Monarch: Gaining Command on Geo-Distributed Graph Analytics

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

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

Abstract

A number of existing and emerging application scenarios generate graph-structured data in a geo-distributed fashion. Although there is a lot of interest in distributed graph processing systems, none of them support geo-distributed graph processing. Geo-distributed analytics, on the other hand, has not focused on iterative workloads such as distributed graph processing.

In this paper, we look at the problem of efficient geo-distributed graph analytics. We find that optimizing the iterative processing style of graph-parallel systems is the key to achieving this goal rather than extending existing geo-distributed techniques to graph processing. Based on this, we discuss our proposal on building Monarch, the first system to our knowledge that focuses on geo-distributed graph processing. Our preliminary evaluation of Monarch shows encouraging results.

1 Introduction

Graph analytics has seen an increasing interest in the recent past, with several systems being proposed, both in the research and the open source community [6, 7, 9, 12, 13, 15, 19, 22, 24, 27, 30, 31, 35–37, 41]. This trend is due to the novel application scenarios that have emerged as a result of the proliferation of smart devices and sensors, and the ability of graph algorithms to derive useful insights from such datasets [1, 2]. Most, if not all, existing graph processing systems focus on analyzing graphs that are aggregated to a central location.

Today, many applications that could benefit from graph analysis are deployed on data centers *across the globe* and generate data in a geo-distributed fashion. For example, users of social networks are located around the globe. Similarly, cellular networks collect data at base stations that are geo-distributed across various locations [18]. This is also true for emerging applications. Internet-of-Things (IoT) applications, such as the much anticipated driverless vehicles, may generate data across multiple aggregation points. In analyzing such datasets, it may not always be feasible to aggregate the data to a central location due to many reasons. First, Wide Area Network (WAN) bandwidth is expensive and transferring large amounts of data may incur high costs. Second, more importantly, many of these scenarios could benefit from timely, low-latency analytics. Finally, political reasons may prevent data from moving to a different location.

The problem of analyzing datasets spanning geographical boundaries is not new; the field of Geo-distributed Analytics (GDA), that has gained much attention recently, focuses on precisely the same problem [34, 39, 40]. GDA brings WAN awareness to big data analytics, thus eliminating the need to move all the data to one location. Existing GDA proposals look at different aspects of this problem, ranging from low-level task placement [34] to higher-level query optimization [39]. However, current works on GDA have focused on simple queries and aggregates, and largely ignored iterative workloads such as machine learning and distributed graph processing, two important and emerging workloads in many applications.

Performing graph analytics in a geo-distributed fashion differs from traditional GDA in many ways. Due to the iterative nature of graph algorithms and the complex dependency in tasks that perform these iterations, simple task placement techniques do not work well as there is a need to deal with task affinity. Further, in traditional GDA, many datasets are amenable to clean sharding. This is not the case in graph processing where locality plays an important role in the performance of graph algorithms. Because of the expensive joins that must be performed at every iteration in a graph-parallel setting, simple join optimizations in GDA may not be effective. Finally, many graph algorithms generate large amounts of intermediate data. Thus, geo-distributed graph analytics solutions need to account for the iterative nature of graph processing.

In this paper, we focus on the problem of *geo-distributed graph analytics*. While combining traditional GDA with graph analytics may seem straightforward, our experience indicates that it is far from trivial. It is tempting to see this as a graph partitioning problem, since the goal of graph partitioning is to improve locality and thus reduce communication. However, due to the nature of data creation, repartitioning may not be feasible. Other similar challenges exist which should be addressed (§2). We observe that the key in geo-distributed graph analytics is to optimize the iterative processing style of graph-parallel systems. Based on this, we propose three techniques. First, we reduce the data to be processed using sampling strategies that leverages graph algorithms. Second, we

```
def Gather(u, v) = Accum
def Apply(v, Accum) = v_new
def Scatter(v, j) = j_new, Accum
```

Listing 1: The Gather-Apply-Scatter (GAS) composition introduced in PowerGraph [13].

remove the inefficiencies in current graph processing models by proposing a modification that reduces data exchange using a simple incremental computation strategy. Finally, we discuss how to bring WAN awareness into the picture (§3). To evaluate our proposal, we are building Monarch, a system that incorporates our proposed techniques. We discuss our early experiences from the system in this paper (§4). To the best of our knowledge, Monarch is the first system to focus on geo-distributed graph analytics.

2 Background & Challenges

We begin the paper with a brief overview of graph processing systems, geo-distributed analytics and then list the challenges in geo-distributed 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 run repeatedly on each of the vertex (in parallel) by the system. Interaction between vertices is implemented using either shared state (e.g., GraphLab [24]) or message passing (e.g., Pregel [28]). A barrier is usually enforced between each iteration of the vertex program. PowerGraph [13] introduced the Gather-Apply-Scatter (GAS) model that captures the conceptual phases of the vertex program as shown in listing 1. In the GAS decomposition, a vertex program consists of three phases: a `gather` phase that collects information about adjacent vertices and edges and applies a function on them, the `apply` phase that uses the function's output to update the vertex, and the `scatter` phase that uses the new vertex value to update adjacent edges. The system executes these phases sequentially. Many popular open-source graph-processing frameworks [12, 13] have adopted the GAS model. Since we are building Monarch on a graph processing framework that uses the GAS model, we focus on it in this paper. However, our techniques are general and can be applied to other models.

2.2 Geo-Distributed Analytics

A number of recent works have made the case for Geo-Distributed Analytics (GDA) [34, 39, 40]. While traditional data analytics assumes that data resides in a single, centralized datacenter, GDA forgoes that assumption. In GDA, data is collected and stored at geographically distributed datacenters. Analytic tasks are run across these datacenters without aggregating data to a central location. The key challenge in GDA systems is to ensure low response times for the analytic tasks being performed.

GDA systems solve this challenge by being Wide Area Network (WAN) aware. Specifically, these systems consider intermediate data movement to be the bottleneck and thus optimize the placement of such data and tasks that operate on them based on the bandwidth available between datacenters. Some systems [17] go further by switching between different join strategies and task coordination.

2.3 Challenges

There are several challenges in building a geo-distributed graph processing system.

First, GDA systems assume simple jobs and queries. In contrast, graph processing systems execute graph algorithms in an iterative manner, with multiple message exchanges in every iteration. Extending this to a geo-distributed setting means that every iteration would generate data exchange across WAN. While traditional GDA's task placement and scheduling can optimize where the tasks are placed, they do not alleviate the problem with the iterative model of graph processing.

Second, in GDA systems, data is susceptible to sharding. Hence, there is fine grained control over data movement that could be beneficial—for instance, a small amount of data could be moved to a different data center for a significant improvement in task placement flexibility. While graph partitioning has similar goals of improving locality and reducing communication between partitions, cleanly partitioning graphs is a hard problem [13]. Additionally, as graph algorithms progress the partitioning may need to be changed for the best performance. On top of this, since partitioning graphs cannot be done at fine granularity, a complete repartitioning may need to be done due to the nature of data generation. Thus, a one-time partitioning or data placement strategy is unlikely to be of help.

Finally, graph algorithms are complex, and their distributed implementations are demanding since they involve expensive operations [12]. The immutability assumption made by many graph-processing frameworks make things worse in terms of bandwidth usage. For task scheduling purposes, some GDA systems assume that intermediate data could be estimated and placed efficiently. This assumption breaks down in graph processing, where the intermediate data size could be large. As an example, running connected components on the openly available Twitter data [3] results in shuffling more than 50GB of data during the initial iterations in GraphX [12], a popular graph processing framework.

3 Our Proposal

We now describe our proposal for geo-distributed graph analytics after discussing our assumptions.

3.1 Assumptions

Geo-Distributed Graph: We assume that the graph is *generated* in a geo-distributed fashion. That is, a graph $G(V,E)$ exists across P partitions, distributed across D data centers ($P >= D$). Each partition $p \in P$ consists of v vertices and e edges. In this setting, it is obvious that aggregating the graph to one data center is expensive.

PowerGraph [13] argued that many naturally occurring graphs follow power-law distribution and hence make a case for vertex-cuts rather than edge-cuts for entities spanning partitions. Following this, we choose vertex-cuts, and mirror vertices which have edges spanning partitions. We note that this is not fundamental to our approach, as our approach could use edge-cuts also.

As discussed earlier, we do not assume that a complete repartitioning (e.g., using a communication efficient partitiong scheme such as 2D partitioning [13]) could be done. Thus, we restrict ourselves to the partitioning provided by the data naturally. Leveraging partitioning flexibility is something that we wish to pursue in the future.

Algorithms: While a large body of algorithms exist for the analysis of graphs, we restrict our scope in this work to algorithms that are implementable in a GAS decomposition model. Most of the commonly used graph algorithms can be expressed in GAS decomposition format; for instance, GraphX [12] provides implementations for six such graph algorithms (connected components, label propagation, page rank, SVD, shortest path, and triangle count).

WAN vs LAN Bandwidth: We assume that the LAN bandwidth is significantly higher than the WAN bandwidth. We further assume that the WAN bandwidth between pairs of DCs can differ significantly. This is true in most cloud provider settings. For instance, [39] notes that inter-DC bandwidth in major cloud providers is 1-2 orders of magnitude less than intra-DC bandwidth, and that the pair-wise WAN bandwidth can vary by over 20×.

3.2 Approach

The overall architecture of the system we are currently building, which we call MONARCH, is depicted in Figure 1. Based on our observation that optimizing iterative processing style of graph-parallel algorithm is the key to efficient geo-distributed graph analytics, the main idea in our approach is to *leverage the characteristics of graph-parallel computation model and the algorithms they support* to reduce WAN usage. To achieve this, we propose three simple, but powerful techniques: first, we reduce the data itself using an accuracy preserving sampling process. This results in less data to exchange. Second, we propose a modification to the GAS model that removes the inefficiencies with iterative processing. Finally, we bring WAN awareness to this modified model. We explain them in the rest of this section.

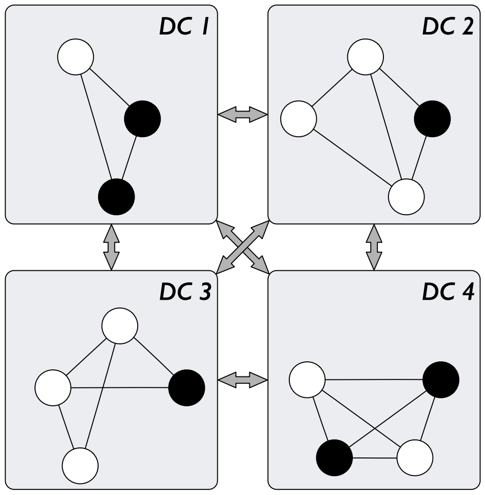

Figure 1: MONARCH system architecture. Each data center (DC) contains part of the graph. Communication between DCs happen through border vertices (shaded vertex in the picture), who exchange and synchronize state as described in §3.

3.2.1 Sparsification without Accuracy Loss

In geo-distributed graph analytics, inter-DC data exchange happens only if there are entities spanning multiple data centers. Hence, our first goal is to reduce these spanning entities and/or reduce the data flowing through them.

In MONARCH, we call vertices that interface with other datacenters *border* vertices. Since we use vertex-cut by default, border vertices are mirrors. In the GAS model, vertices gather (scatter) messages from (to) their neighborhood (§2). Thus, by reducing the size of the graph, we can reduce the amount of data transferred across data centers. Unfortunately, randomly eliminating graph entities to reduce the graph size leads to incorrect results.

To solve this, we plan on using a simple technique. We observe that in iterative graph-parallel model, many graph algorithms generate and exchange redundant intermediate data. This is because the GAS model only considers the immediate neighborhood in each iteration. Leveraging this, we can design a sparsification strategy that eliminates graph entities that will not contribute to the final solution. To illustrate a simple case, consider the connected components algorithm. By examining the connectivity information of border vertices, we can eliminate the need to mirror all but one vertex across DC pairs if the vertices are connected. This reduces the amount of data transferred across DCs. Further improvements can be obtained if only partial results are required (e.g., only required to compute components that contain particular vertices, or only find top components), as such cases can discard large parts of the graph and even eliminate the need for border vertices.

While the sparsification technique is beneficial in our setting, we note two shortcomings with it. First, not all algorithms can leverage such sparsification strtegies. Second, our sparsification strategy may result in a slightly longer convergence time. This is because by dropping entities, we may eliminate a shorter path. However, we assume that the cost of WAN transfer is much higher than this small sacrifice in convergence time.

3.2.2 Geo-distributed Graph Computation Model

Once the graph is sparsified, the next step is to run graph algorithms on it in a geo-distributed manner. As discussed in §2, GAS computations result in two data exchanges (`gather` and `scatter`) in every iteration. In our setting, this means that the border vertices potentially need to exchange data twice per iteration. When the data to be exchanged is large and/or when multiple iterations are involved, these transfers can become the bottleneck.

To solve this problem, we propose a simple modification to the GAS model. In this model, we restrict the execution of the GAS model to each datacenter. We then use a merging strategy to combine the results from each datacenter. Our enhanced GAS model consists of the following stages:

Bootstrap: When a graph algorithm is to be executed, Monarch first invokes the bootstrap stage. In the bootstrap phase, Monarch runs vanilla GAS on every datacenter independently. We consider the subgraph in each data center to be a graph of its own, and compute the algorithm result on this subgraph. At the end of this stage, we end up with local solutions in each datacenter.

Global Sync: After the bootstrap GAS execution has converged, we invoke a global synchronization stage. In this stage, only border vertices participate. A `gather`-like operation is invoked on them which enables the vertices to collect the partial state from other mirrors. Then, an algorithm specific function f_a is used to combine these partial states to generate an updated state for the border vertices. After this stage, all the mirrors of each border vertices have the same state. However, the global graph is in an inconsistent state. This is because the partial results in each DC may no longer be valid because of the updates to the border vertices.

iGAS: A strawman approach to recompute the correct partial results is to reset the local graph's vertices (except the border vertices) and restart the local GAS computation. However, this is wasteful. Hence we propose a different approach. We observe that after the synchronization, each subgraph is equivalent to an updated graph (with just the border vertices updated). Thus, we can leverage an incremental computation model to update the results on the local graph. In Monarch, we design an incremental version of the GAS model, which we call iGAS.

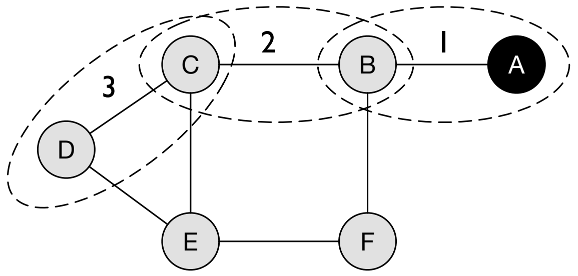

Figure 2: In incremental GAS model, each iteration marks and activates the neighborhood it influences. In this example, border vertex A is updated. It marks and activates B in the first iteration, B marks and activates C in the second iteration and C marks and activates D in the third. By leveraging the characteristics of the algorithm being executed, we avoid marking E and F although they are in the immediate neighborhood of C and B.

The iGAS computation leverages both the GAS computation model and algorithm properties. Specifically, we exploit the fact that GAS computations consider only the immediate neighborhood. In each iteration of the iGAS, we mark the immediate neighborhood of vertices that changed their state, and force computations on them. Obviously, in the first iteration, we mark the neighbors of the border vertices. We then repeat this marking and re-computation step until the change in the border vertex is propagated across the entire local graph. One problem with this approach is that potentially all the vertices may recompute if the graph is fully connected. To avoid this, we leverage algorithm properties. We mark only neighbors which might use the updated value. Figure 2 shows our iGAS approach. At a high level, iGAS can be seen as a backtracking and rectification process.

To summarize, our enhanced GAS model starts with a normal execution of the GAS model, then switches to iterations of global sync followed by iGAS until convergence. Essentially, we are amortizing the cost of synchronization after every iteration of the GAS step by batching multiple GAS iterations. We note that there is one caveat to our approach. The global synchronization step assumes that the algorithm specific function, f_a, is able to correctly combine the partial results for each border vertices. While we have derived f_a for many common graph algorithms, generalizing our technique to any graph algorithm is part of our future work.

3.2.3 Bringing WAN Awareness

While Monarch specifically optimizes for WAN bandwidth, our techniques consider WAN bandwidths to be equal. However, this is not true in practice. To tackle this problem, we envision two approaches.

First, we plan to generalize our computation model to support arbitrary interleaving of the global sync and iGAS phases per datacenter. Thus, depending on the

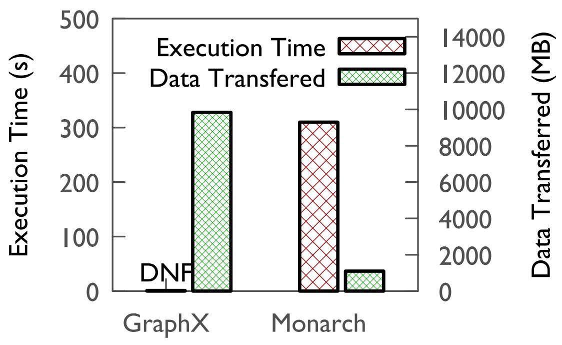

Figure 3: Our proposal is able to complete the execution of connected components when GraphX is unable to complete. This is because it tries to transfer too much data across WAN.

current WAN bandwidth, a datacenter may decide whether to participate in the global sync or not. In the second approach, if the system is given some flexibility in terms of data movement, then we plan to explore moving some border vertices based on the amount of bandwidth they might consume. Both are hard problems, and we are actively exploring different ways to solve them.

4 Preliminary Evaluation

We are currently building Monarch on GraphX. We present our early experiences in this section. We evaluate the usefulness of the enhanced GAS model that includes the local incremental computations and global syncs.

We chose the open source Twitter dataset [21], and use 16 machines on Amazon EC2 that simulates the same setting as in [12]. We picked 4 machines in each region, and then ran the connected components algorithm. The results are depicted in fig. 3. We see that GraphX is unable to complete the computation since it fails during the shuffle stage, while the enhanced model we propose is able to complete the computation in around 310 seconds. While this is longer than the reported numbers in GraphX (251s), we believe a lot of optimization opportunities are left for us to explore. We also see that while GraphX tries to transfer almost 10GB of data across WAN, Monarch only transfers around 1 GB. In this experiment, we did not use the sparsification process.

5 Discussion

Our proposal, Monarch, is a very preliminary attempt at the problem of geo-distributed graph analytics. We would like to improve several aspects of the system, and are actively working on them. First, we are exploring general graph sparsification and approximation techniques. In this respect, we are studying the class of algorithms which can be sparsified without loss in accuracy, or if that is not possible then the impact of sparsification on accuracy. For this, we plan on leveraging graph theory, such as [23], which presents a theoretical analysis of input reduction to some popular graph algorithms. Second, we are studying the convergence properties of various graph algorithms in our computation model. Finally, we wish to improve the computation model. Specifically, we are working on generalizing the model to all graph algorithms (e.g., f_a and incorporating WAN awareness. Towards this, we plan on leveraging the recent advancements in dynamic graph computations, including work on similar graph-parallel abstractions [32]. This will also allow us to extend our work to settings where the graph changes over time.

6 Related Work

A large number of graph processing systems exist in the literature, of which [4–6, 8, 11–14, 22, 24–26, 33, 35–38, 41–52] focus on iterative analytics on static graphs, while [7, 9, 15, 18–20, 27, 29–32] focus on analytics on evolving graphs. However, none of them support geo-distributed processing and thus focus on a single datacenter where the graph is aggregated. While our work focuses on the GAS decomposition model, these techniques can be incorporated into other models. [10] parallelizes sequential graph algorithms using partial evaluations and *algorithm specific* incremental computations, but does not consider geo-distributed settings.

On the other hand, many recent works have proposed techniques for geo-distributed data analytics. Iridium [34] uses WAN aware task placement and scheduling. [40] looks at join algorithm selection strategies for WAN optimization. SWAG [17] coordinates tasks across DCs. Finally, Clarinet [39] argues for WAN aware query optimization. [16] looks at the problem of geo-distributed machine learning using approximation techniques that are specific to ML algorithms. None of these systems consider iterative graph processing.

7 Conclusion

Graph processing and geo-distributed analytics are two areas that have seen increasing interest in the recent past. Yet, neither of them support the other. Geo-distributed graph analytics could be beneficial for many application scenarios that generate graph-structured data. In this paper, we took the first step towards marrying geo-distributed analytics with graph-parallel processing. We listed the challenges in doing so, and proposed a solution to address these challenges. We are actively building Monarch, a system that incorporates 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] Enterprise DBMS, Q1 2014. https://www.forrester.com/report/TechRadar+Enterprise+DBMS+Q1+2014/-/E-RES106801.

[2] Graph DBMS increased their popularity by 500% within the last 2 years. http://db-engines.com/en/blog_post//43.

[3] Stanford large network dataset collection. https://snap.stanford.edu/data/index.html.

[4] Ai, Z., Zhang, M., Wu, Y., Qian, X., Chen, K., and Zheng, W. Squeezing out all the value of loaded data: An out-of-core graph processing system with reduced disk i/o. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)* (Santa Clara, CA, 2017), USENIX Association, pp. 125–137.

[5] Ai, Z., Zhang, M., Wu, Y., Qian, X., Chen, K., and Zheng, W. Squeezing out all the value of loaded data: An out-of-core graph processing system with reduced disk i/o. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)* (Santa Clara, CA, 2017), USENIX Association, pp. 125–137.

[6] Buluç, A., and Gilbert, J. R. The combinatorial BLAS: design, implementation, and applications. *IJHPCA 25*, 4 (2011), 496–509.

[7] Cai, Z., Logothetis, D., and Siganos, G. Facilitating real-time graph mining. In *Proceedings of the Fourth International Workshop on Cloud Data Management* (New York, NY, USA, 2012), CloudDB '12, ACM, pp. 1–8.

[8] Chen, R., Shi, J., Chen, Y., and Chen, H. Powerlyra: Differentiated graph computation and partitioning on skewed graphs. In *Proceedings of the Tenth European Conference on Computer Systems* (New York, NY, USA, 2015), EuroSys '15, ACM, pp. 1:1–1:15.

[9] Cheng, R., Hong, J., Kyrola, A., Miao, Y., Weng, X., Wu, M., Yang, F., Zhou, L., Zhao, F., and Chen, E. Kineograph: Taking the pulse of a fast-changing and connected world. In *Proceedings of the 7th ACM European Conference on Computer Systems* (New York, NY, USA, 2012), EuroSys '12, ACM, pp. 85–98.

[10] Fan, W., Xu, J., Wu, Y., Yu, W., Jiang, J., Zheng, Z., Zhang, B., Cao, Y., and Tian, C. Parallelizing sequential graph computations. In *Proceedings of the 2017 ACM International Conference on Management of Data* (New York, NY, USA, 2017), SIGMOD '17, ACM, pp. 495–510.

[11] Gao, P., Zhang, M., Chen, K., Wu, Y., and Zheng, W. High performance graph processing with locality oriented design. *IEEE Transactions on Computers 66*, 7 (July 2017), 1261–1267.

[12] Gonzalez, J., Xin, R., Dave, A., Crankshaw, D., and Franklin, Stoica, I. Graphx: Graph processing in a distributed dataflow framework. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (Broomfield, CO, Oct. 2014), USENIX Association.

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

[14] Grossman, S., Litz, H., and Kozyrakis, C. Making pull-based graph processing performant. In *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (New York, NY, USA, 2018), PPoPP '18, ACM, pp. 246–260.

[15] Han, W., Miao, Y., Li, K., Wu, M., Yang, F., Zhou, L., Prabhakaran, V., Chen, W., and Chen, E. Chronos: A graph engine for temporal graph analysis. In *Proceedings of the Ninth European Conference on Computer Systems* (New York, NY, USA, 2014), EuroSys '14, ACM, pp. 1:1–1:14.

[16] Hsieh, K., Harlap, A., Vijaykumar, N., Konomis, D., Ganger, G. R., Gibbons, P. B., and Mutlu, O. Gaia: Geo-distributed machine learning approaching LAN speeds. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (Boston, MA, 2017), USENIX Association, pp. 629–647.

[17] Hung, C.-C., Golubchik, L., and Yu, M. Scheduling jobs across geo-distributed datacenters. In *Proceedings of the Sixth ACM Symposium on Cloud Computing* (New York, NY, USA, 2015), SoCC '15, ACM, pp. 111–124.

[18] Iyer, A., Li, L. E., and Stoica, I. Celliq : Real-time cellular network analytics at scale. In *Proceedings of the 12th USENIX conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2015), NSDI'15, USENIX Association.

[19] Khurana, U., and Deshpande, A. Efficient snapshot retrieval over historical graph data. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on* (April 2013), pp. 997–1008.

[20] Khurana, U., and Deshpande, A. Storing and analyzing historical graph data at scale. *CoRR abs/1509.08960* (2015).

[21] Kwak, H., Lee, C., Park, H., and Moon, S. What is Twitter, a social network or a news media? In *WWW '10: Proceedings of the 19th international conference on World wide web* (New York, NY, USA, 2010), ACM, pp. 591–600.

[22] Kyrola, A., Blelloch, G., and Guestrin, C. 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)* (Hollywood, CA, 2012), USENIX, pp. 31–46.

[23] Lattanzi, S., Moseley, B., Suri, S., and Vassilvitskii, S. Filtering: A method for solving graph problems in mapreduce. In *Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures* (New York, NY, USA, 2011), SPAA '11, ACM, pp. 85–94.

[24] Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J. M. Graphlab: A new framework for parallel machine learning. In *UAI* (2010), P. Grünwald and P. Spirtes, Eds., AUAI Press, pp. 340–349.

[25] Maass, S., Min, C., Kashyap, S., Kang, W., Kumar, M., and Kim, T. Mosaic: Processing a trillion-edge graph on a single machine. In *Proceedings of the Twelfth European Conference on Computer Systems, EuroSys 2017, Belgrade, Serbia, April 23-26, 2017* (2017), pp. 527–543.

[26] Maass, S., Min, C., Kashyap, S., Kang, W., Kumar, M., and Kim, T. Mosaic: Processing a trillion-edge graph on a single machine. In *Proceedings of the Twelfth European Conference on Computer Systems* (New York, NY, USA, 2017), EuroSys '17, ACM, pp. 527–543.

[27] Macko, P., Marathe, V. J., Margo, D. W., and Seltzer, M. I. Llama: Efficient graph analytics using large multiversioned arrays. In *2015 IEEE 31st International Conference on Data Engineering* (April 2015), pp. 363–374.

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

[29] Miao, Y., Han, W., Li, K., Wu, M., Yang, F., Zhou, L., Prabhakaran, V., Chen, E., and Chen, W. Immortalgraph: A system for storage and analysis of temporal graphs. *Trans. Storage 11*, 3 (July 2015), 14:1–14:34.

[30] Microsoft Naiad Team. GraphLINQ: A graph library for naiad. `http://bigdataatsvc.wordpress.com/2014/05/08/graphlinq-a-graph-library-for-naiad/`, 2014.

[31] Murray, D. G., McSherry, F., Isaacs, R., Isard, M., Barham, P., and Abadi, M. Naiad: A timely dataflow system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP '13, ACM, pp. 439–455.

[32] Padmanabha Iyer, A., Li, L. E., Das, T., and Stoica, I. Time-evolving graph processing at scale. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems* (2016), ACM, p. 5.

[33] Prabhakaran, V., Wu, M., Weng, X., McSherry, F., Zhou, L., and Haradasan, M. Managing large graphs on multi-cores with graph awareness. In *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)* (Boston, MA, 2012), USENIX, pp. 41–52.

[34] Pu, Q., Ananthanarayanan, G., Bodik, P., Kandula, S., Akella, A., Bahl, P., and Stoica, I. Low latency geo-distributed data analytics. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (New York, NY, USA, 2015), SIGCOMM '15, ACM, pp. 421–434.

[35] Quamar, A., Deshpande, A., and Lin, J. Nscale: Neighborhood-centric large-scale graph analytics in the cloud. *The VLDB Journal 25*, 2 (Apr. 2016), 125–150.

[36] Roy, A., Bindschaedler, L., Malicevic, J., and Zwaenepoel, W. Chaos: Scale-out graph processing from secondary storage. In *Proceedings of the 25th Symposium on Operating Systems Principles* (New York, NY, USA, 2015), SOSP '15, ACM, pp. 410–424.

[37] Roy, A., Mihailovic, I., and Zwaenepoel, W. X-stream: Edge-centric graph processing using streaming partitions. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP '13, ACM, pp. 472–488.

[38] Teixeira, C. H. C., Fonseca, A. J., Serafini, M., Siganos, G., Zaki, M. J., and Aboulnaga, A. Arabesque: A system for distributed graph mining - extended version. *CoRR abs/1510.04233* (2015).

[39] Viswanathan, R., Ananthanarayanan, G., and Akella, A. Clarinet: Wan-aware optimization for analytics queries. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (GA, 2016), USENIX Association, pp. 435–450.

[40] Vulimiri, A., Curino, C., Godfrey, P. B., Jungblut, T., Karanasos, K., Padhye, J., and Varghese, G. Wanalytics: Geo-distributed analytics for a data intensive world. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2015), SIGMOD '15, ACM, pp. 1087–1092.

[41] Wang, G., Xie, W., Demers, A. J., and Gehrke, J. Asynchronous large-scale graph processing made easy. In *CIDR* (2013).

[42] Wang, G., Xie, W., Demers, A. J., and Gehrke, J. Asynchronous large-scale graph processing made easy. In *CIDR* (2013), www.cidrdb.org.

[43] Wu, M., and Jin, R. A graph-based framework for relation propagation and its application to multi-label learning. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2006), SIGIR '06, ACM, pp. 717–718.

[44] Wu, M., Yang, F., Xue, J., Xiao, W., Miao, Y., Wei, L., Lin, H., Dai, Y., and Zhou, L. Gram: Scaling graph computation to the trillions. In *Proceedings of the Sixth ACM Symposium on Cloud Computing* (New York, NY, USA, 2015), SoCC '15, ACM, pp. 408–421.

[45] Xiao, W., Xue, J., Miao, Y., Li, Z., Chen, C., Wu, M., Li, W., and Zhou, L. Tux^2: Distributed graph computation for machine learning. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (Boston, MA, 2017), USENIX Association, pp. 669–682.

[46] Xie, W., Wang, G., Bindel, D., Demers, A., and Gehrke, J. Fast iterative graph computation with block updates. *Proc. VLDB Endow. 6*, 14 (Sept. 2013), 2014–2025.

[47] Zhang, M., Wu, Y., Chen, K., Qian, X., Li, X., and Zheng, W. Exploring the hidden dimension in graph processing. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (GA, 2016), USENIX Association, pp. 285–300.

[48] Zhang, M., Wu, Y., Chen, K., Qian, X., Li, X., and Zheng, W. Exploring the hidden dimension in graph processing. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (Savannah, GA, 2016), USENIX Association, pp. 285–300.

[49] Zhang, M., Wu, Y., Zhuo, Y., Qian, X., Huan, C., and Chen, K. Wonderland: A novel abstraction-based out-of-core graph processing system. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2018), ASPLOS '18, ACM, pp. 608–621.

[50] Zhang, M., Zhuo, Y., Wang, C., Gao, M., Wu, Y., Chen, K., Kozyrakis, C., and Qian, X. Graphp: Reducing communication for pim-based graph processing with efficient data partition. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (Feb 2018), pp. 544–557.

[51] Zhang, Y., Kiriansky, V., Mendis, C., Amarasinghe, S., and Zaharia, M. Making caches work for graph analytics. In *2017 IEEE International Conference on Big Data (Big Data)* (Dec 2017), pp. 293–302.

[52] Zhu, X., Chen, W., Zheng, W., and Ma, X. Gemini: A computation-centric distributed graph processing system. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (GA, 2016), USENIX Association, pp. 301–316.