Graph Stream Zoomer
Distributed grouping of property graph streams

FOSDEM 2023 – Graph Devroom

Christopher Rost
rost@informatik.uni-leipzig.de
4rd February 2023

https://github.com/dbs-leipzig/graph-stream-zoomer
About us

↑ Prof. Dr. Erhard Rahm
Head of database department
University of Leipzig

← Christopher Rost
PhD student since 2018

Max Zimmer →
Master student

Elias Saalmann
Alumnus - University of Leipzig

Rana Noureddin
Alumna - University of Leipzig
What you should take away from this talk

- What is a property graph stream?
- Why should I group a graph stream?
- What is the “graph stream zoomer” and which grouping configuration leads to which results?
- What are the implementation challenges?
- How can I use the graph stream zoomer?
Basics and motivation

- What is an (event-) stream?
- What is a graph stream?
- Why a graph stream?
- Why grouping of graph streams?

**event**
anything that happens at a clearly defined time and that can be specifically recorded

**event stream**
sequence of events ordered by time

**event processing**
identify meaningful events and respond to them as quickly as possible

bike rental events

stock prices

online purchases
Basics and motivation

- What is an (event-) stream?
- **What is a graph stream?**
- Why a graph stream?
- Why grouping of graph streams?

**graph stream**

event stream where an event is a graph element or update

**graph element**

vertex, edge, triple possibly labeled and attributed

**graph update**

modification of the graph structure and content, e.g., edge insertion/deletion
Basics and motivation

- What is an (event-) stream?
- What is a graph stream?
- Why a graph stream?
- Why grouping of graph streams?

execution of graph analysis algorithms (e.g., PageRank) **concurrently** with graph updates

updates of analysis results with a low latency in (near) real time

goal monitoring and/or notification/reactivity
Basics and motivation

- What is an (event-) stream?
- What is a graph stream?
- Why a graph stream?
- Why grouping a graph stream?

- graph streams may be heterogeneous and high frequent
- get overview and reduce complexity on different levels
- summarize graph elements/updates on similar characteristics
  - **time**, **structure**, **content** (label, properties)
  - via grouping key functions: $f(v/e) \rightarrow$ key
- get the grouping result “real-time” after graph update >>> again a graph stream
Applications for graph stream grouping

- **Pre-processing** for graph stream systems (ETL)
  - e.g., before PageRank, group graph stream on city attribute of users
- **Post-processing** after a applied graph stream analysis
  - e.g., after community algorithm, group elements with the same cluster id
- **Understanding** the graph stream (and its evolution)
  - Which vertex/edge types exist in the stream?
  - How frequent the different types arrive?
  - How vertices of different characteristics are connected with edges of certain characteristics?
- **Reveal hidden information** and get instantly notified
  - aggregation of attributes -> deeper insights
  - e.g., how an average value changes over time
  - notification by defining thresholds
Graph Stream by example
Running example - Graph Schemas

**A**

Station
- id: int
- name: string
- bikes: int
- lat: float
- long: float

Trip
- id: int
- user_id: int
- user_type: string
- bike_id: int
- from: datetime
- to: datetime
- duration: int

**B**

Station
- id: int
- name: string
- bikes: int
- lat: float
- long: float

Bike
- id: int

User
- id: int
- name: string
- gender: int
- type: string

Trip
- id: int
- from: datetime
- to: datetime
- duration: int

byBike

byUser

started

ended
Running example - Graph Stream

A

B

time
Example 1 – Zoomed out (Schema A)

Input stream

Grouping config
- W: 10m
- VGK: (v → 1)
- EGK: (e → 1)
- VAgg: count()
- EAgg: count()

Empty label count:
- w1: 42
- w2: 188
- Total: 38

Total:
- 38 + 42 + 242 + 188 = 410
Example 1 – Zoomed out (Schema B)

Input stream

Grouping config
- W: 10m
- VGK: (v → 1)
- EGK: (e → 1)
- VAgg: count()
- EAgg: count()

emptylabel count : 124
emptylabel count : 221
emptylabel count : 145
emptylabel count : 312

w1
w2
Example 2 – Graph Stream Schema (A)

**Grouping config**
- **W**: 10m
- **VGK**: $(v \rightarrow \text{label}(v))$
- **EGK**: $(e \rightarrow \text{label}(e))$
- **VAgg**: count()
- **EAgg**: count()

**Input stream**

Station count: 38
Station count: 42

Trip count: 242
Trip count: 188

w1

w2
Example 2 – Graph Stream Schema (B)

**Grouping config**

- **W**: 10m
- **VGK**: $(v \rightarrow \text{label}(v))$
- **EGK**: $(e \rightarrow \text{label}(e))$
- **VAgg**: count()
- **EAgg**: count()

**Input stream**

- Station count: 64
- Bike count: 22
- Trip count: 26
- User count: 22
- User started count: 26
- User ended count: 26
- Bike byBike count: 22
- Bike byUser count: 22
- Trip started count: 26
Example 3 – Schema with aggregates (A)

Input stream

Grouping config

W: 10m
VGK: (v → label(v))
EGK: (e → label(e))
VAgg: avg(v.bikes)
EAgg: min(e.duration)
       max(e.duration)
       avg(e.duration)

Station
avg_bikes : 52.5

Station
avg_bikes : 22

Station avg_bikes : 52.5

Station avg_bikes : 22

w1

w2

trip
min_duration : 62
max_duration : 21,000
avg_duration : 3,520.5

trip
min_duration : 120
max_duration : 12,420
avg_duration : 3,024.5
Example 4 – Advanced Grouping (A)

**Input stream**

**Grouping config**

<table>
<thead>
<tr>
<th>W:</th>
<th>10m</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGK:</td>
<td>(v \rightarrow \text{label}(v))</td>
</tr>
<tr>
<td></td>
<td>(v \rightarrow \text{getDistrict}(v.\text{lat},v.\text{long}))</td>
</tr>
<tr>
<td>EGK:</td>
<td>(e \rightarrow \text{label}(e))</td>
</tr>
<tr>
<td></td>
<td>(e \rightarrow e.\text{user_type})</td>
</tr>
<tr>
<td>VAgg:</td>
<td>(\text{avg}(v.\text{lat}),\text{avg}(v.\text{long}))</td>
</tr>
<tr>
<td>EAgg:</td>
<td>(\text{count()})</td>
</tr>
<tr>
<td></td>
<td>(\text{min}(e.\text{duration}))</td>
</tr>
<tr>
<td></td>
<td>(\text{max}(e.\text{duration}))</td>
</tr>
<tr>
<td></td>
<td>(\text{avg}(e.\text{duration}))</td>
</tr>
</tbody>
</table>

**Station**

- **Station**
  - **district**: 2
  - **avg_lat**: 50.82
  - **avg_long**: 4.4

- **Station**
  - **district**: 1
  - **avg_lat**: 50.813
  - **avg_long**: 4.382

- **Station**
  - **district**: 3
  - **avg_lat**: 50.83
  - **avg_long**: 4.30

**trip**

- **user_type**: subscriber
- **count**: 25
- **min_duration**: 62
- **max_duration**: 6,421
- **avg_duration**: 2,520

- **trip**
  - **user_type**: customer
  - **count**: 32
  - **min_duration**: 122
  - **max_duration**: 5,420
  - **avg_duration**: 1,520
**Example 5 – Zoomed In (B)**

**Grouping config**
- **W:** 60m
- **VGK:** \((v \rightarrow \text{label}(v))\)
  - \((v \rightarrow v.\text{id})\)
- **EGK:** \((e \rightarrow \text{label}(e))\)
  - \((e \rightarrow e.\text{id})\)
- **VAgg:** -
- **EAgg:** -
Implementation challenges

ONE DOES NOT SIMPLY

GROUP A GRAPH STREAM
Implementation challenges

- Find an optimal **graph representation** in the streaming model
  - triple stream, vertex- and edge streams, adjacency lists/arrays
- Ensure **chronological order** after each step in the processing pipeline
  - use watermarks to prevent out-of-order events
- Ensure **scalability** of the pipeline parts (low communication overhead)
- Ensure a **finite** and minimized internal **state** of each processing step
  - e.g., join needs temporal predicate to clean up state
- Low latency / high throughput / high scalability (scale in/out)
Grouping algorithm overview

Operator encapsulation

Input mapping
- Stream input
- Extract triples
- Time + watermark assign
- Split V/E

Vertex deduplication
- $V \rightarrow V'$
- $\{V_{id}, V_{sid}, V_{u}\}$

V grouping
- $w \rightarrow G_1 \rightarrow V'$
- $w \rightarrow G_2 \rightarrow V''$
- $V'' \rightarrow V^s$

Output mapping
- $V^s \rightarrow V^s$

GS
- Edge enrichment
- $E \rightarrow E'$

GS'
- E grouping
- $E \rightarrow E''$
- $E'' \rightarrow E^s$

E grouping
- $w \rightarrow G_3 \rightarrow E''$
- $e' \rightarrow E^s$

Input mapping
- Stream input
- Extract triples
- Time + watermark assign
- Split V/E

Flink

Department of Computer Science | Database Group
Exemplified operator call

```java
// Init the stream environment
final StreamExecEnvironment env = StreamExecEnvironment.createLocalEnv();
// Create the triple stream from a csv file
DataStream<StreamTriple> citiBikeStream = createInputFromCsv(env);
// Init the StreamGraph - our internal representation of a graph stream
StreamGraph sg = StreamGraph.fromFlinkStream(citiBikeStream, new Config(env));
// Configure and build the grouping operator
GraphStreamGrouping groupingOperator = new TableGroupingBase.GroupingBuilder()
    .setWindowSize(15, WindowConfig.TimeUnit.DAYS)
    .addVertexGroupingKey(":label")
    .addEdgeGroupingKey(":label")
    .addVertexAggregateFunction(new Count())
    .addEdgeAggregateFunction(new AvgProperty("tripduration")).build();

// Execute the grouping and overwrite the input stream with the grouping result
streamGraph = groupingOperator.execute(streamGraph);
// Print the result stream to console
streamGraph.printVertices();
// Trigger the workflow execution
env.execute();
```
Current state and future work

• prototypical implementation using Apache Flink’s Table API at 90%
• bug at the Flink planner not fixed yet -> workaround via SQL API
  • https://issues.apache.org/jira/browse/FLINK-22530
• evaluation planned
  • latency, throughput, scalability, different grouping setups
  • on real-world and synthetic graph streams
• user-defined key and aggregate functions
That's all folks!

Graph Stream Zoomer >> [https://github.com/dbs-leipzig/graph-stream-zoomer](https://github.com/dbs-leipzig/graph-stream-zoomer)
Gradoop >> [https://github.com/dbs-leipzig/gradoop](https://github.com/dbs-leipzig/gradoop)
Temporal Graph Explorer >> [https://github.com/dbs-leipzig/temporal_graph_explorer](https://github.com/dbs-leipzig/temporal_graph_explorer)