Bridging the GAP: Towards Approximate Graph Analytics

Anand Padmanabha Iyer[★], Aurojit Panda[°], Shivaram Venkataraman[•], Mosharaf Chowdhury[†], Aditya Akella[•], Scott Shenker[★], Ion Stoica[★]

[★]*University of California, Berkeley* [°]*NYU* [†]*University of Michigan* [•]*University of Wisconsin*

ABSTRACT

While there has been a tremendous interest in processing data that has an underlying graph structure, existing distributed graph processing systems take several minutes or even hours to execute popular graph algorithms. However, in several cases, providing an approximate answer is good enough. Approximate analytics is seeing considerable attention in big data due to its ability to produce timely results by trading accuracy, but they do not support graph analytics. In this paper, we bridge this gap and take a first attempt at realizing approximate graph analytics. We discuss how traditional approximate analytics techniques do not carry over to the graph usecase. Leveraging the characteristics of graph properties and algorithms, we propose a graph sparsification technique, and a machine learning based approach to choose the apt amount of sparsification required to meet a given budget. Our preliminary evaluations show encouraging results.

1 INTRODUCTION

The recent past has seen a resurgence in the interest in storing and processing massive amounts of graph-structured data. Several sources support this fact; for instance, a ranking among databases revealed that graph databases have grown in popularity by over 500% in the last few years alone [1]. This trend is likely to grow in the future. Fortunately, big data's emphasis on the three V's—volume, velocity and variety—has made existing systems somewhat future-proof. A deluge of graph processing systems exist today [11, 13, 14, 16–18, 20–22, 25, 27–29, 31] that can handle large graphs, some even as large as a trillion edges.

While there are several contributing factors to the renewed popularity of graph analytics, a major one is the emergence of new applications and use-cases. Such scenarios range from existing applications such as social network analytics, recommendations, to upcoming applications such as industrial Internet and the Internet of Things (IoT). A common requirement of many of these applications is the timeliness of the analysis results, a must for actionable analytics [4]. Unfortunately, existing graph-processing systems, in their quest to provide exact answers to graph algorithms, take several minutes or even hours to provide answers even for moderately sized graphs. Surprisingly, several of these applications can benefit from a *rough* answer. For example, in security contexts, it is useful to find patterns, but often it is good enough to just a rough estimate of the number of times the pattern occurs. That is, there is no need to find the exact number of matches. Existing graph-processing systems do not support producing such *approximate* answers.

Approximate analytics is an area that has garnered attention recently in big data analytics [5, 6, 15], where the goal is to let the end-user trade-off accuracy for much faster results. Several proposals for approximate analytics exist, but the underlying key idea is to use a small portion of the dataset to compute the results. Some approximation systems leverage the scheduler, and kill tasks selectively to achieve the desired accuracy or latency budget. However, all of the approximation systems focus on simple aggregate queries or analytics and thus do not consider complex, iterative workloads such as distributed graph processing.

Extending approximate analytics systems to support graph analytics is a challenging task because of the differences in the underlying assumptions. The fundamental assumption of a linear relationship between the sample size and execution time falls apart in graph processing. Further, approximation systems rely on statistical properties of the samples to compose partial results and/or error characteristics. Finally, these systems store multiple samples and cherry pick the right amount based on the linearity assumption. These techniques are difficult to incorporate in distributed graph processing due to the iterative nature of the algorithms.

In this paper, we explore the feasibility of bringing approximate analytics to distributed graph processing. Achieving efficient *approximate graph processing* faces a number of challenges, including the question of how to sample graphs and how to pick the right sampling parameter given a budget (§2). To solve these challenges, we leverage the recent advancements in spectral sparsification theory [30] literature. Specifically, we propose a spectral graph sparsification strategy that reduces the graph size significantly. We then devise a machine learning based approach to modeling performance and picking the right sparsification parameter (§3). We are presently implementing our proposed techniques in a system called GAP (for **G**raph **A**nalytics by **P**roximation). Our preliminary evaluations of GAP has shown encouraging results (§4).

2 BACKGROUND & CHALLENGES

We begin the paper with a brief overview of graph-parallel systems, approximate analytics and then list the challenges in building a system for approximate graph analytics.

2.1 Graph Processing Systems

Most existing general purpose graph processing systems allow end-users to perform graph computations by exposing a *graph-parallel* abstraction. The user provides a vertex program which is

GRADES-NDA'18 , June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-5695-4/18/06…$15.00
https://doi.org/10.1145/3210259.3210269

System	*PageRank Runtime (s)*
PowerGraph [17]	300
GraphX [16]	419
Giraph [7]	596
GraphLab [21]	442

Table 1: Runtimes for pagerank algorithm as reported by popular graph processing systems used in production.

run repeatedly on each of the vertex (in parallel) by the system. Interaction between vertices is implemented using either shared state (e.g., GraphLab [21]) or message passing (e.g., Pregel [23]). A barrier is usually enforced between each iteration of the vertex program. PowerGraph [17] introduced the Gather-Apply-Scatter (GAS) model that captures the conceptual phases of the vertex program. Many popular open-source frameworks [16, 17] have incorporated the GAS model.

The bottleneck in distributed graph-parallel processing arises mainly from the message passing between vertices. In a big data system, these are implemented as shuffles which are quite expensive. As a result, executing graph algorithms take a non-negligible amount of time. Table 1 reproduces the reported results from recent graph processing literature for running 20 iterations of page rank algorithm on a moderately sized graph of 1B edges using 16 machines. We see that the execution time is in the order of several minutes. The performance numbers worsen significantly as the input graph becomes larger.

2.2 Approximate Analytics

Approximate analytics is based on the premise that results from partial execution is often good enough. Systems supporting approximate analytics usually provide bounds on two dimensions—latency and accuracy—and lets users trade-off one for the other. These systems have been used successfully for query processing [5], dataflow jobs and straggler mitigation [6]. To provide this trade-off, they leverage sampling strategies. The basic observation is that the more data the system works on, the more accurate the results and vice-versa. Thus, given a corpus of data, approximation systems save samples of it using various criteria. Given a latency or accuracy budget, the job of the system is then to pick the right amount of samples to process and/or drop tasks when desired result is achieved.

2.3 Challenges

While it may seem straightforward to marry approximate analytics with graph-processing systems, making approximate graph analytics a reality is far from trivial. A system for approximate graph analytics faces a number of challenges. First, approximation systems rely on the fact that there exists a linear relationship between the amount of data in the sample and the execution time. However, such linear relationship does not exist in graph processing. While this could be helpful (i.e., a small reduction in input could lead to a large reduction in execution time), it also means that sampling could lead to undesirable outputs. To illustrate this, consider fig. 1, which shows the result of running connected components on different random samples of a graph. Surprisingly, the execution time does not improve at all, rather it even becomes worse when the sample is small. This is because blind sampling destroys the structure of

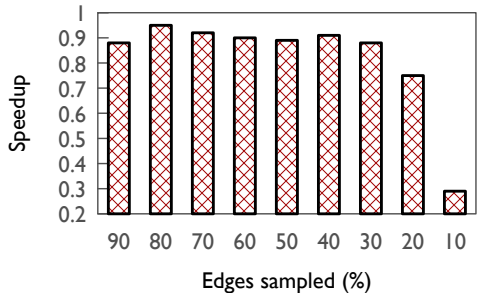

Figure 1: Sampling randomly leads to undesirable effects. Here, execution time (speedup) increases (reduces) with smaller samples.

the graph leading to much longer paths. Thus, simple sampling strategies employed by existing approximate analytics systems are not applicable in our setting.

Second, due to this non-linearity, picking the right amount of samples is difficult. Traditional approximation systems create, store and precompute query results on samples based on the assumption that partial results and errors could be composed. However, this may not hold true in graphs. Thus, precomputing aggregates by creating and storing samples is not a feasible approach.

Finally, existing approximation systems support only simple queries, such as aggregates, where computing the error on the result is intuitive. However, graph algorithms are executed in an iterative manner and thus estimating error on the output of a graph algorithm operating on a sampled graph is hard. Theoretical bounds exist for a few specific algorithms, but to the best of our knowledge, there are no general guarantees.

3 OUR APPROACH

We now describe our vision and approach for an approximate graph analytics system. In addition to solving the challenges listed earlier, we wish to achieve the following goals in our quest towards an efficient approximate graph analytics solution:

- A large body of graph theoretical work exists in the area of approximation algorithms. These works propose efficient approximate versions of various graph processing algorithms. We do not want to depend on such approximate version of any graph algorithms. In other words, we would like to be *approximation algorithm agnostic*. If an approximate version of the algorithm we support is available, we discuss how to leverage them in §5.
- Similarly, several flavors of distributed graph-processing engines exist. Some of them offer asynchronous processing mode [17], while some of them offer the favorable properties of dataflow [16]. We would like to propose techniques that are generic and not specific to one graph-parallel model.
- Finally, existing graph processing systems support varied workloads. In this respect, we would like our solution to have low overhead when it needs to accommodate new workloads.

The overall architecture of our solution GAP is depicted in fig. 2. It consists of two main components. Leveraging the work in spectral graph theory, a graph sparsifier is used to reduce the input graph's size. Based on the observation that a graph workload's performance characteristics is majorly dependent on the input graph [8], a machine learning (ML) based model is used to learn and predict the

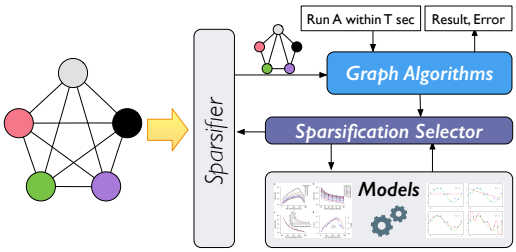

Figure 2: GAP **System Architecture.**

amount of sparsification required for a given budget. When an input graph is provided, we map it to one of the benchmark models by a simple mapping technique. Our intuition is that since the number of graph algorithms are limited and graph characteristics are described by a few variables, ML models are apt at this job. We discuss these components in detail in the rest of this section.

3.1 Graph Sparsification

The fundamental building block of any approximation system is sampling. Carrying this over to graphs, a straightforward approach is to sample edges and vertexes using some criteria. This approach, commonly referred to as *graph sparsification*[1] has been studied extensively in the literature on graph theory. The main idea in this body of work is to compute a (much) smaller graph that preserves crucial properties of the input graph.

While several proposals on the type of sparsifier exists, many of them are either computationally intensive, or are not amenable to a distributed implementation (which is the focus of our work)[2]. As an initial solution, we developed a simple sparsifier adapted from the work of Spielman and Teng [30] that is based on vertex degrees. The sparsifier uses the following probability to decide to keep an edge between vertex a and b:

$$\frac{d_{AVG} \times s}{min(d_a^o, d_b^i)} \tag{1}$$

where d_{AVG} is the average degree of the graph, d_a^o is the out-degree of vertex a and d_b^i is the in-degree of vertex b and s is a tunable parameter that controls the level of sparsification.

Intuitively, we would like to drop one of *many* edges from a vertex with large degree as opposed to dropping the *only* edge from a vertex with low degree. The sparsifier in eq. (1) does exactly this. The cost of running the sparsifier is negligible. We further reduce this cost by computing vertex degrees when the graph is first loaded into the system. One potential problem with the sparsifier is that it takes decision solely on local information. To reduce the ill effects of this, we leverage how the algorithm operates. For instance, we can avoid removing an edge it is in the spanning tree and so on.

3.1.1 Estimating Error due to Sparsification. An important task when using sampling strategies is to estimate the error in the output. In a non-graph setting, error estimation is straightforward. However, it is unclear how to estimate the error due to sparsification on the output of graph algorithms. We take a simple approach to this problem: we define a few error metrics, and leave the flexibility of defining additional error metrics to the user. In our system, one default error metric is the *degree of reordering*. This metric is applicable to algorithms that output a ranking for the vertexes, for example page rank or triangle count. In these algorithms, we can define the degree of reordering as the amount of reordering of the ranking compared to the ground truth. This flexibility exists because we learn the relation between error and sparsification.

3.2 Picking Sparsification Parameter, s

Once the sparsification strategy is in place, the next question is how to pick the right sparsification parameter s for a given accuracy requirement. To the best of our knowledge, theoretical bounds on error for the graph sparsification in a general setting is an open problem, hence we develop heuristics to solve this problem. Specifically, we use simple machine learning techniques to learn a model for the relation between s and performance (latency/error).

3.2.1 Building a Model for s. At the simplest level, one can build a model for s by running every possible algorithm on a given graph at varying values of s and then feeding the observed results to a learning algorithm. However, this requires too much time and effort. Thus, an approximation system needs a smarter solution.

In GAP, we take a simple approach. We consider a set of standard graph algorithms. These algorithms are then run on a set of representative graphs at varying values of s. The objective of this task is to learn a function H that maps s and the characteristics of the graph and algorithm to the performance profile. That is, we would like to learn:

$$H : (s, a, g) \implies e/p$$

where a is the algorithm specific features (if any), g is the graph specific features and e/p is the error / performance. Since there is no standard benchmark for distributed graph processing, we choose the representative set of algorithms and workloads from the Graph500 benchmark [2]. Our observation (§2.3) indicates that performance profiles are non-linear, hence we pick learning techniques that can accommodate discontinuity (e.g., random forests).

3.2.2 Accommodating New Workloads. Once models are built, the final step is to use the model to pick the sparsification parameter s when the system needs to run a graph algorithm on an unknown/new graph workload. We do not want to build a model per workload online (the model building phase is intensive and hence is typically done offline). Thus, we need to find an existing model that can operate on the new workload.

For this, we propose a light-weight mechanism[3]. We randomly pick a few values of sparsification parameters and run the algorithm on the new workload in an online fashion. Simultaneously, we use the models to predict the output. We then pick the model(s) with the least error. The random values of s could be chosen to complete the tests within a given time budget. For every new workload, we also use the results of running analytics as a feedback to our learning component. This lets us refine and improve our models over time.

4 PRELIMINARY EVALUATION

We are actively working on refining and evaluating our approach by implementing GAP at this time. We present our early experiences

[1] Also referred to as graph sketching.
[2] We are actively investigating several sparsification strategies.
[3] We are pursuing better techniques here at the time of writing.

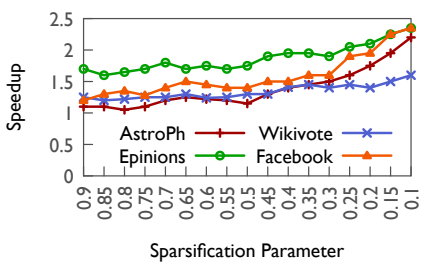

Figure 3: In triangle counting, we see similar trends in performance in graphs with similar characteristics (e.g., AstroPh and Facebook).

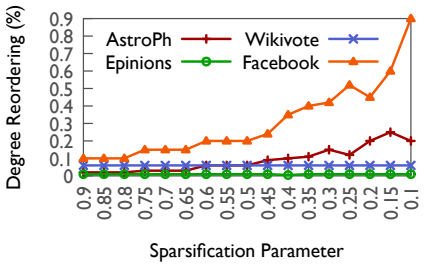

Figure 4: Error due to sparsification. Like the speedup, we see similarity in the error profile of graphs with similar characteristics.

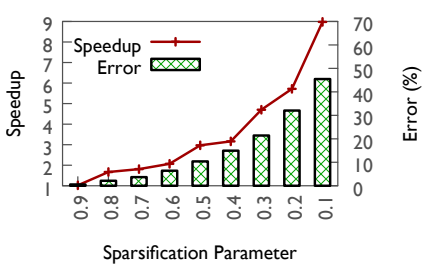

Figure 5: Larger graph (uk-2007-05 [9, 10] with 3.7B edges) sees better speedup due to the distributed nature of the execution.

in this section. We chose to build GAPon GraphX [16], but note that our techniques are not restricted to it. We picked five openly available graph datasets [3, 9, 10] (with number of edges up to 3.7 billion for the largest graph) based on the characteristics of the underlying graph, such as the diameter and clustering coefficient.

We evaluate our hypothesis of building a model for sampling based on the characteristics of the graph in the following way. We ran two algorithms, page rank and triangle count, on the datasets with varying values of the sparsification parameter s. We then recorded the speedup of the algorithm compared to the execution on the complete graph. We also note the error by evaluating the degree of reordering (§3) for each value of s.

Figure 3 shows the speedup obtained on triangle count algorithm, while fig. 4 depicts the error in terms of reordering. We see that even with a small reduction in input, the system is able to speedup the execution. As seen in the error characteristics, this speedup does not come at the expense of large errors. The error remains small for a wide range of the sparsification parameter. It may be troubling to see the diminishing returns with increase in sparsification, but this is due to the use of small datasets and also due to the fact that the experiment was done on a single machine (GraphX executes the same way as in a distributed setting but does not incur network penalties). As the graph grows larger and the computation is spread across many machines, sparsification reduces the shuffled data, and we see much larger gains as shown in fig. 5.

A more intriguing question is if our proposed approach is feasible. That is, is it possible to learn a model at all for approximate analytics? In our dataset, the Facebook and AstroPh datasets share similarity in the diameter and clustering coefficients. Similarly, Wikivote and Epinions share similar graph characteristics. Moreover, Facebook and AstroPh are social relationships while Wikivote and Epinions represent voting/rating relationships. In our results, we see that the performance and error trends follow the same observation—-the performance and error curves of the Facebook and AstroPh datasets exhibit similar trends, the same is true for the Wikivote and Epinions datasets. Results from our experiments using page rank algorithm also show similar trends. Thus, we believe that our approach is feasible.

5 DISCUSSION

We are presently working on improving all areas in our proposal First, we are investigating better sparsifiers that we could leverage, including the possibility of using machine learning techniques such as deep learning to find such sparsifiers. We plan on making the sparsifier pluggable to study the feasibility of choosing sparsifier on demand. Second, a large body of theoretical work exist on approximation techniques on specific graph algorithms. If it is possible to study the performance and error characteristics of different proposals on the same algorithm, it may be possible to cherry pick a proposal given a latency/error bound. Third, we are looking at programming language techniques to evaluate our wild idea of *synthesizing* approximate versions given an exact graph program. Finally, since real-world graphs are dynamic, an interesting direction is *incremental approximate* graph analytics.

6 RELATED WORK

Our work is related to distributed graph processing systems and approximate analytics systems.

A large number of graph processing systems exist in the literature. [11, 16, 17, 20, 21, 27–29, 31] focus on iterative analytics on static graphs. [12, 13, 18, 19, 22, 24–26] focus on analytics on evolving graphs. None of these systems support approximate analytics. GraphTau [26], which focuses on evolving graph processing, supports approximate page rank computations. However, it does not allow user to specify a budget. Our techniques can be used to bring approximation to several of these graph processing systems.

Approximate analytics systems have gained much popularity in the big data analytics community recently, and thus several proposals exist. BlinkDB [5] uses stratified sampling to generate samples and then chooses samples to satisfy the query budget. [6] uses approximation techniques to mitigate stragglers. ApproxHadoop [15] enables approximation enabled map-reduce jobs. These systems do not support graph processing.

7 CONCLUSION

For many graph-processing application scenarios, computing an approximate answer is good enough. Yet, existing graph processing frameworks, in an effort to compute the exact answer, take several minutes or even hours to execute popular graph algorithms. In this paper, we looked at the problem of approximate graph analytics. We presented our proposal, which uses a spectral sparsifier to reduce the size of the graph, and a machine learning model to pick the right amount of sparsification given a budget. We are currently building GAP, a system that implements our proposals, and plan on open-sourcing the system.

ACKNOWLEDGMENTS

We would like to thank the reviewers for their valuable feedback. In addition to NSF CISE Expeditions Award CCF-1730628, this research is supported in part by DHS Award HSHQDC-16-3-00083, and gifts from Alibaba, Amazon Web Services, Ant Financial, CapitalOne, Ericsson, Facebook, Google, Huawei, Intel, Microsoft, Scotiabank, Splunk and VMware.

REFERENCES

[1] [n. d.]. Graph DBMS increased their popularity by 500% within the last 2 years. http://db-engines.com/en/blog_post//43. ([n. d.]).
[2] [n. d.]. Graph500 Benchmarks. http://www.graph500.org. ([n. d.]).
[3] [n. d.]. Stanford Large Network Dataset Collection. https://snap.stanford.edu/. ([n. d.]).
[4] 2016. Graph Data Use-cases. https://neo4j.com/resources/2016-state-of-the-graph/. (2016).
[5] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys '13)*. ACM, New York, NY, USA, 29–42. https://doi.org/10.1145/2465351.2465355
[6] Ganesh Ananthanarayanan, Michael Chien-Chun Hung, Xiaoqi Ren, Ion Stoica, Adam Wierman, and Minlan Yu. 2014. GRASS: Trimming Stragglers in Approximation Analytics.. In *NSDI*. 289–302.
[7] Apache Giraph. [n. d.]. http://giraph.apache.org. ([n. d.]).
[8] Scott Beamer, Krste Asanovic, and David Patterson. 2015. Locality Exists in Graph Processing: Workload Characterization on an Ivy Bridge Server. In *Proceedings of the 2015 IEEE International Symposium on Workload Characterization (IISWC '15)*. IEEE Computer Society, Washington, DC, USA, 56–65. https://doi.org/10.1109/IISWC.2015.12
[9] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. 2011. Layered Label Propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks. In *Proceedings of the 20th international conference on World Wide Web*. ACM Press.
[10] Paolo Boldi and Sebastiano Vigna. 2004. The WebGraph Framework I: Compression Techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*. ACM Press, Manhattan, USA, 595–601.
[11] Aydin Buluç and John R. Gilbert. 2011. The Combinatorial BLAS: design, implementation, and applications. *IJHPCA* 25, 4 (2011), 496–509. https://doi.org/10.1177/1094342011403516
[12] Zhuhua Cai, Dionysios Logothetis, and Georgos Siganos. 2012. Facilitating Real-time Graph Mining. In *Proceedings of the Fourth International Workshop on Cloud Data Management (CloudDB '12)*. ACM, New York, NY, USA, 1–8. https://doi.org/10.1145/2390021.2390023
[13] Raymond Cheng, Ji Hong, Aapo Kyrola, Youshan Miao, Xuetian Weng, Ming Wu, Fan Yang, Lidong Zhou, Feng Zhao, and Enhong Chen. 2012. Kineograph: Taking the Pulse of a Fast-changing and Connected World. In *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys '12)*. ACM, New York, NY, USA, 85–98. https://doi.org/10.1145/2168836.2168846
[14] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. 2015. One Trillion Edges: Graph Processing at Facebook-scale. *Proc. VLDB Endow.* 8, 12 (Aug. 2015), 1804–1815. https://doi.org/10.14778/2824032.2824077
[15] Inigo Goiri, Ricardo Bianchini, Santosh Nagarakatte, and Thu D. Nguyen. 2015. ApproxHadoop: Bringing Approximations to MapReduce Frameworks. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '15)*. ACM, New York, NY, USA, 383–397. https://doi.org/10.1145/2694344.2694351
[16] Joseph Gonzalez, Reynold Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. 2014. GraphX: Graph Processing in a Distributed Dataflow Framework. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, Broomfield, CO. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/gonzalez
[17] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. PowerGraph: Distributed Graph-parallel Computation on Natural Graphs. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI'12)*. USENIX Association, Berkeley, CA, USA, 17–30. http://dl.acm.org/citation.cfm?id=2387880.2387883
[18] Wentao Han, Youshan Miao, Kaiwei Li, Ming Wu, Fan Yang, Lidong Zhou, Vijayan Prabhakaran, Wenguang Chen, and Enhong Chen. 2014. Chronos: A Graph Engine for Temporal Graph Analysis. In *Proceedings of the Ninth European Conference on Computer Systems (EuroSys '14)*. ACM, New York, NY, USA, Article 1, 14 pages. https://doi.org/10.1145/2592798.2592799
[19] Anand Iyer, Li Erran Li, and Ion Stoica. 2015. CellIQ : Real-Time Cellular Network Analytics at Scale. In *Proceedings of the 12th USENIX conference on Networked Systems Design and Implementation (NSDI'15)*. USENIX Association, Berkeley, CA, USA.
[20] Aapo Kyrola, Guy Blelloch, and Carlos Guestrin. 2012. GraphChi: Large-Scale Graph Computation on Just a PC. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. USENIX, Hollywood, CA, 31–46. https://www.usenix.org/conference/osdi12/technical-sessions/presentation/kyrola
[21] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. 2010. GraphLab: A New Framework For Parallel Machine Learning.. In *UAI*, Peter Grünwald and Peter Spirtes (Eds.). AUAI Press, 340–349. http://dblp.uni-trier.de/db/conf/uai/uai2010.html#LowGKBGH10
[22] P. Macko, V. J. Marathe, D. W. Margo, and M. I. Seltzer. 2015. LLAMA: Efficient graph analytics using Large Multiversioned Arrays. In *2015 IEEE 31st International Conference on Data Engineering*. 363–374. https://doi.org/10.1109/ICDE.2015.7113298
[23] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: A System for Large-scale Graph Processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD '10)*. ACM, New York, NY, USA, 135–146. https://doi.org/10.1145/1807167.1807184
[24] Microsoft Naiad Team. 2014. GraphLINQ: A graph library for Naiad. http://bigdataatsvc.wordpress.com/2014/05/08/graphlinq-a-graph-library-for-naiad/. (2014).
[25] Derek G. Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. 2013. Naiad: A Timely Dataflow System. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13)*. ACM, New York, NY, USA, 439–455. https://doi.org/10.1145/2517349.2522738
[26] Anand Padmanabha Iyer, Li Erran Li, Tathagata Das, and Ion Stoica. 2016. Time-evolving graph processing at scale. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems*. ACM, 5.
[27] Abdul Quamar, Amol Deshpande, and Jimmy Lin. 2016. NScale: Neighborhood-centric Large-scale Graph Analytics in the Cloud. *The VLDB Journal* 25, 2 (April 2016), 125–150. https://doi.org/10.1007/s00778-015-0405-2
[28] Amitabha Roy, Laurent Bindschaedler, Jasmina Malicevic, and Willy Zwaenepoel. 2015. Chaos: Scale-out Graph Processing from Secondary Storage. In *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP '15)*. ACM, New York, NY, USA, 410–424. https://doi.org/10.1145/2815400.2815408
[29] Amitabha Roy, Ivo Mihailovic, and Willy Zwaenepoel. 2013. X-Stream: Edge-centric Graph Processing Using Streaming Partitions. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13)*. ACM, New York, NY, USA, 472–488. https://doi.org/10.1145/2517349.2522740
[30] Daniel A. Spielman and Shang-Hua Teng. 2008. Spectral Sparsification of Graphs. *CoRR* abs/0808.4134 (2008). http://arxiv.org/abs/0808.4134
[31] Guozhang Wang, Wenlei Xie, Alan J Demers, and Johannes Gehrke. 2013. Asynchronous Large-Scale Graph Processing Made Easy.. In *CIDR*.