SCALABLE GRAPH DATA ANALYTICS WITH GRADOOP

ERHARD RAHM,
MARTIN JUNGHANNS, ANDRÉ PETERMANN, ERIC PEUKERT

www.scads.de
$Graph = (Vertices, Edges)$
"GRAPHS ARE EVERYWHERE"

Graph = (Users, Followers)
Graph = (Users, Friendships)
Graph = (Users ∪ Bands, Friendships ∪ Likes)
Graph = (Users ∪ Bands, Friendships ∪ Likes)
"GRAPHS CAN BE ANALYZED"

Assuming a social network
1. Determine subgraph
2. Find communities
3. Filter communities
4. Find common subgraph
GRAPH DATA ANALYTICS: REQUIREMENTS

- *all V challenges (volume, variety, velocity, veracity)*
- *ease-of-use*
- *high cost-effectiveness*
- powerful but easy to use graph data model
  - support for heterogeneous, schema-flexible vertices and edges
  - support for collections of graphs (not only 1 graph)
  - powerful graph operators
- graph-based integration of many data sources
- versioning and evolution (dynamic /temporal graphs)
- interactive, declarative graph queries
- scalable graph mining
- comprehensive visualization support
<table>
<thead>
<tr>
<th>Feature</th>
<th>Neo4j, OrientDB</th>
<th>Neo4j, OrientDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>data model</td>
<td>rich graph models (PGM)</td>
<td></td>
</tr>
<tr>
<td>focus</td>
<td>queries</td>
<td></td>
</tr>
<tr>
<td>query language</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>graph analytics</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>scalability</td>
<td>vertical</td>
<td></td>
</tr>
<tr>
<td>Workflows</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>persistency</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>dynamic graphs / versioning</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>data integration</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>visualization</td>
<td>(yes)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Graph Database Systems Neo4j, OrientDB</td>
<td>Graph Processing Systems (Pregel, Giraph)</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>data model</td>
<td>rich graph models (PGM)</td>
<td>generic graph models</td>
</tr>
<tr>
<td>focus</td>
<td>queries</td>
<td>analytic</td>
</tr>
<tr>
<td>query language</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>graph analytics</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>scalability</td>
<td>vertical</td>
<td>horizontal</td>
</tr>
<tr>
<td>Workflows</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>persistency</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>dynamic graphs / versioning</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>data integration</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>visualization</td>
<td>(yes)</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>Graph Database Systems Neo4j, OrientDB</td>
<td>Graph Processing Systems (Pregel, Giraph)</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>----------------------------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>data model</td>
<td>rich graph models (PGM)</td>
<td>generic graph models</td>
</tr>
<tr>
<td>focus</td>
<td>queries</td>
<td>analytic</td>
</tr>
<tr>
<td>query language</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>graph analytics</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>scalability</td>
<td>vertical</td>
<td>horizontal</td>
</tr>
<tr>
<td>Workflows</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>persistency</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>dynamic graphs / versioning</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>data integration</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>visualization</td>
<td>(yes)</td>
<td>no</td>
</tr>
</tbody>
</table>
An end-to-end framework and research platform for efficient, distributed and domain independent graph data management and analytics.
Graph Databases

Graph Dataflow Systems

Graph Processing Systems

Ease-of-use

Data Volume and Problem Complexity

neo4j

Gradoop

GraphX

Gelly
AGENDA

- Intro Graph Analytics
  - Graph data
  - Requirements
  - Graph database vs graph processing systems

- Gradoop
  - Architecture
  - Extended Property Graph Model (EPGM)
  - Implementation
  - Evaluation

- Summary/Outlook
Hadoop-based framework for graph data management and analysis
  - persistent graph storage in scalable distributed store (Hbase)
  - utilization of powerful dataflow system (Apache Flink) for parallel, in-memory processing

Extended property graph data model (EPGM)
  - operators on graphs and sets of (sub) graphs
  - support for semantic graph queries and mining

Declarative specification of graph analysis workflows
  - Graph Analytical Language - GrALa

End-to-end functionality
  - graph-based data integration, data analysis and visualization

Open-source implementation: www.gradoop.org
- **integrate data** from one or more sources into a dedicated **graph store** with **common graph data model**

- **definition of analytical workflows** from **operator algebra**

- **result representation in meaningful way**
HIGH LEVEL ARCHITECTURE

Data flow

Control flow

Workflow Declaration

Visual

GrALa DSL

Representation

Extended Property Graph Model

Flink Operator Implementations

Data Integration

Graph Analytics

Representation

Flink Operator Execution

HBase Distributed Graph Store

HDFS/YARN Cluster
GRADOOP AS A FLINK EXTENSION

Graph Operators
Extended Property Graph Model
Graph Store

HadoopMR Table Gelly FlinkML ...

DataSet
DataStream

Streaming Dataflow Runtime

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

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

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

CEP...
EXTENDED PROPERTY GRAPH MODEL (EPGM)

- includes PGM as special case
- support for collections of logical graphs / subgraphs
  - can be defined explicitly
  - can be result of graph algorithms / operators
- support for graph properties
- powerful operators on both graphs and graph collections
- Graph Analytical Language – GrALa
  - domain-specific language (DSL) for EPGM
  - flexible use of operators with application-specific UDFs
  - plugin concept for graph mining algorithms
• Vertices and directed Edges
- Vertices and directed Edges
- Logical Graphs
• Vertices and directed Edges
• Logical Graphs
• Identifiers
• Vertices and directed Edges
• Logical Graphs
• Identifiers
• Type Labels
- Vertices and directed Edges
- Logical Graphs
- Identifiers
- Type Labels
- Properties
Combination

Overlap

Exclusion

LogicalGraph graph3 = graph1.combine(graph2);
LogicalGraph graph4 = graph1.overlap(graph2);
LogicalGraph graph5 = graph1.exclude(graph2);
```python
udf = (graph => graph['vertexCount'] = graph.vertices.size())
graph3 = graph3.aggregate(udf)
```
LogicalGraph graph4 = graph3.subgraph((vertex => vertex[:label] == 'green'))
LogicalGraph graph5 = graph3.subgraph((edge => edge[:label] == 'blue'))
LogicalGraph graph6 = graph3.subgraph(
  (vertex => vertex[:label] == 'green'),
  (edge => edge[:label] == 'orange'))
GraphCollection collection = graph3.match("(:Green)-[:orange]->(:Orange)");
LogicalGraph grouped = graph3.groupBy(
    [:label], // vertex keys
    [:label]) // edge keys
LogicalGraph grouped = graph3.groupBy([:label], [COUNT()], [:label], [MAX('a')])
GROUPING: TYPE LEVEL (SCHEMA GRAPH)

vertexGrKeys = [:label]
edgeGrKeys = [:label]
sumGraph = databaseGraph.groupby(vertexGrKeys, [COUNT()], edgeGrKeys, [COUNT()])
personGraph = databaseGraph.subgraph((vertex => vertex[:label] == 'Person'),
  (edge => edge[:label] == 'knows'))
vertexGrKeys = [:label, “city”]
edgeGrKeys = [:label]
sumGraph = personGraph.groupBy(vertexGrKeys, [COUNT()], edgeGrKeys, [COUNT()])
GraphCollection filtered = collection.select((graph => graph[‘vertexCount’] > 4));
GraphCollection frequentPatterns = collection.callForCollection(new TransactionalFSM(0.5))
Implementation
### EPGMGraphHead

<table>
<thead>
<tr>
<th>Id</th>
<th>Label</th>
<th>Properties</th>
<th>POJO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### EPGMVertex

<table>
<thead>
<tr>
<th>Id</th>
<th>Label</th>
<th>Properties</th>
<th>SourceId</th>
<th>TargetId</th>
<th>Graphs</th>
<th>POJO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### EPGMEdge

<table>
<thead>
<tr>
<th>Id</th>
<th>Label</th>
<th>Properties</th>
<th>SourceId</th>
<th>TargetId</th>
<th>Graphs</th>
<th>POJO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### EPGMVertex

<table>
<thead>
<tr>
<th>Id</th>
<th>Label</th>
<th>Properties</th>
<th>Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **GradoopId**: UUID 128-bit
- **PropertyList**: List(Property)
- **Property**: (String, PropertyValue)
- **PropertyValue**: byte[]
### 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<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<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>
Exclusion

// input: firstGraph (G[1]), secondGraph (G[2])

1: DataSet<GradoopId> graphId = secondGraph.getGraphHead();
2: .map(new Id<G>();
3: 
4: DataSet<V> newVertices = firstGraph.getVertices().filter(new NotInGraphBroadCast<V>()).
5: .withBroadcastSet(graphId, GRAPH_ID);
6: 
7: DataSet<E> newEdges = firstGraph.getEdges().filter(new NotInGraphBroadCast<E>()).
8: .withBroadcastSet(graphId, GRAPH_ID).
9: 
10: DataSet<E> newEdges = firstGraph.getEdges().filter(new NotInGraphBroadCast<E>()).
11: .withBroadcastSet(graphId, GRAPH_ID).
12: 
13: .join(newVertices).
14: .where(new SourceId<E>().equalTo(new Id<V>())).
15: .with(new LeftSide<E, V>().
16: .join(newVertices).
17: .where(new TargetId<E>().equalTo(new Id<V>())).
18: .with(new LeftSide<E, V>().}
IMPLEMENTATION OF GRAPH GROUPING

**V**
- Map
- Extract attributes

**GroupBy(1) + GroupReduce***
- Assign vertices to groups
- Compute aggregates
- Create super vertex tuples
- Forward updated group members

**V1**

**V2**
- Filter + Map
- Extract super vertex tuples
- Build super vertices

**V3**
- Filter + Map
- Extract group members
- Reduce memory footprint

**E**
- Map
- Extract attributes

**Join***
- Replace Source/TargetId with corresponding super vertex id

**E1**

**E2**
- GroupBy(1,2,3) + GC + GR* + Map
- Assign edges to groups
- Compute aggregates
- Build super edges

**E’**

*requires worker communication*
ITERATIVE COMPUTATION OF FREQUENT SUBGRAPHS

collecting intermediate iteration results

G: grow frequent patterns
R: report supported patterns
C: count global frequency
F: filter by min frequency
Evaluation
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
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

```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.setProperty("city", current.getPropertyValue("city");
        transformed.setProperty("gender", current.getPropertyValue("gender");
        transformed.setProperty("birthday", current.getPropertyValue("birthday");
        return transformed;
    })
    .returnResults();

    // 2a) compute communities
    .callForGraph(new GellyLabelPropagation<GraphHeadPojo, VertexPojo, EdgePojo>());
    // 2b) separate communities
    .splitBy(label);
    // 4) compute vertex count per community
    .apply(new ApplyAggregation<>());
    // 5) select graphs with more than minClusterSize vertices
    .select((g) -> {
        return g.getPropertyValue("vertexCount").getLong() > threshold;
    });
    // 6) reduce filtered graphs to a single graph using combination
    .reduce(new ReduceCombination<>());
    // 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>());
```

https://git.io/vgozj
### Benchmark Results

#### Dataset Details

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

#### Hardware Configuration
- 16x Intel(R) Xeon(R) 2.50GHz (6 Cores)
- 16x 48 GB RAM
- 1 Gigabit Ethernet
- Hadoop 2.6.0
- Flink 1.0-SNAPSHOT

#### Graph Analytics

**Runtime**

![Runtime Graph](chart1.png)

**Speedup**

![Speedup Graph](chart2.png)
**Datasets**

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

- 16x Intel(R) Xeon(R) 2.50GHz (6 Cores)
- 16x 48 GB RAM
- 1 Gigabit Ethernet
- Hadoop 2.6.0
- Flink 1.0-SNAPSHOT
EVALUATION OF GROUPING: SCALABILITY

Speedup for grouping on type

Runtime for grouping on type
AGENDA

- Intro Graph Analytics
  - Graph data
  - Graph databases vs graph processing systems

- Gradoop
  - Architecture
  - Extended Property Graph Model (EPGM)
  - Use cases
  - Evaluation

- Summary/Outlook
- **Big Graph Analytics**
  - Hadoop-based graph processing frameworks based on generic graphs
  - Spark/Flink: batch/streaming-oriented workflows (rather than interactive OLAP)
  - graph collections not generally supported
  - generally missing: graph-based data integration, built-in support for dynamic graph data

- **GraDoop** ([www.gradoop.org](http://www.gradoop.org))
  - open-source infrastructure for entire processing pipeline: graph acquisition, storage, integration, transformation, analysis (queries + graph mining), visualization
  - extended property graph model (EPGM) with powerful operators (e.g., grouping, pattern matching) and support for graph collections
  - leverages Hadoop ecosystem
    - Apache HBase for permanent graph storage
    - Apache Flink to implement operators
  - ongoing implementation
<table>
<thead>
<tr>
<th></th>
<th>Graph Database Systems Neo4j, OrientDB</th>
<th>Graph Processing Systems (Pregel, Giraph)</th>
<th>Distributed Dataflow Systems (Flink Gelly, Spark GraphX)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>data model</td>
<td>rich graph models (PGM)</td>
<td>generic graph models</td>
<td>generic graph models</td>
<td>Extended PGM</td>
</tr>
<tr>
<td>focus</td>
<td>queries</td>
<td>analytic</td>
<td>analytic</td>
<td>analytic</td>
</tr>
<tr>
<td>query language</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>graph analytics</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>scalability</td>
<td>vertical</td>
<td>horizontal</td>
<td>horizontal</td>
<td>horizontal</td>
</tr>
<tr>
<td>Workflows</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>persistency</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>dynamic graphs / versioning</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>data integration</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>(yes)</td>
</tr>
<tr>
<td>visualization</td>
<td>(yes)</td>
<td>no</td>
<td>no</td>
<td>limited</td>
</tr>
</tbody>
</table>
LESSONS LEARNED ABOUT FLINK

- instrumental to develop Gradoop in relatively short time
- elegant and intuitive DataSet API
- very good out-of-the-box performance for non-custom types
- stumbling blocks
  - collecting intermediate results during iterations requires non-intuitive workarounds
  - missing possibility to reuse datasets in data flow programs
  - missing multicast operator with multiple outputs of possibly different types (to replace filter hierarchies causing duplication of previous outputs)
  - missing support for theta-joins (e.g., via user-defined join predicates)
  - missing adaptive configuration of parallelism (e.g., to keep data local as long as possible)
OUTLOOK / CHALLENGES

- Graph-based data integration
  - unified approach for knowledge graphs and regular data graphs
  - holistic data integration for many sources

- Graph analytics
  - automatic optimization of analysis workflows
  - optimized graph partitioning approaches
  - visualization of graphs and analysis results
  - interactive graph analytics
  - dynamic graph data


A. Petermann, M. Junghanns, R. Müller, E. Rahm: *BIIIG: Enabling Business Intelligence with Integrated Instance Graphs*. Proc. 5th Int. Workshop on Graph Data Management (GDM 2014)

A. Petermann, M. Junghanns, R. Müller, E. Rahm: *Graph-based Data Integration and Business Intelligence with BIIIG*. Proc. VLDB Conf., 2014

A. Petermann, M. Junghanns, R. Müller, E. Rahm: *FoodBroker - Generating Synthetic Datasets for Graph-Based Business Analytics*. Proc. 5th Int. Workshop on Big Data Benchmarking (WBDB), 2014

A. Petermann; M. Junghanns: *Scalable Business Intelligence with Graph Collections*. it - Information Technology Special Issue: Big Data Analytics, 2016