GRAPH-BASED DATA INTEGRATION AND ANALYSIS FOR BIG DATA

ERHARD RAHM

www.scads.de
Big is changing quickly
- Gigabytes
- Terabytes ($10^{12}$)
- Petabytes ($10^{15}$)
- Exabytes ($10^{18}$)
- Zettabytes ($10^{21}$),
- Yottabytes ($10^{24}$),
- Brontobytes ($10^{27}$), ...

by 2020 about 40 ZB of data will be generated

Source: IDC
BIG DATA CHALLENGES

**Volume**
Petabytes / exabytes of data

**Velocity**
fast analysis of data streams

**Variety**
heterogeneous data of different kinds

**Veracity**
high data quality

**Value**
useful analysis results
BIG DATA ANALYSIS PIPELINE

Data acquisition → Data extraction/cleaning → Data integration/annotation → Data analysis and visualization → Interpretation

Variety  Volume  Velocity  Veracity  Privacy
"GRAPHS ARE EVERYWHERE"

Social science

Facebook
ca. 1.3 billion users
ca. 340 friends per user
Twitter
ca. 300 million users
cia. 500 million tweets per day

Engineering

Internet
ca. 2.9 billion users

Life science

Gene (human)
20,000-25,000
cia. 4 million individuals
Patients
> 18 millions (Germany)
Illnesses
> 30,000

Information science

World Wide Web
ca. 1 billion Websites
LOD-Cloud
ca. 90 billion triples
Graph = (Vertices, Edges)
Graph = (Users, Followers)
"GRAPHS ARE EVERYWHERE"

Graph = (Users, Friendships)
"GRAPHS ARE HETEROGENEOUS"

Graph = (Users ∪ Bands, Friendships ∪ Likes)
“GRAPHS CAN BE ANALYZED”

Graph = (Users ∪ Bands, Friendships ∪ Likes)
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>
</tbody>
</table>
## COMPARISON (2)

<table>
<thead>
<tr>
<th></th>
<th>Graph Database Systems Neo4j, OrientDB</th>
<th>Graph Processing Systems (Pregel, Giraph)</th>
</tr>
</thead>
<tbody>
<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
AGENDA

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

- Gradoop architecture and data integration

- Extended Property Graph Model (EPGM)
  - Data organization and operators
  - Implementation

- Performance 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

UNIVERSITÄT LEIPZIG
BIIIG: Business Intelligence on Integrated Instance Graphs

- heterogeneous data sources are integrated within an instance graph by preserving original relationships between data objects
  - transactional and master data

- largely automated extraction of metadata and instance data and transformation into graphs
  - fusion of matching entities and relations

- extraction of subgraphs (business transaction graphs) related to interrelated business activities

- analysis of graphs/subgraphs with aggregation queries, pattern mining etc.
„Business Intelligence on Integrated Instance Graphs (BIIIG)“ (PVLDB 2014)
Intro Graph Analytics
- Graph data
- Requirements
- Graph database vs graph processing systems

Gradoop architecture and data integration

Extended Property Graph Model (EPGM)
- Data organization and operators
- Implementation

Performance Evaluation

Summary/Outlook
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

Graph:

- Vertex 1: Person name: Alice, born: 1984
- Vertex 2: Band name: Metallica, founded: 1981
- Vertex 3: Person name: Bob
- Vertex 4: Band name: AC/DC, founded: 1973
- Vertex 5: Person name: Eve

Edges:
- Alice likes Metallica since 2014
- Bob likes AC/DC since 2013
- Alice knows Eve since 2014
- Person 2 and 3 are in Community 1: interest: Heavy Metal
- Person 2 and 5 are in Community 2: interest: Hard Rock
<table>
<thead>
<tr>
<th>Operators</th>
<th>Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unary</strong></td>
<td><strong>Gelly Library</strong></td>
</tr>
<tr>
<td>Aggregation</td>
<td><strong>BTG Extraction</strong></td>
</tr>
<tr>
<td>Pattern Matching</td>
<td><strong>Adaptive Partitioning</strong></td>
</tr>
<tr>
<td>Transformation</td>
<td><strong>Frequent Subgraphs</strong></td>
</tr>
<tr>
<td>Grouping</td>
<td></td>
</tr>
<tr>
<td>Subgraph</td>
<td></td>
</tr>
<tr>
<td>Call *</td>
<td></td>
</tr>
<tr>
<td><strong>Binary</strong></td>
<td></td>
</tr>
<tr>
<td>Combination</td>
<td></td>
</tr>
<tr>
<td>Overlap</td>
<td></td>
</tr>
<tr>
<td>Exclusion</td>
<td></td>
</tr>
<tr>
<td>Equality</td>
<td></td>
</tr>
<tr>
<td><strong>Graph Collection</strong></td>
<td></td>
</tr>
<tr>
<td>Selection</td>
<td></td>
</tr>
<tr>
<td>Distinct</td>
<td></td>
</tr>
<tr>
<td>Sort</td>
<td></td>
</tr>
<tr>
<td>Limit</td>
<td></td>
</tr>
<tr>
<td>Apply *</td>
<td></td>
</tr>
<tr>
<td>Reduce *</td>
<td></td>
</tr>
<tr>
<td>Call *</td>
<td></td>
</tr>
</tbody>
</table>

* auxiliary
BASIC BINARY OPERATORS

Combination

Overlap

Exclusion

LogicalGraph graph3 = graph1.combine(graph2);
LogicalGraph graph4 = graph1.overlap(graph2);
LogicalGraph graph5 = graph1.exclude(graph2);
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)"));

- new: support of Cypher query language for pattern matching*

q = "MATCH (p1: Person) -[e: knows *1..3] -> (p2: Person)
WHERE p1.gender <> p2.gender RETURN *
GraphCollection matches = g.cypher (q)

* Junghanns et al.: Cypher-based Graph Pattern Matching in Gradoop. Proc. GRADES 2017
LogicalGraph grouped = graph3.groupBy([:label], // vertex keys
                               [:label]) // edge keys
LogicalGraph grouped = graph3.groupBy([:label], [COUNT()], [:label], [MAX('a')])
SAMPLE GRAPH
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>
</tr>
</thead>
</table>

POJO

DataSet<EPGMGraphHead>

EPGMVertex

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

POJO

DataSet<EPGMVertex>

EPGMEdge

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

POJO

DataSet<EPGMEdge>

EPGMVertex

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

GradoopId := UUID 128-bit
PropertyList := List<Property>
PropertyValue := byte[]
GradoopIdSet := Set<GradoopId>
**Graph Representation: Example**

<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<EPGMVertex>**

<table>
<thead>
<tr>
<th>Id</th>
<th>Label</th>
<th>Properties</th>
<th>Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>likes</td>
<td>since: 2014</td>
<td>{1}</td>
</tr>
<tr>
<td>2</td>
<td>likes</td>
<td>since: 2013</td>
<td>{1}</td>
</tr>
<tr>
<td>3</td>
<td>knows</td>
<td></td>
<td>{2}</td>
</tr>
<tr>
<td>4</td>
<td>likes</td>
<td>since: 2015</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>
\[
\begin{align*}
\text{Exclusion} & \\
// & \text{input: firstGraph (G[1]), secondGraph (G[2])} \\
1: & \text{DataSet<GradoopId> graphId = secondGraph.getGraphHead();} \\
2: & \text{.map(new Id<GradoopId>());} \\
3: & \text{} \\
4: & \text{DataSet<V> newVertices = firstGraph.getVertices()} \\
5: & \text{.filter(new NotInGraphBroadcast<V>());} \\
6: & \text{.withBroadcastSet(graphId, GRAPH_ID);} \\
7: & \text{.filter(new NotInGraphBroadcast<E>());} \\
8: & \text{.withBroadcastSet(graphId, GRAPH_ID);} \\
9: & \text{.join(newVertices)} \\
10: & \text{.where(new SourceId<E>().equalTo(new Id<V>());} \\
11: & \text{.with(new LeftSide<E, V>();} \\
12: & \text{.join(newVertices)} \\
13: & \text{.where(new TargetId<E>().equalTo(new Id<V>());} \\
14: & \text{.with(new LeftSide<E, V>();} \\
15: & \text{} \\
16: & \text{.with(new LeftSide<E, V>();)} \\
\end{align*}
\]
IMPLEMENTATION OF GRAPH GROUPING (PROC. BTW2017)

- **V**
  - Map
  - Extract attributes

- **V1**
  - GroupBy(1) + GroupReduce*
  - Assign vertices to groups
  - Compute aggregates
  - Create super vertex tuples
  - Forward updated group members

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

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

- **E**
  - Map
  - Extract attributes

- **E1**
  - Join*
  - Replace Source/TargetId with corresponding super vertex id

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

*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
### AGENDA

- Intro Graph Analytics
  - Graph data
  - Requirements
  - Graph database vs graph processing systems
- Gradoop architecture and data integration
- Extended Property Graph Model (EPGM)
  - Data organization and operators
  - Implementation
- Performance Evaluation
- Summary/Outlook
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

```
return socialNetwork
// 1) extract subgraph
   .subgraph((vertex) -> {
       return vertex.getIs().toLowerCase().equals("person");
   }, (edge) -> {
       return edge.getIs().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");
   })
// 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) 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<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>())
```

https://git.io/vgozj
### BENCHMARK RESULTS

<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
### 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
  - Requirements
  - Graph database vs graph processing systems

- Gradoop architecture and data integration

- Extended Property Graph Model (EPGM)
  - Data organization and operators
  - Implementation

- Performance 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)

- 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
## COMPARISON

<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>Extended PGM</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>(yes)</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>
OUTLOOK / CHALLENGES

- Graph-based data integration
  - refined operators for data import, matching and fusion
  - holistic data integration for many sources (clusters of matching entities instead of binary “sameAs“ links)

- Graph analytics
  - optimized graph partitioning approaches
  - automatic load balancing techniques
  - visualization of graphs and analysis results
  - interactive graph analytics
  - dynamic graph data
REFERENCES

- M. Junghanns, M. Kießling, A. Averbuch, A. Petermann, E. Rahm: *Cypher-based Graph Pattern Matching in Gradoop*. Proc. ACM SIGMOD workshop on Graph Data Management Experiences and Systems (GRADES), 2017
- A. Petermann; M. Junghanns: *Scalable Business Intelligence with Graph Collections*. it - Information Technology Special Issue: Big Data Analytics, 2016
- 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, E. Rahm: *DIMSpan - Transactional Frequent Subgraph Mining with Distributed In-Memory Dataflow Systems*. arXiv 2017