Taking the Pulse of Financial Activities with Online Graph Processing

Xiaowei Zhu[1], Zhisong Fu[1], Zhenxuan Pan[1], Jin Jiang[1], Chuntao Hong[1], Yongchao Liu[1], Yang Fang[1], Wenguang Chen[2,1], and Changhua He[1]

[1]Ant Group
[2]Tsinghua University

Abstract

Graph processing has been widely adopted in various financial scenarios at Ant Group to detect malicious and prohibited user behaviors. The low latency requirement under big data volume and high throughput raises rigorous challenges for efficient online graph processing. This paper gives a brief introduction of our encountered issues, the current solutions, and some future directions we are exploring.

1 Introduction

Graph processing is a powerful tool in financial scenarios, widely adopted within Ant Group [1]. Various types of data sources can be modelled as graphs: transactions recording fund transfers between user accounts, and social networks connecting users are representatives that naturally form graph-structured datasets; besides, there are many cases in which graph objects can be extracted from "flat" data, e.g. activities in Alipay [2] can link users through commonly used medium, such as devices, mini-programs, etc. In fact, sometimes multiple graphs may be used together, while some graphs are shared by multiple scenarios.

Many applications have been built on top of these graphs, with risk control being the largest domain in use. Graph analytics can effectively detect malicious and prohibitive financial activities such as fraud, money laundering, gambling, and many others as these actions are usually performed by densely connected groups of entities. While community detection algorithms can be used on periodically dumped graph snapshots, the long latencies caused by the offline processing nature make them difficult to keep up with the fast pace of massive mobile and online payments or other actions made in Alipay everyday.

To this end, we have developed our own systems to enable online dynamic graph analytics, so that risk control decisions can be made timely enough to meet the quality of service. We also build a simulation framework so that developers can try out new strategies on historical graph snapshots and test whether proposed new queries could bring desired benefits. This paper gives a brief overview of the major challenges we have been facing, our current working solutions and tradeoffs, as well as our long-term vision towards more efficient online graph processing.

2 Challenges

Querying on a graph that is changing frequently is currently the most widely adopted way how graph-structured data is used in financial risk control scenarios. The challenges come from various aspects, described as follows.

Big data volume generated with high velocity. There are hundreds of millions of daily active users, and a total of billions of users in Alipay. During promotion events, there can be up to billions of payments each day (e.g. Singles' Day [3]), with hundreds of thousands per second at peak times.

Power-law degree distribution. This is a common feature of most real-world graphs [4]. The hotspots (e.g. top merchant accounts) can have much more associated relationships than normal personal accounts.

Complex query patterns. Compared to relational queries in OLTP scenarios, complexities of graph queries are significantly higher, sometimes involving deep explorations with up to six-hop traversals. Some queries like shortest path and cycle detection involve iterative computations.

Strong need for simulation. In some business scenarios such as credit payments, a new graph processing strategy needs to be tested for a long duration to see whether it could bring stable and consistent improvements. Query simulation using historical data is thus in great need to enable fast development.

Strict tail latency requirement. The risk control decisions usually need to be made within hundreds of milliseconds, and only a portion of these can be allocated to graph processing. Many workloads require tail latencies to be shorter than 20 milliseconds [5].

High service availability requirement. This is a common and important requirement of all financial applications. Repli-

cation and consensus protocols are employed to avoid or reduce serving failures.

3 Present Solutions

We perform both synchronous and asynchronous graph computation in our online scenarios, tailored to simple and complex queries respectively. Their design decisions and tradeoffs will be described next.

3.1 Synchronous Graph Computation

For scenarios relying on synchronous graph computation, clients send requests (i.e. updates and queries) to servers, and wait for corresponding responses synchronously (Figure 1). No existing open-source graph storage systems could meet our demands of low-latency and high-throughput query processing on large-scale graphs, which led us to develop GeaBase [6], our own graph database solution.

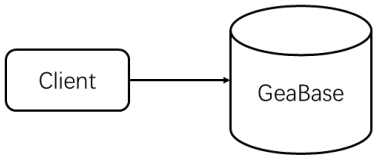

Figure 1: Synchronous graph computation with GeaBase

GeaBase adopts a shared-nothing architecture, spliting the whole graph into multiple shards. Each shard manages a portion of vertices and their attached edges with a local graph storage built on top of RocksDB [7]. Raft [8]-assisted synchronous replication is employed to achieve strong data consistency among multiple replicas of each shard.

While GeaBase is able to manage huge graphs with up to hundreds of billions of relationships, we still need to apply some skills or make some tradeoffs to satisfy the business requirements.

Pre-computation. We can perform aggregations (e.g. count, sum, max, etc.) on some properties along incident edges or neighboring vertices of each vertex in advance (i.e. modified upon each related update), so that later queries can just read the aggregated result rather than doing (at least) one more hop of traversals then performing computations. This strategy makes reads faster, at the cost of both space and write amplification.

Windowing. Due to the high velocity of data ingestion, it would be very expensive and even impossible to hold the complete graph evolution history. Thus we usually limit the lifespan of each entity by setting its TTL (Time-to-Live) upon creation, and sometimes only keep the latest version to reduce space consumption.

Restricted expanding. It is sometimes inevitable to come across high-degree vertices during graph traversals. Developers can add a hint in the query to use only a limited of number of edges when expanding as a workaround, so that latencies can be more stable and predictable.

Asynchronous updates. Sometimes updates can be applied in an asynchronous fashion, i.e. batched, to amortize the synchronous replication cost between data centers (possibly distributed in different cities). This strategy enables higher write throughputs. The downside is that queries may occasionally read stale data.

3.2 Asynchronous Graph Computation

Our current graph database solution is unable to process very complex queries, e.g. those more than three hops, under the tight latency requirements, as the complexity increases exponentially with growing depths of exploration. We therefore develop GeaFlow, a streaming graph processing system, to handle more complex queries in an asynchronous manner. In addition to the general streaming operators, GeaFlow provides a vertex-centric API [9] for developers to write graph processing logics.

The methodology is simple yet effective (Figure 2): clients write computing requests to message queues ahead of querying, i.e. predictively; GeaFlow takes requests from message queues, applying updates and performing queries; computed results are written to external storage systems, which can be read by clients later.

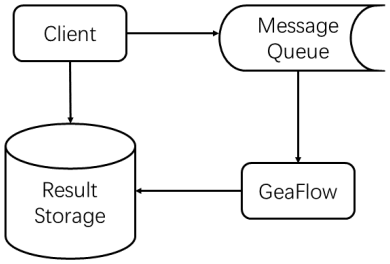

Figure 2: Asynchronous graph computation with GeaFlow

We should note that compared to synchronous mode, asynchronous computation has some restrictions. It is not usable for situations when graph processing and result retrieval cannot be separated into two events with significant intervals.

There are two major design tradeoffs in GeaFlow:

Mini-batch processing. Currently, graph processing operators in GeaFlow are executed in BSP super-steps [10], as batching could increase parallelism to provide adequate throughputs. The downside is that batching inevitably introduces longer latencies, sometimes leading to stale results (i.e. latest results not available yet when retrieving).

Decoupled processing and serving. The dynamic graph is maintained as states of the streaming system, while results are written to external storage systems. This decoupling of processing and serving enables each component to excel at their own areas. Result retrievals can be just as fast as a single key-value lookup, which can easily satisfy the tail latency requirement. However, when results are subsets of states, there would be data duplication.

While query simulation appears to be an offline workload, its temporal processing nature makes it very suitable for stream processing. The main distinctions between online processing and query simulation include following aspects:

Source and sink. Typically in online scenarios, message queues are used as data sources, while offline logs or data warehouse tables serve as the inputs to query simulation. As for sinks, online streaming writes results to external (usually key-value) storage systems, while query simulation dumps all results to data warehouses for further in-depth analytics.

Compaction window. The TTLs set in online scenarios mean physical time durations. Hence query simulation cannot expire data with the same mechanism, and a much smaller window size according to how fast simulation runs is set for data compaction.

4 Further Steps

While the existing adoptions of our systems within Ant Group have proven to be effective in many scenarios, we keep working on enhancement of our systems and try to open up new possiblities for existing and potentially new workloads.

4.1 Ongoing Actions

Our ongoing actions include the following:

Faster synchronous processing. We are exploring the use of non-volatile memory like Optane Persistent Memory [11] as a part of the backing storage to provide more stable and even lower latencies. The database team is also working on a new parallel computing engine which aims to enable more complex queries under the same latency requirement.

Faster asynchronous processing. We are prototyping a new dynamic graph computing system which aimes for lower complex query latencies by processing each request separately rather than in mini-batches.

Higher cost efficiency. Both GeaBase and GeaFlow currently use RocksDB, a tree-structured key-value store for graph storage. We are working on the replacement of old storage modules with a new hash-based engine whose memory consumption would be a lot smaller.

Better flexibility. We are seeking the possibilities for new replication strategies which enable multi-master writes with eventual consistency, so that writes do not need to be batched for high throughput.

Better usability. We have been making great efforts to ease the use of our systems. Many utilities like visualization, hotspots detection, and performance estimation are continuously being developed. We are also trying to unify the query languages of various graph processing systems, even those for analytical processing (i.e. iterative computation).

Benchmark proposal. To provide guidelines on system requirements and performance optimization towards online graph processing in financial scenarios, we are also developing a benchmark which is modelled on top of the key characteristics. Compared to existing graph querying benchmarks such as LDBC SNB [12], the proposed benchmark differs in various aspects, e.g. more strict latency requirements, inclusion of read-write operations, multi-edge generations with richer edge properties, etc.

4.2 Long-Term Vision

Besides the above ongoing actions, we have some long-term visionary goals:

Convergence of synchronous and asynchronous processing. We expect a single system that is capable of efficiently handling both synchronous computation of simple queries and asynchronous computation of complex queries to be the ultimate solution. Storage, computation, scheduling, and many other components need to be re-architected.

Connecting online and offline processing. We also have offline graph processing systems that perform global graph analytics and graph neural network computation. The current ETL pipelines can be improved through a better connection of online and offline systems, so that the batch processing latency could be significantly reduced.

Integration with other types of storage systems. Currently in many use cases, we have to duplicate another copy of data inside graph processing systems other than original data sources. A future graph processing system with better interoperability with e.g. relational databases, key-value stores, etc. would greatly reduce developing/deployment complexities as well as costs.

5 Conclusion

This paper aims to make a brief introduction of the challenges in online graph processing under the big picture of financial workloads and requirements, as well as our present solutions and future directions. We hope the content could bring inspirations of new techniques and open up new research topics.

References

[1] https://www.antgroup.com/.

[2] https://www.alipay.com/.

[3] https://en.wikipedia.org/wiki/Singles%27_Day.

[4] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, pages 17–30, 2012.

[5] Xiafei Qiu, Wubin Cen, Zhengping Qian, You Peng, Ying Zhang, Xuemin Lin, and Jingren Zhou. Real-time constrained cycle detection in large dynamic graphs. *Proceedings of the VLDB Endowment*, 11(12):1876–1888, 2018.

[6] Zhisong Fu, Zhengwei Wu, Houyi Li, Yize Li, Min Wu, Xiaojie Chen, Xiaomeng Ye, Benquan Yu, and Xi Hu. Geabase: a high-performance distributed graph database for industry-scale applications. *International Journal of High Performance Computing and Networking*, 15(1-2):12–21, 2019.

[7] Siying Dong, Mark Callaghan, Leonidas Galanis, Dhruba Borthakur, Tony Savor, and Michael Strum. Optimizing space amplification in rocksdb. In *CIDR*, volume 3, page 3, 2017.

[8] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, pages 305–319, 2014.

[9] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146, 2010.

[10] Leslie G Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[11] https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html.

[12] Renzo Angles, János Benjamin Antal, Alex Averbuch, Peter Boncz, Orri Erling, Andrey Gubichev, Vlad Haprian, Moritz Kaufmann, Josep Lluís Larriba Pey, Norbert Martínez, et al. The ldbc social network benchmark. *arXiv preprint arXiv:2001.02299*, 2020.