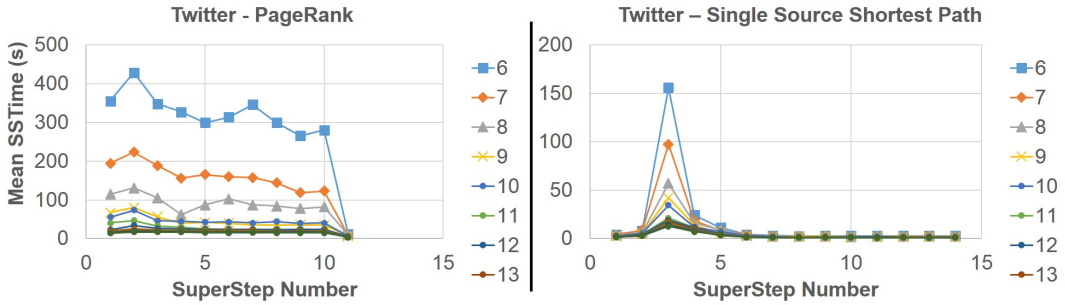

Figure 2. Performance gains due to scaling (Twitter dataset) cluster size, which are unevenly distributed and decrease with additional resources (limited range of configurations shown for visualization clarity)

time benefits of scaling resource allocations are dependent on the Skew of the input graph dataset, which we will rely on during modeling in Section IV.

Within a single job, superstep-level effects follow a near-logarithmically decreasing trend themselves and provide decreasing returns to scale in only a few steps, similar to **SSTime** at the job level. The incremental time benefits of adding resources at a superstep-level granularity for the Twitter graph and PR and SSSP algorithms (Figure 2) decrease quickly (example run on the 20-node cluster). The same holds for the USARoad graph and GC algorithm (we omit other figures due to a page constraint), with near-overlap between the times of the supersteps with lower workload. In addition, the benefits of increasing the cluster size do not have a uniform effect across supersteps, and in the case of the algorithms with clustered iterations, their effect is limited to the workload-heavy supersteps with the most vertex activation (Figures 1, 2).

A key insight gleaned from this analysis is that multiple nearby points have similar performance times within an acceptable margin of error due to performance variation (we omit plots of mean behavior due to a page constraint). Moreover, since the performance stops improving and potentially worsens after a point, the tail end of the curve contains consecutive points where the performance is stagnant. Hence, predicting a single "sweet spot", where the maximum benefits of additional resource benefits have been obtained and additional increases bring no further benefit is not the right modeling objective. It is more fitting to predict a "sweet range" or group of points within which the behavior will be optimal as compared to points outside the range. This requires modeling trends in the entire measurement set, which is both more challenging and more useful: once we find the "sweet range", resource minutes or other considerations such as available working memory can be used to make a final decision based on cost or energy considerations. Since resource minutes is derived from **SSTime**, we focus on accurately modeling primarily **SSTime** from the other variables (except Compute Time and Network Bytes which are also measured at the end of a job).

IV. Modeling Superstep Time

In this section, we build a simple, interpretable model incorporating the different variables we measured, train it using supervised learning and test it on data not seen during training. Our aim is to accurately predict the "sweet range" for a given benchmark using the intuitions from our observations in the previous section. A salient feature of the model's application is that it will be used to find a range of optimal cluster sizes - hence, the relative ordering of predictions is more important than their absolute values, compared to the ground truth measured data. Attempting to make behavioral decisions with more complex tools such as neural networks might be accurate but is hard to interpret in terms of domain-specific insights.

A. Developing a Formal Model

We model the **SSTime** by creating terms that capture a job's workload from combinations of the variables/parameters. To begin, we need a way to connect the workload to the randomized hash partitioning of the graph dataset. With p representing the number of worker machines used for the job under a randomized hash partitioning, we can represent the edge cut in expectation as P(edge is cut) = $1 - \frac{1}{p}$. Hence, we can write the expected number of edges cut as E(edge cuts) = $|E|(1 - \frac{1}{p}) = \frac{|E|(p-1)}{p}$.

With this representing the communication load per machine, we can write the total load in a superstep as the sum of *per-worker* computation and communication components defined as *SSComm* ($= \frac{E(p-1)}{p}$) and *SSComp* ($= \frac{V+E}{p}$).

These equations incorporate our intuition that workload depends on both edges and vertices of a graph and enable us to account for the overlap in time between computation and communication in Giraph, as the edge count is included in both terms. Therefore, every machine processes all the edges assigned to it as it iterates over vertices and performs additional computation for message-related functions, proportional to the number of vertices and edges assigned to it. Each superstep or iteration will see near-identical load for any identical iterations-algorithm, hence we can simply multiply the superstep time by a constant to approximate the job time. However, this is not the case for algorithms with clustered iterations. For this reason, a statistical model must have a means of capturing the activation pattern of each algorithm, which we represent as the *Algorithm* variable.

The above terms do not account for the rate of computation processing per machine (the threads assigned for computation), or the rate of network processing per machine (network bandwidth) respectively. Thus, we can update both terms as follows, naming them *SSCommBand* and *SSCompThreaded*:

$$SSCommBand = \frac{E(p-1)}{p * NetBandwidth} \quad (1)$$

$$SSCompThreaded = \frac{V + E}{p * Threads} \quad (2)$$

Finally, these equations implicitly assume a uniform distribution of the graph's edges - they do not account for skewed edge distribution characteristics of the graph dataset (since hash partitioning distributes vertices almost equally among machines, skew is not much of a problem there). We incorporate graph dataset skew to account for this, based on our initial hypothesis about the scaling behavior. The skew term scales the benefits provided by a resource increase, based on the skew characteristics of the graph. Hence, we can write down equations for the computation and communication workloads which capture the behavior of a benchmark for a skewed graph distribution compared to the ideal scaling scenario:

$$SSCommSkewed = SSCommBand \times log(Skew) \quad (3)$$

$$SSCompSkewed = SSCompThreaded \times log(Skew) \quad (4)$$

Equations 3 and 4 are simple and interpretable since they rely on domain insights. In addition, they are powerful enough to represent the variety of scaling behaviors we observed using a *linear* model with *non-linear* terms/features.

B. Validating and Training

To show that the above terms above have the power to explain the **SSTime** accurately, we use a feature selection procedure on the measurement set, comparing the base variables against our model's terms. We rely on forward stepwise regression[23] and bootstrapping[24], a non-parametric method (we use non-parametric methods as our data displayed statistical features that challenged traditional parametric modeling assumptions), to find the variables that are statistically useful in modeling the behaviors in the data. All our analyses are implemented in R. We perform feature selection after pre-processsing the data to calculate the above equations for the 36-node cluster training set. We then perform bootstrap sampling (sample size of 20%) and forward stepwise regression ten times to obtain the ten best-fitting models, chosen with the Akaike Information Criterion (AIC)[23]. The AIC is an information-theoretic criterion that rewards goodness of fit measured using likelihood, and penalizes larger numbers of parameters. We examined the ten best-fitting models to discover that the following variables are significant: Algorithm, SSCompSkewed and SSCommSkewed and NettyOn. We choose to include the Partition Count variable to provide a numerical estimate of its impact. Since the default setting in Giraph is n^2 partitions, we learn the impact of n^2 partitions via the *IncreasedPartitionCount* factor, by using $4n$ as our base setting. Equation 5 below represents the model designed using features selected from Section IV-A and above.

$$SSTime = \begin{cases} \theta_1 \times SSCommSkewed \\ +\theta_2 \times SSCompSkewed \\ +\theta_3(Algorithm) \\ +\theta_4 \times NettyOn \\ +\theta_5 \times IncreasedPartitionCount \end{cases} \quad (5)$$

For training, we log-transformed the **SSTime** variable to reduce its range and performed bootstrap sampling (sample size of 10%) one thousand times to obtain coefficient estimates along with boostrap confidence intervals. The coefficient distributions (rounded values in Table II) show the stability of the estimated values, with the average mean squared error on the training data approximately one-two seconds only. Finally, we perform ten-fold cross-validation (CV) and find a low prediction error estimate (ranging from approximately one and a half to 2 seconds), demonstrating a good fit and an ability to generalize.

C. Results of Modeling

Table II shows the learned coefficients for our model (see equation 5). We also learned coefficients for a model where the skew was replaced by the average degree (computed as $\frac{edges}{vertices}$) and found that the former had lower training and testing error. These coeffficients coupled with our data analysis led us to the following conclusions for typical small-to-medium size distributed graph analytics use-cases:

-*SSCommSkewed and SSCompSkewed* (θ_1 and θ_2): The coefficients of the feature capturing network traffic show it has a larger impact than the computation load, all other factors being equal. These indicate that the graph dataset skew has a major impact on the performance under scaling.

-*Algorithm* (θ_3): We run PR and DE for ten supersteps, SSSP, MSSP, BFS and CC until convergence (number of supersteps will depend on graph diameter) and GC for one thousand supersteps. The identical iteration algorithms have close coefficient values while the other group has a much broader range of coefficients depending on the workload (e.g. MSSP requires more work than SSSP due to the nature of the algorithm). Low standard deviation values for the algorithm coefficients show the stability of this behavior across graphs, allowing us to characterize algorithm behavior robustly. We can conclude that the most important factors to predict the impact of scaling are the algorithm behavior and the graph dataset skew.

-*NettyOn* (θ_4): We visualized the data (plots omitted due to a page constraint) to gauge the impact of Netty's optimizations in more detail and observed that they are only meaningful at smaller cluster sizes. There is greater memory pressure per machine in this case, also explaining the relatively larger standard deviation. Beyond that, the impact on performance is small enough to indicate little benefit to performance times, and possibly even increasing them very slightly.

-*Increased Partition Count* (θ_5): The mean overhead of an asymptotically-greater partition count is small but non-zero. It can have negligible improvement on the actual performance in some cases, as seen by the range of coefficients, but is generally a contributor to performance degradation.

V. Prediction with the Model

To show the validity of our model in predicting the sweet range, we trained the model on 36-node cluster data and test it on data from the 20-node cluster using the Mean coefficient values (Table II) . Hence, even with point prediction inaccuracy that can be captured via the range of learned coefficients, we can test the ability of the model to predict the sweet range among all cluster sizes. Recall (from Section III-B) that our objective is primarily to capture the relative ordering of performance scaling trends, not the absolute values. Note that for the 20-node cluster data, the Network Bandwidth, Cluster size effects (captured via the

Table II
LEARNED COEFFICIENTS

	θ_1	θ_2	$\theta_3(PR)$	$\theta_3(DE)$	$\theta_3(SSSP)$	$\theta_3(MSSP)$	$\theta_3(BFS)$	$\theta_3(CC)$	$\theta_3(GC)$	θ_4	θ_5
Max	1.01	0.59	4.74	4.94	3.16	6.46	1.83	2.98	7.69	0.06	0.43
Mean	0.80	0.33	4.49	4.54	2.93	5.45	0.94	2.42	7.48	-0.18	0.19
Median	0.80	0.33	4.49	4.54	2.93	5.46	0.94	2.43	7.48	-0.18	0.19
Min	0.61	0.13	4.24	4.05	2.72	4.58	-0.18	1.79	7.29	-0.41	-0.004
Std.Dev.	0.064	0.068	0.074	0.15	0.069	0.27	0.29	0.17	0.059	0.07	0.074

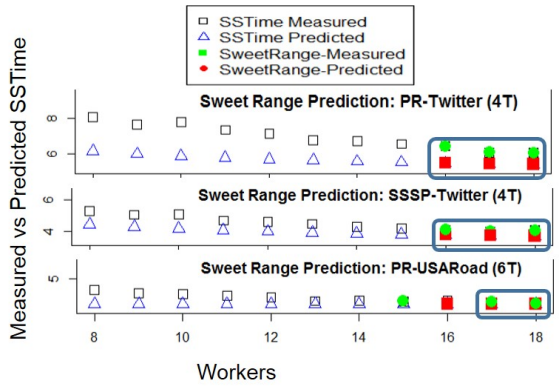

Figure 3. Optimal range prediction examples on unseen cluster and configurations, with areas of agreement within box

Cluster factor variable) and Thread counts (6 and 8) were never seen in the training data, so this is a practical way of testing the model. Despite this, predictions of the sweet range (Figure 3) are robust, with the 4 thread predictions agreeing exactly for both PR-Twitter and SSSP-Twitter, and the 6 thread prediction for PR-USARoad off by one worker (on very close absolute values). With 8 threads (figures omitted due to a page constraint), due to higher variance, the predicted range has higher inaccuracy although the values are close. All sweet range predictions on the Twitter and USARoad graphs, with PR and SSSP for 4 or 6 threads, either match exactly or are off by one worker. This shows that the model is able to generalize effectively to previously unseen environments proving its worth.

VI. CONCLUSIONS AND FUTURE WORK

The conclusions presented in this work make it clear that for performance metrics, understanding the impact of system parameter choices and their relationship with the graph dataset characteristics is essential. A preliminary step for future work would be expanding our measurement set to include measurements for different partitioning methods and algorithm behaviors. Analyzing this data can provide additional insights on the relative effects of these variables. The results of this work enable users to make choices for distributed graph data processing in a static setting without considering dynamic graphs that may change frequently. An interesting area for future exploration could enable leveraging the ability of resource management frameworks to acquire and release resources elastically, making flexible real-time decisions via relying on our proposed model.

ACKNOWLEDGMENTS

The authors would like to thank the Statistics Consulting Center at the University of Pittsburgh and Vineet Raghu for their feedback on the methods used. 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] M. Han *et al.*, "An experimental comparison of pregel-like graph processing systems," *Proc. VLDB Endowment*, 2014.
[3] Y. Lu *et al.*, "Large-scale distributed graph computing systems: An experimental evaluation," *Proc. VLDB Endowment*, 2014.
[4] Y. Low *et al.*, "Distributed graphlab: a framework for machine learning and data mining in the cloud," *Proc. VLDB Endowment*, 2012.
[5] J. E. Gonzalez *et al.*, "Graphx: Graph processing in a distributed dataflow framework." *OSDI*, 2014.
[6] ——, "Powergraph: Distributed graph-parallel computation on natural graphs." *OSDI*, 2012.
[7] A. Ching *et al.*, "One trillion edges: Graph processing at facebook-scale," *Proc. VLDB Endowment*, 2015.
[8] V. K. Vavilapalli *et al.*, "Apache hadoop yarn: Yet another resource negotiator," *Proc. of the 4th Annual Symp. on Cloud Computing*, 2013.
[9] B. Graham and M. Rangaswami, "Do you hadoop? a survey of big data practitioners," *Sand Hill Group, San Francisco, CA, USA*, 2013.
[10] A. Nadkarni and L. DuBois, "Trends in enterprise hadoop deployments," *Survey, IDC*, 2013.
[11] G. Malewicz *et al.*, "Pregel: a system for large-scale graph processing," *Proc. of the ACM SIGMOD*, 2010.
[12] A. Barker *et al.*, "Academic cloud computing research: Five pitfalls and five opportunities." *HotCloud*, 2014.
[13] N. Maurer, "Netty best practices." [Online]. Available: http://normanmaurer.me/presentations/2014-facebook-eng-netty/slides
[14] C. Martella *et al.*, "Spinner: Scalable graph partitioning in the cloud," *IEEE 33rd Intl. Conf. on Data Engineering*, 2017.
[15] S. Salihoglu and J. Widom, "Gps: A graph processing system," *Proc. of the 25th Intl. Conf. on Scientific and Statistical Database Management*, 2013.
[16] O. Batarfi *et al.*, "Large scale graph processing systems: survey and an experimental evaluation," *Cluster Computing*, 2015.
[17] Y. Guo *et al.*, "How well do graph-processing platforms perform? an empirical performance evaluation and analysis," *IEEE 28th Intl. Parallel and Distributed Processing Symp.*, 2014.
[18] P. Boldi *et al.*, "Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks," *Proc. of the 20th Intl. Conf. on World Wide Web*, 2011.
[19] "soc-sinaweibo — massive network data." [Online]. Available: http://networkrepository.com/soc-sinaweibo.php
[20] "Web data commons hyperlink graph." [Online]. Available: http://webdatacommons.org/hyperlinkgraph/2012-08/
[21] M. Hammoud *et al.*, "Mc2: Map concurrency characterization for mapreduce on the cloud," *IEEE Intl. Conf. on Cloud Computing*, 2013.
[22] P. Leitner and J. Cito, "Patterns in the chaos—a study of performance variation and predictability in public iaas clouds," *ACM TOIT*, 2016.
[23] J. Neter *et al.*, *Applied linear statistical models*. Irwin Chicago, 1996.
[24] B. Efron, *The jackknife, the bootstrap, and other resampling plans*. SIAM, 1982.