SCALABLE GRAPH ANALYTICS WITH GRADOOP AND BIIG

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

Life science

Engineering

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

1. HDFS Cluster
2. HBase Distributed Graph Store
3. Extended Property Graph Model
4. Operator Implementations
   - Data Integration
   - Graph Analytics
   - Representation
5. Workflow Execution
6. Workflow Declaration
   - Visual
   - GrALa DSL
7. Data Integration
8. Control flow
9. Data flow
1. Collection Operators
   - Select, Union, Intersect, Difference
   - Sort by, Top, Distinct

2. Graph Operators
   - Pattern Matching
   - Combination, Overlap, Exclusion
   - Aggregation, Summarization, Projection

3. Auxillary Operators
   - Apply, Reduce, Call
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)
SUMMARY & ROADMAP: GRADOOP

- **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
BIIIG OVERVIEW

1. Metadata Acquisition
2. Automated Graph Integration

Enterprise Service Bus

RDB - XML - WWW

Data Sources

Unified Metadata Graph

Expert
BUSINESS TRANSACTION GRAPH

- **Employee**
  - Name: Alice
  - processedBy
  - sentBy

- **Employee**
  - Name: Bob
  - createdBy

- **Purchase Invoice**
  - Expense: 2,000

- **Purchase Order**
  - serves
  - bills

- **Sales Quotation**
  - basedOn

- **Sales Order**
  - openedFor
  - serves

- **Sale Invoice**
  - Revenue: 5,000
  - processedBy

- **Employee**
  - Name: Dave
  - processedBy

- **Employee**
  - Name: Carol
  - createdBy

---

**CIT**

**ERP**
CLUSTER-CHARACTERISTIC PATTERNS

BTG 1

BTG 2

BTG 3

BTG 4

BTG 5

BTG 6

BTG n
// generate base collection
btgs = iig.callForCollection(:BusinessTransactionGraphs, {})
CLUSTER-CHARACTERISTIC PATTERNS

SalesInvoice
Revenue: 5,000

PurchaseInvoice
Expense: 2,000

PurchaseInvoice
Expense: 1,500

Employee
Name: Alice

Employee
Name: Bob

Employee
Name: Dave

Employee
Name: Carol

Ticket
Expense: 500

processedBy

processedBy

sentBy

createdBy

createdBy

basedOn

serves

serves

openedFor

bills

bills

bills

processedBy
// 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>BTG 2</td>
<td>9,000</td>
<td>- 3,000</td>
<td>6,000</td>
</tr>
<tr>
<td>BTG 3</td>
<td>2,000</td>
<td>- 1,500</td>
<td>500</td>
</tr>
<tr>
<td>BTG 4</td>
<td>4,000</td>
<td>- 4,500</td>
<td>- 500</td>
</tr>
<tr>
<td>BTG 5</td>
<td>5,000</td>
<td>- 7,000</td>
<td>- 2,000</td>
</tr>
<tr>
<td>BTG 6</td>
<td>10,000</td>
<td>- 15,000</td>
<td>-5,000</td>
</tr>
<tr>
<td>BTG n</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 )
)
CLUSTER-CHARACTERISTIC PATTERNS

Diagram showing relationships between entities such as Employee, SalesQuotation, SalesOrder, PurchaseOrder, and SalesInvoice. Each entity is connected with arrows indicating relationships like createdBy, sentBy, basedOn, serves, and processedBy. The diagram includes details such as names, relationships, and financial data like revenue and expense amounts.
// 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

#### BTG 1

#### BTG 2

#### BTG 3

#### BTG 4

#### BTG 5

#### BTG 6

#### BTG n

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

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

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