Gradoop - Scalable Graph Analytics with Apache Flink @ GRIDKA 2019

28th August 2019

Kevin Gómez
University of Leipzig
Motivation
“Graphs are everywhere“

\[ \text{Graph} = (\text{Vertices}, \text{Edges}) \]
„Graphs are everywhere“

\[ \text{Graph} = (\text{Users}, \text{Followers}) \]
„Graphs are everywhere“

Graph = (Users, Friendships)
„Graphs are heterogeneous“

Graph = (Users ∪ Bands, Friendships ∪ Likes)
Graphs can be analyzed

\[ \text{Graph} = (\text{Users} \cup \text{Bands}, \text{Friendships} \cup \text{Likes}) \]
Graphs can be analyzed

Graph = (Users ∪ Bands, Friendships ∪ Likes)
„Graphs can be analyzed“

Assuming a social network
"Graphs can be analyzed"

Assuming a social network
1. Determine subgraph
„Graphs can be analyzed“

Assuming a social network
1. Determine subgraph
Graphs can be analyzed

Assuming a social network
1. Determine subgraph
2. Find communities
Graphs can be analyzed

Assuming a social network
1. Determine subgraph
2. Find communities
“Graphs can be analyzed”

Assuming a social network
1. Determine subgraph
2. Find communities
3. Filter communities
„Graphs can be analyzed“

Assuming a social network
1. Determine subgraph
2. Find communities
3. Filter communities
Graphs can be analyzed

Assuming a social network
1. Determine subgraph
2. Find communities
3. Filter communities
4. Find common subgraph
„Graphs can be analyzed“

Assuming a social network
1. Determine subgraph
2. Find communities
3. Filter communities
4. Find common subgraph
Graphs can be analyzed

Assuming a social network

1. Determine subgraph
   - Heterogeneous data
   - Apply graph transformation
2. Find communities
   - Handle collections of graphs
3. Filter communities
   - Aggregation, Selection
4. Find common subgraph
   - Apply dedicated algorithm
„Graphs can be analyzed“

Assuming a social network
- Heterogeneous data

1. **Determine subgraph**
   - Apply graph transformation

2. Find communities
   - Handle collections of graphs

3. Filter communities
   - Aggregation, Selection

4. Find common subgraph
   - Apply dedicated algorithm
„Graphs can be analyzed“

Assuming a social network
- Heterogeneous data
1. Determine subgraph
  - Apply graph transformation
2. Find communities
  - Handle collections of graphs
3. Filter communities
  - Aggregation, Selection
4. Find common subgraph
  - Apply dedicated algorithm
„Graphs can be analyzed“

Assuming a social network

1. Determine subgraph
   • Heterogeneous data
   • Apply graph transformation

2. Find communities
   • Handle collections of graphs

3. **Filter communities**
   • **Aggregation, Selection**

4. Find common subgraph
   • Apply dedicated algorithm
„Graphs can be analyzed“

Assuming a social network

1. Determine subgraph
   - Heterogeneous data
   - Apply graph transformation

2. Find communities
   - Handle collections of graphs

3. Filter communities
   - Aggregation, Selection

4. Find common subgraph
   - Apply dedicated algorithm
„And let‘s not forget ...“
“...Graphs are large”
„A framework and research platform for efficient, distributed and domain independent management and analytics of heterogeneous graph data.“
High Level Architecture

Graph Analytical Language (GrALa)

Extended Property Graph Model (EPGM)

Apache Flink Operator Implementation

Apache Flink Distributed Operator Execution

Apache HBase Distributed Graph Store

HDFS/YARN Cluster

Java 8

ALv2
Apache Flink Third-party library

Batch
- Hadoop MR
- Table
- Gelly
- ML
- Dataflow
- MRQL
- Cascading
- Zeppelin
- GRADOOP

DataStream
- Table
- SAMOA
- Dataflow
- Storm

Dataset

DataStream

Streaming Dataflow Runtime

Local
- Cluster (e.g. YARN)
- Cloud (e.g. EC2)

Data Storage (e.g. Files, HDFS, S3, JDBC, Kafka, ...)

Cloud (e.g. EC2)
Extended Property Graph Model (EPGM)
Extended Property Graph Model

- Vertices and directed Edges
- Logical Graphs
- Identifiers
- Type Labels
- Properties

Person name: Alice
born: 1984

Band name: Metallica
founded: 1981

Person name: Bob

Person name: Eve

Band name: AC/DC
founded: 1973

Community interest: Heavy Metal

Community interest: Hard Rock

Person name: Eve

Community interest: Hard Rock

Community interest: Heavy Metal

Person name: Alice
born: 1984

Person name: Bob

Band name: Metallica
founded: 1981

Person name: Eve

likes since: 2013

likes since: 2014

likes since: 2015

knows

likes since: 2014
Why multiple graph support?

- It allows reasoning over multiple versions of the same graph (e.g. comparing daily snapshots)
- It provides an effective grouping mechanism for naturally-partitioned data (e.g. per continent graph)
- It is useful for combining data from disparate data sources in one system (e.g. data integration)
- It fits the paradigm of prominent analytical big-data processing systems (e.g. Apache Flink and Apache Spark)
- It mirrors mathematical graph theory where working with multiple graphs is common

EPGM Operators
Basic Binary Operators

Combination

Overlap

Exclusion
Subgraph Extraction
Graph Grouping

3

4

1 2 3 4 5

+Aggregate

max(a):42
max(a):84
count:2
max(a):13
max(a):21
max(a):21

4

6 7

3

1 2 3 4 5

a:23 a:84 a:13 a:21 a:42 a:12

Keys

count:2 count:2 max(a):21 max(a):84
Cypher Pattern Matching

Which people like the same band that was founded after 1980?

Would they possibly become buddies?
Which people like the same band that was founded after 1980?

Would they possibly become buddies?
Cypher Pattern Matching

MATCH (p1:Person)-[:likes]->(b:Band)  
  (p2:Person)-[:likes]->(b)  
WHERE p1 != p2 AND b.founded > 1980  
CONSTRUCT (p1)-[:possibleBuddy]->(p2)

Which people like the same band that was founded after 1980?
Would they possibly become buddies?
MATCH (p1:Person)-[:likes]->(b:Band) (p2:Person)-[:likes]->(b) WHERE p1 != p2 AND b.founded > 1980
CONSTRUCT (p1)-[new:possibleBuddy]->(p2)
EPGM Operators Overview

Operators

Unary
- Aggregation
- Pattern Matching
- Transformation
- Grouping
- Subgraph
- Call

Binary
- Combination
- Overlap
- Exclusion
- Equality

Algorithms
- Flink Gelly Library
- BTG Extraction
- Adaptive Partitioning
- Frequent Subgraphs
„Graphs can be analyzed“

Assuming a social network

1. Determine subgraph
2. Find communities
3. Filter communities
4. Find common subgraph
„Graphs can be analyzed“

```java
return socialNetwork

// 1) extract subgraph
  .subgraph{{vertex} - { return vertex.getLabel().toLowerCase().equals(person); }}, {edge} - { return edge.getLabel().toLowerCase().equals(knows); }

// project to necessary information
  .transform{{current, transformed} - { return current; }, {current, transformed} - {{
    transformed.setLabel(current.getLabel());
    transformed.setProperty(city, current.getPropertyValue(city));
    transformed.setProperty(gender, current.getPropertyValue(gender));
    transformed.setProperty(label, current.getPropertyValue(birthday));
    return transformed;
  }}, {current, transformed} - {
    transformed.setLabel(current.getLabel());
    return transformed;
  }

// 3a) compute communities
  .callForGraph(new GellyLabelPropagation<GraphHeadPojo, VertexPojo, EdgePojo>(maxIterations, label))

// 3b) separate communities
  .splitBy(label)

// 4) compute vertex count per community
  .apply(new ApplyAggregation<>(vertexCount, new VertexCount<GraphHeadPojo, VertexPojo, EdgePojo>()))

// 5) reduce filtered graphs to a single graph using combination
  .reduce(new ReduceCombination<GraphHeadPojo, VertexPojo, EdgePojo>())

// 7) group that graph by vertex properties
  .groupBy(Lists.newArrayList(city, gender))

// 8a) count vertices of grouped graph
  .aggregate(vertexCount, new VertexCount<GraphHeadPojo, VertexPojo, EdgePojo>());

// 8b) count edges of grouped graph
  .aggregate(edgeCount, new EdgeCount<GraphHeadPojo, VertexPojo, EdgePojo>());
```
EPGM on Apache Flink
Flink DataSet API

- **DataSet** := Distributed Collection of Data Objects
- **Transformation** := Operation on DataSets
- **Flink Program** := Composition of Transformations
Graph Representation

**DataSet<EPGMVertex>**

<table>
<thead>
<tr>
<th>Id</th>
<th>Label</th>
<th>Properties</th>
<th>Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Person</td>
<td>{name:Alice, born:1984}</td>
<td>{1}</td>
</tr>
<tr>
<td>2</td>
<td>Band</td>
<td>{name:Metallica, founded:1981}</td>
<td>{1}</td>
</tr>
<tr>
<td>3</td>
<td>Person</td>
<td>{name:Bob}</td>
<td>{1,2}</td>
</tr>
<tr>
<td>4</td>
<td>Band</td>
<td>{name:AC/DC, founded:1973}</td>
<td>{2}</td>
</tr>
<tr>
<td>5</td>
<td>Person</td>
<td>{name:Eve}</td>
<td>{2}</td>
</tr>
</tbody>
</table>

**DataSet<EPGMGraphHead>**

<table>
<thead>
<tr>
<th>Id</th>
<th>Label</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Community</td>
<td>{interest:Heavy Metal}</td>
</tr>
<tr>
<td>2</td>
<td>Community</td>
<td>{interest:Hard Rock}</td>
</tr>
</tbody>
</table>

**DataSet<EPGMEdge>**

<table>
<thead>
<tr>
<th>Id</th>
<th>Label</th>
<th>Source</th>
<th>Target</th>
<th>Properties</th>
<th>Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>likes</td>
<td>1</td>
<td>2</td>
<td>{since:2014}</td>
<td>{1}</td>
</tr>
<tr>
<td>2</td>
<td>likes</td>
<td>3</td>
<td>2</td>
<td>{since:2013}</td>
<td>{1}</td>
</tr>
<tr>
<td>3</td>
<td>likes</td>
<td>3</td>
<td>4</td>
<td>{since:2015}</td>
<td>{2}</td>
</tr>
<tr>
<td>4</td>
<td>knows</td>
<td>3</td>
<td>5</td>
<td>{}</td>
<td>{2}</td>
</tr>
<tr>
<td>5</td>
<td>likes</td>
<td>5</td>
<td>4</td>
<td>{since:2014}</td>
<td>{2}</td>
</tr>
</tbody>
</table>
Flink DataSet Transformations

**SQL-like Transformations**
- filter
- project
- cross
- union
- distinct
- first-N (limit)
- groupBy
- aggregate
- join
- leftOuterJoin
- rightOuterJoin
- fullOuterJoin

**Hadoop-like Transformations**
- map
- flatMap
- mapPartition
- reduce
- reduceGroup
- coGroup

**Special Flink Operations**
- iterate
- iterateDelta
Operator Implementation

Exclusion

// input: firstGraph (G[1]), secondGraph (G[2])
1: DataSet<GradoopId> graphId = secondGraph.getGraphHead()
2: .map(new Id<>());
3: .map(new Id<>());
4: DataSet newVertices = firstGraph.getVertices()
5: .filter(new NotInGraphBroadcast<>())
6: .withBroadcastSet(graphId, GRAPH_ID);
7: .map(new Id<>());
8: DataSet newEdges = firstGraph.getEdges()
9: .filter(new NotInGraphBroadcast<>())
10: .withBroadcastSet(graphId, GRAPH_ID)
11: .join(newVertices)
12: .where(new SourceId<>().equalTo(new Id<>()))
13: .with(new LeftSide<>(), V<>)
14: .join(newVertices)
15: .where(new TargetId<>().equalTo(new Id<>()))
16: .with(new LeftSide<>(), V<>());
Performance
1. Extract **subgraph** containing only *Persons* and *knows* relations
2. **Transform** *Persons* to necessary information
3. Find communities using **Label Propagation**
4. **Aggregate** vertex count for each community
5. **Select** communities with more than 50K users
6. **Combine** large communities to a single graph
7. **Group** graph by *Persons* *location* and *gender*
8. **Aggregate** vertex and edge count of grouped graph

http://www.ldbcouncil.org/
## Social Network Benchmark – Runtime

### Dataset

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Vertices</th>
<th># Edges</th>
<th>Disk size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphalytics.1</td>
<td>61,613</td>
<td>2,026,082</td>
<td>570 MB</td>
</tr>
<tr>
<td>Graphalytics.10</td>
<td>260,613</td>
<td>16,600,778</td>
<td>4.5 GB</td>
</tr>
<tr>
<td>Graphalytics.100</td>
<td>1,695,613</td>
<td>147,437,275</td>
<td>40.2 GB</td>
</tr>
<tr>
<td>Graphalytics.1000</td>
<td>12,775,613</td>
<td>1,363,747,260</td>
<td>372 GB</td>
</tr>
<tr>
<td>Graphalytics.10000</td>
<td>90,025,613</td>
<td>10,872,109,028</td>
<td>2.9 TB</td>
</tr>
</tbody>
</table>

- 16x Intel(R) Xeon(R) 2.50GHz 6 (12)
- 16x 48 GB RAM
- 1 Gigabit Ethernet
- Hadoop 2.6.0
- Flink 1.0-SNAPSHOT
  - slots (per worker) 12
  - jobmanager.heap.mb 2048
  - taskmanager.heap.mb 40960

### Diagram

- Runtime [s] vs. Number of workers
- Graphalytics.100

- 0
- 200
- 400
- 600
- 800
- 1000
- 1200

- 1
- 2
- 4
- 8
- 16

- Runtimeme [s]
## Social Network Benchmark – Speedup

![Graph analytics performance chart]

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Vertices</th>
<th># Edges</th>
<th>Disk size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphalytics.1</td>
<td>61,613</td>
<td>2,026,082</td>
<td>570 MB</td>
</tr>
<tr>
<td>Graphalytics.10</td>
<td>260,613</td>
<td>16,600,778</td>
<td>4.5 GB</td>
</tr>
<tr>
<td>Graphalytics.100</td>
<td>1,695,613</td>
<td>147,437,275</td>
<td>40.2 GB</td>
</tr>
<tr>
<td>Graphalytics.1000</td>
<td>12,775,613</td>
<td>1,363,747,260</td>
<td>372 GB</td>
</tr>
<tr>
<td>Graphalytics.10000</td>
<td>90,025,613</td>
<td>10,872,109,028</td>
<td>2.9 TB</td>
</tr>
</tbody>
</table>

- 16x Intel(R) Xeon(R) 2.50GHz 6 (12)
- 16x 48 GB RAM
- 1 Gigabit Ethernet
- Hadoop 2.6.0
- Flink 1.0-SNAPSHOT
  - slots (per worker) 12
  - jobmanager.heap.mb 2048
  - taskmanager.heap.mb 40960
### Social Network Benchmark – Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Vertices</th>
<th># Edges</th>
<th>Disk size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphalytics.1</td>
<td>61,613</td>
<td>2,026,082</td>
<td>570 MB</td>
</tr>
<tr>
<td>Graphalytics.10</td>
<td>260,613</td>
<td>16,600,778</td>
<td>4.5 GB</td>
</tr>
<tr>
<td>Graphalytics.100</td>
<td>1,695,613</td>
<td>147,437,275</td>
<td>40.2 GB</td>
</tr>
<tr>
<td>Graphalytics.1000</td>
<td>12,775,613</td>
<td>1,363,747,260</td>
<td>372 GB</td>
</tr>
<tr>
<td>Graphalytics.10000</td>
<td>90,025,613</td>
<td>10,872,109,028</td>
<td>2.9 TB</td>
</tr>
</tbody>
</table>

- 16x Intel(R) Xeon(R) 2.50GHz 6 (12)
- 16x 48 GB RAM
- 1 Gigabit Ethernet
- Hadoop 2.6.0
- Flink 1.0-SNAPSHOT
  - slots (per worker) 12
  - jobmanager.heap.mb 2048
  - taskmanager.heap.mb 40960
Whats next?
Future Work

- Support Temporal Graph Data and Temporal Analysis
- Support Graph Streams and Online analysis
- Inclusion of Apache Flink‘s Table API
  - Additional optimizer (Calcite)
  - Projection- and Filter-Pushdown to data sources
  - Base for Graph Stream support
  - Support for SQL based operators
  - Abstraction layer for framework interchange (Flink vs. Spark)
- Distributed Graph Layouting
Future Work

- Support Temporal Graph Data and Temporal Analysis
- Support Graph Streams and Online analysis
- Inclusion of Apache Flink’s Table API
  - Additional optimizer (Calcite)
  - Projection- and Filter-Pushdown to data sources
  - Base for Graph Stream support
  - Support for SQL based operators
  - Abstraction layer for framework interchange (Flink vs. Spark)
- Distributed Graph Layouting
Temporal Property Graph Model (TPGM)

- Extends the EPGM (downwards compatible)
- **Bitemporal** time representation: valid- and transaction time
- Times can be (1) empty, (2) a timestamp or (3) a time-interval
- Valid times are specified by the application
- Transaction times are maintained by the system
- Whole graph with rollback and historical information
- Chaining of temporal operators → analytical workflow
Future Work
Future Work

- Support Temporal Graph Data and Temporal Analysis
- Support Graph Streams and Online analysis
- **Inclusion of Apache Flink‘s Table API**
  - Additional optimizer (Calcite)
  - Projection- and Filter-Pushdown to data sources
  - Base for Graph Stream support
  - Support for SQL based operators
  - Abstraction layer for framework interchange (Flink vs. Spark)
- Distributed Graph Layouting
Gradoop on Flink Table API

SQL

Table API

DataSet API

DataStream API

Flink Dataflow Runtime
Gradoop on Flink Table API

• Relational API is declarative
  • Knowledge of dataflow processing concepts is not needed
  • Everyone knows Sequel (default language for data analytics)
  • Table-centric: Just think about tables

• Efficient query optimization
  • Less logic in User Defined Functions (UDFs)
  • System optimizes queries more efficiently (Calcite)

COMMING SOON!
Thank you!

www.gradoop.com

http://flink.apache.org
http://ldbcouncil.org

Visit our Wiki!
https://github.com/dbs-leipzig/gradoop/wiki

Try the Gradoop-Demo!
https://github.com/dbs-leipzig/gradoop_demo

Questions? Mail to: gomez@informatik.uni-leipzig.de

