Evolution Analysis of Large Graphs with Gradoop

Christopher Rost[1][0000−0003−4217−9312], Andreas Thor[2][0000−0003−2575−2893], Philip Fritzsche[1], Kevin Gomez[1], and Erhard Rahm[1][0000−0002−2665−1114]

[1] University of Leipzig {rost,fritzsche,gomez,rahm}@informatik.uni-leipzig.de
[2] Leipzig University of Telecommunications thor@hft-leipzig.de

Abstract. The temporal analysis of evolving graphs is an important requirement in many domains. We therefore began with extending the distributed graph analysis framework Gradoop for temporal graph analysis. This short paper contains a brief overview of our work in progress and an example use case from the financial domain demonstrating the flexibility of the temporal graph model and its operators.

Keywords: Temporal Graph Analysis · Distributed Analytical Workflow · Temporal Property Graph Model

1 A brief overview of Gradoop's temporal extension

In this short paper we report on work in progress on our temporal property graph model (TPGM) [5] which is an extension to Gradoop [2,3]. Gradoop is an open source framework for distributed graph analytics based on Apache Flink. It combines and extends features of graph analytical systems with the benefits of distributed graph processing. Gradoop is an implementation of the Extended Property Graph Model (EPGM) and supports a number of generic operators on graphs (for pattern matching, grouping, etc.) that can be used within workflows for graph analysis. The workflows can be specified in a declarative domain-specific language called GrALa. Since the EPGM is built on top of Apache Flink, each Gradoop operator is based on a subset of Flink's transformations (map, flatmap, join, etc.) to achieve a parallel execution and scalability to large graphs.

Extension of data structure: Many applications require time dependent graph models. We therefore extend Gradoop's EPGM by adding additional time attributes *from* and *to* to the schema of vertices, edges and logical graphs. This approach offers a flexible representation of temporal graphs with bitemporal time semantics where the valid time can be empty, a time-stamp or a time interval.[3] An important advantage of our extension is its backward compatibility to the original EPGM since every existing Gradoop operator (that builds upon

[3] For the sake of simplicity, we limit ourselves in this paper to valid times. However, our extension also supports transaction times in a similar way.

the EPGM) can be applied to a temporal graph by disregarding the temporal information of the graph elements.

Extension of existing Gradoop operators: Operators such as transform, subgraph, grouping and pattern matching may benefit from the temporal extension of EPGM. For example, the subgraph operator can identify all vertices and edges where the validity range exceeds a limit. Similarly, the pattern matching operator can extract all subgraphs where the pattern is valid at a given point in time.

Introduction of new temporal operators: We introduce *snapshot* and *difference* as specific temporal operators. The *snapshot* operator allows to retrieve a valid state of the entire temporal graph either at a specific point in time or a subgraph that is valid during a given time range by providing a temporal predicate function. Such predicate functions are adopted from the SQL standard for temporal databases [4]. The *difference* operator computes the difference of two snapshots X and Y by determining the union of X and Y and annotating each vertex and edge if it appears in Y only (i.e., if it has been added), in X only (deleted) or in both X and Y (persistent). Following the philosophy of Gradoop, both operators were implemented on top of Apache Flink: *snapshot* employs Flink's *filter* while *difference* is based on the *flatMap* transformation.

Support of time-specific grouping and aggregation: The time dimension automatically introduces a hierarchy, i.e., graphs can be grouped (summarized) at multiple levels of time-granularity. For example, graph summaries generated by Gradoop's groupBy operator can be additionally "rolled-up" on the time hierarchy to have a aggregations on multiple levels of time granularities.

2 Temporal graph analysis using Gradoop: A use case

Supporting graph analysis at large scale is necessary in various domains like IoT, finance and web to perform risk analysis, customer profiling, etc. In addition, the time plays an important role in such analysis since analysts want to know, e.g., how a specific result of their query changes over time. As a result, a graph processing system has to offer a flexible and rich library of functionalities and algorithms to support a wide range of analysis.

To show the expressiveness and flexibility of Gradoop and its temporal model among with its declarative operator principle, we choose a business case from the customer relationship management domain. More precisely, the interactions in a call center of the banks association of Turkey [1]. The call center is responsible for 25 banks of the association. More than 7,500 agents are employed in about 16 service types (e.g., card, stock, ATM, online banking etc.). Per month, about 46 million incoming calls were answered by agents, 24 million calls are outgoing calls to customers. These entities and their relations form a huge heterogeneous network that continuously changes over time. We can put all the collected data in our temporal property graph model to enable various time-related analysis. Figure 1 shows a simplified example of the resulting graph schema. It includes different types of vertices (entities), like *Bank* and *Customer*,

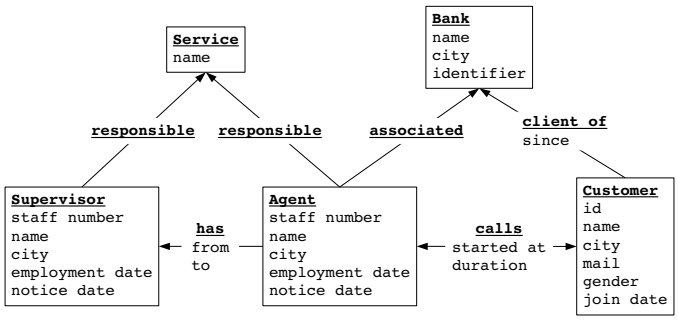

Fig. 1. Simplified example of a call-center network from the financial domain.

as well as edges (relations), like a *call* representing the telephone call between customers and call center agents. Each element includes a variety of properties describing it with additional information, e.g., an *Agent* vertex has a defined *staff number*, a *name* and *city*.

In the following we chose an analytical question that refers to this use case. We will utilize the modularity of our temporal graph operators as well as operators from the reference EPGM implementation and compose them within a simple but powerful workflow to show a way to answer them.

What is the average duration of calls per month, week and day between agents of different cities and customers of Istanbul, where both agents and customers joined the bank in 2018?

This question includes the need of aggregations over time hierarchies besides filters for a subset of entities. The following exemplary workflow definition shows the use of four operators that result in a collection of graphs where each describes one out of the three time granularities month, week and day.

```
1  groupedGraphs = graph
2    .subgraph(
3       v -> { v._label == 'Agent' OR
4              (v._label = 'Customer' AND v.city = 'Istanbul' }),
5       e -> {e._label = 'calls'})
6    .snapshot(CreatedIn(2018))
7    .verify()
8    .groupBy(
9       [Label(), Property('city')],                        // V group keys
10      [Count()],                                          // V aggregates
11      [Month(from)), Week(from), Day(from)] BY ROLLUP,    // E group keys
12      [AvgDuration(), Count()]);                          // E aggregates
```

The first operator named *subgraph* (line 2-5) uses the given vertex and edge predicates to apply a filtering to get a subgraph that contains only *Agent* vertices and *Customer* vertices with a property *city* that is equal to the string *Istanbul*. This operator is part of the EPGM. To receive customers that joined a bank

in 2018, we apply the newly developed TPGM *snapshot* operator (line 6) with a pre-defined predicate. Since the result of the *snapshot* operator can contain dangling edges (i.e., its source or target vertex are not contained in the result set), we apply a *verify* operator (line 7) to remove these from the graph. The final *groupBy* operator (line 8-12) summarizes the graph. It aggregates properties according to the aggregate functions for the specified vertex and edge grouping keys. The vertices will be grouped by their label and the property *city* (line 9). A property with the count is added to each grouped vertex as a result of the given *Count()* vertex aggregate function. The edges representing the calls are grouped by month, week and day through the usage of time-specific value transformation functions of the same name (line 11). The additional *BY ROLLUP* leads to three different aggregations similar to SQL. First, the graph will be grouped on day, then on week and in addition, on the month of the call's beginning. The resulting three logical graphs are contained in a graph collection, which is the result of our workflow. The collection can be stored or visualized by one of Gradoops data sinks.

3 Conclusion

We reported on our work in progress on temporal graph analysis within the distributed graph analytic system Gradoop. We introduced our flexible temporal property graph model TPGM as an extension of Gradoop's powerful EPGM supporting logical abstractions of graphs and collections of them. The extension of its data structure, existing operators and creation of new temporal operators enables answering time-respecting analytical questions on evolving graphs by flexible chaining of the operators. We expect our extensions to be available in Gradoop by the end of this year. We demonstrated how analysts can declaratively express workflows to analyze large evolving graphs by an use case scenario of the financial domain. In future work we will further extend Gradoop by temporal features such as operators and algorithms to make Gradoop a powerful and flexible system for temporal graph analysis.

References

1. The banks association of turkey: Stat. report. `www.tbb.org.tr/en/banks-and-banking-sector-information/statistical-reports/20`, (accessed: 2019-04-30)
2. Junghanns, M., Kießling, M., Teichmann, N., Gómez, K., Petermann, A., Rahm, E.: Declarative and distributed graph analytics with gradoop. PVLDB **11**(12) (2018)
3. Junghanns, M., Petermann, A., Neumann, M., Rahm, E.: Management and analysis of big graph data: current systems and open challenges. In: Handbook of Big Data Technologies, pp. 457–505. Springer (2017)
4. Kulkarni, K., Michels, J.: Temporal features in sql: 2011. ACM Sigmod Record **41**(3), 34–43 (2012)
5. Rost, C., Thor, A., Rahm, E.: Temporal graph analysis using gradoop. BTW 2019–Workshopband (2019)