Efficient Partitioning and Query Processing of Spatio-Temporal Graphs with Trillion Edges

Mengsu Ding
State Key Laboratory of Computer Architecture
Institute of Computing Technology, CAS
University of Chinese Academy of Sciences
dingmengsu@ict.ac.cn

Shimin Chen*
State Key Laboratory of Computer Architecture
Institute of Computing Technology, CAS
University of Chinese Academy of Sciences
chensm@ict.ac.cn

***Abstract*—Real-world graphs often contain spatio-temporal information and evolve over time. Compared with static graphs, spatio-temporal graphs present more significant challenges in data volume, data velocity, and query processing. In this paper, we define a formal spatio-temporal graph model based on real-world applications, and propose PAST, a framework for efficient PArtitioning and query processing of Spatio-Temporal graphs. Our experimental results show that PAST improves query performance by orders of magnitude compared to state-of-the-art solutions.**

***Index Terms*—Spatio-temporal graphs, partitioning for spatio-temporal graphs, graph query processing, PAST**

I. Introduction

Graphs have been widely used to represent real-world entities and relationships. Real-world graphs often contain spatio-temporal information generated by a wide range of hardware devices (e.g., sensors, POS machines, traffic cameras, barcode scanners) and software systems (e.g., web servers).

A. Representative Applications

Understanding customer behaviors is helpful for detecting fraud and providing personalized services. For example, credit card companies track their customers' credit card uses for fraud detection. Internet companies track their users' browsing behaviors to achieve personalized recommendations. In a customer behavior tracking and mining application, people and locations can be modeled as graph vertices, while an edge linking a person vertex to a location vertex represents the event that the person visits the location at a certain time, as illustrated in Fig.1. This forms a spatio-temporal graph. People who visit similar locations around the same time period may have similar personal interests. Therefore, it is desirable to discover groups of people vertices that have similar edge structures in the spatio-temporal graphs.

Other representative applications include clone-plate car detection and shipment tracking. Recognizing car plates at traffic cameras in clone-plate car detection (scanning shipment packages with barcode scanners in shipment tracking) can be represented in spatio-temporal graphs similar to Fig.1, by replacing people with car plates (packages) and replacing locations with traffic cameras (barcode scanners).

*Corresponding Author

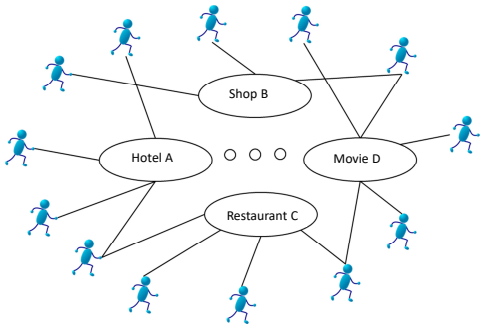

Fig. 1. Customer behavior tracking and mining.

B. Challenges of Supporting Spatio-Temporal Graphs

Compared with static graphs, spatio-temporal graphs pose more significant challenges in data volume, data velocity, and query processing:

Data Volume: ∼10 billion object vertices, ∼10 million location vertices, and ∼100 trillion edges. First, we refer to a vertex that represents a person, a car plate, or a package as an object vertex. The requirement of ∼10 billion object vertices is based on the fact that there are about 7.5 billion people in the world. Second, according to booking.com, there are about 1.9 million hotels and other accommodations in the world. Suppose there are 5 times more shops, restaurants, and theatres than hotels. Then the total number of locations can be on the order of 10 million. Finally, suppose an object vertex generates about 10 new edges per day. If we store the edges generated in the recent 3 years in the graph, there can be about 100 trillion edges in the graph. As a result, the data volume of a spatio-temporal graph can be much larger than static graphs. Suppose the properties of a vertex require at most 100B. Then the object vertices and the location vertices require about 1TB and 1GB space, respectively. An edge is a triplet (object ID, timestamp, location ID). Suppose each field takes 8B. Then an edge can be represented in 24B. Then 100 trillion edges require 2.4PB space. The 3-replica redundancy policy requires a total 7.2PB storage space, or roughly 10PB.

Data Velocity: up to 1 trillion new edges per day. Compared with static graphs, spatio-temporal graphs must support a large number of new edges per day. In the above, we estimate an average of 10 new edges per object vertex daily. Suppose peak

cases (e.g., Black Friday shopping) can see ten times more activities. Therefore, we would like to support 100 new edges per object vertex daily, i.e. 1 trillion new edges per day. This means 24TB new ingestion data per day, which requires a solution to support 290MB/s ingestion throughput.

Query Processing: prohibitive communication cost. The data volume and velocity challenges entail a distributed solution that stores the graph data on a number of machines. However, without careful designs, processing queries (e.g., customer behavior analysis) can easily lead to significant cross-machine communication. For example, in order to find subgroups of people with similar interests (i.e. visiting similar locations at similar time), it is necessary to combine the spatial information in location vertices, the temporal information in edges, and properties of person vertices in the computation. As the data volume is huge, the cross-machine communication cost can be prohibitively high.

C. Related Work

Existing graph partitioning solutions [1]–[5] focus mostly on static graphs, trying to minimize the number of cut edges across partitions and balance the partition sizes. However, these solutions do not take into account spatio-temporal characteristics, which are important for query processing in spatio-temporal graphs.

We compare several existing solutions for storing and processing spatio-temporal graphs in this paper: (i) JanusGraph [6], a state-of-the-art distributed graph database system; (ii) Greenplum [7], a state-of-the-art MPP relational database system; and (iii) Spark [8], a state-of-the-art big data analytics system. Unfortunately, their data storage / partition schemes cannot support spatio-temporal queries well. JanusGraph and Spark store data in key-value stores or distributed file systems (e.g., HDFS). The partition scheme is often random in such systems. Greenplum supports multi-dimensional partitioning. However, the data is first partitioned by the first partition dimension. Then the second partition dimension is applied to each first-level partition to obtain a set of second-level partitions, so on and so forth. Consequently, queries without filters on the first partition dimensions still incur significant cost. The experimental results in Section IV show that our proposed solution can achieve significantly better query performance.

D. Our Solution: PAST

We define a formal model for spatio-temporal graphs based on the representative applications. In this model, a spatio-temporal graph is a bipartite graph with a set of location vertices and a set of object vertices such that every edge connects an object vertex to a location vertex. Then, we present and evaluate PAST, a framework for efficient PArtitioning and query processing of Spatio-Temporal graphs. We vary partitioning and replication methods for location vertices, object vertices, and edges. Moreover, we exploit the multiple replicas of edges to design spatio-temporal partitions and key-temporal partitions. Our partitioning methods can efficiently support graph data storage and query processing. The experimental results show that PAST can successfully achieve the above goals. It improves query performance by orders of magnitude compared to state-of-the-art solutions, including JanusGraph, Greenplum, and Spark.

II. Spatio-temporal Graphs

We define a formal model for spatio-temporal graphs based on the representative applications:

Definition 1 (Spatio-temporal Graph): A spatio-temporal graph $G = (V_L, V_O, E)$. V_L is a set of location vertices. Every location vertex contains a location property. V_O is a set of object vertices that represent objects being tracked. Every vertex in V_L and V_O is assigned a globally unique vertex ID. E is a set of edges. Every edge in E connects an object vertex to a location vertex, and contains a time property.

In customer behavior tracking and mining, location vertices represent the locations that customers visit, and object vertices represent people. Every location vertex contains a location property such that given two location vertices u and v, their distance $dist(u, v)$ is well defined. For example, if the application is concerned about geographic locations, then the location property is the latitude and longitude of the location vertex. An edge is a (object ID, timestamp, location ID) triplet.

In essence, a spatio-temporal graph as defined in Definition 1 is a bi-parity graph. That is, we do not consider edges between object vertices and edges between location vertices. This abstraction captures the key characteristics and the main challenges of the representative applications.

A. Query Workload

We consider the following four types of queries based on the representative applications:

- **Q1: object trace**: Given an object and a time range, list the locations visited by the object during the time range. For example, Q1 can display activities of a customer or the trace of a shipment package in a period of time.
- **Q2: trace similarity**: Given two objects and a time range, compute the similarity of the two object traces during the time range. Consider edge ($o1$, $t1$, $l1$) in object $o1$'s trace and edge ($o2$, $t2$, $l2$) in object $o2$'s trace. The two edges are considered similar if $|t2 - t1| \leq TH_{time}$ and $dist(l2, l1) \leq TH_{dist}$, where TH_{time} and TH_{dist} are predefined thresholds on time and location distance, respectively. The similarity of the two traces is the count of similar edge pairs in the two traces.
- **Q3: similar object discovery**: Given an object o and a time range, list the objects that have similar traces compared to o. Display the list in the descending order of trace similarity with o. Q3 can be used to discover people with similar interests in the customer behavior tracking and mining application.
- **Q4: clone object detection**: Given a time range, discover all the clone objects. An object o is a clone object if there exists two incident edges (o, $t1$, $l1$) and (o, $t2$, $l2$) such that the computed velocity is beyond a predefined threshold: $\frac{dist(l2,l1)}{|t2-t1|} > TH_{velocity}$. Q4 supports detection of

duplicate credit cards in customer behavior tracking and mining or cloned car plates in clone-plate car detection.

III. PAST Overview

We propose PAST, a framework for efficient PArtitioning and query processing of Spatio-Temporal graphs. The system architecture of PAST is shown in Fig.2.

There is a coordinator machine and a large number of (e.g., 1000) worker machines. The coordinator keeps track of meta information of the graph partitions and coordinates data ingestion. The workers store the graph data, handle incoming updates, and process queries.

Graph Partitioning and Storage. The GraphStore component in Fig.2 implements PAST's graph partitioning methods and supports compressed storage of the main graph data. The GraphStore utilizes an underlying DB/storage system (e.g., the Cassandra key-value store in our implementation).

The number of location vertices is 1/1000 of that of object vertices. As the spatio-temporal graph is a bipartite graph, the average degree of a location vertex is 1000 times of that of an object vertex. Consequently, they have very different impact on the communication patterns in query processing. We choose different partition methods for the two types of vertices. For location vertices, we replicate all the location vertices at every worker so that location lookup and computation can be fast. For object vertices, we employ hash-based partitions.

The number of edges is 4 orders of magnitude higher than the number of vertices. Therefore, edges consume much more space than vertices, and are often the performance critical factor in query processing. Consider the four query types. All four queries filter the edges with a given time range. Therefore, the storage of edge data should support time range filters efficiently. Q1, Q2, and Q4 access edges for a given object, two objects, and all objects, respectively. Hence, it would be nice to organize edges according to object IDs. On the other hand, Q3 can be more efficiently computed if edges are stored in spatio-aware orders. In this way, the system can filter out a large number of edges that are not relevant to the trace of the given object. It seems that the requirements of the four query types are contradicting.

We solve this problem by taking advantage of the multiple replicas of edges. For the purpose of fault tolerance, PAST stores multiple replicas for edges (e.g., 3 replicas). Therefore, we propose a spatio-temporal edge partition method and a key-temporal edge partition method for different edge replicas.

High-throughput Streaming Edge Ingestion. New edge updates are streamed in rapidly. As shown in Fig. 2, the IngestStore component maintains an in-memory staging buffer for incoming new edges. All the worker machines handle incoming edges in rounds. They keep loosely synchronized clocks. In every round, the IngestStores collect the incoming edges in the current round. At the same time, IngstStores send edges collected in the previous round to their destination GraphStores based on the partitions computed by PAST's partition methods. We design an efficient algorithm to perform the data ingestion in the paper.

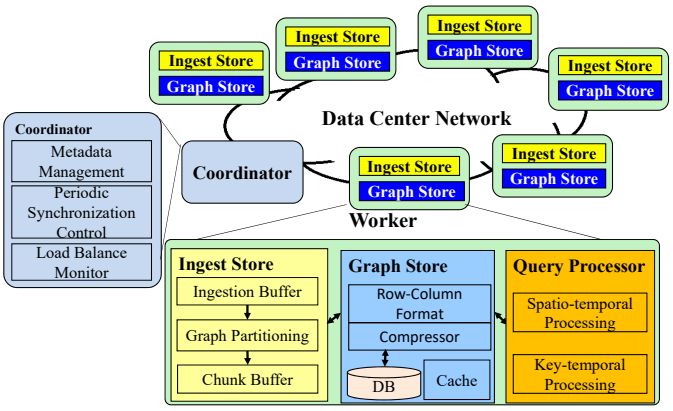

Fig. 2. PAST system architecture.

Query Processing and Optimization. Given PAST's partition methods, we design a cost-based query optimizer to choose the best partitions for an input query. Our goal is to reduce cross-machine communication and edge data access as much as possible. Our edge partitions divide the spatio-temporal space and key-temporal space into discretized blocks. We perform block-level filtering to avoid reading irrelevant edge data. Then we also take advantage of triangle inequalities for finer-grain filtering if geographic locations are used in the application.

IV. Experimental Evaluation

In this section, we present preliminary evaluation results for understanding the performance of PAST. We study (i) edge ingestion throughput and (ii) query performance.

A. Experimental Setup

Machine Configuration. The experiments are performed on a cluster of 11 nodes. Each machine is a Dell PowerEdge blade server equipped with two Intel(R) Xeon(R) E5-2650 v3 2.30GHz CPUs (10 cores/20 threads, 25.6MB cache), 128GB memory, a 1TB HDD and a 180GB SSD, running 64-bit Ubuntu 16.04 LTS with 4.4.0-112-generic Linux kernel. The blade servers are connected through 10Gbps Ethernet. We use Oracle Java 1.8, Cassandra 2.1.9, JanusGraph 0.2.1, Greenplum 5.9.0, Spark 2.2.0 in our experiments.

Workload Generation. We generate a synthetic data set of customer shopping events. The goal to support 10 billion object vertices and 10 million location vertices is designed for clusters with 1000 machines. Given the machine cluster size in our experiments, we scale down the number of vertices by a factor of 100. Therefore, we would like to generate 100 million object vertices and 100 thousand location vertices.

First, we crawled 450 thousand hotel locations in China from ctrip.com (a popular travel booking web site in China). The locations are distributed across about 1100 areas (cities / counties / districts). We randomly choose 10 thousand locations from the real-world hotel locations as shopping locations. Second, we generate 100 million customers. As the number of shopping locations and the population of an area are often correlated, we assign customers to areas so that the number of customers is proportional to the number of locations in an

area. Third, we would like to generate a data set that covers a 2-3 year period and can be stored in the cluster in the experiments. We assume 40% people are frequent shoppers and 60% people are infrequent shoppers. A frequent shopper and an infrequent shopper visit a randomly chosen shopping location in his or her area every week with probability 0.8 and 0.2, respectively. (Note that the week period is necessary to reduce the total data volume to fit into the cluster storage capacity. Our ingestion experiments will send the edge data as fast as possible to saturate the system.) Finally, we produce a small number of cloned objects that visit shopping locations in far-away areas. The resulting graph contains 57 billion edges, which cover 800 days.

B. Edge Ingestion Throughput and Scalability

The edge ingestion throughput is the number of new edges streamed into the GraphStores per second. We send edges in the data set as fast as possible, and measure the sustained ingestion throughput. We vary the number of worker machines from 1 to 10 in PAST to study the scalability of our solution.

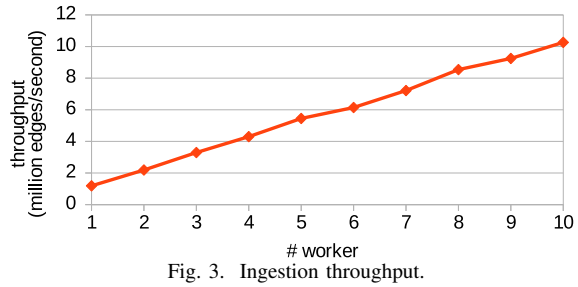

Fig. 3. Ingestion throughput.

In Fig.3, the Y-axis is the throughput in million edges per second) and the X-axis is the number of workers. The higher the better. From the figure, we see that the sustained ingestion throughput increases linearly as the worker number grows. The PAST design achieves good scalability for new edge ingestion. Every worker in PAST can support an additional 1 million new edges per second. As an edge takes 24 bytes, every worker in PAST can support additional 24MB/s ingestion bandwidth. Therefore, the design goal of 1 trillion new edges per day (or 290MB/s) for a full-scale spatio-temporal graph can be achieved with about 12 machines. This gives a lower bound of the actual number of nodes in a design, whose choice must also consider the performance of query processing.

C. Query Performance

In the next set of experiments, we compare the query performance for four solutions: (1) JanusGraph, where spatio-temporal graph data is stored as a property graph; (2) Greenplum, where graph data is stored as relational tables, and multi-dimensional partitioning is used (vertex ID is the first dimension, and time is the second dimension); (3) Cassandra+Spark, where data is stored in Cassandra and loaded into Spark for query processing; and (4) our solution PAST.

We run Q1–Q4. We set the time range to be 32 days. For Q2 and Q3, we set the thresholds on time and location distance

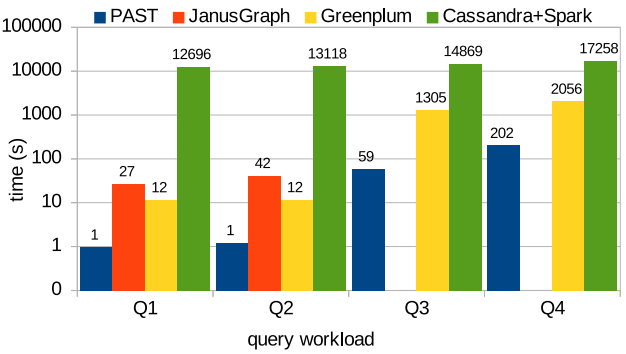

Fig. 4. Comparison with state-of-the-art systems.

to be 7 hours and 100 meters, respectively. For Q4, we set the velocity threshold to be 120km/h.

Fig.4 compares the query performance for the four solutions. The Y-axis is the execution time in the logarithmic scale. We do not run Q3 and Q4 on JanusGraph as it mainly focuses on simple traversal queries and employs Spark for complex queries, which can be represented by Cassandra+Spark. From the figure, we see that PAST achieves 1–4 orders of magnitude better performance compared with the three existing solutions. The partition and query processing schemes in PAST can effectively reduce the amount of data accessed from the underlying storage and the data communication cost.

V. Conclusion

In conclusion, we define a bipartite graph model for spatio-temporal graphs based on the commonalities of representative real-world applications, i.e., customer behavior tracking and mining, clone-plate car detection, and shipment tracking. We propose and evaluate PAST, a framework for efficient PArtitioning and query processing of Spatio-Temporal graphs. Our preliminary experimental evaluation shows that PAST can meet the requirements of the above applications. For typical queries on spatio-temporal graphs, PAST can outperform state-of-art platforms (e.g., JanusGraph, Greenplum, Spark) by 1–4 orders of magnitude.

Acknowledgment

This work is partially supported by NSFC project No. 61572468, by Huawei collaboration project No. HO2017050001B5, and by K.C.Wong Education Foundation.

References

[1] Metis, http://glaros.dtc.umn.edu/gkhome/metis/metis/overview.
[2] R. L. B. Hendrickson, "The chaco user's guide, version 2.0," Sandia National Labs, Albuquerque, NM, Tech. Rep. SAND95-2344, 1995.
[3] Scotch, http://www.labri.u-bordeaux.fr/perso/pelegrin/scotch.
[4] B. Hendrickson and T. G. Kolda, "Graph partitioning models for parallel computing," *Parallel Computing*, vol. 26, pp. 1519–1534, Jun. 2000.
[5] S. Verma, L. M. Leslie, Y. Shin, and I. Gupta, "An experimental comparison of partitioning strategies in distributed graph processing," *PVLDB*, vol. 10, pp. 493–504, Aug. 2017.
[6] JanusGraph, http://janusgraph.org/.
[7] Greenplum, https://greenplum.org/.
[8] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *NSDI 2012*, San Jose, CA, USA, Apr. 2012, pp. 15–28.