Wave : A Substrate for Distributed Incremental Graph Processing on Commodity Clusters

Swapnil Gandhi
Indian Institute of Science, Bangalore
gandhis@IISc.ac.in

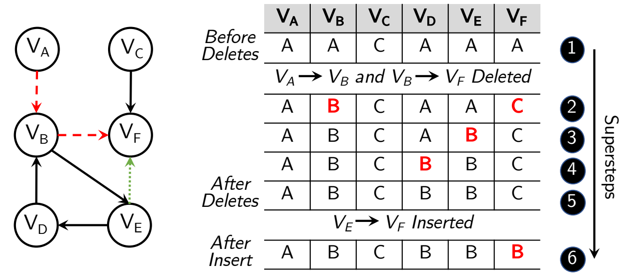
1 Research Problem and Motivation

Linked data in diverse fields of science, engineering and business tend to be continuously evolving, leading to significant interest in *dynamic graph processing* [1]. At the heart of this is a graph whose structure and/or properties change rapidly ($> 10^3/sec$) through a stream of *non-monotonic updates*[1], and a user algorithm which consistently *maintains insights* on the dynamic graph, with low latency. E.g., fraud detection analyzes and recognizes patterns between customers (vertices) and financial transactions (edges, properties) in real-time to preempt losses [2].

While such *large dynamic graphs* are ubiquitous, there are few abstractions and distributed platforms for analyzing them at scale. Some [12, 15, 26] discretize the dynamic graph into a sequence of snapshots and recompute them from scratch. These leverage existing offline graph platforms [10, 17, 19, 22, 23] and algorithms [30], but cause substantial redundancy in computation and distributed communication which limits the update rate that can be maintained [7].

A fundamental approach [11, 13, 20] to minimize such redundancy is to perform *incremental processing* [21]. However, naïvely resuming computation from an initial state or the neighborhood state of the modified vertices/edges may cause the algorithm to produce incorrect results. E.g., in Fig. 1(a), we label vertices with the smallest vertex ID of the *connected component* they are part of. Initially, all are part of the component A. When deleting edge V_A–V_B and V_B–V_F, vertices V_B and V_F are affected. But updating their component labels just based on their neighbors causes V_B to be incorrectly labeled with A by V_D, while V_F is correctly labeled as C by V_C. This is due to V_B and V_D being part of a cycle. Such inaccuracies will also cascade with future updates. Prior works [24, 25, 29] have examined incremental computation for monotonic graph updates and a sub-class of graph algorithms, but these do not support non-monotonic algorithms like PageRank.

We present Wave, a principled approach for generalized incremental graph processing over distributed machines. Users design graph algorithms in the familiar vertex-centric programming model [17], which execute iteratively over *supersteps* using Bulk Synchronous Parallel (BSP) [28]. Wave avoids redundant computation and communication by *dynamically tracking state dependencies among vertices* to decide if incremental computation is required. If so, it *transparently schedules vertex execution and state inheritance* at an appropriate superstep. The results are identical to recomputing the algorithm on the new graph, but *orders of magnitude faster.*



	V_A	V_B	V_C	V_D	V_E	V_F	
Before Deletes	A	A	C	A	A	A	1
	V_A → V_B and V_B → V_F Deleted						
	A	B	C	A	A	C	2
	A	B	C	A	B	C	3
	A	B	C	B	B	C	4
After Deletes	A	B	C	B	B	C	5
	V_E → V_F Inserted						
After Insert	A	B	C	B	B	B	6

(a) Dynamic Graph

(b) Vertex state updates over supersteps when finding *Connected Components*. Changes shown in **red**.

Figure 1. Incremental processing using Wave

2 Related Work

Existing *static graph processing systems* [10, 16, 17, 19, 22, 23, 31] can be used to process a snapshot-at-a-time. These assume the graph updates have been applied to create a snapshot at a point in time. However, fully recomputing a snapshot duplicates the compute and messaging, limits scalability and increases latency. Several batch systems and abstractions [8, 9, 12, 18, 27] try to mitigate this inefficiency. Other software frameworks, including GraphInc [4], STINGER [6], DegAwareRHH [14], and Tegra [15] provide point-in-time snapshot views to implement algorithms. Instead, Wave is designed for online and incremental processing.

A simple means for *incremental graph processing* is to reset the state of vertices affected by the graph update and their dependent vertices, and recompute on all modified vertices. But this can reset and recompute a majority of vertices [5, 24]. Tornado [25] processes streams of graph updates by forking execution to process user-queries while the graph structure updates in the main branch. Kickstarter [29] uses a global dependence tree to maintain state dependencies over RDMA. *However, all of these are limited to the subclass of monotonic graph updates and algorithms.*

[1] *Non-monotonic updates* allow both vertex/edge additions *and* deletions (mutations), while *monotonic* limits these to only additions.

3 Approach

Two key challenges in incrementally processing of dynamic graphs are to identify (1) which vertices of the graph are affected by updates and require recomputation, and (2) what prior states should be used in the recomputation. We make two observations that help address these:

1. *We say vertex v is dependent on vertex u if there is a path from u to v.* Here, if the state of a vertex or its adjacent edge changes, or they are mutated, the affected vertex has to be recomputed. This *may* cascade a recomputation to all its dependent vertices.
2. *For vertices with cyclical dependencies, all vertices that are part of the cycle in which a vertex is updated need to be recomputed.* For linearly dependent vertices, the updated vertex state depends only on the state of the immediate new neighbors of the vertex that is mutated, and can cascade. For vertices with cyclical dependency, updated state depends on the state and neighbors of all vertices in the cycle. E.g., for the deletes in Fig. 1(a), the cycle V_B–V_F–V_D means V_B cannot directly use V_D's state A, while the linear dependence between V_F–V_C means that V_F can directly use V_C's state C.

Naïvely tracking transitive dependencies is expensive, taking $O(V^2)$ space. Instead, we maintain a *level* information for each vertex that is the path-length from the "source" vertex that its current state is causally dependent on. Intuitively, if $level(v) < level(u)$, then v is not dependent on u; else, v may be dependent on u. While Kickstarter [29] too uses levels, it maintains a costly global level tree.

Levels also help detect cycles in vertex-centric computation, where message updates traverse one edge per superstep. If a vertex is not dependent on an updated vertex, it is not part of its cycle. Else, any update on a vertex should propagate and wait for $s = level(u) - level(v)$ supersteps to see if it is returned back. If so, there is a cycle and a recompute of all vertices in the cycle may be needed. In the worst case, $2s-1$ supersteps are required to converge. E.g., in Fig. 1(b), deleting the edges causes Wave to wait for 3 supersteps for V_B's updates to propagate through the 3-cycle, and subsequently converge in the 4^{th} superstep. But the edge delete and add that affect V_F can converge in 1 superstep as it is not part of a cycle.

Besides levels, vertices also maintain the last received update message from its adjacent vertex. This *memoization* ensures that causally dependent vertices that are not part of a cycle can immediately reuse an alternative recent message from a neighbor to converge to a solution in one superstep. This trades off memory but saves communication and synchronization costs. The number of vertices for which messages are memoized is dynamically tuned. Further, using just level information to detect cycles is *conservative*, causing up to $2s-1$ supersteps. As an optimization, we maintain a fixed-length *Bloom filter* at each vertex that tracks vertices that are part of a cycle it is part of. Since Bloom filters have no false negatives, we know that the absence of an updated vertex at a given vertex's filter means it is not dependent or part of its cycle. This can save s supersteps.

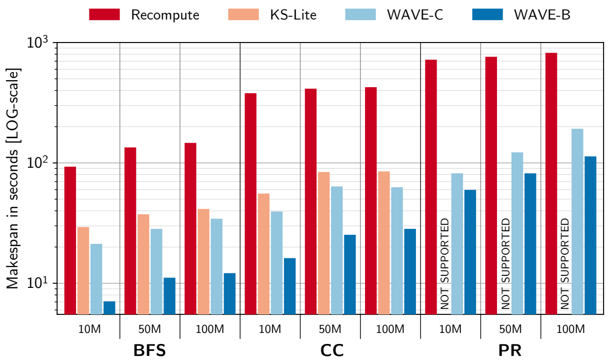


Figure 2. Incremental Graph Processing on Twitter [3] Dataset. Size of update batch is shown on inner X axis.

4 Results and Contributions

We evaluate the performance of Wave for Breadth First Search (BFS), Connected Components (CC) and PageRank (PR) algorithms, on the Twitter graph with 41M vertices and 1.4B edges initially [3]. Graph updates batched into 10M, 50M and 100M edge adds and deletes, in equal number, are applied before the incremental computation [15, 25, 29]. We compare it against two baselines. *Recompute* reruns the full user algorithm on the updated snapshot. *KS-Lite* is a vertex-centric version of the incremental Kickstarter approach [29], but like the original, does not support the non-monotonic PR algorithm. Besides the conservative Wave-C that defers compute supersteps based on levels, we also evaluate the efficient Wave-B that uses Bloom filters to identify if vertices are not in a cycle before awaiting superteps.

Fig. 2 shows the time taken by these 4 strategies on 8 machines. Wave-C is 7–23× faster than Recompute and 3–4× faster than KS-Lite. Wave-B is even faster than Wave-C by 1.37–3×. These benefits correlate with fewer (approx. 40%) vertex recomputes. Larger batch sizes increases the throughput (average updates/sec) for the incremental approaches, with Wave-B supporting a peak of 8.3M updates/sec for BFS; close to a 1 million updates/sec per machine.

Contributions. In summary, Wave makes three primary contributions: (1) It is one of the first platforms to support incremental graph processing for both monotonic and non-monotonic algorithms, with minimal development overheads; (2) It uses a light-weight, dynamic tracking of vertex state dependency to determine if recompute is required and if so, the state to use and supersteps to schedule vertex processing to ensure correctness; and (3) A prototype implementation of Wave outperforms comparable baselines by ≈ 6×. Together, it offers a novel, intuitive and scalable strategy for incremental graph processing.

References

[1] Charu Aggarwal and Karthik Subbian. 2014. Evolutionary Network Analysis: A Survey. *ACM Comput. Surv.* 47, 1, Article 10 (May 2014), 36 pages. https://doi.org/10.1145/2601412

[2] Alaa Mahmoud. 2017. (2017). https://developer.ibm.com/dwblog/2017/detecting-complex-fraud-real-time-graph-databases/ [Accessed online on 12 August 2019].

[3] 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*, Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar (Eds.). ACM Press, 587–596.

[4] Zhuhua Cai, Dionysios Logothetis, and Georgos Siganos. 2012. Facilitating Real-time Graph Mining. In *CloudDB*.

[5] 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

[6] D. Ediger, R. McColl, J. Riedy, and D. A. Bader. 2012. STINGER: High performance data structure for streaming graphs. In *2012 IEEE Conference on High Performance Extreme Computing*. 1–5. https://doi.org/10.1109/HPEC.2012.6408680

[7] Wenfei Fan, Jianzhong Li, Jizhou Luo, Zijing Tan, Xin Wang, and Yinghui Wu. 2011. Incremental Graph Pattern Matching. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD '11)*. ACM, New York, NY, USA, 925–936. https://doi.org/10.1145/1989323.1989420

[8] Swapnil Gandhi and Yogesh Simmhan. 2019. *Graphite: An Interval-centric Model for Distributed Computing over Temporal Graphs*. Technical Report. Indian Institute of Science. https://www.w3id.org/dream-lab/pubs/icm.pdf.

[9] Swapnil Gandhi and Yogesh Simmhan. 2020. An Interval-centric Model for Distributed Computing over Temporal Graphs. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*.

[10] Joseph E. Gonzalez, Reynold S. 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, 599–613. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/gonzalez

[11] Pradeep Kumar Gunda, Lenin Ravindranath, Chandramohan A. Thekkath, Yuan Yu, and Li Zhuang. 2010. Nectar: Automatic Management of Data and Computation in Datacenters. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*. USENIX Association, Berkeley, CA, USA, 75–88. http://dl.acm.org/citation.cfm?id=1924943.1924949

[12] 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

[13] Bingsheng He, Mao Yang, Zhenyu Guo, Rishan Chen, Bing Su, Wei Lin, and Lidong Zhou. 2010. Comet: Batched Stream Processing for Data Intensive Distributed Computing. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC '10)*. ACM, New York, NY, USA, 63–74. https://doi.org/10.1145/1807128.1807139

[14] K. Iwabuchi, S. Sallinen, R. Pearce, B. V. Essen, M. Gokhale, and S. Matsuoka. 2016. Towards a Distributed Large-Scale Dynamic Graph Data Store. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 892–901. https://doi.org/10.1109/IPDPSW.2016.189

[15] 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 (GRADES '16)*. ACM, New York, NY, USA, Article 5, 6 pages. https://doi.org/10.1145/2960414.2960419

[16] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. 2012. Distributed GraphLab: a framework for machine learning and data mining in the cloud. *PVLDB* 5, 8 (2012), 716–727.

[17] 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

[18] Youshan Miao, Wentao Han, Kaiwei Li, Ming Wu, Fan Yang, Lidong Zhou, Vijayan Prabhakaran, Enhong Chen, and Wenguang Chen. 2015. ImmortalGraph: A System for Storage and Analysis of Temporal Graphs. *Trans. Storage* 11, 3, Article 14 (July 2015), 34 pages. https://doi.org/10.1145/2700302

[19] Donald Nguyen, Andrew Lenharth, and Keshav Pingali. 2013. A Lightweight Infrastructure for Graph Analytics. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13)*. ACM, New York, NY, USA, 456–471. https://doi.org/10.1145/2517349.2522739

[20] Daniel Peng and Frank Dabek. 2010. Large-scale Incremental Processing Using Distributed Transactions and Notifications. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*. USENIX Association, Berkeley, CA, USA, 251–264. http://dl.acm.org/citation.cfm?id=1924943.1924961

[21] G. Ramalingam and Thomas Reps. 1993. A Categorized Bibliography on Incremental Computation. In *Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '93)*. ACM, New York, NY, USA, 502–510. https://doi.org/10.1145/158511.158710

[22] 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

[23] 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

[24] S. Sallinen, R. Pearce, and M. Ripeanu. 2019. Incremental Graph Processing for On-Line Analytics. In *2019 IEEE International Parallel and Distributed Processing Symposium*.

[25] Xiaogang Shi, Bin Cui, Yingxia Shao, and Yunhai Tong. 2016. Tornado: A System For Real-Time Iterative Analysis Over Evolving Data. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. ACM, New York, NY, USA, 417–430. https://doi.org/10.1145/2882903.2882950

[26] Y. Simmhan, N. Choudhury, C. Wickramaarachchi, A. Kumbhare, M. Frincu, C. Raghavendra, and V. Prasanna. 2015. Distributed Programming over Time-Series Graphs. In *IPDPS*.

[27] Manuel Then, Timo Kersten, Stephan Günnemann, Alfons Kemper, and Thomas Neumann. 2017. Automatic Algorithm Transformation for Efficient Multi-snapshot Analytics on Temporal Graphs. *PVLDB* 10, 8 (April 2017), 877–888. https://doi.org/10.14778/3090163.3090166

[28] Leslie G. Valiant. 1990. A Bridging Model for Parallel Computation. *Commun. ACM* 33, 8 (Aug. 1990), 103–111. https://doi.org/10.1145/79173.79181

[29] Keval Vora, Rajiv Gupta, and Guoqing Xu. 2017. KickStarter: Fast and Accurate Computations on Streaming Graphs via Trimmed Approximations. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '17)*. ACM, New York, NY, USA, 237–251. https://doi.org/10.1145/3037697.3037748

[30] Da Yan, James Cheng, Kai Xing, Yi Lu, Wilfred Ng, and Yingyi Bu. 2014. Pregel Algorithms for Graph Connectivity Problems with Performance Guarantees. *Proc. VLDB Endow.* 7, 14 (Oct. 2014), 1821–1832. https://doi.org/10.14778/2733085.2733089

[31] Xiaowei Zhu, Wenguang Chen, Weimin Zheng, and Xiaosong Ma. 2016. Gemini: A Computation-Centric Distributed Graph Processing System. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 301–316. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/zhu