GRADOOP: Scalable Graph Analytics with Apache Flink

Martin Junghanns
@kc1s
Leipzig University
About the speaker and the team

Martin, PhD Student

André, PhD Student

Prof. Dr. Erhard Rahm
Database Chair

Niklas, M.Sc. Student

Kevin, M.Sc. Student
Motivation
"Graphs are everywhere"

Graph = (Vertices, Edges)
“Graphs are everywhere”

Graph = (Users, Followers)
“Graphs are everywhere”

Graph = (Users, Friendships)
“Graphs are heterogeneous”

\[ \text{Graph} = (\text{Users } \cup \text{ Bands}, \text{Friendships } \cup \text{ Likes}) \]
“Graphs can be analyzed”

\[ \text{Graph} = (\text{Users} \cup \text{Bands}, \text{Friendships} \cup \text{Likes}) \]
“Graphs can be analyzed”

Graph = (Users ∪ Bands, Friendships ∪ Likes)
“Graphs can be analyzed”

Assuming a social network
“Graphs can be analyzed”

Assuming a social network
1. Determine subgraph
“Graphs can be analyzed”

Assuming a social network
1. Determine subgraph
2. Find communities
“Graphs can be analyzed”

Assuming a social network
1. Determine subgraph
2. Find communities
3. Filter communities
“Graphs can be analyzed“

Assuming a social network
1. Determine subgraph
2. Find communities
3. Filter communities
4. Find common subgraph
“Graph data models must be expressive”

Assuming a social network

1. Determine subgraph
   - Heterogeneous data
2. Find communities
   - Apply graph transformation
3. Filter communities
   - Handle collections of graphs
4. Find common subgraph
   - Aggregation, Selection
5. Apply dedicated algorithms
„And let’s not forget ...“
“...Graphs are large”
A framework and research platform for **efficient**, **distributed** and domain independent graph data management and **analytics**.
High Level Architecture

Graph Analytical DSL

Extended Property Graph Model

Flink Operator Implementation

HBase Distributed Graph Store

- Java
- 25K (33K) LOC
- GPLv3
Extended Property Graph Model (EPGM)
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

Graph Collection

- Selection
- Distinct
- Sort
- Limit
- Apply *
- Reduce *
- Call *

Logical Graph

- Limit
- Selection
- Distinct
- Sort
- Limit
- Apply *
- Reduce *
- Call *
Combination

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

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

Unary
- Aggregation
- Pattern Matching
- Transformation
- Grouping
- Subgraph
  - Call *

Binary
- Combination
- Overlap
- Exclusion
- Equality

Algorithms

Gelly Library
- BTG Extraction
- Adaptive Partitioning

Graph Collection
- Selection
- Distinct
- Sort
- Limit
- Apply *
- Reduce *
- Call *

Adaptive Partitioning

* auxiliary
Combination + Grouping

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)
Combination + Grouping

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)
Combination + Grouping + Aggregation

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)
7: aggFunc = (g => |g.E|)
8: aggGraph = sumGraph.aggregate("edgeCount", aggFunc)
Combination + Grouping + Aggregation

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)
7: aggFunc = (g => |g.E|)
8: aggGraph = sumGraph.aggregate(“edgeCount”, aggFunc)
### 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
- Frequent Subgraphs

#### Call *
- Apply *
- Reduce *
- Call *
Selection

1: resultColl = db.G[0,1,2].select((g => g["vertexCount"] > 3))
1: resultColl = db.G[0,1,2].select((g => g["vertexCount"] > 3))
Apache Flink
Apache Flink

„Streaming Dataflow Engine that provides
• data distribution,
• communication,
• and fault tolerance
for distributed computations over data streams.“
Apache Flink – DataSet API

- **DataSet** := Distributed Collection of Data
- **Transformation** := Operation applied on DataSet
- **Flink Program** := Composition of Transformations
Apache Flink – DataSet Transformations

- aggregate
- coGroup
- cross
- distinct
- filter
- first-N
- flatMap
- groupBy
- join
- leftOuterJoin
- rightOuterJoin
- fullOuterJoin
- map
- mapPartition
- project
- reduce
- reduceGroup
- union
- iterate
- iterateDelta
The „Hello World“ of Big Data – Word Count

1: ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
2: 
3: DataSet<String> text = env.fromElements( // or env.readTextFile("hdfs://...")
4:     „He who controls the past controls the future.“, 
5:     „He who controls the present controls the past.“);
6: 
7: DataSet<Tuple2<String, Integer>> wordCounts = text
8:     .flatMap(new LineSplitter()) // splits the line and outputs (word, 1) tuples
9:     .groupBy(0)
10:     .sum(1);
11: 
12: wordCounts.print(); // trigger execution

„He who controls the past controls the future.“
„He who controls the present controls the past.“
EPGM on Apache Flink
EPGM on Apache Flink – User facing API

**GraphBase**
- graphHeads : DataSet<EPGMGraphHead>
- vertices : DataSet<EPGMVertex>
- edges : DataSet<EPGMEdge>

- getVertices() : DataSet<EPGMVertex>
- getEdges() : DataSet<EPGMEdge>

// ...

**EPGMDatabase**
- fromCollections(...) : EPGMDatabase
- fromJSONFile(...) : EPGMDatabase
- fromHBase(...) : EPGMDatabase

- writeAsJSON(...) : void
- writeToHBase(...) : void
- getDatabaseGraph() : LogicalGraph

// ...

**LogicalGraph**
- fromCollections(...) : LogicalGraph
- fromDataSets(...) : LogicalGraph
- fromGellyGraph(...) : LogicalGraph

- getGraphHead() : DataSet<EPGMGraphHead>
- toGellyGraph() : Graph
- combine(...) : LogicalGraph
- intersect(...) : LogicalGraph
- groupBy(...) : LogicalGraph
- match(...) : GraphCollection

// ...

**GraphCollection**
- fromCollections(...) : GraphCollection
- fromDataSets(...) : GraphCollection

- getGraphHeads() : DataSet<EPGMGraphHead>
- getGraph(...) : LogicalGraph
- getGraphs(...) : GraphCollection
- select(...) : GraphCollection
- union(...) : GraphCollection
- distinct(...) : GraphCollection
- sortBy(...) : GraphCollection

// ...
EPGM on Apache Flink – DataSets

EPGMGraphHead

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

EPGMVertex

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

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>

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>

Property := (String, PropertyValue)

PropertyValue := byte[]

GradoopIdSet := Set<GradoopId>
EPGM on Apache Flink – Exclusion

```java
// input: firstGraph (G[0]), secondGraph (G[2])
1: DataSet<GradoopId> graphId = secondGraph.getGraphHead()
2: .map(new Id<G>())
3:
4: DataSet<V> newVertices = firstGraph.getVertices()
5: .filter(new NotInGraphBroadCast<V>())
6: .withBroadcastSet(graphId, GRAPH_ID)
7:
8: DataSet<E> newEdges = firstGraph.getEdges()
9: .filter(new NotInGraphBroadCast<E>())
10: .withBroadcastSet(graphId, GRAPH_ID)
11: .join(newVertices)
12: .where(new SourceId<E>().equalTo(new Id<V>())
13: .with(new LeftSide<E, V>())
14: .join(newVertices)
15: .where(new TargetId<E>().equalTo(new Id<V>())
16: .with(new LeftSide<E, V>());
```

db.G[0].exclude(db.G[2])
graphId =
  secondGraph.getGraphHead()

  .map(new Id<G>());

newVertices =
  firstGraph.getVertices()

  .filter(new NotInGraphBroadcast<V>())
  .withBroadcastSet(graphId, GRAPH_ID);
newEdges =
    firstGraph.getEdges()

    .filter(new NotInGraphBroadcast<E>())
    .withBroadcastSet(graphId, GRAPH_ID)

    .join(newVertices)
    .where(new SourceId<E>().equalTo(new Id<V>())
    .with(new LeftSide<E, V>())

    .join(newVertices)
    .where(new TargetId<E>().equalTo(new Id<V>())
    .with(new LeftSide<E, V>())

<table>
<thead>
<tr>
<th>Id</th>
<th>Label</th>
<th>SourceId</th>
<th>TargetId</th>
<th>Properties</th>
<th>Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>knows</td>
<td>5</td>
<td>6</td>
<td>since: 2014</td>
<td>[0, 2]</td>
</tr>
<tr>
<td>1</td>
<td>knows</td>
<td>6</td>
<td>5</td>
<td>since: 2014</td>
<td>[0, 2]</td>
</tr>
<tr>
<td>6</td>
<td>knows</td>
<td>9</td>
<td>5</td>
<td>since: 2013</td>
<td>[0]</td>
</tr>
<tr>
<td>7</td>
<td>knows</td>
<td>9</td>
<td>6</td>
<td>since: 2015</td>
<td>[0]</td>
</tr>
</tbody>
</table>
Social Network Example
LDBC Social Network Data

http://ldbcouncil.org/
LDBC Social Network Data

```
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|))
```
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
  - slots (per worker) 12
  - jobmanager.heap.mb 2048
  - taskmanager.heap.mb 40960
### 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
  - slots (per worker) 12
  - jobmanager.heap.mb 2048
  - taskmanager.heap.mb 40960
Benchmark Results

Dataset 
<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
  - slots (per worker) 12
  - jobmanager.heap.mb 2048
  - taskmanager.heap.mb 40960
Current State & Future Work
Current State

- **0.0.1 First Prototype (May 2015)**
  - Hadoop MapReduce and Giraph for operator implementations
  - Too much complexity
  - Performance loss through serialization in HDFS/HBase
- **0.0.2 Using Flink as execution layer (June 2015)**
  - Basic operators
- **0.1 December 2015**
  - System-side identifiers (UUID)
  - Improved property handling
  - More operator implementations (e.g., Equality, Bool operators)
  - Code refactoring
- **0.2-SNAPSHOT**
  - Graph Pattern Matching
  - Frequent Subgraph Mining
  - Memory optimization (96-bit ID, Dictionary Encoding, ...)
  - Tuple Implementation
Contributions to Flink

- FLINK-2411 Add basic graph summarization algorithm
- FLINK-2590 DataSetUtils.zipWithUniqueID creates duplicate IDs
- FLINK-2905 Add intersect method to Graph class
- FLINK-2910 Combine tests for binary graph operators
- FLINK-2941 Implement a neo4j - Flink/Gelly connector
- FLINK-2981 Update README for building docs
- FLINK-3064 Missing size check in GroupReduceOperatorBase leads to NPE
- FLINK-3118 Check if MessageFunction implements ResultTypeQueryable
- FLINK-3122 Generalize value type in LabelPropagation
- FLINK-3272 Generalize vertex value type in ConnectedComponents

- Flink Forward (October 2015)
- Meetup Big Data Usergroup Saxony (December 2015)
Contributions welcome!

- **Code**
  - Operator implementations
  - Performance Tuning
  - Extend HBase Storage

- **People**
  - Bachelor / Master Thesis
  - Open PhD positions in Leipzig, Germany

- **Data! and Use Cases**
  - We are researchers, we assume ...
Thank you!

www.gradoop.com

https://flink.apache.org
http://ldbcouncil.org/

http://dbs.uni-leipzig.de/file/GradoopTR.pdf
http://dbs.uni-leipzig.de/file/biiig-vldb2014.pdf

https://github.com/dbs-leipzig/gradoop
https://github.com/s1ck/gdl
https://github.com/s1ck/ldbc-flink-import
https://github.com/s1ck/flink-neo4j