An Introduction to Graph Analytics Platforms
(Very Short Version)

M. Tamer Özsu

University of Waterloo
David R. Cheriton School of Computer Science
Outline

1. Introduction – Graph Types
2. Property Graph Processing
   - Classification
   - Online querying
   - Offline analytics
3. Graph Analytics Approaches
   - MapReduce & Variants
   - Classification of Native Approaches
4. Graph Analytics Systems
5. OLAP-Style Analytics
   - Graph Summarization
   - Snapshot-based Aggregation
   - Graph Cube
   - Pagrol
   - Gagg Model
Outline

1. Introduction – Graph Types
2. Property Graph Processing
   - Classification
   - Online querying
   - Offline analytics
3. Graph Analytics Approaches
   - MapReduce & Variants
   - Classification of Native Approaches
4. Graph Analytics Systems
5. OLAP-Style Analytics
   - Graph Summarization
   - Snapshot-based Aggregation
   - Graph Cube
   - Pagrol
   - Gagg Model
Graph Types

Property graph
Graph Types

Property graph

RDF graph

- Workload: Online queries and analytic workloads
- Query execution: Varies

- Workload: SPARQL queries
- Query execution: subgraph matching by homomorphism
Outline

1. Introduction – Graph Types
2. Property Graph Processing
   - Classification
   - Online querying
   - Offline analytics
3. Graph Analytics Approaches
   - MapReduce & Variants
   - Classification of Native Approaches
4. Graph Analytics Systems
5. OLAP-Style Analytics
   - Graph Summarization
   - Snapshot-based Aggregation
   - Graph Cube
   - Pagrol
   - Gagg Model
Outline

1. Introduction – Graph Types
2. Property Graph Processing
   - Classification
   - Online querying
   - Offline analytics
3. Graph Analytics Approaches
   - MapReduce & Variants
   - Classification of Native Approaches
4. Graph Analytics Systems
5. OLAP-Style Analytics
   - Graph Summarization
   - Snapshot-based Aggregation
   - Graph Cube
   - Pagrol
   - Gagg Model
Example Design Points

Graph Dynamism
- Static Graphs
- Dynamic Graphs
- Streaming Graphs
- Evolving Graphs

Algorithm Types
- Offline
- Online
- Dynamic
- Incremental
- Streaming

Workload Types
- Online Queries
- Analytics Workloads

Compute the query result/perform analytic computation over the graph as it exists.
Example Design Points

Graph Dynamism
- Static Graphs
- Dynamic Graphs
  - Streaming Graphs
  - Evolving Graphs

Algorithm Types
- Offline
- Online
  - Dynamic
    - Streaming
    - Incremental
- Batch

Workload Types
- Online Queries
- Analytics Workloads

Compute the query result/perform analytic computation on each snapshot from scratch.
Continuous compute the query result/perform analytic computation as the input changes.
Dynamic (or batch-dynamic) algorithms do not make sense for static graphs.
Scale-up or Scale-out?

- Scale-up: Single machine execution
  - Graph datasets are small and can fit in a single machine – even in main memory
  - Single machine avoids parallel execution complexities
Scale-up or Scale-out?

- **Scale-up**: Single machine execution
  - Graph datasets are small and can fit in a single machine – even in main memory
  - Single machine avoids parallel execution complexities
- **Scale-out**: Parallel execution
  - Graph data sets grow when they are expanded to their storage formats

| Dataset          | |V|  | |E|  | Regular size | Single Machine* |
|------------------|-----------------|-----------------|----------------|-----------------|
| Live Journal     | 4,847,571       | 68,993,773      | 1.08GB         | 6.3GB           |
| USA Road         | 23,947,347      | 58,333,344      | 951MB          | 9.09GB          |
| Twitter          | 41,652,230      | 1,468,365,182   | 26GB           | 128 GB          |
| UK0705           | 82,240,700      | 2,829,101,180   | 48GB           | 247GB           |
| World Road       | 682,496,072     | 717,016,716     | 15GB           | 194GB           |
| CommonCrawl2014  | 1,727,000,000   | 64,422,000,000  | 1.3TB          | Out of memory   |

* Using (PowerLyra)
Scale-up or Scale-out?

- **Scale-up:** Single machine execution
  - Graph datasets are small and can fit in a single machine – even in main memory
  - Single machine avoids parallel execution complexities

- **Scale-out:** Parallel execution
  - Graph data sets grow when they are expanded to their storage formats
  - Workstations big enough to handle even smaller datasets are still expensive
  - Some graphs are very large: Alibaba: several billion vertices, > 100 million edges
  - Dataset size may not be the determinant ⇒ parallelizing computation is important

Dataset | V  | E  | Size
---|---|---|---
Live Journal | 4,847,571 | 68,993,773 | 1.08GB
USA Road | 23,947,347 | 58,333,344 | 951MB
Twitter | 41,652,230 | 1,468,365,182 | 26GB
UK0705 | 82,240,700 | 2,829,101,180 | 48GB
World Road | 682,496,072 | 717,016,716 | 15GB
CommonCrawl2014 | 1,727,000,000 | 64,422,000,000 | 1.3TB

*Using (PowerLyra)*

We focus on parallel graph analytics systems
Scale-up: Single machine execution
- Graph datasets are small and can fit in a single machine – even in main memory
- Single machine avoids parallel execution complexities

Scale-out: Parallel execution
- Graph data sets grow when they are expanded to their storage formats
- Workstations big enough to handle even smaller datasets are still expensive
- Some graphs are very large: Alibaba: several billion vertices, > 100 million edges
- Dataset size may not be the determinant ⇒ parallelizing computation is important

We focus on parallel graph analytics systems
Graph Partitioning

- **Edge-cut (vertex-disjoint)**
  - Achieve disjoint partitions by allocating each vertex to a partition
  - **Objective 1:** Partitions should be balanced
  - **Objective 2:** Minimize edge-cuts (to reduce communication)
  - Good for graphs with low-degree vertices, not for power-law graphs
  - Examples: Hashing, METIS [Karypis and Kumar, 1995], label propagation algorithms [Ugander and Backstrom, 2013]

- **Vertex-cut (edge-disjoint)**
  - Achieve disjoint partitions by allocating each edge to a partition and replicating vertices as necessary
  - **Objective 1:** Partitions should be balanced
  - **Objective 2:** Minimize vertex-cuts (to reduce replica cost)
  - Perform better on power-law graphs
  - Examples: Hashing, greedy algorithms [Gonzalez et al., 2012]
Graph Partitioning

- **Edge-cut (vertex-disjoint)**
  - Achieve disjoint partitions by allocating each vertex to a partition
  - **Objective 1**: Partitions should be balanced
  - **Objective 2**: Minimize edge-cuts (to reduce communication)
  - Good for graphs with low-degree vertices, not for power-law graphs
  - Examples: Hashing, METIS [Karypis and Kumar, 1995], label propagation algorithms [Ugander and Backstrom, 2013]

- **Vertex-cut (edge-disjoint)**
  - Achieve disjoint partitions by allocating each edge to a partition and replicating vertices as necessary
  - **Objective 1**: Partitions should be balanced
  - **Objective 2**: Minimize vertex-cuts (to reduce replica cost)
  - Perform better on power-law graphs
  - Examples: Hashing, greedy algorithms [Gonzalez et al., 2012]
Graph Partitioning

- **Edge-cut (vertex-disjoint)**
  - Achieve disjoint partitions by allocating each vertex to a partition
  - **Objective 1:** Partitions should be balanced
  - **Objective 2:** Minimize edge-cuts (to reduce communication)
  - Good for graphs with low-degree vertices, not for power-law graphs
  - Examples: Hashing, METIS [Karypis and Kumar, 1995], label propagation algorithms [Ugander and Backstrom, 2013]

- **Vertex-cut (edge-disjoint)**
  - Achieve disjoint partitions by allocating each edge to a partition and replicating vertices as necessary
  - **Objective 1:** Partitions should be balanced
  - **Objective 2:** Minimize vertex-cuts (to reduce replica cost)
  - Perform better on power-law graphs
  - Examples: Hashing, greedy algorithms [Gonzalez et al., 2012]

- **Hybrid**
  - Edge-cut for low-degree vertices/vertex-cut for high-degree ones
  - PowerLyra [Chen et al., 2015]
Graph Workloads

Online graph querying
- Reachability
- Single source shortest-path
- Subgraph matching
- SPARQL queries

Offline graph analytics
- PageRank
- Clustering
- Connected components
- Diameter finding
- Graph colouring
- All pairs shortest path
- Graph pattern mining
- Machine learning algorithms (Belief propagation, Gaussian non-negative matrix factorization)
Can you reach film_1267 from film_2014?
Is there a book whose rating is > 4.0 associated with a film that was directed by Stanley Kubrick?
Think of Facebook graph and finding friends of friends.
Outline

1. Introduction – Graph Types
2. Property Graph Processing
   • Classification
   • Online querying
   • Offline analytics
3. Graph Analytics Approaches
   • MapReduce & Variants
   • Classification of Native Approaches
4. Graph Analytics Systems
5. OLAP-Style Analytics
   • Graph Summarization
   • Snapshot-based Aggregation
   • Graph Cube
   • Pagrol
   • Gagg Model
A web page is important if it is pointed to by other important pages.

\[ r(P_i) = (1 - d) + d \sum_{P_j \in B_{P_i}} \frac{r(P_j)}{|F_{P_j}|} \]

(let \( d = 1 \))

\[ r(P_2) = \frac{r(P_1)}{2} + \frac{r(P_3)}{3} \]

\[ r_{k+1}(P_i) = \sum_{P_j \in B_{P_i}} \frac{r_k(P_j)}{|F_{P_j}|} \]

\( B_{P_i} \): in-neighbours of \( P_i \)
\( F_{P_i} \): out-neighbours of \( P_i \)
A web page is important if it is pointed to by other important pages.

\[ r_{k+1}(P_i) = \sum_{P_j \in B_{P_i}} \frac{r_k(P_j)}{|F_{P_j}|} \]

<table>
<thead>
<tr>
<th>Iteration 0</th>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Rank at Iter. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_0(P_1) = 1/6 )</td>
<td>( r_1(P_1) = 1/18 )</td>
<td>( r_2(P_1) = 1/36 )</td>
<td>5</td>
</tr>
<tr>
<td>( r_0(P_2) = 1/6 )</td>
<td>( r_1(P_2) = 5/36 )</td>
<td>( r_2(P_2) = 1/18 )</td>
<td>4</td>
</tr>
<tr>
<td>( r_0(P_3) = 1/6 )</td>
<td>( r_1(P_3) = 1/12 )</td>
<td>( r_2(P_3) = 1/36 )</td>
<td>5</td>
</tr>
<tr>
<td>( r_0(P_4) = 1/6 )</td>
<td>( r_1(P_4) = 1/4 )</td>
<td>( r_2(P_4) = 17/72 )</td>
<td>1</td>
</tr>
<tr>
<td>( r_0(P_5) = 1/6 )</td>
<td>( r_1(P_5) = 5/36 )</td>
<td>( r_2(P_5) = 11/72 )</td>
<td>3</td>
</tr>
<tr>
<td>( r_0(P_6) = 1/6 )</td>
<td>( r_1(P_6) = 1/6 )</td>
<td>( r_2(P_6) = 14/72 )</td>
<td>2</td>
</tr>
</tbody>
</table>
Outline

1. Introduction – Graph Types
2. Property Graph Processing
   - Classification
   - Online querying
   - Offline analytics
3. Graph Analytics Approaches
   - MapReduce & Variants
   - Classification of Native Approaches
4. Graph Analytics Systems
5. OLAP-Style Analytics
   - Graph Summarization
   - Snapshot-based Aggregation
   - Graph Cube
   - Pagrol
   - Gagg Model
1 Introduction – Graph Types

2 Property Graph Processing
   - Classification
   - Online querying
   - Offline analytics

3 Graph Analytics Approaches
   - MapReduce & Variants
   - Classification of Native Approaches

4 Graph Analytics Systems

5 OLAP-Style Analytics
   - Graph Summarization
   - Snapshot-based Aggregation
   - Graph Cube
   - Pagrol
   - Gagg Model

© M. Tamer Özsu
Can MapReduce be Used for Graph Analytics?

- Yes; map and reduce functions can be written for graph analytics workloads
  - Scalable Graph processing Class $SGC$ [Qin et al., 2014]
  - Connected component computation [Kiveris et al., 2014; Rastogi et al., 2013]

- Not suitable for iterative processing due to data movement at each stage
  - No guarantee that computation will be assigned to the same worker nodes in the next round

- High I/O cost
  - Need to save in storage system (HDFS) intermediate results of each iteration
Yes; map and reduce functions can be written for graph analytics workloads
- Scalable Graph processing Class \textit{SGC} [Qin et al., 2014]
- Connected component computation [Kiveris et al., 2014; Rastogi et al., 2013]

Not suitable for iterative processing due to data movement at each stage
- No guarantee that computation will be assigned to the same worker nodes in the next round

High I/O cost
- Need to save in storage system (HDFS) intermediate results of each iteration

There are systems that address these concerns
- HaLoop [Bu et al., 2010, 2012]
- GraphX over Spark [Gonzalez et al., 2014]
Spark objectives

- Better support for iterative programs
- Provide a complete ecosystem
- Similar abstraction (to MapReduce) for programming
- Maintain MapReduce fault-tolerance and scalability
Spark objectives

- Better support for iterative programs
- Provide a complete ecosystem
- Similar abstraction (to MapReduce) for programming
- Maintain MapReduce fault-tolerance and scalability

Fundamental concepts

- RDD: Reliable Distributed Datasets
- Caching of working set
- Maintaining lineage for fault-tolerance
GraphX

- Built on top of Spark
- Objective is to combine data analytics with graph processing
  - Unify computation on tables and graphs
- Carefully convert graph to tabular representation
- Native GraphX API or can accommodate vertex-centric computation

Apache Spark

Native Spark Apps

Spark SQL
Spark Streaming
MLlib (machine learning)
GraphX (graph processing)

Vertex-centric API

© M. Tamer Özsu TD-LSG (2018/08/31) 22 / 46
GraphX: Representation of Graphs as Tables
GraphX: Representation of Graphs as Tables

Partition 1

Partition 2

Edge-disjoint partitioning
GraphX: Representation of Graphs as Tables

Partition 1

Partition 2

Edge-disjoint partitioning

Vertex Table

(RDD)

v-prop: vertex prop.
GraphX: Representation of Graphs as Tables

Partition 1

A
B
C
D
E
F
G
H
I

Partition 2

Machine 1

Vertex Table

A
B
C
D
E
F
G
H
I

Edge Table

A
B
C
D
E
F
G
H
I

Edge-disjoint partitioning

(RDD)
v-prop: vertex prop.

(RDD)
e-prop: edge prop.
GraphX: Representation of Graphs as Tables

Partition 1

Partition 2

Edge-disjoint partitioning

Vertex Table (RDD)

A v-prop
B v-prop
C v-prop
D v-prop
E v-prop
F v-prop
G v-prop
H v-prop
I v-prop

Edge Table (RDD)

A e-prop B
A e-prop C
B e-prop A
C e-prop A
D e-prop A
E e-prop D
F e-prop G
G e-prop F
H e-prop E
I e-prop F

Joining vertices and edges
Move vertices to edges

© M. Tamer Özsu

TD-LSG

(2018/08/31) 23 / 46
### GraphX: Representation of Graphs as Tables

**Partition 1**

- **Machine 1**
  - **Routing Table** (RDD)
    - A: 1 2
    - B: 1
    - ... I: 1
    - F: 1 2
    - D: 2
    - E: 2
    - J: 2

- **Vertex Table** (RDD)
  - A: v-prop
  - B: v-prop
  - C: v-prop
  - D: v-prop
  - E: v-prop
  - F: v-prop
  - G: v-prop
  - H: v-prop
  - I: v-prop

- **Edge Table** (RDD)
  - A: e-prop B
  - A: e-prop C
  - ... F: e-prop G

**Partition 2**

- **Machine 2**
  - **Routing Table** (RDD)
    - A: 1 2
    - B: 1
    - ... I: 1
    - F: 1 2
    - D: 2
    - E: 2
    - J: 2

- **Vertex Table** (RDD)
  - A: v-prop
  - B: v-prop
  - C: v-prop
  - D: v-prop
  - E: v-prop
  - F: v-prop
  - G: v-prop
  - H: v-prop
  - I: v-prop

- **Edge Table** (RDD)
  - A: e-prop D
  - A: e-prop E
  - ... E: e-prop F

**Edge-disjoint partitioning**

v-prop: vertex prop.

© M. Tamer Özsu
Outline

1. Introduction – Graph Types
2. Property Graph Processing
   - Classification
   - Online querying
   - Offline analytics
3. Graph Analytics Approaches
   - MapReduce & Variants
   - Classification of Native Approaches
4. Graph Analytics Systems
5. OLAP-Style Analytics
   - Graph Summarization
   - Snapshot-based Aggregation
   - Graph Cube
   - Pagrol
   - Gagg Model

© M. Tamer Özsu
TD-LSG (2018/08/31) 24 / 46
Classification of Graph Processing Systems

- Programming model
- Computation model
Classification of Graph Processing Systems

- Programming model
- Computation model

<table>
<thead>
<tr>
<th>Programming Model</th>
<th>Computation Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex-centric</td>
<td>Asynchronous</td>
</tr>
<tr>
<td>Partition-centric</td>
<td>Block Synchronous Parallel (BSP)</td>
</tr>
<tr>
<td>Edge-centric</td>
<td>Gather-Apply Scatter (GAS)</td>
</tr>
</tbody>
</table>
Classification of Graph Processing Systems

- Programming model
- Computation model

Diagram:

- **Programming Model**
  - Vertex-centric
  - Partition-centric
  - Edge-centric

- **Computation Model**
  - Block Synchronous Parallel (BSP)
  - Asynchronous
  - Gather-Apply Scatter (GAS)

- Vertex-centric
  - BSP
  - Partition-centric
  - Edge-centric
  - ??? (Asynchronous)

- Partition-centric
  - ??? (BSP)
  - Edge-centric
  - ???

- Edge-centric
  - ??? (BSP)
  - ???

[Han, 2015]
Vertex-centric
- Computation on a vertex is the focus
- “Think like a vertex”
- Vertex computation depends on its own state + states of its neighbors
- `Compute(vertex v)`
- `GetValue()`, `WriteValue()`
Programming Models

- Vertex-centric
- Partition-centric (Block-centric)
  - Computation on an entire partition is specified
  - “Think like a block” or “Think like a graph”
  - Aim is to reduce the communication cost among vertices
Programming Models

- **Vertex-centric**
- **Partition-centric (Block-centric)**
- **Edge-centric**
  - Computation is specified on each edge rather than on each vertex or block
  - `Compute(edge e)`
Computational Models

- Block Synchronous Parallel (BSP) [Valiant, 1990]
Computational Models

- Block Synchronous Parallel (BSP) [Valiant, 1990]

  Computation

Each machine performs computation on its graph partition. At the end of each superstep, results are pushed to other workers. Communication barriers are used to synchronize the computation.
Computational Models

- Block Synchronous Parallel (BSP) [Valiant, 1990]

Each machine performs computation on its graph partition.

At the end of each superstep results are pushed to other workers.

<table>
<thead>
<tr>
<th>Superstep 1</th>
<th>Superstep 2</th>
<th>Superstep 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine 1</td>
<td>Machine 2</td>
<td>Machine 3</td>
</tr>
</tbody>
</table>

Communication Barrier
Block Synchronous Parallel (BSP) [Valiant, 1990]

At the end of each superstep, results are pushed to other workers.

Each machine performs computation on its graph partition.

Communication Barrier

Communication Barrier
Computational Models

- **Block Synchronous Parallel (BSP)** [Valiant, 1990]

  Each machine performs computation on its graph partition. At the end of each superstep, results are pushed to other workers.

  - **Superstep 1**
    - Machine 1
    - Machine 2
    - Machine 3
  
  - **Superstep 2**
    - Machine 1
    - Machine 2
    - Machine 3
  
  - **Superstep 3**
    - Machine 1
    - Machine 2
    - Machine 3

  Communication Barrier
Computational Models

- Block Synchronous Parallel (BSP) [Valiant, 1990]
- Asynchronous Parallel
Computational Models

- Block Synchronous Parallel (BSP) [Valiant, 1990]
- Asynchronous Parallel
  - No communication barriers. ✓
  - Uses the *most recent* values. ✓
  - Implemented via distributed locking

![Diagram of machine connections]

© M. Tamer Özsu
Computational Models

- Block Synchronous Parallel (BSP) [Valiant, 1990]
- Asynchronous Parallel
  - No communication barriers. ✓
  - Uses the most recent values. ✓
  - Implemented via distributed locking
  - Consider vertex-centric program

![Graph Diagram]

Compute State
Gather-Apply-Scatter (GAS)
Similar to BSP, but pull-based
Gather: pull state
Apply: Compute function
Scatter: Update state
Updates of states separated from scheduling
Computational Models

- **Block Synchronous Parallel (BSP)** [Valiant, 1990]
- **Asynchronous Parallel**
  - No communication barriers. ✓
  - Uses the *most recent* values. ✓
  - Implemented via distributed locking
  - Consider vertex-centric program

![Graph](image)
Computational Models

- Block Synchronous Parallel (BSP) [Valiant, 1990]
- Asynchronous Parallel
  - No communication barriers. ✓
  - Uses the most recent values. ✓
  - Implemented via distributed locking
  - Consider vertex-centric program

Gather-Apply-Scatter (GAS)
Similar to BSP, but pull-based
Gather: pull state
Apply: compute function
Scatter: update state
Updates of states separated from scheduling
Computational Models

- **Block Synchronous Parallel (BSP)** [Valiant, 1990]
- **Asynchronous Parallel**
  - No communication barriers. ✓
  - Uses the *most recent* values. ✓
  - Implemented via distributed locking
  - Consider vertex-centric program

```
<table>
<thead>
<tr>
<th>Machine 1</th>
<th>Machine 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine 2</td>
<td>Machine 2</td>
</tr>
<tr>
<td>Machine 3</td>
<td>Machine 3</td>
</tr>
</tbody>
</table>
```

```
V1 <--- V0 <--- V2

V3 ---|     |--- V4
```

© M. Tamer Özsu
Computational Models

- Block Synchronous Parallel (BSP) [Valiant, 1990]
- Asynchronous Parallel
  - No communication barriers. ✓
  - Uses the most recent values. ✓
  - Implemented via distributed locking
  - Consider vertex-centric program

![Diagram of vertex-centric program]

Calculations:
- Gather: pull state
- Apply: compute function
- Scatter: update state

Updates of states separated from scheduling
Computational Models

- **Block Synchronous Parallel (BSP)** [Valiant, 1990]
- **Asynchronous Parallel**
  - No communication barriers. ✓
  - Uses the *most recent* values. ✓
  - Implemented via distributed locking
  - Consider vertex-centric program

![Diagram](diagram.png)
Computational Models

- Block Synchronous Parallel (BSP) [Valiant, 1990]
- Asynchronous Parallel
  - No communication barriers. ✓
  - Uses the most recent values. ✓
  - Implemented via distributed locking
  - Consider vertex-centric program

![Diagram of vertex-centric program](image)
Computational Models

- Block Synchronous Parallel (BSP) [Valiant, 1990]
- Asynchronous Parallel
- Gather-Apply-Scatter (GAS)
  - Similar to BSP, but pull-based
  - Gather: pull state
  - Apply: Compute function
  - Scatter: Update state
  - Updates of states separated from scheduling
Real-World Graph Characteristics

- Read-world graphs have skewed vertex degree distribution
  - Common in power-law graphs
  - Problem: imbalanced communication workloads

- Real-world graphs have large diameters
  - Common in road networks, web graphs, terrain meshes
  - Problem: one superstep per hop $\Rightarrow$ too many supersteps

- Real-world graphs have high average vertex degree
  - Common in social networks, mobile communication networks
  - Problem: heavy average communication workloads
Outline

1. Introduction – Graph Types
2. Property Graph Processing
   - Classification
   - Online querying
   - Offline analytics
3. Graph Analytics Approaches
   - MapReduce & Variants
   - Classification of Native Approaches
4. Graph Analytics Systems
5. OLAP-Style Analytics
   - Graph Summarization
   - Snapshot-based Aggregation
   - Graph Cube
   - Pagrol
   - Gagg Model
Vertex-Centric BSP Systems

- “Think like a vertex”
- `Compute(vertex v)`
“Think like a vertex”

Compute(vertex v)

BSP Computation – push state to neighbor vertices at the end of each superstep
“Think like a vertex”
Compute(vertex v)
BSP Computation – push state to neighbor vertices at the end of each superstep
Continue until all vertices are inactive
Vertex state machine
Vertex-Centric BSP Systems

- “Think like a vertex”
- `Compute(vertex v)`
- BSP Computation – push state to neighbor vertices at the end of each superstep
- Continue until all vertices are inactive
- Vertex state machine
- Example systems: Pregel [Malewicz et al., 2010], Apache Giraph, GPS [Salihoglu and Widom, 2013], Mizan [Khayyat et al., 2013], Trinity [?]
“Think like a vertex”
Compute(vertex v)
“Think like a vertex”

*Compute*(vertex v)

Supersteps exist along with synchronization barriers, but ...

*Compute*(vertex v) function can see messages it was sent in the same superstep as well as those that come at the end of the previous superstep
“Think like a vertex”

Compute(vertex v)

Supersteps exist along with synchronization barriers, but ...

Compute(vertex v) function can see messages it was sent in the same superstep as well as those that come at the end of the previous superstep

Consistency of vertex states: distributed locking
“Think like a vertex”

Compute(\(v\))

Supersteps exist along with synchronization barriers, but ...

Compute(\(v\)) function can see messages it was sent in the same superstep as well as those that come at the end of the previous superstep

Consistency of vertex states: distributed locking
"Think like a vertex"

Compute(vertex v)

Supersteps exist along with synchronization barriers, but ...

Compute(vertex v) function can see messages it was sent in the same superstep as well as those that come at the end of the previous superstep

Consistency of vertex states: distributed locking
Vertex-Centric Asynchronous Systems

- “Think like a vertex”
- Compute(vertex v)
- Supersteps exist along with synchronization barriers, but ...
- Compute(vertex v) function can see messages it was sent in the same superstep as well as those that come at the end of the previous superstep
- Consistency of vertex states: distributed locking
“Think like a vertex”
Compute(vertex v)
Supersteps exist along with synchronization barriers, but ...
Compute(vertex v) function can see messages it was sent in the same superstep as well as those that come at the end of the previous superstep
Consistency of vertex states: distributed locking
Consistency issues: no guarantee about input to Compute()
Example systems: GRACE [Wang et al., 2013], GiraphCU [Han and Daudjee, 2015]
Vertex-Centric GAS Systems

- “Think like a vertex”
- Gather phase
  - Gather local computation from neighbours if vertex $v$: called scope $S_v$
- Apply phase
  - $\text{Compute}(v, S_v)$
- Scatter phase
  - $\text{Compute}(v, S_v)$ produces $S'_v$ (scattering state)
- Example: GraphLab [Low et al., 2012]
- Synchronous version
  - Similar to vertex-centric BSP, except pulling $S_v$ rather than pushing
- Asynchronous version different
procedure $\text{GraphLab Async}(G = (V, E, D), T)$

return Modified $G = (V, E, D')$

end procedure

- Graph mutation restricted to vertex states
procedure $\text{GraphLab Async}(G = (V, E, D), \mathcal{T})$

while $\mathcal{T}$ is not empty do

end while

return Modified $G = (V, E, D')$

end procedure

- Graph mutation restricted to vertex states
procedure $\text{GRAPHLAB\_ASYNC}(G = (V, E, D), \mathcal{T})$

while $\mathcal{T}$ is not empty do
    $v \leftarrow \text{RemoveNext}(\mathcal{T})$

end while

return Modified $G = (V, E, D')$

end procedure

- Graph mutation restricted to vertex states

Computing $S_{v}'$ updates (scatters) states of the vertices in scope

Computation of new states $S_{v}'$ separated from computation of new $T'$

RemoveNext() can remove any vertex $\rightarrow$ scheduling separated from state scatter
procedure GraphLabAsync\( (G = (V, E, D), T) \)

while \( T \) is not empty do

\( v \leftarrow \text{RemoveNext}(T) \)

Compute\( (v, S_v) \rightarrow (T', S'_v) \)

end while

return Modified \( G = (V, E, D') \)

end procedure

- Graph mutation restricted to vertex states
- Computing \( S'_v \) updates (scatters) states of the vertices in scope
Vertex-Centric GAS Systems – Asynchronous

**procedure** `GRAPHLAB_ASYNC(G = (V, E, D), T)`

while $T$ is not empty do

$v \leftarrow $ RemoveNext$(T)$

Compute$(v, S_v) \rightarrow (T', S'_v)$

$T \leftarrow T \cup T'$

end while

**return** Modified $G = (V, E, D')$

end procedure

- Graph mutation restricted to vertex states
- Computing $S'_v$ updates (scatters) states of the vertices in scope
- Computation of new states $S'_v$ separated from computation of new $T'$
  - RemoveNext() can remove any vertex → scheduling separated from state scatter
Partition- (Block-)Centric BSP Systems

- Blogel [Yan et al., 2014]: “Think like a block”; also “think like a graph” [Tian et al., 2013]
- Better handles the characteristics of real-world graphs by reducing communication
- Exploit the partitioning of the graph
- Message exchanges only among blocks
- Within a block, run a serial in-memory algorithm; BSP between partitions
Benefits of Partition- (Block-)Centric BSP

- High-degree vertices inside a block send no messages
- Fewer number of supersteps
- Fewer number of blocks than vertices
“Think like an edge”

\textbf{Compute}(\text{edge } e)

- Number of edges $\gg$ number of vertices
  - More computation but perhaps fewer messages
  - Operate on unsorted sequence of edges $\Rightarrow$ no random access

\textbf{X-Stream} [Roy et al., 2013]
Outline

1. Introduction – Graph Types
2. Property Graph Processing
   - Classification
   - Online querying
   - Offline analytics
3. Graph Analytics Approaches
   - MapReduce & Variants
   - Classification of Native Approaches
4. Graph Analytics Systems
5. OLAP-Style Analytics
   - Graph Summarization
   - Snapshot-based Aggregation
   - Graph Cube
   - Pagrol
   - Gagg Model
OLAP Over Graphs

- **OLAP in RDBMS**
  - Usage: Data Warehousing + Business Intelligence
  - Model: Multidimensional cube
  - Operations: Roll-up, drill-down, and slice and dice

- Analytics that we discussed over graphs is much different
- Can we do OLAP-style analytics over graphs?
  - There is some work
    - Graph summarization [Tian et al., 2008]
    - Snapshot-based Aggregation [Chen et al., 2008]
    - Graph Cube [Zhao et al., 2011]
    - Pagrol [?]
    - Gagg Model [Maali et al., 2015]
Acknowledgements

This presentation draws upon collaborative research and discussions with the following colleagues:

Khaled Ammar, U. Waterloo

Xiaofei Zhang, U. Waterloo (U. Memphis)

Khuzaima Daudjee, U. Waterloo

Young Han, U. Waterloo (Google)
Thank you!


References II


