Graph Databases and more

Wei BAI, Lie YAN

April 6, 2016
Motivation

- Applications where data are modeled as graphs, eg.
  - transport networks
  - semantic web, social network
  - cheminformatics, bioinformatics
Graph Databases and Graph Analytic Systems

Some people regard the latter as graph databases as well.

<table>
<thead>
<tr>
<th>Graph database</th>
<th>OLTP</th>
<th>Queries</th>
<th>Operational data</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph analytic system</td>
<td>OLAP</td>
<td>Analytics</td>
<td>Consolidation data</td>
<td>Offline</td>
</tr>
</tbody>
</table>
Outline

• Graph Database ... Models
  - Data models
  - Three components of graph data model
    - Data structures
    - Integrity constraints
    - Query and manipulation languages
• Graph Analytic Systems
Data Models
# Data Models

<table>
<thead>
<tr>
<th>Data model</th>
<th>Base data structure</th>
<th>Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network</td>
<td>pointers + records</td>
<td>/</td>
</tr>
<tr>
<td>Relational</td>
<td>relations</td>
<td>attributes</td>
</tr>
<tr>
<td>Semantic</td>
<td>graph</td>
<td>standard abstractions</td>
</tr>
<tr>
<td>Object-Oriented</td>
<td>objects</td>
<td>complex objects</td>
</tr>
<tr>
<td>Semistructured</td>
<td>tree</td>
<td>composition relations</td>
</tr>
<tr>
<td>Graph</td>
<td>graph</td>
<td>connectivity</td>
</tr>
</tbody>
</table>

[AG08]
Graph Data Models

[AG08]
Current Graph Data Models

- AllegroGraph
- SparkSee (aka. DEX)
- HypergraphDB
- InfiniteGraph
- Neo4j
- Sones (inactive)
- ...

[Logos of AllegroGraph, HypergraphDB, Neo4j, InfiniteGraph, Sones]
Data Models

- A **data model** is a collection of conceptual tools used to model representations of real-world entities and the relationships among them. [Silberschatz et al. 1996]

- Three components: data structure types, operators, and integrity rules
Graph Data Models

Characterized as follows:

- **Data and/or the schema** are represented by graphs, hypergraphs, or other generalized variants.

- **Data manipulation** is expressed by graph transformation, or operations described with graph-theoretic objects and properties, eg. edges, paths, neighborhoods, connected components, etc.
Outline

• Graph Database ... Models
  - Data models
    - Three components of graph data model
      - Data structures
      - Integrity constraints
      - Query and manipulation languages
  • Graph Analytic Systems
Data Structures

- A **schema** graph defines
  - entity types (as nodes)
  - primitive entities (as nodes, not shown here)
  - relation types (as edges)

- An **instance** graph contains
  - concrete entities (entity nodes labeled with entity type name or object identifier)
  - primitive values (value nodes labeled with value)
  - relations (edges labeled with relation names)
Data Structures

- simple graph
- nested graph (hypernodes)
- hypergraph

- directed
- undirected

- labeled node
- unlabeled node

- labeled edge
- unlabeled edge

- attributed edge
- unattributed edge

- attributed node
- unattributed node
Example

<table>
<thead>
<tr>
<th>NAME</th>
<th>LASTNAME</th>
<th>PERSON</th>
<th>PARENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>George</td>
<td>Jones</td>
<td>Julia</td>
<td>George</td>
</tr>
<tr>
<td>Ana</td>
<td>Stone</td>
<td>Julia</td>
<td>Ana</td>
</tr>
<tr>
<td>Julia</td>
<td>Jones</td>
<td>David</td>
<td>James</td>
</tr>
<tr>
<td>James</td>
<td>Deville</td>
<td>David</td>
<td>Julia</td>
</tr>
<tr>
<td>David</td>
<td>Deville</td>
<td>Mary</td>
<td>James</td>
</tr>
<tr>
<td>Mary</td>
<td>Deville</td>
<td>Mary</td>
<td>Julia</td>
</tr>
</tbody>
</table>
Logical Data Model

<table>
<thead>
<tr>
<th>NAME</th>
<th>LASTNAME</th>
<th>PERSON</th>
<th>PARENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>George</td>
<td>Jones</td>
<td>Julia</td>
<td>George</td>
</tr>
<tr>
<td>Ana</td>
<td>Stone</td>
<td>Julia</td>
<td>Ana</td>
</tr>
<tr>
<td>Julia</td>
<td>Jones</td>
<td>David</td>
<td>James</td>
</tr>
<tr>
<td>James</td>
<td>Deville</td>
<td>David</td>
<td>Julia</td>
</tr>
<tr>
<td>David</td>
<td>Deville</td>
<td>Mary</td>
<td>James</td>
</tr>
<tr>
<td>Mary</td>
<td>Deville</td>
<td>Mary</td>
<td>Julia</td>
</tr>
</tbody>
</table>

![Diagram of the Logical Data Model](image)

**Schema**

- **PP**: Person-Parent
- **NL**: Name-Lastname

**Instance**

<table>
<thead>
<tr>
<th>l</th>
<th>val (l)</th>
<th>l</th>
<th>val (l)</th>
<th>l</th>
<th>val (l)</th>
<th>l</th>
<th>val (l)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>George</td>
<td>7</td>
<td>Jones</td>
<td>10</td>
<td>(1,7)</td>
<td>16</td>
<td>(12,10)</td>
</tr>
<tr>
<td>2</td>
<td>Ana</td>
<td>8</td>
<td>Stone</td>
<td>11</td>
<td>(2,8)</td>
<td>17</td>
<td>(12,11)</td>
</tr>
<tr>
<td>3</td>
<td>Julia</td>
<td>9</td>
<td>Deville</td>
<td>12</td>
<td>(3,7)</td>
<td>18</td>
<td>(14,13)</td>
</tr>
<tr>
<td>4</td>
<td>James</td>
<td>13</td>
<td></td>
<td>19</td>
<td>(4,9)</td>
<td>14</td>
<td>(14,12)</td>
</tr>
<tr>
<td>5</td>
<td>David</td>
<td>14</td>
<td></td>
<td>20</td>
<td>(5,9)</td>
<td>15</td>
<td>(15,13)</td>
</tr>
<tr>
<td>6</td>
<td>Mary</td>
<td>15</td>
<td></td>
<td>21</td>
<td>(6,9)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hypernode Model

<table>
<thead>
<tr>
<th>NAME</th>
<th>LASTNAME</th>
<th>PERSON</th>
<th>PARENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>George</td>
<td>Jones</td>
<td>Julia</td>
<td>George</td>
</tr>
<tr>
<td>Ana</td>
<td>Stone</td>
<td>Julia</td>
<td>Ana</td>
</tr>
<tr>
<td>Julia</td>
<td>Jones</td>
<td>David</td>
<td>James</td>
</tr>
<tr>
<td>James</td>
<td>Deville</td>
<td>David</td>
<td>Julia</td>
</tr>
<tr>
<td>David</td>
<td>Deville</td>
<td>Mary</td>
<td>James</td>
</tr>
<tr>
<td>Mary</td>
<td>Deville</td>
<td>Mary</td>
<td>Julia</td>
</tr>
</tbody>
</table>

![Graph showing relationships between family members](image-url)
GROOVY

<table>
<thead>
<tr>
<th>NAME</th>
<th>LASTNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>George</td>
<td>Jones</td>
</tr>
<tr>
<td>Ana</td>
<td>Stone</td>
</tr>
<tr>
<td>Julia</td>
<td>Jones</td>
</tr>
<tr>
<td>James</td>
<td>Deville</td>
</tr>
<tr>
<td>David</td>
<td>Deville</td>
</tr>
<tr>
<td>Mary</td>
<td>Deville</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PERSON</th>
<th>PARENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Julia</td>
<td>George</td>
</tr>
<tr>
<td>Julia</td>
<td>Ana</td>
</tr>
<tr>
<td>David</td>
<td>James</td>
</tr>
<tr>
<td>David</td>
<td>Julia</td>
</tr>
<tr>
<td>Mary</td>
<td>James</td>
</tr>
</tbody>
</table>

Schema

Instance

CHILD–PARENT

PERSON

1

NAME Lastname

George Jones

PARENTS

2

NAME Lastname

Ana Stone

PARENTS

3

NAME Lastname

Julia Jones

VAL(1) VAL(2)

PARENTS

4

NAME Lastname

James Deville

PARENTS

5

NAME Lastname

David Deville

VAL(3) VAL(4)

PARENTS

6

NAME Lastname

Mary Deville

VAL(3) VAL(4)

PARENTS
Simatic-XT

<table>
<thead>
<tr>
<th>NAME</th>
<th>LASTNAME</th>
<th>PERSON</th>
<th>PARENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>George</td>
<td>Jones</td>
<td>Julia</td>
<td>George</td>
</tr>
<tr>
<td>Ana</td>
<td>Stone</td>
<td>Julia</td>
<td>Ana</td>
</tr>
<tr>
<td>Julia</td>
<td>Jones</td>
<td>David</td>
<td>James</td>
</tr>
<tr>
<td>James</td>
<td>Deville</td>
<td>David</td>
<td>Julia</td>
</tr>
<tr>
<td>David</td>
<td>Deville</td>
<td>Mary</td>
<td>James</td>
</tr>
<tr>
<td>Mary</td>
<td>Deville</td>
<td>Mary</td>
<td>Julia</td>
</tr>
</tbody>
</table>

Abstraction Level 1

Abstraction Level 2
GGL

<table>
<thead>
<tr>
<th>NAME</th>
<th>LASTNAME</th>
<th>PERSON</th>
<th>PARENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>George</td>
<td>Jones</td>
<td>Julia</td>
<td>George</td>
</tr>
<tr>
<td>Ana</td>
<td>Stone</td>
<td>Julia</td>
<td>Ana</td>
</tr>
<tr>
<td>Julia</td>
<td>Jones</td>
<td>David</td>
<td>James</td>
</tr>
<tr>
<td>James</td>
<td>Deville</td>
<td>David</td>
<td>Julia</td>
</tr>
<tr>
<td>David</td>
<td>Deville</td>
<td>Mary</td>
<td>James</td>
</tr>
<tr>
<td>Mary</td>
<td>Deville</td>
<td>Mary</td>
<td>Julia</td>
</tr>
</tbody>
</table>

Instance

Person1

name
George

lastname
Jones

Person2

name
Ana

lastname
Stone

Person3

name
Julia

lastname
Jones

Person4

name
James

lastname
Deville

Person5

name
David

lastname
Deville

Person6

name
Mary

lastname
Deville
GOOD

<table>
<thead>
<tr>
<th>NAME</th>
<th>LASTNAME</th>
<th>NAME</th>
<th>LASTNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>George</td>
<td>Jones</td>
<td>Julia</td>
<td>George</td>
</tr>
<tr>
<td>Ana</td>
<td>Stone</td>
<td>Julia</td>
<td>Ana</td>
</tr>
<tr>
<td>Julia</td>
<td>Jones</td>
<td>David</td>
<td>James</td>
</tr>
<tr>
<td>James</td>
<td>Deville</td>
<td>David</td>
<td>Julia</td>
</tr>
<tr>
<td>David</td>
<td>Deville</td>
<td>Mary</td>
<td>James</td>
</tr>
<tr>
<td>Mary</td>
<td>Deville</td>
<td>Mary</td>
<td>Julia</td>
</tr>
</tbody>
</table>

**Schema**

```
CP  
   ↓ parent
   Pe
       ↓ n
       Pe
          ↓ n
          N
             ↓ ln
             L
                ↓ CP
```

**Instance**

```
CP  
   ↓ parent
   Pe
       ↓ n
       Pe
          ↓ n
          N
             ↓ ln
             L
                ↓ CP
```

```
CP  
   ↓ child
   Pe
       ↓ n
       Pe
          ↓ n
          N
             ↓ ln
             L
                ↓ CP
```

```
CP  
   ↓ child
   Pe
       ↓ n
       Pe
          ↓ n
          N
             ↓ ln
             L
                ↓ CP
```

```
CP  
   ↓ child
   Pe
       ↓ n
       Pe
          ↓ n
          N
             ↓ ln
             L
                ↓ CP
```

```
CP  
   ↓ child
   Pe
       ↓ n
       Pe
          ↓ n
          N
             ↓ ln
             L
                ↓ CP
```

```
CP  
   ↓ child
   Pe
       ↓ n
       Pe
          ↓ n
          N
             ↓ ln
             L
                ↓ CP
```

```
CP  
   ↓ child
   Pe
       ↓ n
       Pe
          ↓ n
          N
             ↓ ln
             L
                ↓ CP
```

n = name    ln = lastname

George Jones   Ana Stone
James Deville  Julia Jones
David Deville  Mary Deville
GMOD

<table>
<thead>
<tr>
<th>NAME</th>
<th>LASTNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>George</td>
<td>Jones</td>
</tr>
<tr>
<td>Ana</td>
<td>Stone</td>
</tr>
<tr>
<td>Julia</td>
<td>Jones</td>
</tr>
<tr>
<td>James</td>
<td>Deville</td>
</tr>
<tr>
<td>David</td>
<td>Deville</td>
</tr>
<tr>
<td>Mary</td>
<td>Deville</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PERSON</th>
<th>PARENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Julia</td>
<td>George</td>
</tr>
<tr>
<td>Julia</td>
<td>Ana</td>
</tr>
<tr>
<td>David</td>
<td>James</td>
</tr>
<tr>
<td>David</td>
<td>Julia</td>
</tr>
<tr>
<td>Mary</td>
<td>James</td>
</tr>
<tr>
<td>Mary</td>
<td>Julia</td>
</tr>
</tbody>
</table>

**Schema**

- **Person**
  - name
  - lastname

**Instance**

- **Person**
  - name
  - lastname
  - parent
- **George Jones**
- **Ana Stone**
- **James Deville**
- **Julia Jones**
- **David Deville**
- **Mary Deville**
PaMaL

<table>
<thead>
<tr>
<th>NAME</th>
<th>LASTNAME</th>
<th>PERSON</th>
<th>PARENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>George</td>
<td>Jones</td>
<td>Julia</td>
<td>George</td>
</tr>
<tr>
<td>Ana</td>
<td>Stone</td>
<td>Julia</td>
<td>Ana</td>
</tr>
<tr>
<td>Julia</td>
<td>Jones</td>
<td>David</td>
<td>James</td>
</tr>
<tr>
<td>James</td>
<td>Deville</td>
<td>David</td>
<td>Julia</td>
</tr>
<tr>
<td>David</td>
<td>Deville</td>
<td>Mary</td>
<td>James</td>
</tr>
<tr>
<td>Mary</td>
<td>Deville</td>
<td>Mary</td>
<td>Julia</td>
</tr>
</tbody>
</table>

**Schema**

- Person
  - typ
  - parent
  - name
  - lastname

**Reduced Instance Graph**

1. George Jones
2. Ana Stone
3. Julia Jones
4. James Deville
5. Mary Deville
6. P5
7. P6

- George
  - lastname
  - parent
- Ana
  - lastname
  - parent
- Julia
  - lastname
  - parent
- James
  - lastname
  - parent
- Mary
  - lastname
  - parent
- P5
  - name
  - lastname
  - parent
- P6
  - name
  - lastname

- George Jones
  - lastname
  - parent
- Ana Stone
  - lastname
  - parent
- Julia Jones
  - lastname
  - parent
- James Deville
  - lastname
  - parent
- Mary Deville
  - lastname
  - parent
GOAL

<table>
<thead>
<tr>
<th>NAME</th>
<th>LASTNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>George</td>
<td>Jones</td>
</tr>
<tr>
<td>Ana</td>
<td>Stone</td>
</tr>
<tr>
<td>Julia</td>
<td>Jones</td>
</tr>
<tr>
<td>James</td>
<td>Deville</td>
</tr>
<tr>
<td>David</td>
<td>Deville</td>
</tr>
<tr>
<td>Mary</td>
<td>Deville</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PERSON</th>
<th>PARENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Julia</td>
<td>George</td>
</tr>
<tr>
<td>Julia</td>
<td>Ana</td>
</tr>
<tr>
<td>David</td>
<td>James</td>
</tr>
<tr>
<td>David</td>
<td>Julia</td>
</tr>
<tr>
<td>Mary</td>
<td>James</td>
</tr>
<tr>
<td>Mary</td>
<td>Julia</td>
</tr>
</tbody>
</table>

**Schema**

**Instance**

[Diagram showing relationships between individuals with names and last names, and their parent-child connections.]
### Gram

#### Schema

<table>
<thead>
<tr>
<th>NAME</th>
<th>LASTNAME</th>
<th>PERSON</th>
<th>PARENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>George</td>
<td>Jones</td>
<td>Julia</td>
<td>George</td>
</tr>
<tr>
<td>Ana</td>
<td>Stone</td>
<td>Julia</td>
<td>Ana</td>
</tr>
<tr>
<td>Julia</td>
<td>Jones</td>
<td>David</td>
<td>James</td>
</tr>
<tr>
<td>James</td>
<td>Deville</td>
<td>David</td>
<td>Julia</td>
</tr>
<tr>
<td>David</td>
<td>Deville</td>
<td>Mary</td>
<td>James</td>
</tr>
<tr>
<td>Mary</td>
<td>Deville</td>
<td>Mary</td>
<td>Julia</td>
</tr>
</tbody>
</table>

#### Instance

![Instance Diagram]

1. **George Jones**
2. **Ana Stone**
3. **Julia Jones**
4. **James Deville**
5. **David Deville**
6. **Mary Deville**
<table>
<thead>
<tr>
<th>Model</th>
<th>Directed</th>
<th>Labeled Nodes/Edges</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOOD</td>
<td>directed</td>
<td>labeled nodes/edges</td>
<td>simple</td>
</tr>
<tr>
<td>GMOD</td>
<td>directed</td>
<td>labeled nodes/edges</td>
<td>simple</td>
</tr>
<tr>
<td>G-Log</td>
<td>directed</td>
<td>labeled nodes/edges</td>
<td>simple</td>
</tr>
<tr>
<td>Gram</td>
<td>directed</td>
<td>labeled nodes/edges</td>
<td>simple</td>
</tr>
<tr>
<td>Hypernode Model</td>
<td>directed</td>
<td>labeled nodes/edges</td>
<td>hypernodes</td>
</tr>
<tr>
<td>Simatic-XT</td>
<td>directed</td>
<td>labeled nodes/edges</td>
<td>hypernodes</td>
</tr>
<tr>
<td>GGL</td>
<td>directed</td>
<td>labeled nodes/edges</td>
<td>hypernodes</td>
</tr>
<tr>
<td>GROOVY</td>
<td>directed</td>
<td>labeled nodes/edges</td>
<td>hypergraphs</td>
</tr>
</tbody>
</table>
Data Structures

Simple flat graphs and hypernodes [AG08]:

- Simple flat graphs make it hard to present the information clearly.
- Hypernodes allow one to view the information at different scales, and allow one to compose complex relations (objects) from primitive objects.
- Thus, a drastic difference in expressiveness and extensibility.
Integrity Constraints

• Schema-instance consistency
  
  (1) the instance should only contain entities defined by the schema
  
  (2) the entities of the instance should have only relations defined for its entity types

• Object-identity and referential integrity
  
  - similar to primary key and foreign key

• Functional dependencies
  
  - similar to functional dependencies in the relational data model
Query Languages

• Adjacency queries
  - Node/edge adjacency
  - neighborhood
• Reachability queries (between two nodes)
  - fixed-length paths
  - regular simple paths
  - shortest path
• Pattern matching queries
• Summarization queries
Other queries [Woo12]

\[ \text{conjunctive query} \]

\[ \text{ans}(x) \leftarrow (x, \text{hasWon}, \text{Nobel}), (x, \text{hasWon}, \text{Booker}) \]
Other queries [Woo12]

regular path query

\[
\text{ans}(x,y) \leftarrow (x, (\text{citizenOf} \lor ((\text{bornIn} \lor \text{livesIn}) \cdot \text{locatedIn}^*)), y)
\]
Other queries [Woo12]

conjunctive regular path query

\[
\text{ans}(x, y) \leftarrow (x, \text{hasWon}, \text{Nobel}), (x, \text{hasWon}, \text{Booker}) \\
(x, (\text{citizenOf}|((\text{bornIn}\mid\text{livesIn}) \cdot \text{locatedIn}^*)), y)
\]
Other queries \[\text{[Woo12]}\]

An extended conjunctive regular path query is defined as:

\[
\text{ans}(x, y) \leftarrow (\text{Coetzee}, \pi, y), (x, \pi, y), \Sigma^*(\pi)
\]
Summary

• Graph data model in three components:
  • data structure types
  • integrity rules
  • operators

• Graph data model provides support for data of more complexity and operations of more expressiveness

• Specialized graph storage structure and efficient graph algorithms.

• Unlike relational data model, graph data models have not converged to a standard. So some time before the "graph SQL".
Graphs in Our Daily Life

• Graphs in real world are typically very big:
  – Billions of vertices and edges and rich meta data.
How to efficiently process large graphs?
Potential Old Solutions

• Leverage existing general-purpose distributed dataflow frameworks.
Potential Old Solutions

• Leverage existing general-purpose distributed dataflow frameworks.

Hadoop MapReduce
Potential Old Solutions

• Leverage existing general-purpose distributed dataflow frameworks.
  – MapReduce: single-stage & on-disk processing
Potential Old Solutions

• Leverage existing general-purpose distributed dataflow frameworks.
  – MapReduce: single-stage & on-disk processing

Read from HDFS (disk)
Potential Old Solutions

- Leverage existing general-purpose distributed dataflow frameworks.
  - MapReduce: single-stage & on-disk processing

Write intermediate results to disk
Potential Old Solutions

• Leverage existing general-purpose distributed dataflow frameworks.
  – MapReduce: single-stage & on-disk processing

Read intermediate results from disk
Potential Old Solutions

• Leverage existing general-purpose distributed dataflow frameworks.
  – MapReduce: single-stage & on-disk processing

  Write final results to HDFS (disk)
Potential Old Solutions

• Leverage existing general-purpose distributed dataflow frameworks.
  – MapReduce: single-stage & on-disk processing
Potential Old Solutions

• Leverage existing general-purpose distributed dataflow frameworks.
  – MapReduce: single-stage & on-disk processing.

• Graph algorithms:
  – Repeatedly and randomly access graphs.
Potential Old Solutions

• Leverage existing general-purpose distributed dataflow frameworks.
  – MapReduce: single-stage & on-disk processing.

• Graph algorithms:
  – Repeatedly and randomly access graphs.

Require specialized graph processing systems
Rest of Talk

• Pregel [SIGMOD 2010]
• PowerGraph [OSDI 2012]
• Other Works
Rest of Talk

• Pregel [SIGMOD 2010]
• PowerGraph [OSDI 2012]
• Other Works

The first well-known distributed graph processing system
Pregel Abstraction
Pregel Abstraction

- A **vertex program** runs on each vertex.
Pregel Abstraction

• A vertex program runs on each vertex.
• Vertex programs are run in parallel.
Pregel Abstraction

• A *vertex program* runs on each vertex.
• Vertex programs are run *in parallel*.
• Vertex programs interact with other vertices using *messages* along edges.
Pregel Abstraction

- A **vertex program** runs on each vertex.
- Vertex programs are run in parallel.
- Vertex programs interact with other vertices using messages along edges.
Model of Computation: Input

• A directed graph
Model of Computation: Parallel

• A typical Pregel computation consists of a sequence of supersteps separated by global synchronization points.
Model of Computation: Compute

- Within each superstep, vertices compute (run the same vertex programs) in parallel.
Model of Computation: Compute

• Within each superstep, vertices compute (run the same vertex programs) in parallel.

• A vertex can:
  – Modify its vertex value
  – Modify values of its outgoing edges
  – Receive messages sent to it in previous superstep
  – Send messages to other vertices (to be received in next superstep)
  – Mutate the topology of the graph
Model of Computation: Terminate

- Vertex State Machine

- The algorithm as a whole terminates when all vertices are simultaneously inactive and there are no messages to transmit.
Page Rank Algorithm

\[ R[i] = 0.15 + \sum_{j \in Nbrs(i)} W_{ji}R[j] \]

- Rank of user \( i \)
- Weight sum of neighbors’ sum
Example: Page Rank in Pregel

while (superstep() < 30) :

![Graph](image)
Example: Page Rank in Pregel

```python
while (superstep() < 30) :
    // Receive all messages
    total = 0
    foreach (msg in messages) :
        total = total + msg
```
Example: Page Rank in Pregel

```plaintext
while (superstep() < 30):
    // Receive all messages
    total = 0
    foreach (msg in messages):
        total = total + msg

    // Update the rank of this vertex
    R[i] = 0.15 + total
```
Example: Page Rank in Pregel

\[
\text{while } (\text{superstep}() < 30) : \\
    // Receive all messages \\
    \text{total} = 0 \\
    \text{foreach } (\text{msg} \text{ in messages}) : \\
        \text{total} = \text{total} + \text{msg} \\
    // Update the rank of this vertex \\
    R[i] = 0.15 + \text{total} \\
    // Send new messages to neighbors \\
    \text{foreach } (j \text{ in out\_neighbors}[i]) : \\
        \text{Send } \text{msg} \text{ (}R[i] * W_{ij}\text{) to vertex } j
\]
Example: Page Rank in Pregel

```python
while (superstep() < 30):
    // Receive all messages
    total = 0
    foreach (msg in messages):
        total = total + msg

    // Update the rank of this vertex
    R[i] = 0.15 + total

    // Send new messages to neighbors
    foreach (j in out_neighbors[i]):
        Send msg (R[i] * W_{ij}) to vertex j
```

One Superstep
Rest of Talk

• Pregel [SIGMOD 2010]
• PowerGraph [OSDI 2012]
• Other Works

A distributed graph processing system for Natural Graphs
Natural Graphs

• Graphs derived from natural phenomena
Natural Graphs

- Graphs derived from natural phenomena
- Property: power-law degree distribution

![Graph showing power-law degree distribution](image-url)
Natural Graphs

- Graphs derived from natural phenomena
- Property: power-law degree distribution
Pregel on Natural Graphs

• Note: a single vertex program in Pregel cannot be distributed over multiple machines.
Pregel on Natural Graphs

• High-degree vertices are difficult to process.

Sequentially process adjacent vertices
Pregel on Natural Graphs

- High-degree vertices are difficult to process.

Send/Receive too many messages
Pregel on Natural Graphs

• High-degree vertices are difficult to process.

Edge meta-data too large for single machine
Pregel on Natural Graphs

- High-degree vertices are difficult to process.

Straggler problem due to synchronous execution
Pregel on Natural Graphs

• High-degree vertices are difficult to process.
• Natural graphs are difficult to partition.
Pregel on Natural Graphs

- High-degree vertices are difficult to process.
- Natural graphs are difficult to partition.
Pregel on Natural Graphs

- High-degree vertices are difficult to process.
- Natural graphs are difficult to partition.

Partition the graph to 3 machines
Pregel on Natural Graphs

- High-degree vertices are difficult to process.
- Natural graphs are difficult to partition.

Balance computation and storage
Minimize network communication

Partition the graph to 3 machines
Pregel on Natural Graphs

• High-degree vertices are difficult to process.
• Natural graphs are difficult to partition.

Balanced $P$-way edge-cut: *evenly* assign vertices to $P$ machines and *minimize* the number of edges spanning machines.
Pregel on Natural Graphs

- High-degree vertices are difficult to process.
- Natural graphs are difficult to partition.

Balanced $P$-way edge-cut: *evenly* assign vertices to $P$ machines and \textit{minimize} the number of edges spanning machines.

\textbf{NP-hard}
Pregel on Natural Graphs

• High-degree vertices are difficult to process.
• Natural graphs are difficult to partition.

Pregel resorts to random hashed partitioning:
\[ \text{hash}(\text{Vertex ID}) \mod N \]
Pregel on Natural Graphs

- High-degree vertices are difficult to process.
- Natural graphs are difficult to partition.

Pregel resorts to random hashed partitioning:
\[ \text{hash(Vertex ID)} \mod N \]

Easy to implement
Pregel on Natural Graphs

• High-degree vertices are difficult to process.
• Natural graphs are difficult to partition.

10 Machines → 90% of edges cut
100 Machines → 99% of edges cut
PowerGraph in 1 Slide
PowerGraph in 1 Slide

• **GAS Decompositions**: distribute vertex-programs
  – Move computation to data
  – Parallelize **high-degree** vertices
PowerGraph in 1 Slide

• **GAS Decompositions**: distribute vertex-programs
  – Move computation to data
  – Parallelize *high-degree* vertices

• **Vertex Partitioning**
  – Effectively distribute large power-law graphs
A Common Pattern for Vertex-Programs

```
while (superstep() < 30) :
    // Receive all messages
    total = 0
    foreach (msg in messages) :
        total = total + msg

    // Update the rank of this vertex
    R[i] = 0.15 + total

    // Send new messages to neighbors
    foreach (j in out_neighbors[i]) :
        Send msg (R[i] * W_{ij}) to vertex j
```
A Common Pattern for Vertex-Programs

\[
\text{while (superstep() < 30) :}
\]

// Receive all messages
total = 0
foreach (msg in messages) :
    total = total + msg

// Update the rank of this vertex
R[i] = 0.15 + total

// Send new messages to neighbors
foreach (j in out_neighbors[i]) :
    Send msg (R[i] \times W_{ij}) to vertex j

**Gather**: accumulate information about neighborhood
A Common Pattern for Vertex-Programs

while (superstep() < 30) :

    // Receive all messages
    total = 0
    foreach (msg in messages) :
        total = total + msg

    // Update the rank of this vertex
    R[i] = 0.15 + total

    // Send new messages to neighbors
    foreach (j in out_neighbors[i]) :
        Send msg (R[i] * W_{ij}) to vertex j

Gather: accumulate information about neighborhood

Apply: apply the accumulated value to central vertex
A Common Pattern for Vertex-Programs

**while** (superstep() < 30) :

// Receive all messages
total = 0
foreach (msg in messages) :
    total = total + msg

// Update the rank of this vertex
R[i] = 0.15 + total

// Send new messages to neighbors
foreach (j in out_neighbors[i]) :
    Send msg (R[i] * W_{ij}) to vertex j
A Common Pattern for Vertex-Programs

```python
while (superstep() < 30):
    // Receive all messages
    total = 0
    foreach (msg in messages):
        total = total + msg
    // Update the rank of this vertex
    R[i] = 0.15 + total
    // Send new messages to neighbors
    foreach (j in out_neighbors[i]):
        Send msg (R[i] * Wij) to vertex j
```

**GAS Decomposition**

- **Gather**: accumulate information about neighborhood
- **Apply**: apply the accumulated value to central vertex
- **Scatter**: update adjacent edges and vertices
Distributed Execution of a PowerGraph Vertex-Program

\[ R[i] = 0.15 + \sum_{j \in Nbrs(i)} W_{ji} R[j] \]
Distributed Execution of a PowerGraph Vertex-Program
Distributed Execution of a PowerGraph Vertex-Program
Distributed Execution of a PowerGraph Vertex-Program
Distributed Execution of a PowerGraph Vertex-Program

Machine 1

Machine 2

Machine 3

Machine 4

Master

Mirror

Mirror
Distributed Execution of a PowerGraph Vertex-Program

Gather

Machine 1

Master

Machine 2

Mirror

Machine 3

Mirror

Machine 4

Mirror
Distributed Execution of a PowerGraph Vertex-Program

Gather
Distributed Execution of a PowerGraph Vertex-Program

\[ \sum_1 \sum_2 \sum_3 \sum_4 \]

Gather

Machine 1

Master

Machine 2

Mirror

Machine 3

Mirror

Machine 4
Distributed Execution of a PowerGraph Vertex-Program

\[ \sum 1 + \sum 2 + \sum 3 + \sum 4 \]

Gather

Machine 1

Master

Machine 2

Mirror

Machine 3

Mirror

Machine 4

Mirror
Distributed Execution of a PowerGraph Vertex-Program

Gather

Machine 1

Master

Machine 2

Mirror

Machine 3

Mirror

Machine 4
Distributed Execution of a PowerGraph Vertex-Program

**Gather**

**Apply**

Machine 1

Machine 2

Machine 3

Machine 4
Distributed Execution of a PowerGraph Vertex-Program

Gather

Apply

Machine 1

Machine 2

Machine 3

Machine 4
Distributed Execution of a PowerGraph Vertex-Program

Gather

Apply
Distributed Execution of a PowerGraph Vertex-Program

Gather

Apply
Distributed Execution of a PowerGraph Vertex-Program

Gather

Apply

Scatter
New Approach to Partitioning
New Approach to Partitioning

• Rather than cut edges
New Approach to Partitioning

• Rather than cut edges, we cut vertices.
Constructing Vertex Cuts

• *Evenly* assign *edges* to machines
  
  – Minimize machines spanned by each vertex

Balanced $p$-way Vertex-Cut
Constructing Vertex Cuts

• **Evenly** assign *edges* to machines
  – Minimize machines spanned by each vertex

• Assign each edge *as it is loaded*
  – Touch each edge only once
Constructing Vertex Cuts

• **Evenly** assign *edges* to machines
  – Minimize machines spanned by each vertex

• Assign each edge *as it is loaded*
  – Touch each edge only once

• Propose three *distributed* approaches
  – The simplest *Random* Edge Placement performs well on power graphs
Rest of Talk

• Pregel [SIGMOD 2010]
• PowerGraph [OSDI 2012]
• Other Works
GraphX [OSDI 2014]

- Modern data analysis
GraphX [OSDI 2014]

- Graphs

![Diagram of GraphX process]

1. Raw Wikipedia
2. XML
3. Link Table
4. Hyperlinks
5. PageRank
6. Top 20 Pages
7. Top Communities
8. Discussion Table
9. Editor Graph
10. Community Detection
11. User Community
GraphX [OSDI 2014]

- Tables
GraphX [OSDI 2014]

- Separate systems to process graphs
GraphX [OSDI 2014]

• An **embedded** graph processing framework built on top of Apache Spark
PowerLyra [EuroSys 2015]

• Differentiate high-degree and low-degree vertices on natural graphs
  – Graph placement:
    • Hybrid-cuts (vertex & edge)
  – Vertex computation:
    • Local for low-degree vertices (Pregel)
    • Distributed for high-degree vertices (PowerGraph)