SCALABLE GRAPH ANALYTICS WITH GRADOOP AND BIIGG

MARTIN JUNGHANNS, ANDRE PETERMANN, ERHARD RAHM
Graph Analytics on Hadoop (Gradoop)
- Distributed graph data management
- Rich graph data model with powerful operators
- Domain independent

Business Intelligence with Integrated Instance Graphs (BIIIG)
- Graph-based data integration
- Graph OLAP, Mining and visualization
- Improved Scalability on Gradoop
„GRAPHS ARE EVERYWHERE“ AND LARGE

Social science

Engineering

Life science

Information science
Integrate data from one or more sources into a dedicated graph storage with common graph data model

Definition of analytical workflows from operator algebra

Result representation in meaningful way
An end-to-end framework and research platform for efficient, distributed and domain independent graph data management and analytics.
HIGH LEVEL ARCHITECTURE

Data flow
Control flow

Workflow Declaration
Visual
GrALa DSL

Representation

Workflow Execution

Flink Operator Implementations

Data Integration
Graph Analytics
Representation

Flink Operator Execution

Extended Property Graph Model

HBase Distributed Graph Store

HDFS Cluster
1. Simple but powerful
   • intuitive graphs are flat structures of vertices and binary edges

2. Logical graphs
   • support of multiple, possibly overlapping graphs in one database is advantageous for analytical applications

3. Attributes and type labels
   • type labels and custom properties for vertices, edges and graphs

4. Parallel edges and loops
   • allow multiple relations between two vertices and self-connected relations
## Graph Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Definition</th>
<th>GrALa notation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unary</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pattern Matching</td>
<td>$\mu_{G^*,\varphi} : G \rightarrow G^n$</td>
<td>graph.match(patternGraph, predicate) : Collection</td>
</tr>
<tr>
<td>Aggregation</td>
<td>$\gamma_a : G \rightarrow G$</td>
<td>graph.aggregate(propertyKey, aggregateFunction) : Graph</td>
</tr>
<tr>
<td>Projection</td>
<td>$\pi_{v,e} : G \rightarrow G$</td>
<td>graph.project(vertexFunction, edgeFunction) : Graph</td>
</tr>
<tr>
<td>Summarization</td>
<td>$\varsigma_{v,e} : G \rightarrow G$</td>
<td>graph.summarize(vertexGroupKeys, vertexAggregateFunction, edgeGroupKeys, edgeAggregateFunction) : Graph</td>
</tr>
<tr>
<td><strong>Binary</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combination</td>
<td>$\sqcup : G^2 \rightarrow G$</td>
<td>graph.combine(otherGraph) : Graph</td>
</tr>
<tr>
<td>Overlap</td>
<td>$\sqcap : G^2 \rightarrow G$</td>
<td>graph.overlap(otherGraph) : Graph</td>
</tr>
<tr>
<td>Exclusion</td>
<td>$\neg : G^2 \rightarrow G$</td>
<td>graph.exclude(otherGraph) : Graph</td>
</tr>
</tbody>
</table>
1: personGraph = db.G[0].combine(db.G[1]).combine(db.G[2])
2: vertexGroupingKeys = {':type, "city"'}
3: edgeGroupingKeys = {':type'}
4: vertexAggFunc = (Vertex vSum, Set vertices => vSum["count"] = |vertices|)
5: edgeAggFunc = (Edge eSum, Set edges => eSum["count"] = |edges|)
6: sumGraph = personGraph.summarize(vertexGroupingKeys, edgeGroupingKeys, vertexAggFunc, edgeAggFunc)
**WORKFLOW EXAMPLE: SUMMARIZATION**

1: `personGraph = db.G[0].combine(db.G[1]).combine(db.G[2])`
2: `vertexGroupingKeys = {:type, “city”}`
3: `edgeGroupingKeys = {:type}`
4: `vertexAggFunc = (Vertex vSum, Set vertices => vSum[“count”] = |vertices|)`
5: `edgeAggFunc = (Edge eSum, Set edges => eSum[“count”] = |edges|)`
6: `sumGraph = personGraph.summarize(vertexGroupingKeys, edgeGroupingKeys, vertexAggFunc, edgeAggFunc)`
<table>
<thead>
<tr>
<th>Operator</th>
<th>Definition</th>
<th>GrALa notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Selection</td>
<td>$\sigma_{\varphi} : G^n \to G^n$</td>
<td>collection.select(predicate) : Collection</td>
</tr>
<tr>
<td>Distinct</td>
<td>$\delta : G^n \to G^n$</td>
<td>collection.distinct() : Collection</td>
</tr>
<tr>
<td>Sort by</td>
<td>$\xi_{k,d} : G^n \to G^n$</td>
<td>collection.sortBy(key, [:asc</td>
</tr>
<tr>
<td>Top</td>
<td>$\beta_n : G^n \to G^n$</td>
<td>collection.top(limit) : Collection</td>
</tr>
<tr>
<td>Union</td>
<td>$\cup : (G^n)^2 \to G^n$</td>
<td>collection.union(otherCollection) : Collection</td>
</tr>
<tr>
<td>Intersection</td>
<td>$\cap : (G^n)^2 \to G^n$</td>
<td>collection.intersect(otherCollection) : Collection</td>
</tr>
<tr>
<td>Difference</td>
<td>$\setminus : (G^n)^2 \to G^n$</td>
<td>collection.difference(otherCollection) : Collection</td>
</tr>
<tr>
<td>auxiliary</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apply</td>
<td>$\lambda_o : G^n \to G^n$</td>
<td>collection.apply(unaryGraphOperator) : Collection</td>
</tr>
<tr>
<td>Reduce</td>
<td>$\rho_o : G^n \to G$</td>
<td>collection.reduce(binaryGraphOperator) : Graph</td>
</tr>
<tr>
<td>Call</td>
<td>$\eta_{a,P} : G \cup G^n \to G \cup G^n$</td>
<td>[graph</td>
</tr>
</tbody>
</table>
2: predicate = (Graph g => |g.V| > 3
3: result = collection.select(predicate)
2: predicate = (Graph g => |g.V| > 3
3: result = collection.select(predicate)
<table>
<thead>
<tr>
<th>Operator</th>
<th>Implementation</th>
<th>Operator</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>unary</td>
<td></td>
<td>collection</td>
<td></td>
</tr>
<tr>
<td>Pattern Matching</td>
<td></td>
<td>Selection</td>
<td></td>
</tr>
<tr>
<td>Aggregation</td>
<td></td>
<td>Distinct</td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td></td>
<td>Sort by</td>
<td></td>
</tr>
<tr>
<td>Summarization</td>
<td></td>
<td>Top</td>
<td></td>
</tr>
<tr>
<td><strong>binary</strong></td>
<td></td>
<td>Union</td>
<td></td>
</tr>
<tr>
<td>Combination</td>
<td></td>
<td>Intersection</td>
<td></td>
</tr>
<tr>
<td>Overlap</td>
<td></td>
<td>Difference</td>
<td></td>
</tr>
<tr>
<td>Exclusion</td>
<td></td>
<td><strong>auxiliary</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Apply</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reduce</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Call</td>
<td></td>
</tr>
</tbody>
</table>
### Summary
- end-to-end framework for graph data management and analytics
- extended property graph model (EPGM) with powerful operators
- initial implementation running (HBase, MapReduce and Giraph)

### Roadmap
- WIP: workflow execution layer (Flink, Spark, ...)
- WIP: reference implementation for all operators
- optimized graph partitioning approaches
- graph-based data integration (DeDoop)
- Fitting data model

- Complex Analytics composed of Gradoop Operators

- Example: Cluster Characteristic Patterns in Business Process Executions
  - Quantify clusters by business measure (e.g., profitable and lossy)
  - Characteristic = frequent within one but not in other clusters
// generate base collection
btgs = iig.callForCollection( :BusinessTransactionGraphs , {} )
// generate base collection
btgs = iig.callForCollection( :BusinessTransactionGraphs , {} )

// aggregate profit
aggFunc = ( Graph g =>
    g.V.values("Revenue").sum() - g.V.values("Expense").sum()
)

CLUSTER-CHARACTERISTIC PATTERNS

<table>
<thead>
<tr>
<th>BTG 1</th>
<th>Total Rev.</th>
<th>Expenses</th>
<th>Net Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5,000</td>
<td>-3,000</td>
<td>2,000</td>
</tr>
<tr>
<td></td>
<td>9,000</td>
<td>-3,000</td>
<td>6,000</td>
</tr>
<tr>
<td></td>
<td>2,000</td>
<td>-1,500</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>4,000</td>
<td>-4,500</td>
<td>-500</td>
</tr>
<tr>
<td></td>
<td>5,000</td>
<td>-7,000</td>
<td>-2,000</td>
</tr>
<tr>
<td></td>
<td>10,000</td>
<td>-15,000</td>
<td>-5,000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BTG n</th>
<th>Total Rev.</th>
<th>Expenses</th>
<th>Net Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8,000</td>
<td>-4,000</td>
<td>4,000</td>
</tr>
</tbody>
</table>
// generate base collection
btgs = iig.callForCollection( :BusinessTransactionGraphs , {} )

// aggregate profit
aggFunc = ( Graph g =>
    g.V.values("Revenue").sum() - g.V.values("Expense").sum()
)
btgs = btgs.apply( Graph g =>
    g.aggregate("Profit", aggFunc )
)
// specific projection

vertexFunc = (Vertex v => new Vertex(
  (v["IsMasterData"] ? v["SourceID"] : v[":type"]),
  {"Result":v["Result"]}
)
)
edgeFunc = (Edge e => new Edge(
  (e[":type"]), {}
)
)
btgs = btgs.apply( Graph g =>
  g.project( vertexFunc , edgeFunc )
)
CLUSTER-CHARACTERISTIC PATTERNS

<table>
<thead>
<tr>
<th>BTG 1</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BTG 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTG 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTG 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTG 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTG 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTG n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Rev.</th>
<th>Expenses</th>
<th>Net Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,000</td>
<td>- 3,000</td>
<td>2,000</td>
</tr>
<tr>
<td>9,000</td>
<td>- 3,000</td>
<td>6,000</td>
</tr>
<tr>
<td>2,000</td>
<td>- 1,500</td>
<td>500</td>
</tr>
<tr>
<td>4,000</td>
<td>- 4,500</td>
<td>- 500</td>
</tr>
<tr>
<td>5,000</td>
<td>- 7,000</td>
<td>- 2,000</td>
</tr>
<tr>
<td>10,000</td>
<td>- 15,000</td>
<td>- 5,000</td>
</tr>
<tr>
<td>8,000</td>
<td>- 4,000</td>
<td>4,000</td>
</tr>
</tbody>
</table>
// select profit and loss clusters

profitBtgs = btgs.select( Graph g => g["Result"] >= 0 )
lossBtgs = btgs.difference(profitBtgs)
CLUSTER-CHARACTERISTIC PATTERNS

Ticket → Alice
processedBy

BTG 1
BTG 2
BTG 3
BTG 4
BTG 5
BTG 6
BTG n

Total Rev.  Expenses  Net Profit
5,000  - 3,000  2,000
9,000  - 3,000  6,000
2,000  - 1,500  500
4,000  - 4,500  - 500
5,000  - 7,000  - 2,000
10,000  - 15,000  -5,000
8,000  - 4,000  4,000
// select profit and loss clusters
profitBtgs = btgs.select( Graph g => g[“Result”] >= 0 )
lossBtgs = btgs.difference(profitBtgs)

profitFreqPats = profitBtgs.callForCollection(
 :FrequentSubgraphs , {“Threshold”:0.7}
)
lossFreqPats = lossBtgs.callForCollection(
 :FrequentSubgraphs , {“Threshold”:0.7}
)

// determine cluster characteristic patterns
trivialPats = profitFreqPats.intersect(lossFreqPats)
profitCharPatterns = profitFreqPats.difference(trivialPats)
lossCharPatterns = lossFreqPats.difference(trivialPats)
SUMMARY & ROADMAP: BIIIG

- **Summary**
  - Graph-based business intelligence framework
  - Graph transformations of business information systems
  - Concept of Business Transaction Graphs

- **Roadmap**
  - WIP: distributed frequent pattern mining
  - Summarization-based Graph OLAP
  - Meaningful result representation
  - Real-world evaluation


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

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

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

Jindal, A. et.al.: *Vertexica: your relational friend for graph analytics!*. PVLDB 7(13), 2014

Rudolf, M. et.al.: *The Graph Story of the SAP HANA Database*. BTW, 2013
Thank you!

www.gradoop.org
www.biiig.org