GRAPH-BASED DATA INTEGRATION AND ANALYSIS
WITH GRADOOP

ERHARD RAHM,
MARTIN JUNGHANNS, ANDRÉ PETERMANN, ERIC PEUKERT

www.scads.de
- Project period: 4 years (10/2014 – 09/2018), option for +3 more years after evaluation
- Many involved research groups + many associated partners
- Focal point for new research activities
- Specialists from computer and domain sciences
COLLABORATION WITH APPLICATION SCIENCES

- Life-Sciences
- Material Sciences
- Environmental and Traffic Sciences
- Digital Humanities
- Business Data

- Service-Center

- Big Data Life Cycle Management and Workflows

- Data Quality/Data Integration

- Knowledge Extraction

- Visual Analysis

- Efficient Big Data Architecture

- K.-P. Fähnrich
- M. Bogdan
- C. Rother
- P. Stadler
- G. Heyer
- G. Scheuermann

- W.E. Nagel
- W. Lehner
- S. Gumhold
Life Sciences
- Gene-Sequence Alignment (Consulting, Measurement, Parallelization), Myers-Bit-Vector with dedicated hardware
- Parallelization of Gene-Imputation-Tasks
- UFZ- Mass-Spectrometry-Data Workflows

Digital Humanities
- Text repository services for Digital Humanities (CTS)
- OCR-Workflow on HPC (in progress)

Environmental/Traffic sciences
- Deep Learning & structure recognition in spatial planning

Material and Engineering Sciences
- Multi-Scale Visualisation

Business Data
- Condition Monitoring of Renewable Energy Power Plants
- Real-Estate Data Fusion
- Deduplication of Person Data
COMPUTER SCIENCE RESEARCH (METHODS)

- Big Data Lifecycle and Workflows
  - KNIME & HPC Integration
- Data Quality and Integration
  - Privacy-Preserving Data Matching
  - Hadoop-based deduplication (Dedoop)
  - Graph-based data integration
  - Holistic data integration
- Visual Analysis
  - Porting CV-Algorithms to GPU
  - Improve Visualization of Large Particle Data
  - Muliti-Scale Visualization for Big Data
- Knowledge Extraction
  - Deep Learning for Structure Recognition in Spatial Planning
- Big Data Infrastructure
  - Hardening computation infrastructure (Security)
  - Flexible Cluster management
  - Big Data Framework Execution & Monitoring on HPC
  - Time Series Management and Forecasting
  - Geotemporal Data Storage
- Graph-based data analysis (Gradoop)
Graph = (Vertices, Edges)
“GRAPHS ARE EVERYWHERE”

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)
"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
## COMPARISON

<table>
<thead>
<tr>
<th></th>
<th>Graph Database Systems Neo4j, OrientDB</th>
<th></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.
Data Volume and Problem Complexity

Ease-of-use

Graph Databases

Graph Dataflow Systems

Graph Processing Systems

Gelly

Gradoop

GraphX

neod4j
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
GRADOOP CHARACTERISTICS

- 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

22
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)
SCREENSHOT OF NEO4J IMPLEMENTATION
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
- 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
Operators
**BASIC BINARY OPERATORS**

**Combination**

LogicalGraph graph3 = graph1.combine(graph2);

**Overlap**

LogicalGraph graph4 = graph1.overlap(graph2);

**Exclusion**

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)");
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()])
GROUPING: PROPERTY-SPECIFIC

```javascript
personGraph = databaseGraph.subgraph((vertex => vertex[:label] == 'Person'),
  (edge => edge[:label] == 'knows'))
vertexGrKeys = [:label, "city"]
edgeGrKeys = [:label]
sumGraph = personGraph.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>

\[\text{POJO}\]

\[\text{DataSet}\langle\text{EPGMGraphHead}\rangle\]

EPGMVertex

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

\[\text{POJO}\]

\[\text{DataSet}\langle\text{EPGMVertex}\rangle\]

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>

\[\text{POJO}\]

\[\text{DataSet}\langle\text{EPGMEdge}\rangle\]

EPGMVertex

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

\[\text{POJO}\]

\[\text{GradoopId} \leftarrow \text{UUID}\]

\[\text{128-bit}\]

\[\text{PropertyList} \leftarrow \text{List}\langle\text{Property}\rangle\]

\[\text{Property} \leftarrow (\text{String}, \text{PropertyValue})\]

\[\text{PropertyValue} \leftarrow \text{byte[]}\]

\[\text{GradoopIdSet} \leftarrow \text{Set}\langle\text{GradoopId}\rangle\]
**GRAPH REPRESENTATION: EXAMPLE**

**DataSet<EPGMGraphHead>**

<table>
<thead>
<tr>
<th>Id</th>
<th>Label</th>
<th>Properties</th>
<th>Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Community</td>
<td>{interest:Heavy Metal}</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Community</td>
<td>{interest:Hard Rock}</td>
<td></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<>());
3: 
4: DataSet<V> newVertices = firstGraph.getVertices();
5: .filter(new NotInGraphBroadcast<>())
6: .withBroadcastSet(graphId, GRAPH_ID);
7: 
8: DataSet<E> 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<E, V>())
14: .join(newVertices)
15: .where(new TargetId<>().equalTo(new Id<>()))
16: .with(new LeftSide<E, V>())

IMPLEMENTATION OF GRAPH GROUPING (PROC. BTW2017)

Map
Extract attributes

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

Filter + Map
Extract super vertex tuples
Build super vertices

Filter + Map
Extract group members
Reduce memory footprint

Map
Extract attributes

Join*
Replace Source/TargetId with corresponding super vertex id

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

search space

1-edge

2-edge

3-edge

n-edge

result

1-edge

2-edge

3-edge

n-edge

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

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.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(birthday, 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) 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
```
### Dataset Summary

<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

### Performance Metrics

#### Runtime

- Graphalytics.100

#### Speedup

- Linear

---

**Runtime**

- **Number of workers** vs. **Runtime [s]**

**Speedup**

- **Number of workers** vs. **Speedup**
### BENCHMARK RESULTS 2

#### 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>graadoop</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>
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
REFERENCES

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