Backlogs & Interval Timestamps:
Building blocks for supporting temporal queries in graph DBs

Gabriel Campero Durand, Marcus Pinnecke, David Broneske, Gunter Saake
Temperance: The virtue of keeping track of time

1439-1442
Porta della Carta (Between St. Mark’s Church and the Ducal Palace)
Maestro Bortolo Tajapietra
Temporal models in data mgmt

A plethora of temporal models exists...

**Point-Stamped Temporal Models**

David Toman  
University of Waterloo, Waterloo, ON, Canada

**Synonyms**  
Point-based temporal models

**Period-Stamped Temporal Models**

Nikos A. Lorentzos  
Agricultural University of Athens, Athens, Greece

**Synonyms**  
Interval-based temporal models

They help to express clearly:
- the temporal **validity** of items
- the sequence of **changes** on the items
Temporal queries in graph DBs: A technology in evolution

- Work w/focus on dynamic props [HUA16]
- Frame node [CAT13][SEM16T]
- Indexing + algorithmic tuning [HUO14][SEM16D]
- Time in graph proc. frameworks [MOF16][CHRONOS14][IMMORTALGRAPH15]
- Graph deltas [DES13][KOL13]

Attribute timestamp | Point-stamped | Period-stamped | Backlogs

Today: Some very modest, work-in-progress insights towards a common-ground approach
1. Building blocks for temporal views

2. Query rewriting to use views (Snapshot -> Differential processing)

3. Towards adaptive temporal views
Design choices related to time support

- Separating snapshots, aggregation to a single graph, or hybrids.

- Proposal #1: (atelic) validity can be fully captured with interval timestamps.
  - More expressive than point timestamps, but special care required for telic case.

- After adopting interval timestamps as a model, other choices:
  - One interval of validity per item*, or several.
  - Tracking attribute or topology changes.

In bold, the choices for our study.

*Arguably easier to index.
Proposal #2: backlogs can be understood as an emergent “view”, which comes for free by not having all snapshots separate.

- **Differential processing.**
- **Supported w/meta-vertices or global indexes over times.**

Backlogs with meta-items (Using the graph as an index, GRAIN)

Backlogs with global indexes
Note:

- The concept of using the graph itself as an index, to create certain “views”, builds on the possibilities for combining models with meta-models in graph databases.
- We believe that this is a powerful aspect of these databases & that it can be crucial to adaptive features (e.g. cracking views to accelerate graph traversals).
- This paper provides an early example of opportunities from this technique.
2. Query rewriting to use views
(Snapshot -> Differential processing)

1. Building blocks for temporal views
Density Calculation

Density: \[
\frac{|E|}{|V|(|V| - 1)}
\]

Basic query:

```scala
1  titanGraph.V().
2    hasLabel("device").
3    has("deletedAt", P.gt(date)).
4    has("createdAt", P.lte(date)).
5    count().next();
```

Incremental- Using global indexes:

```scala
1  titanGraph.V().
2    hasLabel("device").
3    has("createdAt", date).
4    count().next();
5  titanGraph.V().
6    hasLabel("device").
7    has("deletedAt", date).
8    count().next();
```

Incremental- GRAIN:

```scala
1  Map<Object,Long> creationCount =
2    (Map<Object, Long>) titanGraph.V(logId).
3    outE("created").
4    has("createdAt", date).
5    groupCount("created").by("createdType").
6    cap("created").next();
7    numCreatedVertices = creationCount.get(Vertex.class);
8    numCreatedEdges = creationCount.get(Edge.class);
```
SNAP as-733 dataset.

From 2-100x speedups by incremental processing.
The global was more heap intensive than the local (GRAIN), which was compute-bound.

The gains from incremental processing generalize to other methods.
3. Towards adaptive temporal views

2. Query rewriting to use views (Snapshot -> Differential processing)

1. Building blocks for temporal views
Towards adaptive temporal views

After establishing the use of interval timestamps, one-per-item.

The knobs we (as data managers) see:
- Aggregated vs. Snapshot-by-snapshot (SBS)
- Incremental vs. SBS processing
- Different approaches for backlogs (room for improvement here).

What we need to figure out about using these knobs:
- Trade-off between the detrimental effect of structural locality loss vs. the opportunities from incremental processing in the aggregated version?
  - How to gauge this on a live system?
  - Effect of the different backlog approaches?
- On live systems, will SBS be better for new data, whereas aggregated copies could be for past (read-only) data?
- Agreed-upon benchmarks and relevant use cases.

Our goal: Using these knobs adaptively:
- What are the best triggers and methods to go from SBS to aggregated?
- How to achieve automatic query re-writing from SBS to the best incremental processing?
- Extensions for attributes?

This research is part of our work on an evolving H³TAP database: Mondrian
Questions? Ideas?
campero@ovgu.de

Thanks for the attention :)
References