SCALABLE AND PRIVACY-PRESERVING DATA INTEGRATION
- PART 3 -

ERHARD RAHM
MARTIN JUNGHANNS, ANDRÉ PETERMANN
UNIVERSITY OF LEIPZIG

www.scads.de
- ScaDS Dresden/Leipzig

- Big Data Integration
  - Scalable entity resolution / link discovery
  - Large-scale schema/ontology matching
  - Holistic data integration

- Privacy-preserving record linkage
  - Encryption of sensitive information
  - PPRL with linkage unit
  - Secure multi-party approaches

- Graph-based data integration and analytics
  - Introduction
  - Graph-based data integration / business intelligence (BIIIG)
  - Hadoop-based graph analytics (GRADOOP)
"GRAPHS ARE EVERYWHERE"

Social science
- Facebook: ca. 1.3 billion users, ca. 340 friends per user
- Twitter: ca. 300 million users, ca. 500 million tweets per day

Engineering
- Internet: ca. 2.9 billion users

Life science
- Gene (human): 20,000-25,000 ca. 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)
“GRAPHS ARE EVERYWHERE”

Graph = (Users, Followers)
Graph = (Users, Friendships)
Graph = (Users $\cup$ Bands, Friendships $\cup$ Likes)
Graph = (Users \cup Bands, Friendships \cup Likes)
Support for heterogeneous vertices and edges

Good semantic expressiveness
  - Typed (labeled) vertices and edges
  - Properties for vertices and edges
  - Support for collections of graphs (not only 1 graph)

Flexibility: (semi-) structured data without strict schema

Analysis support
  - Powerful graph operators / queries

Easy usability

Property Graph Data Model (PGM) supports most requirements
Graphs are not only useful to represent and analyze existing networks / graph data

Support easy linking / integration of existing data sources
- utilized in LOD based on semantic web technology / RDF
- can be utilized for other data models such as PGM
- Enables graph-based analysis on relational databases and other data sources, e.g., for business intelligence

Business intelligence usually based on relational data warehouses
- enterprise data is integrated within dimensional schema
- analysis limited to predefined relationships
- no support for relationship-oriented data mining
- 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" (PVLDB 2014)
METADATA REPRESENTATION WITH UMG

Unified Metadata Graph

metadata

(1) Metadata Acquisition

RDB XML WWW

Data Sources

Class

Association

relationship_type
start_class
end_class
direction_indicator
start_reference_attribute
end_reference_attribute
(property_attributes)
schema_translation

data_source
class_name
class_type
identity_attribute
(property_attributes)
schema_translation

UNIVERSITÄT LEIPZIG
Schema translations:

Class **Employee**

```sql
SELECT number, name, dob, address, phone
FROM CIT.employees
```

Class **Ticket**

```sql
SELECT id, description
FROM CIT.tickets
```

Association **processedBy** (m:n)

```sql
SELECT ticket_id, empl_id
FROM CIT.ticket_employee
```
RESULTING UMG (UNIFIED METADATA GRAPH)

Ticket \(\text{sameAs}\) Employee

processedBy \(\text{openedFor}\) Employee

SalesQuotation \(\text{basedOn}\) SalesOrder

sentBy \(\text{processedBy}\) SalesInvoice

createdBy \(\text{createdBy}\) PurchaseOrder

bills \(\text{bills}\) PurchaseInvoice

CIT \(\rightarrow\) ERP
SCREENSHOT OF NEO4J IMPLEMENTATION
GRAPH DATA MANAGEMENT

- Relational database systems
  - can be used to implement a graph store
  - SQL alone insufficient for graph processing (need for graph operators and graph mining)

- RDF data management
  - flexible management of semantic web data
  - Data integration support (linking of entities / concepts)
  - SPARQL query processing
  - insufficient scalability of triple stores
  - insufficient support for graph mining

- Graph database system, e.g. Neo4J
  - use of *property graph data model*: vertices and edges have arbitrary set of properties (represented as key-value pairs)
  - focus on simple transactions and queries
  - insufficient scalability
  - insufficient support for graph mining
Parallel *graph processing* systems, e.g., Google Pregel, Apache Giraph
- in-memory storage of graphs in shared nothing clusters
- parallel processing of general graph algorithms, e.g., page rank, connected components, ...
- little support for semantically expressive graphs
- no end-to-end approach with data integration and persistent graph storage

Newer approaches (Apache Spark, Apache Flink):
- analysis workflow with graph operators
- no end-to-end solution with data integration and persistent graph storage
<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>
</tr>
</thead>
<tbody>
<tr>
<td>Data Model</td>
<td>Rich Graph Models (PGM)</td>
<td>Generic Graph Models</td>
<td>Generic Graph Models</td>
</tr>
<tr>
<td>Focus</td>
<td>transactional</td>
<td>analytic</td>
<td>analytic</td>
</tr>
<tr>
<td>Query Language</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Persistency</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Scalability</td>
<td>Vertical</td>
<td>Horizontal</td>
<td>Horizontal</td>
</tr>
<tr>
<td>Workflows</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Data Integration</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>
</tr>
<tr>
<td>Visualization</td>
<td>Yes</td>
<td>No</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.
- Hadoop-based framework for graph data management and analysis
- Graph storage in scalable distributed store, e.g., HBase
- Extended property graph data model (EPGM)
  - operators on graphs and sets of (sub) graphs
  - support for semantic graph queries and mining
- Leverages powerful components of Hadoop ecosystem
  - initially: Apache HBase, MapReduce, Apache Giraph
  - now: Apache HBase, Apache Flink, ...
- New functionality for graph-based processing workflows and graph mining

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

Extended Property Graph Model

Flink Operator Implementations
Data Integration
Graph Analytics
Representation

Flink Operator Execution

HBase Distributed Graph Store

HDFS/YARN Cluster

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
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
- Operators on both graphs and graph collections
EPGM – GRAPH REPRESENTATION

[0] Tag
name: Databases

[1] Tag
name: Graphs

[2] Tag
name: Hadoop

title: Graph Databases

title: Graph Processing

[5] Person
name: Alice
gender: f
city: Leipzig
age: 23

[6] Person
name: Bob
gender: m
city: Leipzig
age: 30

[7] Person
name: Carol
gender: f
city: Dresden
age: 30

[8] Person
name: Dave
gender: m
city: Dresden
age: 42

[9] Person
name: Eve
gender: f
city: Dresden
age: 35
speaks: en

[10] Person
name: Frank
gender: m
city: Berlin
age: 23
IP: 169.32.1.3
EPGM – OPERATORS AND ALGORITHMS

Operators

Unary

Aggregation

Pattern Matching

Transformation

Grouping

Subgraph

Call *

Binary

Combination

Overlap

Exclusion

Equality

Algorithms

Gelly Library

BTG Extraction

Adaptive Partitioning

Logical Graph

Graph Collection

Selection

Distinct

Sort

Limit

Apply *

Reduce *

Call *
COMBINATION

1: personGraph =
   db.G[0].combine(db.G[1]).combine(db.G[2])

[0] Tag
   name: Databases
   hasInterest 16

[1] Tag
   name: Graphs
   hasTag 19 13

[2] Tag
   name: Hadoop
   hasTag 20

   title: Graph Databases
   hasMember 12
   hasModerator 10
   hasInterest

   title: Graph Processing
   hasMember 13
   hasModerator
   hasInterest

[5] Person
   name: Alice
   gender: f
   city: Leipzig
   age: 23
   knows since: 2014
   knows since: 2014

[6] Person
   name: Bob
   gender: m
   city: Leipzig
   age: 30
   knows since: 2013
   knows since: 2013

[7] Person
   name: Carol
   gender: f
   city: Dresden
   age: 30
   knows since: 2014
   knows since: 2014

[8] Person
   name: Dave
   gender: m
   city: Dresden
   age: 42
   knows since: 2014
   knows since: 2015

[9] Person
   name: Eve
   gender: f
   city: Dresden
   age: 35
   speaks: en
   knows since: 2013
   knows since: 2015

[10] Person
    name: Frank
    gender: m
    city: Berlin
    age: 23
    IP: 169.32.1.3
    knows since: 2015
    knows since: 2015

[0] Community | interest: Databases | vertexCount: 3
[1] Community | interest: Hadoop | vertexCount: 3
1: personGraph = db.G[0].combine(db.G[1]).combine(db.G[2])
GROUPING (GRAPH SUMMARIZATION)

1: personGraph = db.G[0].combine(db.G[1]).combine(db.G[2])
2: vertexGroupingKeys = [:label, "city"]
3: edgeGroupingKeys = [:label]
4: vertexAggFunc = (superVertex, vertices => superVertex["count"] = |vertices|)
5: edgeAggFunc = (superEdge, edges => superEdge["count"] = |edges|)
6: sumGraph = personGraph.groupBy(vertexGroupingKeys, vertexAggFunc, edgeGroupingKeys, edgeAggFunc)
GROUPING (2)

1: personGraph = db.G[0].combine(db.G[1]).combine(db.G[2])
2: vertexGroupingKeys = [:label, “city”]
3: edgeGroupingKeys = [:label]
4: vertexAggFunc = (superVertex, vertices => superVertex[“count”] = |vertices|)
5: edgeAggFunc = (superEdge, edges => superEdge[“count”] = |edges|)
6: sumGraph = personGraph.groupBy(vertexGroupingKeys, vertexAggFunc, edgeGroupingKeys, edgeAggFunc)
GROUPING: TYPE LEVEL *(SCHEMA GRAPH)*

1: vertexGroupingKeys = [:label]
2: edgeGroupingKeys = [:label]
3: vertexAggFunc = (superVertex, vertices => superVertex["count"] = |vertices|)
4: edgeAggFunc = (superEdge, edges => superEdge["count"] = |edges|)
5: sumGraph = databaseGraph. **groupBy** (vertexGroupingKeys, vertexAggFunc, edgeGroupingKeys, edgeAggFunc)
1: resultColl = 
   db.G[0,1,2].select((g => g[“vertexCount”] > 3))
1: resultColl =
db.G[0,1,2].select((g => g["vertexCount"] > 3))
1. Large-scale graphs
   • Support for real-world graphs with millions of vertices and billions of edges

2. Graph partitioning
   • Efficient data distribution to balance load and minimize communication during computation

3. Data versioning
   • Enable time-based graph analytics on properties and graph structure

4. Fault tolerance
   • Prevent data loss in case of cluster failures
- Open Source implementation of Google BigTable
- **Distributed**, persistent, **sparse**, **multidimensional** sorted map based on HDFS
- Data distribution based on row key (i.e., horizontal **partitioning**)
- **Flexible** storage layout (handles only byte[], no types, no schema)
- **Fault tolerance** through data replication (HDFS)
- **Data versioning** on cell level

### HTable

<table>
<thead>
<tr>
<th>Row Key 1</th>
<th>Column Family 1</th>
<th>Column Family 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Column Identifier</td>
<td>C. Identifier</td>
</tr>
<tr>
<td></td>
<td>Versioned Value</td>
<td>V. Value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row Key 2</th>
<th>Column Family 1</th>
<th>Column Family 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C. Identifier</td>
<td>C. Identifier</td>
</tr>
<tr>
<td></td>
<td>V. Value</td>
<td>V. Value</td>
</tr>
</tbody>
</table>

Cell: `<rowkey>.<column_family>.<column_identifier>[.<version>]`
### Table  `vertices`:

<table>
<thead>
<tr>
<th>Type</th>
<th>Idx</th>
<th>Graphs</th>
<th>Meta</th>
<th>Properties</th>
<th>Out Edges</th>
<th>In Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0-0</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>[0]</td>
<td>k₁</td>
<td>〈t₁, a₁〉</td>
<td>(k₁, (t₁, a₁))</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>k₂</td>
<td>〈t₂, a₂〉</td>
<td>[k₁, (t₁, a₁)]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>0-1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>[0,1]</td>
<td>k₁</td>
<td>〈t₂, a₁〉</td>
<td>(k₁, (t₂, a₁))</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>k₂</td>
<td>〈t₂, a₂〉</td>
<td>[k₁, (t₂, a₂)]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>0-2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>[1]</td>
<td>k₁</td>
<td>〈t₂, a₂〉</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
VERTEX TABLE

Table \(\text{\textquoteleft}\text{vertices}\text{\textquoteright}\):

<table>
<thead>
<tr>
<th></th>
<th>meta</th>
<th>properties</th>
<th>out edges</th>
<th>in edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\theta \rightarrow)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>idx</td>
<td>graphs</td>
<td>(k_1)</td>
<td>(k_2)</td>
</tr>
<tr>
<td>(A)</td>
<td>1</td>
<td>([0])</td>
<td>(\langle t_1, a_1 \rangle)</td>
<td>(\langle t_2, a_2 \rangle)</td>
</tr>
<tr>
<td>(\theta - 1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>idx</td>
<td>graphs</td>
<td>(k_1)</td>
<td>(k_2)</td>
</tr>
<tr>
<td>(B)</td>
<td>1</td>
<td>([0,1])</td>
<td>(\langle t_2, a_3 \rangle)</td>
<td>(\langle t_2, a_2 \rangle)</td>
</tr>
<tr>
<td>(\theta - 2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>idx</td>
<td>graphs</td>
<td>(k_1)</td>
<td></td>
</tr>
<tr>
<td>(A)</td>
<td>2</td>
<td>([1])</td>
<td>(\langle t_2, a_2 \rangle)</td>
<td></td>
</tr>
</tbody>
</table>
### Vertex Table

<table>
<thead>
<tr>
<th>Type</th>
<th>Idx</th>
<th>Graphs</th>
<th>$k_1$</th>
<th>$k_2$</th>
<th>Out Edges</th>
<th>In Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0-0$</td>
<td>$A 1$</td>
<td>${t_1, a_1}$</td>
<td>${t_2, a_2}$</td>
<td>${k_1, (t_1, a_1)}$</td>
<td>${(k_1, (t_1, a_1)), (k_2, (t_2, a_2))}$</td>
<td></td>
</tr>
<tr>
<td>$0-1$</td>
<td>$B 1$</td>
<td>${t_2, a_1}$</td>
<td>${t_2, a_2}$</td>
<td>${b, 0 - 0,0}$</td>
<td></td>
<td>${(a, 0 - 0,0), (a, 0 - 2,0)}$</td>
</tr>
<tr>
<td>$0-2$</td>
<td>$A 2$</td>
<td>${t_2, a_2}$</td>
<td></td>
<td>${a, 0 - 1,0}$</td>
<td>${b, 0 - 2,1}$</td>
<td>${b, 0 - 2,1}$</td>
</tr>
</tbody>
</table>
Table `vertices`

<table>
<thead>
<tr>
<th></th>
<th>meta</th>
<th>properties</th>
<th>out edges</th>
<th>in edges</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>type</td>
<td>idx</td>
<td>graphs</td>
<td>k₁</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td></td>
<td>t₁, a₁</td>
</tr>
<tr>
<td></td>
<td>meta</td>
<td>idx</td>
<td>graphs</td>
<td>k₁</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0,1</td>
<td></td>
<td>t₂, a₃</td>
</tr>
<tr>
<td></td>
<td>meta</td>
<td>idx</td>
<td>graphs</td>
<td>k₁</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>1</td>
<td></td>
<td>t₂, a₂</td>
</tr>
</tbody>
</table>
### Vertex Table

<table>
<thead>
<tr>
<th>Type</th>
<th>Idx</th>
<th>Graphs</th>
<th>Properties</th>
<th>Out Edges</th>
<th>In Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-0</td>
<td></td>
<td></td>
<td>$k_1$</td>
<td>$a_1$</td>
<td>$(a, 0 - 1, 0)$</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>[0]</td>
<td>$\langle t_1, a_1 \rangle$</td>
<td>$\langle t_2, a_2 \rangle$</td>
<td>$[{ k_1, (t_1, a_1) }]$</td>
</tr>
<tr>
<td>0-1</td>
<td></td>
<td></td>
<td>$k_1$</td>
<td>$k_2$</td>
<td>$(b, 0 - 0, 0)$</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>[0,1]</td>
<td>$\langle t_2, a_3 \rangle$</td>
<td>$\langle t_2, a_2 \rangle$</td>
<td>$[{ k_1, (t_1, a_1) }]$</td>
</tr>
<tr>
<td>0-2</td>
<td></td>
<td></td>
<td>$k_1$</td>
<td>$a_1$</td>
<td>$(a, 0 - 1, 0)$</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>[1]</td>
<td>$\langle t_2, a_2 \rangle$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. Business Intelligence
   • Top Revenue Subgraph
   • Find the common subgraph of the top 100 revenue business transaction graphs

2. Social Network Analysis
   • “Summarized Communities”
   • Find communities by label propagation
   • Filter communities by number of users
   • Summarize vertices per community and edges between community members
// compute logical graphs
1: btgs = db.callForCollection( :BusinessTransactionGraphs , {} )

// define and apply aggregate function (number of invoices per graph)
2: aggFuncInvoiceCount = ( Graph g =>
   |g.V.filter( Vertex v => v[:type] == "Invoice")|)
3: btgs = btgs.apply(
   Graph g => g.aggregate( "invoiceCount",aggFuncInvoiceCount ) )

// select logical graphs with at least one invoice
4: invBtgs = btgs.select( 
   Graph g => g["invoiceCount"] > 0)

// define and apply aggregate function (revenue per graph)
5: aggFuncRevenue = ( Graph g =>
   g.V.values("revenue").sum())

6: invBtgs = invBtgs.apply( 
   Graph g => g.aggregate( "revenue",aggFuncRevenue ) )

// sort graphs by revenue and return top 100
7: topBtgs = invBtgs.sortBy( "revenue", :desc ).top( 100 )

// compute overlap to find master data objects (e.g., Employees)
8: topBtgOverlap = invBtgs.reduce( 
   Graph g, Graph h => g.overlap(h))
SUMMARIZED COMMUNITIES

socialNetwork

.subgraph(
    (v => v.label == 'Person'),
    (e => e.label == 'knows'))

.transform(
    (gIn, gOut => gOut = gIn),
    (vIn, vOut => {
        vOut.label = vIn.label,
        vOut['city'] = vIn['city'],
        vOut['gender'] = vIn['gender'],
        vOut['key'] = vIn['birthday']},
    (eIn, eOut) => eOut.label = eIn.label)

.callForCollection(:LabelPropagation, ['key', 4])

.apply(g =>
    g.aggregate('vertexCount', (h => |h.V|)),
    .select(g => g['vertexCount'] > 50000)
    .reduce(g, h => g.combine(h))
    .groupBy(
        ['city', 'gender'],
        (superVertex, vertices =>
            superVertex['count'] = |vertices|),
        [], (superEdge, edges =>
            superEdge['count'] = |edges|)
    .aggregate('vCount', (g => |g.V|))
    .aggregate('eCount', (g => |g.E|))
### 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
<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>

### Diagram:

- **Graphalytics.100**: Linear speedup
- **Graphalytics.10**: Linear speedup

### System 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**
### Dataset 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>

#### Datasets
- 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-based data integration
- centralized „linked data“ using PGM rather than RDF
- data/metadata extraction and transformation into graphs
- linking / matching + fusion

Big Graph Analytics
- high potential even for business intelligence (BIIIG)
- Hadoop-based graph processing frameworks based on generic graphs
- Spark/Flink: batch-oriented workflows (rather than OLAP)
- Graph collections not generally supported
GraDoop

- 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) and support for graph collections
- leverages Hadoop ecosystem
  - Apache HBase for permanent graph storage
  - Apache Flink to implement operators
- ongoing implementation
OUTLOOK / CHALLENGES

- **Graph-based data integration**
  - unified approach for knowledge graphs and regular data graphs
  - evaluate/improve scalability and data quality

- **Graph analytics**
  - automatic optimization of analysis workflows
  - optimized graph partitioning approaches
  - load balancing
  - interactive graph analytics
  - visualization of graphs and analysis results
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)
- 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
- Z. Wang et. al. *Pagrol: Parallel Graph OLAP over Large-scale Attributed Graphs*. ICDE, 2014
- P. Zhao et. al. *Graph Cube: On Warehousing and OLAP Multidimensional Networks*. Sigmod, 2011