Time-Evolving Graph Processing on Commodity Clusters

Anand Padmanabha Iyer
UC Berkeley

1. Problem & Motivation

Graph-structured data is on the rise, in size, complexity and dynamism [1]. This growth has spurred the development of a large number of graph processing systems [5, 6, 8, 10–12, 14, 16–18, 21, 24–27, 29] in both academia and open-source community. By leveraging specialized abstractions and careful optimizations, these systems have the ability to analyze large, *static* graphs, some in the order of trillion edges [9]. However, real-world graphs are seldom static. Consider, for instance, the familiar example of social network graphs such as in Facebook and Twitter. In such networks, "friends" relations, tweets with "mentions" are created continuously resulting in the graph's constant evolution. The dynamic aspect of such graphs makes it more difficult to answer questions like, *"What are the trending topics at the moment?"*, or *"Who was Bob's most active friend in 2016?"*. In another example, cellular operators collect massive amounts of real-time data useful for network diagnostics [13]. We can also find evolving graphs in Internet of Things (IoT) applications such as connected cars [3], transaction graph in financial networks, and disease propagation graph. Mining these *time-evolving graphs* can be useful, from both scientific and commercial perspectives.

Ideally, a system for time-evolving graph processing must be able to provide the same analysis abilities as that on static graphs, but on any arbitrary point or points in the history of the graph. Intuitively, a time-evolving graph can be seen as a series of static graphs, called *snapshots*. Thus, time-evolving graph processing systems must be able to support interactive ad-hoc, incremental and streaming analysis on any arbitrary snapshot, or snapshot windows. The big challenge in achieving this ability is in efficiently accessing arbitrary snapshots and performing computations on them. There are two main techniques to provide access to snapshots: (1) store each snapshot that the user might ask for, (2) log all changes to the graph (commonly referred as *deltas*), and then use these logs to compute the requested snapshots. Unfortunately, the former technique becomes quickly infeasible due to the prohibitive storage requirements, while the later technique becomes computationally expensive and slow over time. While recent work in graph systems has made considerable progress to address these needs, they often only address a subset or are fundamentally inefficient.

In this work, we present Tegra[1], a time-evolving graph processing system that addresses this challenge. The key idea in Tegra is to share storage, computation and communication across snapshots in a time-evolving graph. Tegra hides the intricacies of state management and sharing from the end-user using *Timelapse*, a new abstraction for time-evolving graph processing. At a high level, a timelapse is formed by a series of snapshots starting from the original graph. This enables users (and computations) to work on independent *versions* of the graph, without having to worry about changes to the underlying evolving graph. Timelapses can be explicitly created by users, or implicitly by the system during computations. For efficiently sharing state and computation, Tegra implements Timelapse using a persistent indexing datastructure. Tegra exposes the Timelapse abstraction to the end-users using a very simple API that supports a wide-variety of time-evolving graph analysis tasks. For ad-hoc analysis, it allows retrieval of arbitrary snapshots. For windowing operations, it provides API to access all snapshots in a particular window. Finally, to support incremental and streaming analysis, as the graph is being updated[2] , Tegra provides APIs to enable higher level graph computational models to implement this functionality.

2. Background & Related Work

Analytics on Static Graphs: There are a large number of graph processing systems [5, 7, 10, 11, 16, 17, 25–30] that focus on iterative analytics of static graphs. Of these, some [16, 17, 27, 29] are single machine systems, while the rest support distributed processing. While many of the techniques can be applied to Tegra, they do not consider the requirements for dynamic graph processing such as incremental maintainence or ad-hoc analysis on snapshot.

ISBN [to be supplied]. . . $15.00

[1] for **T**ime **E**volving **GRA**ph.

[2] We also support deletions, as they represent a more realistic scenario. Deletion increases the complexity of updating previous computation results and in some cases may be sub-optimal compared to restarting the computation

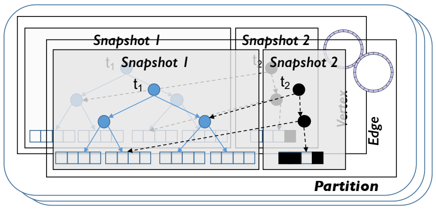

Figure 1: Tegra uses persistent datastructures to store and share snapshots as versions.

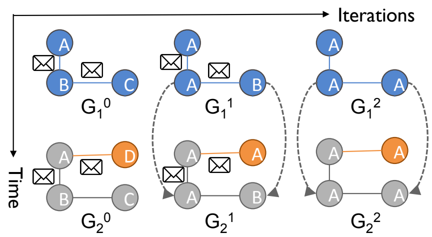

Figure 2: Tegra proivdes a general incremental computation model.

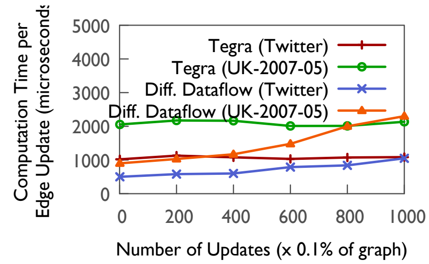

Figure 3: Streaming incremental computation performance against differential dataflow.

Graph Store for Evolving Graphs: The problem of managing time-evovling graph has been studied in the context of graph store [2, 4, 22, 23]. Those systems focus on supporting and optimizing point queries or simple graph queries.

Managing Graph Snapshots: A lot of systems took the idea to manage snapshots for a evolving graphs, so the problem is converted to analytics on a series of static graphs. Kineograph [8] supports constructing consistent snapshots of an evolving graph. However, it does not provide retrieving earlier version. Further, it relies on replicating the updates that are common among snapshots. DeltaGraph [14] proposes a hierarchical index that can manage multiple snapshots of a graph using deltas and event lists for efficient retrievals, but lacks the ability to do windowed iterative analytics. TAF [15] fixes this, but it is a specialized framework that does not provide a generalized incremental model. LLAMA [18] uses a multiversion array to support incremental ingestion. However, it is a single machine system, and it is unclear how the multiversion array can be extended to support data parallel operations required for iterative graph analytics.

Incremental Maintenence on Evolving Graphs: Another important body of work are the streaming systems. CellIQ [13] is a specialized system for cellular network analytics that shares some of the same objectives as Tegra. It does not support temporal analytics or snapshot management. GraphInc [6] supports incremental graph processing using memoization of the messages in graph parallel computation, but does not support snapshot generation or maintenance. Chronos [12] and ImmortalGraph [20] optimizes for efficient computation across snapshots. While similar in spirit to Tegra's Timelapse abstraction, it is not a general purpose graph processing engine nor does it support streaming ingestion of updates or window operations. Differential dataflow [19, 21, 24] enables fully dynamic computations on streaming datasets, but doesn't support ad-hoc analysis. While technically it can recreate a collection at any given time, this requires reconstruction.

3. Approach

Tegra is the first streaming graph processing system, to our best knowledge, that uniquely shares storage, computation and communication:

Sharing Storage: Tegra needs to store evolving graphs efficiently. It uses a persistent version of the adaptive radix tree to store entire snapshots without duplication (fig. 1). This datastructure is designed for real-time ingestion and retrieval of arbitrary snapshots. Tegra stores graphs and intermediate state as versions, which are branches. A version can be branched at any point from any other version. A version can be a single update, or a batch of updates. This is completely up to the user. There is no one total ordering of the versions. But the system can provide ordered access if needed. Version IDs can be composite keys, and can be partially matched.. Versions can be cached/uncached and can be written-to/read-from external sources (e.g., Parquet/RocksDB/Neo4j). Tegra exposes these versions to the user using a compact API.

Sharing Computation: Tegra provides generic incremental computations that accommodates edge deletions and gives the same semantics as complete re-execution of the algorithm (fig. 2). It stores intermediate computation state (e.g., iterations) using the same storage engine. When the graph is updated in the future, Tegra shares this saved computation by reusing it for parts of the graph that doesn't need recomputation. Further, it can branch saved intermediate state.

Sharing Communication: When Tegra has access to all the snapshots in a window, it is able to drastically reduce communication among graph entities across snapshots by sharing messages exchanged. Tegra simultaneously starts computation on all the snapshots, and eliminates duplicate messages before transmitting. It further reduces the message size by leveraging delta encoding.

4. Results

Figure 3 shows the performance of Tegra's streaming incremental computations by depicting the time taken to update the results of a connected component computation on two graphs when a fraction of the graph is updated. We see consistent performance compared to differential dataflow, whose performance suffers over time. Tegra can sustain an ingestion rate of over 1 million edge updates per second per machine. Finally, we have found that since Tegra only needs two snapshots to be in memory for streaming computations at any time, it can support millions of updates with virtually no degradation in performance.

References

[1] Graph dbms increased their popularity by 500 http://db-engines.com/en/blog_post//43.

[2] Weaver: A scalable, fast, consistent graph store. http://weaver.systems.

[3] Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing* (New York, NY, USA, 2012), MCC '12, ACM, pp. 13–16.

[4] Bronson, N., Amsden, Z., Cabrera, G., Chakka, P., Dimov, P., Ding, H., Ferris, J., Giardullo, A., Kulkarni, S., Li, H., Marchukov, M., Petrov, D., Puzar, L., Song, Y. J., and Venkataramani, V. Tao: Facebook's distributed data store for the social graph. In *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)* (San Jose, CA, 2013), USENIX, pp. 49–60.

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

[6] 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.

[7] 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.

[8] 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.

[9] Ching, A., Edunov, S., Kabiljo, M., Logothetis, D., and Muthukrishnan, S. One trillion edges: Graph processing at facebook-scale. *Proc. VLDB Endow. 8*, 12 (Aug. 2015), 1804–1815.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] McSherry, F., Murray, D. G., Isaacs, R., and Isard, M. Differential dataflow. In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings* (2013).

[20] 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.

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

[22] Mondal, J., and Deshpande, A. Eagr: Supporting continuous ego-centric aggregate queries over large dynamic graphs. *CoRR abs/1404.6570* (2014).

[23] Mondal, J., and Deshpande, A. Stream querying and reasoning on social data. In *Encyclopedia of Social Network Analysis and Mining* (2014), pp. 2063–2075.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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).

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

[30] 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.